INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO RIO GRANDE DO SUL – CAMPUS PORTO ALEGRE
MESTRADO PROFISSIONAL EM INFORMÁTICA NA EDUCAÇÃO

KAREN CRISTINA BRAGA

CODEIN'PLAY: UM AMBIENTE DE MEDIAÇÃO DO ERRO A PARTIR DA AVALIAÇÃO DE EXERCÍCIOS DE PROGRAMAÇÃO DE COMPUTADORES

PORTO ALEGRE - RS
2020

KAREN CRISTINA BRAGA

CODEIN'PLAY: UM AMBIENTE DE MEDIAÇÃO DO ERRO A PARTIR DA AVALIAÇÃO DE EXERCÍCIOS DE PROGRAMAÇÃO DE COMPUTADORES

Dissertação apresentada junto ao Mestrado Profissional em Informática na Educação do Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul – campus Porto Alegre, como requisito parcial para obtenção do título de Mestre.

Orientador: Fabio Yoshimitsu Okuyama.
Coorientadora: Márcia Amaral Corrêa Ughini Villarroel

PORTO ALEGRE - RS
2020

Dados Internacionais de Catalogação na Publicação (CIP)

B813c Braga, Karen Cristina.
Codein'Play: um ambiente de mediação do erro a partir da avaliação de exercícios de programação de computadores. / Karen Cristina Braga; orientador Fábio Yoshimitsu Okuyama; coorientador: Márcia Amaral Corrêa Ughini Villarroel. – Porto Alegre: 2020.

250 f.

Dissertação (Mestrado) – Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul – Campus Porto Alegre. Mestrado Profissional em Informática na Educação. Porto Alegre, 2020. Orientador: Prof Dr. Fábio Yoshimitsu Okuyama. Coorientador: Profª Drª: Márcia Amaral Corrêa Ughini Villarroel

1. Informática na Educação 2. Ensino de Programação. 3. Programação de Computadores. 4. Mediação. I. Okuyama, Fabio Yoshimitsu. II. Villarroel, Márcia Amaral Corrêa Ughini. III. Título

CDU: 004:37

Bibliotecário responsável: Filipe Xerxeneski da Silveira – CRB-10/1497

AGRADECIMENTOS

A conquista desta etapa não foi somente minha. Muitas pessoas me ajudaram, de alguma forma, e eu as agradeço nesta parte do trabalho.

Primeiramente, agradeço ao meu marido, Marcos Weiss, pela força e pelo ombro disponível nos momentos de dificuldade.

Ao meu colega e amigo, Márcio Fabiano de Carvalho, que foi um companheiro importante ao longo de toda a minha jornada no programa MPIE. Agradeço sua participação durante as aulas de LPI, que foi fundamental para a coleta de dados desta pesquisa, as trocas de ideias e os cafés na cantina para discutir as atividades a serem aplicadas na turma.

Aos meus colegas de trabalho, Eduardo Hartmann, Bruna Vuicik Mocelin, Rossano Fenner, Ana Cristina Schuch e Vitor Hugo Oliveira, pela paciência e disponibilidade em discutir assuntos técnicos relacionados a Linux, servidores web e PHP. Em particular, à Bruna, pelo batismo do nome do juiz *online* após um concurso às pressas para eleger um novo nome para a ferramenta.

Aos professores do Mestrado Profissional em Informática na Educação e, em especial, ao meu orientador, Professor Fabio Yoshimitsu Okuyama, e minha coorientadora, Márcia Amaral Corrêa Ughini Villarroel, pela paciência, incentivo e confiança ao longo das orientações.

Ao Professor André Peres, pela disponibilidade em me atender toda vez que a máquina virtual, na qual está instalado o juiz *online* no IFRS, estava fora do ar.

Aos Professores Evandro Manara Miletto, Sílvia de Castro Bertagnolli e Patrícia Brandalise Scherer Bassani, pelas contribuições no aprimoramento de meu trabalho.

Aos amigos que fiz na turma MPIE 2018/02, que me auxiliaram na discussão sobre Piaget e compartilharam ideias para a ferramenta.

Por fim, aos alunos das turmas de LPI - 2018/02, 2019/01 e 2019/02, pelo acolhimento e disposição em participar desta pesquisa.

A todos, muito obrigada!

RESUMO

A formação do estudante em programação de computadores ocorre por intermédio de um conjunto de disciplinas introdutórias do curso de Computação, que tem por objetivo fornecer os conceitos iniciais que utilizam uma linguagem de programação adotada, para que os alunos possam usar em comandos e estruturas para resolver os exercícios propostos no cronograma de ensino. Contudo, muitos discentes encontram barreiras ao tentarem desenvolver um programa, tais como: propor uma lógica que atenda a sua resolução; encontrar os erros e compreendê-los de modo a corrigi-los. Aliado a isso, os professores encontram dificuldades no acompanhamento individual das atividades realizadas pelo aluno devido ao elevado número de estudantes por turma. Isso também dificulta o aprendizado, haja vista que esses não conseguem esclarecer suas dúvidas e avaliar os erros no momento em que ocorrem, o que frustra e desmotiva; contribuindo no aumento dos índices de abandono e repetência nas disciplinas introdutórias de programação. Nesse contexto, com o propósito de melhorar essa situação, pesquisadores investigam o uso de ferramentas chamadas de juízes *online* no contexto educacional, para auxiliar na tarefa de avaliação automática dos códigos desenvolvidos pelos estudantes. Os juízes *online* são capazes de compilar, executar e testar o código-fonte submetido ao ambiente e informar, automaticamente, se o programa funcionou corretamente ou não, exibindo orientações (*feedback*) em caso de falha. Diante disso, tendo como base a Epistemologia Genética, mais precisamente a teoria da equilibração, e o estudo das dificuldades enfrentadas pelos alunos iniciantes em programação, esta pesquisa pretende elaborar uma estratégia de *feedback* que auxilie na sistematização e padronização do processo de mapeamento e de registro dos erros pelo professor, buscando clarificar o que é o erro, porque ele ocorreu e como corrigi-lo. Essas ações devem ser aplicadas a um juiz *online*, chamado de Codein'play, que permita avaliar se a estratégia proposta realmente elucida os equívocos encontrados durante a resolução dos exercícios e promova uma melhora na aprendizagem do discente. Esta pesquisa teve uma abordagem quali-quantitativa e foi aplicada na disciplina de Linguagem de Programação I do Curso Superior de Tecnologia em Sistemas para Internet no Instituto Federal do Rio Grande do Sul - Campus Porto Alegre/RS. O instrumento de coleta dos dados é constituído de observações, questionários aplicados em diferentes momentos durante o semestre e pesquisas bibliográfica e documental. Intenciona-se, com este estudo, mapear um número considerável de erros cometidos pelos estudantes iniciantes e seus *feedbacks*, além de medir a eficácia deste como ferramenta de auxílio no ensino de programação. O juiz *online* foi avaliado no segundo semestre de 2019 e o resultado evidenciou a contribuição da ferramenta ao proporcionar um ambiente que ajuda o aluno em seus estudos mediante a resolução de exercícios de programação. Nesta aplicação, observou-se uma redução de 6,82% de reprovação na disciplina. Destaca-se, todavia, que são necessárias novas avaliações para que se possa verificar a influência desta pesquisa isoladamente. A ferramenta foi avaliada positivamente, com destaque nas recomendações sobre o erro por meio de *feedback* fornecido aos alunos; na execução automática do código aplicando testes de caixa preta, na exibição da cobertura do código mostrando por onde o compilador passou e nos relatórios disponíveis para professor e alunos.

Palavras-chave: *feedback*, juiz *online*, ensino de programação, erro construtivo, execução automática de código

ABSTRACT

The formation of the student in computer programming occurs through a set of introductory subjects of the Computer courses, whose objective is to provide the introductory concepts with the use of a programming language adopted in the discipline, where the students must use the commands and structures of language to solve the proposed exercises. However, many students face difficulties when trying to develop a program, such as: proposing a logic that meets the problem's resolution, finding the errors and understanding them to correct them. Allied to this, teachers find it difficult to individually monitor the activities carried out by the student due to the high number of students per class, which also hinders their learning, since they are unable to clarify their doubts and errors when they occur, which frustrates and demotivates even more; thus contributing to the increase in dropout rates and repetition in the discipline. In this context, in order to improve this situation, researchers are investigating the use of tools called online judges in the educational context to assist in the task of automatic assessment of codes developed by students. Online Judges are able to compile, execute and test the source code submitted to the environment and automatically inform if the program worked correctly or not, displaying guidelines (feedback) in case of failure. Therefore, based on Genetic Epistemology, more precisely on the theory of equilibrium, and on the study of the difficulties faced by students beginning in programming, this aimed to develop a feedback strategy that would help in the systematization and standardization of the process of mapping and recording errors and their recommendations by the teacher, in order to clarify what the error is, why it occurred and how to correct it, applied to an online judge, called Codein'play, that would allow to evaluate if the proposed strategy really elucidates the errors found during the resolution of the exercises and promotes an improvement in student learning. The research used both quantitative and qualitative approaches and was applied in the discipline of Programming Language I of the Superior Course of Technology in Internet Systems at the Instituto Federal do Rio Grande do Sul Campus Porto Alegre/RS. The data collection instrument consisted of observations, questionnaires that were applied at different times during the semester, bibliographic research and documentary research. The aim of this research was to map a considerable number of mistakes made by beginning students and their feedbacks, in addition to measuring the effectiveness of feedback as an aid tool in teaching programming. The online judge was evaluated in the second half of 2019 and the result showed the contribution of the tool in providing an environment that would help the student in his studies from the resolution of programming exercises. In this application of the approach, it was found a 6.82% reduction in failure, however further investigation is required in order to isolate the reasons of such reduction. In general, the tool was positively evaluated, with emphasis on recommendations about the error through feedback provided to students, automatic code execution by applying black box tests, display of code coverage showing where the compiler went through and the reports available to the teacher and students.

Keyword: feedback, online judge, teaching programming, constructive error, automatic code execution.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE SIGLAS E ACRÔNIMOS

SIGLA	SIGNIFICADO
ANSI	*American National Standards Institute*
API	*Application Programming Interface*
AVA	Ambiente Virtual de Aprendizagem
IDE	*Integrated Development Environment*
IFRS	Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul
INEP	Instituto Nacional de Estudos e Pesquisas Educacionais Anísio Teixeira Legislação
ISO	*International Standards Organization*
LDBEN	Lei de Diretrizes e Bases da Educação Nacional
LPI	Linguagem de Programação I
MPIE	Mestrado Profissional em Informática na Educação
OD	Organização Didática
OO	Orientação a Objetos
PHP	PHP: *Hypertext Preprocessor*
PLEA	Planejamento, Execução e Avaliação
PPC	Projeto Pedagógico de Curso
PROEJA	Programa Nacional de Integração da Educação Profissional
SBC	Sociedade Brasileira de Computação
STSI	Superior de Tecnologia em Sistemas para Internet
TI	Tecnologia da Informação
UFRGS	Universidade Federal do Rio Grande do Sul

SUMÁRIO

1 INTRODUÇÃO

O mercado de tecnologia da informação (TI) vem crescendo exponencialmente e se tornou exigente na busca por profissionais cada vez mais qualificados. Segundo a SOFTEX (2018), esse mercado deixará de faturar bilhões de reais nos próximos cinco anos em decorrência da carência de trabalhadores qualificados para atender essa demanda.

De acordo com o INEP (2017), existem, no Brasil, 2.451 cursos na área de TI, distribuídos nos cursos de: Gestão da Informação, Administração de Redes, Banco de dados, Ciência da Computação, Formação de Professor de Computação (Informática), Informática, Tecnologia da Informação, Tecnologia em Desenvolvimento de Softwares, Análise de Sistemas, Análise e Desenvolvimento de Sistemas (Tecnólogo), Segurança da Informação, Sistemas de Informação e Engenharia da Computação, todos ministrados em instituições de ensino públicas e privadas. Em 2017, foram ofertadas 687.551 vagas na área de TI, com 163.599 ingresso e, desse montante, somente 43.260 concluíram o curso. Um dado que chama a atenção é o número de estudantes que trancam a matrícula e trocam de curso - 73.470 discentes cancelaram suas matrículas e 5.669 optaram por trocar de curso. Uma informação preocupante, pois o número de evasão é superior à quantidade de alunos que finalizaram o curso.

A baixa taxa de concluintes nos cursos de TI não é um acontecimento recente. No ano de 2015, a Associação Brasileira das Empresas de Tecnologia da Informação e Comunicação (BRASSCOM, 2015) realizou uma pesquisa buscando identificar o motivo da evasão escolar e constatou que, dentre as causas coletadas e analisadas entre 2006 e 2013, estão: abandono em função das responsabilidades profissionais em razão da carga horária muito extensa que atrapalha os estudos; e entendimento do estudante de não haver necessidade para finalizar a graduação, uma vez que não consegue uma boa posição no mercado de trabalho. Outra justificativa muito pertinente está relacionada às empresas, quando valorizam mais a certificação do que o diploma acadêmico no momento da seleção dos candidatos, uma vez que compreendem que o conhecimento técnico, adquirido por meio dessas certificações, é de suma importância no momento da contratação, colocando a formação acadêmica do candidato em segundo plano. Essa importância dada pelas empresas sobre o valor das certificações está relacionada ao reconhecimento do domínio em uma tecnologia, por um determinado período ou versão, que é valorizada mundialmente. Além disso, outros estudos como o do Instituto

Federal do Rio Grande do Sul (IFRS, 2018b), Lahtinen et al. (2005), Bosse & Gerosa (2016) e Sevella et al. (2013) apontam outras causas que levam a evasão/retenção escolar que dizem respeito a fatores educacionais como o nível de complexidade em aprender a programar.

A aprendizagem de algoritmos é fundamental nas disciplinas iniciais de programação dos cursos de Computação. Gomes et al. (2015a) enfatiza que o ensino de linguagens de programação nas disciplinas tenciona desenvolver um conjunto de habilidades nos estudantes que são necessárias para projetar programas capazes de resolver problemas que surgem no dia a dia. No entanto, a disciplina é considerada um 'funil' no curso, por efeito da grande dificuldade de compreensão e aplicação de alguns conceitos abstratos de programação.

Na maioria das vezes, no início do curso de graduação, o aluno possui quase nenhum ou pouco contato com alguma linguagem de programação e, ao chegar em sala de aula, depara-se com a necessidade de aprender rapidamente a linguagem adotada na disciplina. Enquanto alguns estudantes possuem grande facilidade em aprender, outros apresentam muitas dificuldades (AMBROSIO, 2011). Para estes, Ambrosio (2011) analisa que, no decorrer da disciplina, um número reduzido de discentes conseguem dar um 'salto' o que lhes possibilita acompanhar e se desenvolver até o final do semestre. Outros, no entanto, apesar de demonstrarem um enorme esforço, não conseguem atingir o mesmo objetivo.

Pimentel & Nizam (2008) e Jenkins (2002) citam como causas para o baixo desempenho nas disciplinas de programação a falta de habilidade para resolução de problemas. Borges (2000) e Giraffa (2003) corroboram essa ideia e evidenciam como obstáculos a base fraca em matemática, a dificuldade em compreender o problema e entender do que se trata o assunto. Gomes et al. (2015a) destacam a dificuldade em entender e interpretar as mensagens de erro exibidas pelo compilador por estarem no idioma inglês, o que dificulta a compreensão, além da forma como ela é organizada e apresentada ao aluno na tela do computador, o que igualmente impossibilita o entendimento.

Em função dessas e de outras dificuldades que ocorrem durante o aprendizado nas disciplinas introdutórias de programação é que ocorrem os erros, que para muitos alunos é um momento de extrema frustração, acarretando, muitas vezes, o abandono ou a reprovação na disciplina. Para minimizar essas tribulações, pesquisadores como Alves & Jaques (2014); Carvalho et al. (2016) e Chaves et al. (2014) discutem ferramentas computacionais que

possam auxiliar o estudante durante seus estudos de programação de computadores. Esses ambientes são conhecidos como juízes *online*, originados das competições de programação, mas adaptados para o contexto educacional, fornecendo *feedbacks* como forma de orientação. Essas ferramentas são capazes de compilar, executar e testar o código-fonte submetido ao ambiente e informar, automaticamente, se o programa funcionou corretamente ou falhou.

Mas qual é a posição do erro em relação ao conhecimento? É algo natural, ou apenas representa o insucesso de um processo educativo? Indica problemas do aluno? Diferentemente da perspectiva tradicional de ensino, onde o erro é considerado uma barreira na formação do conhecimento e que deveria ser eliminado do processo de ensino-aprendizagem, o construtivismo com Piaget, de acordo com Kutzke (2015), introduz uma nova visão sobre o erro e seu significado no meio pedagógico. Cerqueira (2009) afirma que "na perspectiva de Piaget o erro é uma fonte rica de aprendizagem e desenvolvimento, é uma fonte e não a única, por isso toda pedagogia não deve ser gerada em torno do erro, o erro terá uma função e uma utilidade na construção do saber da criança e cabe ao educador uma metodologia para auxiliar no processo de construção do conhecimento, e através do erro chegar ao acerto". Nascimento (2012) comenta que "para Piaget, o erro não interessa o que interessa são as ações física e mental. Erro e acerto são detalhes dessas ações."

Em sua teoria, Piaget (1983, p. 211) distingue dois aspectos no desenvolvimento intelectual da criança: o primeiro é chamado de psicossocial e compreende tudo o que a criança recebe do exterior que seja motivador de aprendizado, tais como transmissão familiar, escolar, interação entre amigos; e o segundo é o desenvolvimento chamado de espontâneo, ou seja, quando o conhecimento é descoberto sozinho, levando mais tempo para adquiri-lo. Consoante Abreu et al. (2010), o verdadeiro conhecimento é aquele construído pela experiência do próprio indivíduo, em um ambiente estimulante e social, onde a sua aquisição surge da relação entre o sujeito e o objeto. Na epistemologia genética, a capacidade de aprendizagem do indivíduo depende do grau de maturidade de suas estruturas cognitivas. Desta forma, existem quatro estágios do desenvolvimento cognitivo, identificados por Piaget, pelos quais o indivíduo passa gradativamente, da condição de um mundo desconhecido que o cerca ao um outro com diversas possibilidades pronto para ser descoberto e explorado. Assim, é na atuação sobre o ambiente que o indivíduo constrói seu conhecimento, mediante os processos de assimilação, acomodação (e equilibração), em um desenvolvimento mútuo e

sucessivo.

Conforme Piaget (1999, p. 16), a ação humana é formada por um movimento contínuo de reajustamento ou de equilibração. O autor chamou de adaptação o processo de equilíbrio entre assimilação e acomodação, representando o equilíbrio psíquico. Conhecer é interpretar e a qualidade dessas interpretações depende dos níveis de estrutura da inteligência, porém, quando o assunto é o erro, essa afirmação sofre algumas suposições.

Kutzke (2015) comenta que o erro é um indicador do motivo gerador da falha e chega a duas hipóteses: ou o aluno não está pronto (não possui a maturidade cognitiva) para desenvolver a atividade proposta pelo professor, ou o conteúdo não gerou o desequilíbrio necessário à aprendizagem. É o processo de equilibração, que ocorre entre a assimilação e a acomodação, o grande gerador da evolução da inteligência e do conhecimento, na visão de Piaget (1967). No entanto, para haver o conflito, o indivíduo tem que perceber que as suas formas de assimilação não estão dando conta daquilo que ele pretende fazer ou resolver.

La Taille (1997) comenta que o erro só terá valor como fonte de enriquecimento se ele for 'observável' pelo estudante; ou seja, este deve ter acesso à qualidade do erro mediante informações que o façam refletir e o ajudem a definir estratégias para chegar ao resultado, não somente saber que errou. Esse retorno deve acontecer de forma natural, com o intuito de trabalhar o erro, disponibilizando elementos em torno dele para que a regulação, em virtude do erro negativo, ocorra. Assim, Deitos (2018, p. 31) sublinha que "o erro precisa ser estabelecido como parte integrante desse processo, sendo capaz de contribuir para a construção do conhecimento".

Como trabalhar o erro, de forma construtiva, no aprendizado de programação de computadores para que promova uma reflexão na formação de novos conceitos sobre a unidade de aprendizagem de programação de computadores? Com base nesse questionamento, este estudo apresenta uma abordagem para mediar o erro durante a aprendizagem em programação, mediante *feedbacks* exibidos aos discentes durante a resolução de exercícios. Por meio de uma pesquisa quali-quantitativa com alunos maiores de 18 anos, matriculados na disciplina de Linguagem de Programação I do curso Superior Tecnológico em Sistemas para Internet no Instituto Federal do Rio Grande do Sul, pretende-se identificar os erros mais cometidos pelos estudantes iniciantes em programação e validar a

abordagem de mediação do erro idealizada, verificando se os alunos conseguem conhecer e aprender com ele.

1.1 Objetivos

Esta pesquisa tem por objetivo primário organizar uma estratégia de *feedback* aplicada a um juiz *online* que apoie a aprendizagem de programação estruturada, apresentando informações relevantes que ajudem o aluno a entender, corrigir e aprender com o erro.

Como objetivos secundários:

1. identificar os erros mais cometidos pelos alunos iniciantes em programação;
2. avaliar os ambientes com *feedback* imediato e realizar uma pesquisa bibliográfica para identificar os requisitos para uma ferramenta de *feedback* automático;
3. desenvolver um juiz *online* que possibilite o uso da estratégia de *feedback* automático;
4. desenvolver os *feedbacks* para os erros encontrados;
5. reduzir o esforço de professores na análise dos erros e na correção dos exercícios; e
6. investigar a contribuição da ferramenta no processo de ensino-aprendizagem de programação.

1.2 Justificativa

A formação do estudante em programação de computadores, geralmente ocorre por meio de um conjunto de disciplinas introdutórias dos cursos de Computação, que são identificadas por diversos nomes, tais como: Algoritmos, Lógica de Programação, Linguagem de Programação, Técnicas de Programação, entre outras. Essas têm como meta fornecer os conceitos introdutórios de programação por intermédio de comandos e definições sobre a linguagem e as estruturas, que devem ser utilizados pelos discentes para implementar soluções para um determinado problema, de maneira lógica, e representá-las em um ambiente computacional.

Para Forbellone (2005), a lógica de programação significa a utilização correta das leis do pensamento e dos processos de raciocínio e a simbolização formal na programação de computadores, delineando a racionalidade e o desenvolvimento de técnicas que ajudem na produção de soluções logicamente válidas e coerentes, que resolvam com qualidade os

problemas que se deseja elucidar. Do mesmo modo como as pessoas se expressam com a fala e a escrita alicerçadas em um determinado idioma, seguindo padrões lexicais, sintáticos e gramaticais (semânticos), existem convenções para a representar a lógica de programação, com linguagens de programação que utilizam, para isso, os algoritmos. Segundo Forbellone (2005, p.3), "algoritmo é uma sequência clara e precisa de passos que visa atingir um objetivo previamente definido, ou seja, o algoritmo fixa um padrão de comportamento a ser seguido, uma norma de execução a ser percorrida, visando alcançar a solução de um problema como resultado final, garantindo que sempre que executado, sob as mesmas condições, produza o mesmo resultado". Tudo isso, respeitando os padrões lexicais, sintáticos e semânticos definidos em cada linguagem de programação.

Para Lahtinen et al. (2005), o estudo da programação não é fácil, pois requer uma compreensão correta dos conceitos abstratos inerentes à natureza do conteúdo. A carga de conhecimento inicial, que é preciso adquirir e dominar para começar um programa é enorme para um programador iniciante, principalmente se ele nunca teve contato com a programação. Viegas (2017) argumenta que o estudante precisa dominar alguns conceitos para ser aprovado na disciplina de Programação, entre eles: tipo de dados, variáveis, entrada e saída de dados, expressões aritméticas, operadores lógicos e relacionais, estruturas de decisão e de repetição, vetores, matrizes, definição de funções, entre outros. Além disso, ele precisa saber aplicar esses conhecimentos na prática dos exercícios, o que acaba sendo muito complexo e difícil, criando uma barreira na aprendizagem da disciplina.

É neste momento que os erros nos códigos aparecem. De acordo com Sevella et al. (2013), a maioria dos programadores iniciantes encontram obstáculos em várias fases do desenvolvimento ao tentarem completar um programa, como, por exemplo: dificuldades na construção do código, identificar erros no código-fonte e compreendê-los para posterior reflexão e correção. Contudo, os erros prováveis de acontecer durante a construção de um programa estão classificados em quatro tipos, conforme Hoshino (2004):

- **Erros léxicos:** quando há um erro na grafia do comando, por exemplo, if por fi;
- **Erros sintáticos:** quando a sintaxe do comando não é respeitada, como por exemplo, a quantidade de parênteses abertos não corresponde à mesma quantidade de parênteses fechados;

- **Erros semânticos:** quando envolve códigos tecnicamente corretos, mas que possuem problemas com o significado interno da linguagem de programação, ou seja, não se respeita as regras da linguagem de programação, como dividir um número por zero ou atribuir um valor flutuante (*float*) para uma variável do tipo inteira (*int*);
- **Erros lógicos:** quando o código está sintatica e semanticamente correto, mas o resultado da execução do código não é o esperado, ou seja, não executam de maneira correta. Esse é um erro muito comum de acontecer para programadores iniciantes.

Os erros sintáticos, léxicos e semânticos são identificados pelos compiladores das linguagens de programação, porém, os erros lógicos são mais complexos de detectar, principalmente, quando os códigos são extensos. Nesta pesquisa, os erros semânticos e léxicos serão tratados como erro sintático.

Conforme Chaves et al. (2014), o aumento dos índices de reprovação e a evasão escolar são reflexos das dificuldades no aprendizado da disciplina de programação. Neste tipo de matéria, para que o aprendizado seja eficaz, é primordial a prática dos conteúdos aprendidos em aula, esta é uma metodologia chamada por Anzai et al. (1979) de "*learning by doing*". O professor elabora um conjunto de exercícios de programação com níveis de dificuldade graduais e disponibiliza para seus alunos resolverem. Após a conclusão desses exercícios, o professor inicia a árdua função de correção, verificando se a solução proposta pelos alunos está de acordo com o enunciado, apontando os problemas nos códigos e sugerindo melhorias.

Alves & Jaques (2014) asseveram que é durante a resolução dos exercícios que se espera que os estudantes acionem o professor para esclarecer suas dúvidas e dificuldades, tentando entender as falhas encontradas durante a execução dos programas. Todavia, em virtude do grande volume de trabalho em função do número elevado de estudantes por turma, o professor não consegue atender de imediato as solicitações dos alunos, o que acaba desestimulando-os e contribuindo para a ação de deixar de lado o aprendizado da disciplina (CHAVES et al., 2014; ANDRADE, 2018). Além do problema da quantidade de alunos por turma, há outros fatores que contribuem na dificuldade do ensino de programação, como expressam Bosse & Gerosa (2016): a baixa participação em sala de aula, a linguagem de

programação adotada na disciplina, a baixa frequência dos estudantes e o desinteresse pelo aprendizado.

Para apoiar o ensino de programação e reduzir as dificuldades na aprendizagem em programação, pesquisadores investigam e propõem ferramentas computacionais que possam auxiliar nas tarefas de *feedback* e de avaliação automática dos códigos desenvolvidos pelos estudantes, um exemplo são os ambientes chamados de juízes *online*. Estes, geralmente utilizados em competições de programação (ALVES & JAQUES, 2014), são capazes de executar o código-fonte submetido ao ambiente e informar, automaticamente, se o programa funcionou corretamente ou falhou. Adaptado ao contexto escolar, em caso de falha, esses sistemas fornecem ao aluno um *feedback* descritivo do motivo da falha e onde ela ocorreu, provocando uma reflexão sobre o erro, buscando o equilíbrio da estrutura de inteligência. Sabe-se que o processo de equilibração, que ocorre entre a assimilação e acomodação, é o grande gerador da evolução da inteligência e do conhecimento, na visão de Piaget (1967), e as reequilibrações representam a modificação e o melhoramento dos esquemas do sujeito. Carvalho et al. (2016) apresentam que o juiz *online* utilizado em sala de aula permitiu a prática de exercícios de programação e proporcionou um aumento na taxa de aprovação.

1.3 Estrutura deste Trabalho

Este estudo está organizado em seções e subseções. A **primeira seção** apresenta uma visão da pesquisa, os objetivos e a sua justificativa. O contexto institucional onde ela é realizada é tratado na **segunda seção,** abordando o Instituto Federal e o Instituto Federal do Rio Grande do Sul *Campus* Porto Alegre/RS; apresentando o curso Superior Tecnológico em Sistema para Internet e a disciplina de Linguagem de Programação I; e por fim, expondo dados sobre a evasão/abandono e retenção/repetência na disciplina pesquisada. Na **terceira seção** apresenta a metodologia utilizada, a população dos indivíduos participantes e como os dados foram coletados e gerados. A fundamentação teórica é discutida na **quarta seção**, possuindo cinco subseções. A primeira mostra a Epistemologia Genética utilizada para embasar a construção da ferramenta. A segunda aborda os conceitos entorno do *feedback*, apresentando seus tipos e sua importância. Na terceira são delineadas as dificuldades encontradas pelos estudantes iniciantes em programação durante seus estudos. Na quarta o conceito de juiz *online* é exibido. E, na quinta, são indicadas as noções sobre teste de software e cobertura de linhas de

código. Na **quinta seção**, são evidenciados alguns estudos relacionados referentes às ferramentas de execução automática conhecidas por juízes *online*, desenvolvidas para ambiente educacional para aprendizagem em programação de computadores. Na **sexta seção**, está descrita a estratégia de *feedback* criada para aplicar no juiz *online* Codein'play. A **sétima seção** demonstra o protótipo, seu funcionamento e suas funcionalidades. Na **oitava seção** estão narradas as análises e resultados dos dados levantados. Na **nona seção**, as considerações finais são alinhavadas. Por fim, as Referências Bibliográficas, os Apêndices e Anexos são balizados.

2 CONTEXTO INSTITUCIONAL

Os Institutos Federais de Educação, Ciência e Tecnologia, criados pela Lei nº 11.892/2008, representam um novo viés da Educação Profissional e Tecnológica brasileira, e o art. 2º dessa Lei dispõe (BRASIL, 2008):

> Os Institutos Federais são instituições de educação superior, básica e profissional, pluricurriculares e multicampi, especializadas na oferta de educação profissional e tecnológica nas diferentes modalidades de ensino, com base na conjugação de conhecimento técnicos e tecnológicos com as suas práticas pedagógicas, nos termos desta Lei.

Para efeitos de regulação, avaliação e supervisão das instituições e dos cursos de educação superior ofertados, os Institutos são equiparados às Universidades Federais e possuem autonomia para criar e extinguir cursos dentro de seus limites de atuação territorial, bem como registrar diplomas dos cursos que oferecem mediante aprovação de seu Conselho Superior.

Conforme Pacheco et al. (2010), os Institutos Federais fazem parte de um projeto inovador, no qual a educação tem o compromisso de transformar e enriquecer conhecimentos, o que os torna capazes de modificar a vida social. Trata-se de uma estratégia de ação política e de transformação social. Pacheco (2011) explicita que os Institutos ressaltam a valorização da educação e das instituições públicas, combatendo as desigualdades, além de reafirmarem a educação profissional tecnológica como política pública, estabelecendo uma interação mais direta junto ao poder público e às comunidades locais.

A estrutura *multicampi* e a definição do território de abrangência das ações dos Institutos Federais afirmam, na missão dessas instituições, o compromisso de intervenção em suas respectivas regiões, identificando problemas e criando soluções técnicas e tecnológicas para o desenvolvimento sustentável com inclusão social. Na proposta desses Institutos, agregar à formação acadêmica a preparação para o trabalho e discutir os princípios das tecnologias a ele relacionados são os elementos essenciais para a definição de um propósito específico na estrutura curricular da educação profissional e tecnológica.

2.1 O Instituto Federal do Rio Grande do Sul (IFRS) - Campus Porto Alegre

A história do IFRS inicia em 1909, com a fundação da Escola de Comércio de Porto Alegre, que foi transformada, em 2008, na Escola Técnica da Universidade Federal do Rio Grande do

Sul (UFRGS), dando origem, mais tarde, ao Campus Porto Alegre do IFRS.

A Escola de Comércio de Porto Alegre pertencia à Faculdade Livre de Direito e foi gerenciada por ela durante 35 anos. Nesse tempo, a Escola possuía dois cursos (IFRS, 2018a): o Curso Geral (início em 1910), que preparava para os cargos da Fazenda, para as funções de guarda-livros e perito judicial; e o Curso Superior, que qualificava para os cargos do Ministério das Relações Exteriores, Corpo Consular, Atuário de Companhias, chefe de Contabilidade de Empresas Bancárias e Grandes Casas Comerciais, e tinha como pré-requisito o Curso Geral. Ambos os cursos tinham dois anos de duração e permitiam o acesso sem necessidade de concurso público aos cargos para o qual os cursos eram propostos.

Em 1916, a Escola foi reconhecida pelo Governo Federal pela sua importância na formação profissional orientada para fins de interesse geral e prestação de serviços à sociedade. A Universidade de Porto Alegre é criada em 1934, incorporando-se a Faculdade de Direito e a Escola de Comércio em seu quadro de cursos de graduação, as quais passaram a ser custeadas pelo Estado. No ano de 1945, a Faculdade de Economia e Administração é criada, momento em que a Escola de Comércio passa a fazer parte dessa instituição, desvinculando-se da Faculdade de Direito.

Foi em 1947 que a Universidade de Porto Alegre passou a ser mantida pelo Governo Federal e começou a ser chamada por Universidade do Rio Grande do Sul (URGS). Porém, ela realmente começa a ser administrada pelo Governo Federal em 1950, passando a ser denominada Universidade Federal do Rio Grande do Sul (UFRGS). A Faculdade de Economia e Administração e, respectivamente, a Escola de Comércio, agora denominada Escola Técnica de Comércio, passam, então, a integrar a UFRGS. Nesse momento, inicia-se uma nova fase para a Escola.

Na década de 60, a Escola passa a ter uma direção própria e diferenciada da Faculdade de Ciências Econômicas. Com o surgimento de novos cursos e a visão reformulada do ensino técnico para atender Lei de Diretrizes e Bases da Educação Nacional (LDBEN), em 1996, a Escola Técnica de Comércio da UFRGS passa a se chamar Escola Técnica da UFRGS. Em 1999, a Escola Técnica começa a ministrar somente cursos de educação profissional, contudo, a admissão nestes exigia a conclusão do ensino médio.

Com a Lei nº 11.892/2008, que cria os Institutos Federais no Brasil, a Escola Técnica

vinculada à UFRGS deixa de existir e passa a se chamar Campus Porto Alegre do IFRS, que, atualmente, conta com 13 cursos técnicos, cinco cursos superiores (três tecnólogos e duas licenciaturas), um Programa Nacional de Integração da Educação Profissional (PROEJA), dois cursos *Lato Sensu*, dois cursos *Stricto Sensu*, e cursos de extensão. Dos três cursos tecnológicos ofertados pela Instituição, destaca-se o Curso Superior de Tecnologia em Sistemas para Internet, objeto desta pesquisa, que é apresentado na próxima subseção.

2.2 O curso Superior de Tecnologia em Sistemas para Internet (STSI)

Implantado no ano de 2010, o curso STSI propõe, segundo o Projeto Pedagógico do Curso (PPC) (IFRS, 2018a, p. 7), formar "[...] profissionais capazes de analisar, projetar, implantar e implementar ações voltadas ao desenvolvimento de aplicações para a Internet, contribuindo com um trabalho qualificado e socialmente comprometido para a utilização de tecnologias ligadas às redes de computadores".

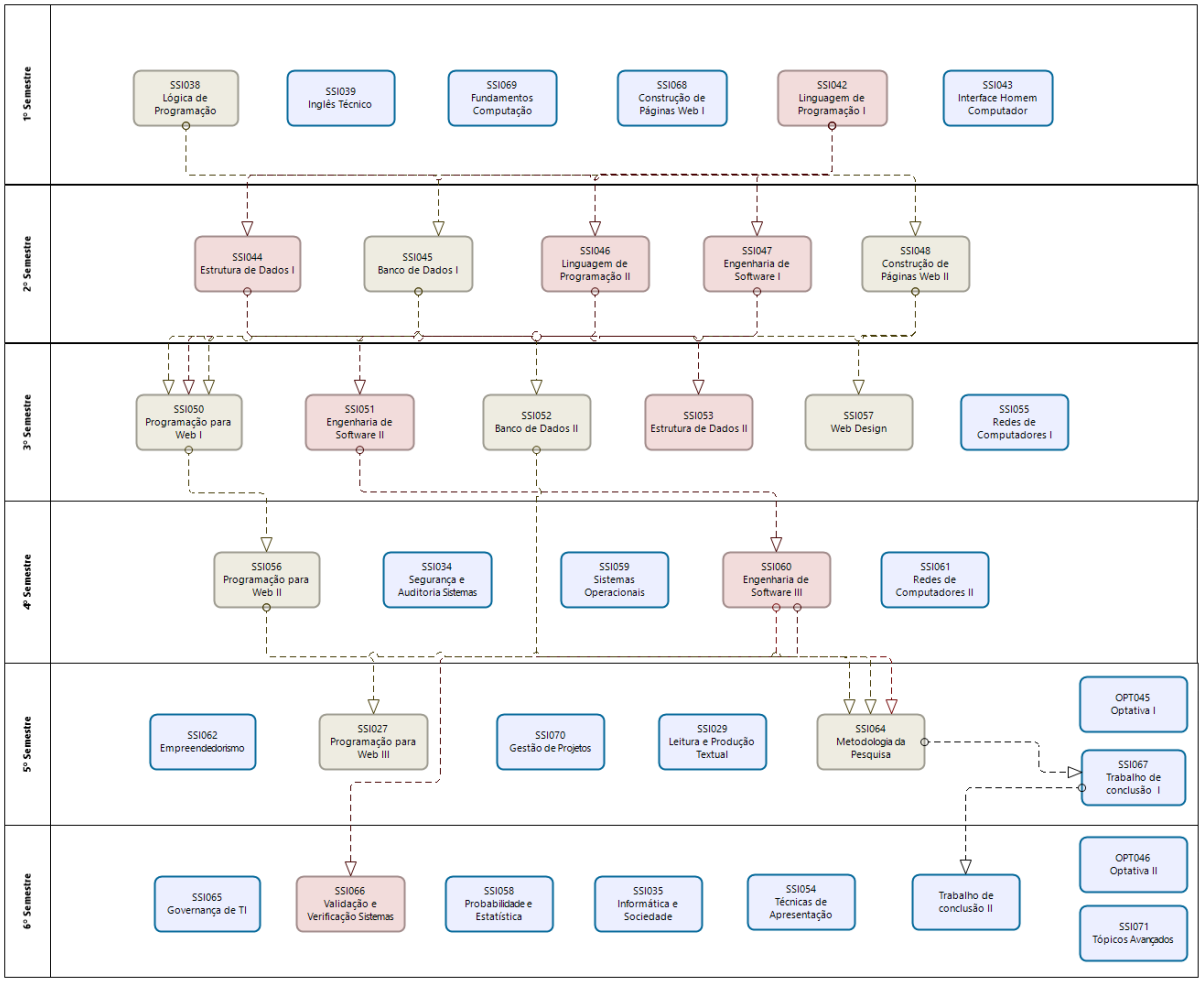
O curso foi organizado em 12 áreas, abrangendo disciplinas que abordam conhecimentos desde linguagens de programação à segurança de sistemas Web, promovendo um encadeamento das matérias para fomentar a integração das áreas ao longo do curso. Com a sua reformulação, em 2013, ele passa a ser ofertado semestralmente em turnos alternados: semestre par, durante a manhã; e semestre ímpar, à noite. Por ano, são oferecidas 72 vagas para ingresso, sendo 36 para o turno da manhã e 36 para o turno da noite.

A proposta do curso é desenvolver conhecimentos necessários para que o estudante esteja apto a trabalhar com diversas tecnologias e consiga ter autonomia e clareza na escolha da melhor solução para determinado projeto ou sistema. Nesta perspectiva, a matriz curricular foi organizada com 2.216 horas relógio, distribuídas ao longo de seis semestres. Conforme o PPC (IFRS, 2018a), o curso possui uma concentração na área de desenvolvimento e engenharia de software, oferecendo várias disciplinas que ensinam diferentes linguagens de programação, o que torna os conceitos introdutórios importantíssimos, uma vez que são a base dessa área.

Na Figura 1, é possível visualizar as disciplinas ofertadas no curso STSI, organizadas por semestre. As células nas cores cinza e vermelha se referem às disciplinas que utilizam os conhecimentos de programação, contemplando as disciplinas introdutórias: **SSI038 Lógica de**

Programação e **SSI042 Linguagem de Programação I**. Observa-se que praticamente 42% das disciplinas ofertadas no curso utilizam os conteúdos ministrados nas duas que são introdutórias do primeiro semestre.

Figura 1 - Matriz Curricular



Fonte: Adaptada do PPC do STSI (IFRS, 2018a, p. 32-35)

Conforme o PPC (IFRS, 2018a), cada ano do curso pretende desenvolver competências e habilidades, articulando os conteúdos estudados entre as disciplinas.

No primeiro semestre, os alunos aplicam os conceitos aprendidos em sala de aula em trabalhos práticos, associando os diferentes componentes curriculares. “Por exemplo, no componente curricular de Interface Homem Computador o aluno elabora protótipos de interfaces e, para esse fim, utiliza os conceitos de programação dos componentes curriculares de Construção de Páginas Web I, Lógica de Programação e Linguagem de Programação I na fase de desenvolvimento do sistema.” (IFRS, 2018a, p. 17).

No segundo semestre, são abordadas as noções referentes à análise e ao projeto de sistema orientado a objeto (OO) na disciplina de Engenharia de Software I, que tem por meta orientar a implementação de pequenas aplicações utilizando o paradigma OO na disciplina Linguagem de Programação II. Ainda, neste semestre, os alunos desenvolvem o modelo de dados (Entidade-Relacionamento) das classes projetadas e implementadas na disciplina de Banco de Dados I.

No terceiro semestre, os discentes são motivados continuar evoluindo o projeto iniciado nos semestres anteriores. Na disciplina de Engenharia de Software II, eles aprendem a modelar sistemas, iniciam o desenvolvimento da aplicação do projeto, utilizando uma linguagem de programação orientada a objetos, na disciplina de Programação para Web I. O acesso ao banco de dados é trabalhado no componente curricular de Banco de Dados II.

No quarto semestre, os estudantes conhecem algumas tecnologias para o desenvolvimento de aplicações Web, aplicando, de modo integrado, metodologias ágeis e *frameworks* diversos nas disciplinas: Programação para Web II e Engenharia de Software III. Além disso, um outro aspecto muito importante é a preocupação com a acessibilidade, a qual garante que os sistemas de informação sejam acessíveis a todos os indivíduos. Assim, questões como: acessibilidade e usabilidade são trabalhadas nas disciplinas de Web Design, Interface Homem Computador e Desenvolvimento de Sistemas Web Acessíveis, que complementa a formação integral dos alunos no IFRS:

> [...] os temas transversais oportunizam que a interdisciplinaridade ocorra no currículo. Destaca-se ainda que, no curso Superior de Sistemas para Internet são trabalhados de forma transversal os seguintes temas: Educação Ambiental, trabalhado no componente curricular de Redes de Computadores II e Direitos Humanos e Cultura Afro brasileira e indígena, discutido no componente curricular de Informática e Sociedade. (IFRS, 2018a, p. 17)

2.3 A disciplina de SSI042-Linguagem de Programação I do curso STSI

No curso STSI, a disciplina de Linguagem de Programação I é ofertada no primeiro semestre do curso, possuindo 80 horas-aula. Consoante o PPC (IFRS, 2018a), a disciplina é presencial e tenciona a "apresentação do paradigma da programação estruturada, com uso da Linguagem C – Padrão ANSI, proporcionando o desenvolvimento do raciocínio na elaboração de soluções de problemas de programação algorítmica." (IFRS, 2018a, p.40).

Os conteúdos que são ministrados na disciplina durante o semestre abordam:

> [...] fundamentos da construção de programas utilizando linguagem C ANSI. Conceitos de variáveis, variáveis homogêneas (vetores e matrizes) e variáveis heterogêneas (registros). Operadores e expressões matemáticas e lógicas. Estruturas de controle de programação. Funções, procedimentos, variáveis locais e globais, passagem de parâmetros por valor e por referência e tratamento de arquivos. (IFRS, 2018a, p. 40)

A disciplina tem como objetivo principal ajudar na compreensão do paradigma de programação estruturada, aplicando a Linguagem C – Padrão ANSI, além de auxiliar no desenvolvimento do raciocínio com a elaboração de programas e apresentar as estruturas de uma linguagem de programação.

Até o segundo semestre de 2018, o critério de aprovação estabelecido pela Instituição era composto por conceitos: A (Ótimo), B (Bom), C (Regular), D (Insatisfatório) e E (Falta de Frequência). O discente que atingisse os conceitos A, B ou C, era considerado APROVADO e estaria apto a se matricular nas disciplinas da sequência curricular. Porém, se a avaliação final estivesse entre os conceitos D ou E, era considerado REPROVADO e deveria se matricular novamente na mesma disciplina. Com a aprovação da Organização Didática (OD), que entrou em vigor em 2019, a nova forma de avaliar o desempenho do estudante em cada componente curricular passa a ser expresso por notas de 0 (zero) a 10 (dez) e não mais por conceitos.

De acordo com a OD (IFRS, 2015), a nota mínima para aprovação é 7 (sete), calculada pela média aritmética das avaliações realizadas durante o semestre. O estudante que não atingir a média para aprovação ao final do período letivo, deverá fazer o exame final. A aprovação do estudante na disciplina se dará mediante a frequência mínima de 75% e média semestral igual ou superior a 7 (sete) ou média final igual ou superior a 5 (cinco), após realização do exame.

2.4 Abandono e Retenção escolar na disciplina LPI do curso STSI

O IFRS, em conformidade com o contexto de sua criação e comprometido com a concepção de Educação Profissional e Tecnológica, atenta-se em desenvolver práticas que objetivam, além do acesso, a permanência e o êxito dos estudantes em seus *Campi*. Em razão disso, em 2018, sistematizou um estudo para identificar os motivos que levam os estudantes à evasão/abandono e à retenção/reprovação escolar, formalizando os dados coletados em um

Plano Estratégico de Permanência e Êxito, aprovado pela Resolução nº 064, de 23 de outubro de 2018, que busca traçar um plano de ação em um conjunto de estratégias que favoreçam o alcance de metas para a permanência e o êxito no processo de formação de seus estudantes.

Em concordância com o Plano, o conceito de evasão pode estar relacionado à "retenção e à repetência do estudante; à saída do estudante da instituição, do sistema de ensino, da instituição acadêmica e posterior retorno; ou ainda, à não conclusão de um determinado nível de ensino" (IFRS, 2018b, p. 15). A escolha de deixar a escola ou instituição acadêmica é um processo que se manifesta de diversas formas (multiformes) ao longo da trajetória escolar/acadêmica do estudante. Com isso, o plano adota classificações de fatores motivacionais que levam evasão/abandono e retenção/reprovação escolar citadas na literatura, organizando-as em três categorias (IFRS, 2018b, p. 19-20): fatores individuais aos estudantes, fatores internos e fatores externos à instituição.

Nos fatores individuais, o estudo destaca algumas características que contribuem na evasão/abandono e na retenção/reprovação: adaptação à vida acadêmica; capacidade de aprendizagem e habilidade de estudo; compatibilidade entre a vida acadêmica e as exigências do mundo do trabalho; encanto ou motivação com o curso escolhido; escolha precoce da profissão; questões de ordem pessoal ou familiar; questões de saúde do estudante ou de familiar; entre outras.

Nos fatores internos às instituições, foram identificadas algumas particularidades que contribuem para a desmotivação e conduzem o aluno a evadir do curso: infraestrutura física, material, tecnológica e de pessoal para o ensino; currículo escolar; gestão administrativa; questões didático-pedagógica da instituição; motivação e formação do professor; existência e abrangência dos programas institucionais para o estudante (assistência estudantil, iniciação científica, monitoria); etc.

Como fatores externos, os motivos estão relacionados às dificuldades financeiras do estudante para permanecer no curso e às questões inerentes ao futuro na profissão, bem como: reconhecimento social do curso; valorização da profissão; oportunidade de trabalho para egressos; conjuntura econômica e social; políticas governamentais para a educação profissional e tecnológica e para a educação superior; e assim por diante.

Para conhecer o quantitativo de evasão/abandono e retenção/repetência na disciplina

de Linguagem de Programação I, analisou-se os diários de classe de 2011 a 2019 e os dados levantados são apresentados no Quadro 1.

Quadro 1 - Percentual de alunos evadidos/retidos na disciplina de LPI de 2011 a 2019

Ano/ Semestre	**2011**	**2012**	**2013**	**2014**		**2015**		**2016**		**2017**		**2018**		**2019**	
	2	**2**	**2**	**1**	**2**	**1**	**2**	**1**	**2**	**1**	**2**	**1**	**2**	**1**	**2**
% Evasão/ Abandono	26,31	34,21	37,84	37,5	25,92	23,25	44,19	53,49	41,86	37,21	32,56	32,56	38,64	29,79	38,64
% Retenção/ Repetência	21,05	15,79	0	12,5	29,63	16,28	13,95	4,65	13,95	16,28	13,95	13,95	15,91	10,64	9,09
Nº Matr.	38	38	37	32	27	43	43	43	43	43	43	43	44	47	44

Fonte: Diários de Classe da disciplina LPI do IFRS Campus Porto Alegre

Nos anos 2011, 2012 e 2013 o curso não era seriado e a disciplina ocorria uma vez ao ano. A partir de 2014, ela passa a ocorrer semestralmente nos turnos da manhã ou da noite, alternadamente. Durante as análises dos diários, observou-se que alguns alunos fizeram a disciplina mais de duas vezes para serem aprovados, tendo casos de discentes que se matricularam sete vezes, abandonando no final do semestre. Pelos dados expostos no Quadro 1, existe uma tendência de os alunos desistirem da disciplina no momento em que percebem que não serão aprovados, ao invés de tentarem ir até o final e reprovar. Essa atitude é constatada presencialmente, durante a realização desta pesquisa.

Com a finalidade de apoiar o ensino de programação e buscar reduzir as dificuldades apresentadas, professores investigam ferramentas computacionais que possam auxiliar nas tarefas de *feedback* e avaliação automática dos códigos desenvolvidos pelos estudantes. Uma das ferramentas utilizadas são os juízes *online*, que será tratado na seção 5 deste estudo.

Na próxima seção, apresenta-se a metodologia que é a base para esta pesquisa.

3 METODOLOGIA DE PESQUISA

Conforme Prodanov e Freitas (2013), a finalidade da pesquisa é resolver algum problema ou dúvida utilizando procedimentos científicos alicerçados em indagações formuladas em relação a pontos ou fatos que permanecem indefinidos e de respostas que possam explicá-las. Para isso, há vários tipos de pesquisas que proporcionam a coleta de dados sobre o que se deseja investigar. A pesquisa científica é a realização de um estudo planejado, sendo o método de abordagem do problema o que caracteriza o aspecto científico da investigação, e tem como finalidade descobrir respostas para questões mediante a aplicação do método científico. Nesse sentido, esta seção possui subseções que apresentam as metodologias utilizadas, os indivíduos pesquisados, o procedimento empregado para gerar e coletar dados e a análise desses dados.

3.1 Fundamentação Teórica da Metodologia

A pesquisa terá uma abordagem quali-quantitativa, onde se observa o vínculo entre o mundo objetivo e subjetivo dos sujeitos e se trabalha com elementos que não podem ser traduzidos somente em números. Gil (2007) destaca que esse tipo de pesquisa é eficiente para a obtenção de dados em profundidade acerca dos mais diversos aspectos da vida social e a mais flexível de todas as técnicas de levantamento de informações. Cabendo, assim, descrever as características que suportam e dão conta da problemática da mediação do erro por meio de *feedbacks* personalizados por tipo de erro, dada a frustração ao fracasso do aluno iniciante em programação devido à sua falha. A pesquisa é de natureza aplicada, haja vista que pretende gerar conhecimento e propor uma solução ao problema investigado.

Os procedimentos qualitativos aplicados à pesquisa serão: bibliográfica, documental, análise de ferramentas e pesquisa-ação. Gil (2007) saliente que a pesquisa bibliográfica é desenvolvida com base em material já elaborado, constituído, principalmente, de livros e artigos científicos já existentes. Buscando entender mais sobre o tema e suas principais características, faz-se necessário realizar estudos bibliográficos, onde serão explorados artigos, livros, revistas, dissertações e periódicos. Como procedimentos técnicos da pesquisa sob um estudo de pesquisa-ação, Fonseca (2002, p. 34-35) define:

> A pesquisa-ação pressupõe uma participação planejada do pesquisador na situação problemática a ser investigada. O processo de pesquisa recorre a uma metodologia sistemática, no sentido de transformar as realidades observadas, a partir da sua compreensão, conhecimento e compromisso para a ação dos elementos envolvidos na pesquisa. O objeto da pesquisa-ação é uma situação social situada em conjunto e

> não um conjunto de variáveis isoladas que se poderiam analisar independentemente do resto. Os dados recolhidos no decurso do trabalho não têm valor significativo em si, interessando enquanto elementos de um processo de mudança social. O investigador abandona o papel de observador em proveito de uma atitude participativa e de uma relação sujeito a sujeito com os outros parceiros. O pesquisador quando participa na ação traz consigo uma série de conhecimentos que serão o substrato para a realização da sua análise reflexiva sobre a realidade e os elementos que a integram. A reflexão sobre a prática implica em modificações no conhecimento do pesquisador.

A investigação foi realizada no IFRS Campus Porto Alegre nos semestres 2018/02, 2019/01 e 2019/02. Em função disso, na próxima seção são abordados: o contexto institucional, explicando um pouco sobre a sua história do IFRS; o curso Superior de Tecnologia em Sistemas para Internet e a disciplina alvo da investigação; e o contexto da pesquisa.

3.2 Contexto da Pesquisa

A formação do estudante em programação de computadores, geralmente ocorre por meio de um conjunto de disciplinas introdutórias, que têm como objetivo fornecer os conceitos iniciais de programação por meio de comandos e definições sobre a linguagem e as estruturas, os quais devem ser utilizados pelos discentes para implementação de soluções de maneira lógica, em um determinado problema, representando-os em um ambiente computacional.

A literatura que aborda os problemas e as dificuldades encontrados pelos estudantes iniciantes nos cursos de computação apontam que a alta taxa de evasão nos cursos superiores de TI está ligada a fatores educacionais, como o nível de complexidade em aprender a programar (LAHTINEN et al., 2005; SEVELLA et al., 2013; ALVES & JAQUES, 2014; CHAVES et al., 2014). Para ser aprovado na disciplina, o estudante precisa dominar alguns conceitos como: tipo de dados, variáveis, entrada e saída de dados, expressões aritméticas, operadores lógicos e relacionais, estruturas de decisão e de repetição, vetores, matrizes, funções, entre outros. Além disso, ele deve saber aplicá-los na prática dos exercícios, o que acaba sendo muito complexo e difícil, formando uma barreira na aprendizagem da disciplina.

Buscando entender o efeito do *feedback* na aprendizagem perante o erro, foram realizadas observações presenciais dos participantes desta pesquisa com a finalidade de colher informações na disciplina de Linguagem de Programação I do curso STSI do IFRS Campus Porto Alegre. Coletou-se os erros mais praticados pelos discentes iniciantes, de forma a

reforçar os conteúdos em torno de tais falhas. Pretende-se disponibilizar o protótipo desenvolvido neste trabalho com uma bateria de exercícios cadastrados pelo professor e seus casos de teste, para que os alunos consigam, autonomamente, junto com os *feedbacks* apresentados*,* resolvê-los sem a necessidade de acionar, instantaneamente, o docente, em caso de erro no código-fonte. O protótipo apresentará dois tipos de *feedbacks* ao estudante: *feedback* avaliativo e *feedback* informativo. Em caso de acerto, o protótipo exibirá um *feedback* avaliativo (positivo), ficando dispensado do *feedback* informativo, cujo acionamento é realizado em caso de erros.

Para que o ambiente forneça *feedbacks* apropriados, de acordo com o tipo de erro gerado, faz-se necessário realizar uma pesquisa bibliográfica sobre as características de um *feedback* eficaz e quais informações pertinentes apresentar ao aluno que o oriente na correção do problema encontrado. Tendo como base as informações coletadas durante a investigação, elabora-se uma estratégia de *feedback,* buscando padronizar o mapeamento e o registro desses erros com a finalidade de automatizá-los.

3.2.1 Participantes

O levantamento de dados foi realizado com aproximadamente 60 estudantes matriculados na disciplina de Linguagem de Programação I no curso STSI do IFRS campus Porto Alegre, que é ofertada no primeiro semestre do curso, totalizando 80 horas-aula. O grupo de estudantes observados está na faixa etária entre 18 e 60 anos de idade e a grande maioria é iniciante na área de programação de computadores.

3.2.2 Coleta de Dados

Antes de iniciar as pesquisas sobre o uso de *feedback* como ferramenta de auxílio de aprendizagem, foi aplicado junto aos estudantes um questionário (**Apêndice A**), propondo mapear as dificuldades enfrentadas na disciplina e saber como os alunos encaram o erro. A ideia central deste primeiro momento é conhecer os erros mais praticados pelos alunos iniciantes em programação e suas dificuldades, buscando-se um ponto de partida para pensar sobre as características do ambiente de *feedback* e nas recomendações para estruturar o maior número de erros. O questionário foi aplicado na turma 2018/02 (manhã), 2019/01 (noite) e 2019/02 (manhã) da disciplina em estudo. Para esquematizar os erros e as recomendações, foi

indispensável analisar os códigos-fontes entregues pelos estudantes durante os semestres. Além disso, com o uso do protótipo, os erros que não foram mapeados anteriormente foram capturados para posterior análise e registro na ferramenta. Em outras palavras, para iniciar a base de erros e seus *feedbacks*, verificou-se os códigos produzidos pelos alunos, e os erros não previstos foram capturados pelo protótipo para anotação posterior do professor.

Para uma segunda fase da coleta de dados foram aplicados dois questionários em momentos diferentes. O primeiro (**Apêndice D**) foi antes de os alunos utilizarem o ambiente proposto, com a intenção de coletar informações a respeito das ferramentas chamadas de *Integrated Development Environment* (IDE) usadas pelos alunos para resolver os exercícios. Em seguida à aplicação deste questionário, foram propostas três atividades no protótipo que ocorreram durante o semestre. Estavam no rol dessas atividades: Simulado (individual), Desafio competitivo (em grupo de até 3 estudantes) e Maratona de Programação (individual). A partir dessas atividades, realizou-se a verificação dos erros mais cometidos pelos alunos durante a resolução da questão, a média geral de tentativas até o acerto sobre as 20 questões e a média de tentativas até o acerto por evento (atividade), extraindo da base de dados do protótipo.

Após essa etapa e já utilizando o protótipo, o estudante respondeu outro questionário (**Apêndice C**), para levantar as percepções do uso da ferramenta, procurando detectar se um sistema com *feedback* automático auxilia ou não no aprendizado. Apesar das observações terem sido realizadas nos semestres 2018/02 (manhã), 2019/01 (noite) e 2019/02 (manhã), somente no semestre 2019/02 (manhã) é que a ferramenta foi utilizada com os alunos, pois foi quando o ambiente ficou pronto para utilização em sala de aula.

Igualmente, fez-se uma análise documental em alguns documentos do IFRS: Projeto Pedagógico do Curso Superior de Tecnologia em Sistemas para Internet (PPC), Organização Didática (OD) e Diários de Classes da disciplina de Linguagem de Programação I, com o intuito de conhecer as competências e habilidades desenvolvidas pelo curso STSI, identificar os componentes curriculares aprendidos na disciplina, determinar a forma de avaliação utilizada no Instituto para os cursos tecnológicos e, ter subsídios para elaborar uma avaliação sobre os dados quantitativos de evasão/abandono e retenção/repetência, desde o início da disciplina de Linguagem de Programação I, alvo deste trabalho, até os momentos atuais.

3.2.3 Análise dos Dados

A análise de dados foi feita sobre a coleta de diversos artefatos: questionário (**Apêndice A**) respondido pelos alunos no decorrer dos três semestres, pretendendo conhecê-los e suas dificuldades; avaliação dos dados capturados pelo protótipo para identificar os erros mais cometidos; cálculo da média de tentativas (execuções) até chegar ao acerto da questão para avaliar se os *feedbacks* informativos ajudaram a conhecer e entender o erro; análise dos códigos-fonte dos alunos nos três semestres observados, que tiveram a nota das provas inferior a 7 (sete), tencionando capturar os erros para formar a base destes do protótipo; questionário (**Apêndice D**) respondido pelos alunos no segundo semestre de 2019, com a finalidade de avaliar a estratégia de *feedback* e se as recomendações que são exibidas pelo protótipo auxiliaram no entendimento do erro e promoveu seu aprendizado; e, observações em sala de aula, a fim de conhecer os estudantes e suas dificuldades com a ideia de identificar funcionalidades no protótipo que auxilie nos exercícios propostos na disciplina.

A fundamentação teórica é apresentada na próxima seção.

4 FUNDAMENTAÇÃO TEÓRICA

Esta seção está dividida em cinco subseções: a primeira aborda a construção do conhecimento segundo a Epistemologia Genética de Jean Piaget. Esta teoria foi escolhida por apresentar que o conhecimento é construído por meio de experiências anteriores dos indivíduo com o meio em que vivem mediante as interações e transmissões sociais, é dependente do grau de maturidade de suas estruturas cognitivas e é adaptado pelo processo de equilibração entre os dois mecanismos estruturantes da inteligência: assimilação e acomodação. Isto é, o novo conhecimento que o estudante deseja alcançar, depende de saber conteúdos relacionados e que precisam ser aprendidos anteriormente, e isso se aplica, similarmente, à programação de computadores. O discente não consegue seguir adiante no aprendizado de uma linguagem de programação, se não entender o conteúdo e a aplicação dos comandos dela dentro do programa. Ademais, os erros ocorridos durante a sua trajetória nos estudos precisam ser observáveis pelo aluno, ou seja, não basta saber que errou, ele precisa ter acesso aos detalhes e aos elementos deste para que a regulação ocorra.

Na sequência, apresenta-se o conceito do termo *feedback*, destacando os tipos encontrados na literatura e sua importância no aprendizado. Apesar de esta pesquisa tratar do papel do *feedback* na aprendizagem em programação de computadores, os conceitos da linguagem de programação C não serão tratados nesta seção, mesmo que este estudo a utiliza, haja vista que existe um acervo inesgotável sobre o tema em livros e materiais *online*. Ainda nesta seção, mostra-se uma subseção referente aos desafios e dificuldades na aprendizagem das noções básicas de programação, identificadas por pesquisadores nas disciplinas introdutórias de Linguagem de Programação. A penúltima subseção aborda o conceito em volta dos juízes *online*, e a última delineia o embasamento teórico para elaboração de casos de teste, abordando algumas técnicas de teste funcional (caixa-preta) e cobertura de linha de código.

4.1 A Epistemologia Genética e o Erro

O precursor da Epistemologia Genética, Jean Piaget nasceu em 9 de agosto de 1896 na Suíça e, precocemente, mergulhou nos estudos, publicando diversos trabalhos relacionados a temas ligados a biologia e zoologia quando jovem. Ele também se interessava por religião, filosofia, lógica, psicologia, metodologia científica, matemática e química (SEBER, 1997) e seus

primeiros estudos foram essenciais para sua formação científica. As questões relativas à religião o levaram a procurar respostas na filosofia e no livro de Bergson chamado "A evolução criadora", que exerceu grande influência sobre sua formação intelectual. O reconhecimento da ideia da existência de Deus à ideia da vida dá uma nova dimensão à biologia, pois ela o ajudou a explicar a construção do conhecimento (epistemologia), sendo este um dos pontos da teoria piagetiana, ou seja, esclarece a forma pela qual o ser humano chega ao conhecimento, dotado de extrema especificidade, sendo este o tema que Piaget se dedicou plenamente (AZENHA, 2001).

Após concluir o doutorado em 1918, Piaget busca um espaço para estabelecer um laboratório de psicologia para servir de campo para as investigações, não conseguindo se estabelecer. Logo, foi convidado a trabalhar no laboratório de Binet, quando estudava em Sorbonne, tendo sido convidado para normatizar os testes de raciocínio (teste de Burt) em crianças parisienses. Assim, tento um campo para observação e experiências com crianças, iniciou seu trabalho com a aplicação do referido teste em crianças de uma escola primária próxima ao laboratório em que trabalhava. Durante os testes, o que lhe chamou a atenção, segundo Azenha (2001, p. 9) "[...] não foi o aspecto substantivo da tarefa, isto é, aquele relacionado à mensuração e os dados resultantes dos testes, mas o que restava como marginal à aplicação do teste em si". Assim, os erros cometidos pelas crianças na solução dos problemas propostos foi o que mais interessou Piaget. O que o fascinava era a compreensão da lógica implícita ao erro e à interpretação do percurso intelectual da criança em relação ao seu desenvolvimento cognitivo.

O desenvolvimento cognitivo, para Piaget (1983), é composto por estágios que seguem uma sequência linear e progressiva, os quais são universais e, portanto, iguais para todos os indivíduos, independentemente de sua cultura. O estágio foi criado como uma maneira de organizar a atividade mental sob dois aspectos: motor/intelectual e afetivo. Cada um desses estágios parte de uma estrutura já existente, mas, possibilita a construção de estruturas próprias que a distingue da anterior. De acordo com o autor, o desenvolvimento cognitivo ocorre em três estágios de desenvolvimento: o primeiro sensório-motor; o segundo se divide em dois sub-estágios: o pré-operatório e o operatório concreto; e, por último, o estágio operatório formal.

- sensório-motor (do nascimento até dois anos) - é o período do

desenvolvimento da inteligência prática;

- pré-operatório (de dois a sete anos) - é o estágio que se caracteriza pelo processo da construção da linguagem, brincadeira de faz de conta, imitação representativa, imagem mental e, assim como nome do estágio, é um período que não envolve operações;
- operatório concreto (dos sete até onze anos) - as operações estão ainda muito próximas das ações onde procedem, ou seja, são as operações realizadas diretamente sobre o objeto e não sobre hipóteses expressas verbalmente. Isso justifica a dificuldade de algumas crianças em compreender certos conteúdos escolares, quando estes são transmitidos apenas via oral; e
- operatório formal (a partir dos doze anos) - está relacionado às operações proposicionais, formais e hipotético-dedutivas. Esse período se situa na adolescência, e, uma vez conquistadas tais operações, o indivíduo consegue raciocinar sobre enunciados verbais e hipóteses, pensa logicamente e abstrair conceitos, diminuindo a necessidade da representação física dos objetos tratados para entendê-los. É a partir da evolução deste estágio que ele começa a desenvolver as condições cognitivas que o torna apto a desenvolver um programa de computador, uma vez que não dependerá mais do concreto para compreender algo, já que consegue relacionar (abstrair) o enunciado ou o problema referente ao que deve ser feito no computador.

Apesar dos estágios seguirem uma sequência progressiva e serem universais, Piaget (1983, p. 224) identificou diferenças de conhecimento em indivíduos de mesma idade localizadas em diferentes localidades, concluindo que o fato pode estar relacionado à sociedade e ao meio em que a pessoa vive. Além disso, a variação na velocidade e na duração do desenvolvimento pode acontecer em função de quatro fatores: o primeiro fator está relacionado à hereditariedade, à maturação interna; o segundo, diz respeito à experiência física, à ação dos objetos; o terceiro, aborda a transmissão social, o fator educativo entre o meio e a criança pela assimilação; e o último, refere-se à equilibração, a fim de equilibrar os fatores entre si, já que se trata de uma ação estruturante e organizadora do sujeito.

Para Piaget (1983, p. XI), "[…] a inteligência é adaptação e sua função é estruturar o Universo, da mesma maneira que o organismo estrutura o meio ambiente, não havendo

diferenças entre os seres vivos, mas com problemas específicos que implicam em níveis diferentes de organização". Assim como os organismos, os sistemas cognitivos são abertos em um sentido (trocas com o meio) e fechados em outro, enquanto ciclos. As partes constituintes destes e que ajudam o sujeito a refletir sobre cada situação do meio são apresentadas como A, B, C, por exemplo. Os elementos que abastecem esse ciclo são denominados de A', B', C', sucessivamente, obtendo-se o seguinte esquema proposto por Piaget (1976, p. 12): (A x A') → B; (B x B') → C; …; (Z x Z') → A, etc.

Nesse esquema, existe certa ação que os elementos ou subsistemas exercem uns sobre os outros, ou seja, quando acontece uma perturbação exterior, o ciclo reage de forma diferente: pode haver a impossibilidade de uma conservação do estado anterior e a rejeição ou morte do organismo, ou pode acontecer uma modificação do estado que compensa o ciclo já constituído, ocorrendo uma adaptação ou um novo equilíbrio, e este pode ser caracterizado por diferenciações e integrações. (PIAGET, 1976, p. 13)

Os ciclos epistêmicos e seu funcionamento estão relacionados a dois processos fundamentais que constituirão os componentes de todo equilíbrio cognitivo, que permitirão o progresso de toda a equilibração (PIAGET, 1976, p. 12-13), que são: assimilação e acomodação. Piaget (1976) define assimilação como a incorporação de um elemento exterior (objeto, acontecimento, etc.) a um esquema existente do sujeito. O autor comenta que há assimilação recíproca quando:

> [...] dois esquemas ou dois subsistemas se aplicarem aos mesmos objetos (por exemplo: olhar e pegar) ou se coordenarem sem mais necessidade de conteúdo atual. Podemos considerar como uma assimilação recíproca as relações entre um sistema total, caracterizado por suas leis próprias de composição, e os subsistemas que ele engloba em sua diferenciação, porque sua integração num todo é uma assimilação a uma estrutura comum e as diferenciações comportam assimilações segundo condições particulares mais dedutíveis a partir de variações possíveis do todo. (PIAGET, 1976, p. 13-14)

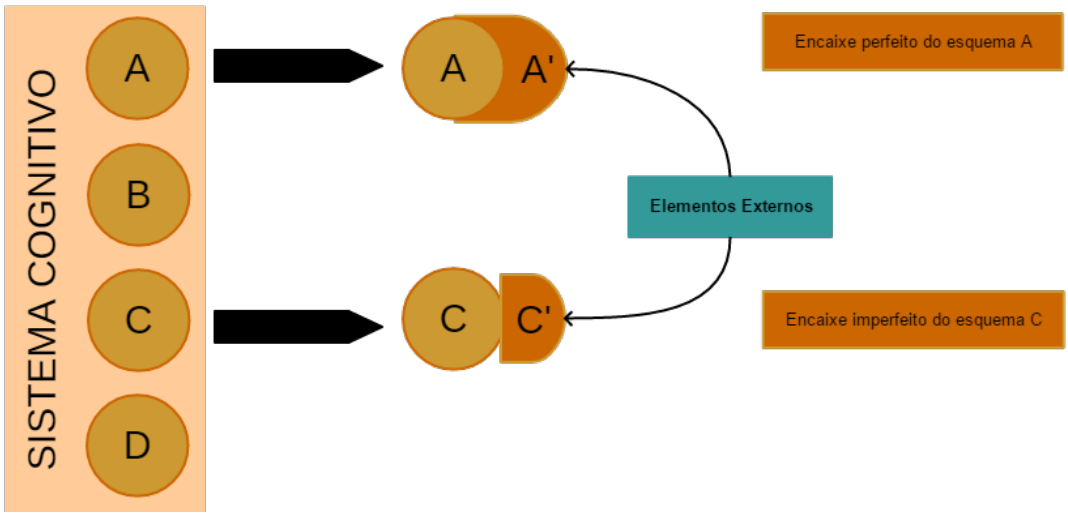
A acomodação se refere à toda modificação dos esquemas de assimilação, por influência de situações desconhecidas, mediante as quais o indivíduo se ajusta - se acomoda - em função das estruturas já existentes: "[...] a acomodação está continuamente subordinada a assimilação (pois é sempre a acomodação de um esquema de assimilação), esta subordinação é mais estreita e sobretudo mais previsível no caso destas acomodações recíprocas que no das adaptações aos objetos exteriores, quando novos dados observáveis surgem de maneira inesperada sob a pressão da experiência." (PIAGET, 1976, p. 14).

Toda vez que um esquema não for suficiente para responder uma situação ou resolver um problema, surge a necessidade de modificá-lo. Por conseguinte, os dois mecanismos são complementares, não havendo um sem o outro. A adaptação do sujeito ocorre pela equilibração desses dois mecanismos que são ativos e dinâmicos e ela possibilita as modificações dos esquemas existentes, para atender à quebra de equilíbrio, provocada pelas situações novas que não tinham um esquema próprio. (PIAGET, 1999)

Conforme apresentado na Figura 2, os elementos A, B, C e D representam o sistema cognitivo do sujeito e a maneira como realizam suas assimilações; os elementos A', B', C' e D' representam os objetos externos que se apresentam como complementares ou modificadores do indivíduo. Em um primeiro momento, o elemento exterior A' se encaixa ao seu correspondente A, o que não acontece com a acomodação do segundo elemento C', que possui diferenças em relação a C e que sofrerá modificações (equilibrações).

Entre a assimilação e a acomodação, há os desequilíbrios como ações para uma nova equilibração: "numa perspectiva de equilibração, uma das fontes de progresso no desenvolvimento dos conhecimentos deve ser procurada nos desequilíbrios como tais, que por si só obrigam um sujeito a ultrapassar seu estado atual e a procurar o que quer que seja em direções novas" (PIAGET, 1976, p. 18), condição representada na Figura 2.

Figura 2 – Processo de assimilação e acomodação de elementos externos



Fonte: Adaptado de TEIXEIRA et al., 2018

Compreender o modo como o ser humano aprende foi uma preocupação e a motivação que desafiou Piaget, e o fez produzir inúmeras obras sobre o desenvolvimento cognitivo, chegando a um tema muito presente na construção do conhecimento: a abstração. Este é um

assunto extremamente relevante quando se fala em programação de computadores, uma vez que o estudante precisa abstrair o problema, coordenar diversas ações cognitivas sobre os conceitos aprendidos em sala de aula, identificar e selecionar os comandos mais apropriados, para, então, apresentar uma solução com um conjunto de instruções algorítmicas para resolver o problema. De acordo com Piaget (1995), é por meio da abstração que o indivíduo extrai propriedades de um determinado objeto e estabelece relações, sendo de dois tipos: empírica e reflexionante.

A **Abstração Empírica** (*empirique*) consiste em retirar qualidades dos objetos, ou das ações, em suas características materiais, isto é, daquilo que pode ser observado, como por exemplo: ouvir um violão, olhar um programa no computador, observar ações de pessoas quando: dirigem um automóvel, programam um código-fonte, plantam uma árvore, entre outros. Tais qualidades, retiradas de objetos (violão, computador, automóveis e aviões) ou de ações (dirigir, programar e plantar), são todas observáveis. Extrair características de objetos/ações observáveis caracteriza as abstrações empíricas.

A **Abstração Reflexionante** (*réfléchissante*) difere da empírica porque o sujeito retira qualidades, não de objetos ou das ações observáveis, mas das coordenações das ações que, por se realizarem internamente ao sujeito (na 'mente'), não são observáveis. A título de exemplo: se um bebê de quatro meses de idade conseguir olhar um objeto, agarrá-lo e levá-lo à boca, ele coordenou três ações ou três esquemas: olhar, agarrar e sugar. Se um programador necessitar resolver um problema, onde precise apresentar uma informação ao usuário final tendo como base a leitura de dados pelo teclado, algoritmicamente, ele coordenará pelo menos cinco ações: ler o material relacionado ao assunto a ser resolvido, entender o problema, traçar o que precisa ser feito, identificar quais comandos utilizar, e elaborar possíveis testes unitários para verificar a corretude do código. Imagina-se que, para chegar ao resultado final, o programador precisará realizar muitas coordenações e operações mentais.

Essas coordenações são realizadas na 'mente' desses indivíduos como operações, oriundas das coordenações de ações. A passagem de uma ou de várias coordenações em uma operação se faz por abstração reflexionante, o que implica equilibração, por assimilações e acomodações, retirando qualidades dessas coordenações ou operações e concebendo, assim, uma nova operação muito mais capaz e abrangente que a anterior.

Piaget (1995, p.193) define abstração reflexionante como:

> [...] um processo que permite construir estruturas novas, em virtude da reorganização de elementos tirados de estruturas anteriores e, como tal, tanto pode funcionar de maneira inconsciente como sob a direção de intenções deliberadas: particularmente, o sujeito de uma investigação ignora, por muito tempo, de que fontes ele tem haurido os mecanismos constitutivos de sua nova construção; e um matemático pode nada saber, sem por isso sentir-se impedido de realizar seu trabalho sobre as raízes psicogenéticas das estruturas elementares que utiliza (como, por ex., a de grupo).

O mecanismo da abstração reflexionante, estudado por Piaget (1995), é justificado pela interação, fundamentada na equilibração. Becker (2014, p. 111) explica esse mecanismo com o processo interação-assimilação-acomodação-equilibração:

> […] uma nova assimilação, cujo esquema foi modificado por acomodação, não consiste numa simples repetição, mas numa nova ação capaz de proporcionar a continuidade das ações anteriores. Se o sujeito agir e o objeto epistemológico não responder, é sinal de que não houve interação. Do mesmo modo, se, fechado um ciclo de ações, não houver modificação dos esquemas assimiladores envolvidos, não houve interação. A ação assimiladora modifica o objeto, enquanto a ação acomodadora modifica o sujeito, seus esquemas assimiladores; o resultado dessas ações é um sujeito mais capaz, mais competente – equilibração majorante. As modificações não provêm diretamente da ação do objeto, mas das modificações que o sujeito faz de seus esquemas assimiladores assim que for impactado ou afetado pelo objeto. A função do objeto, no processo de construção das capacidades ou competências cognitivas, é desequilibrar o sujeito. As novas construções resultam das ações do sujeito em resposta às resistências do objeto.

Os dois tipos de abstração existem em todos os estágios de desenvolvimento do sujeito, desde o início da vida (sensório-motor) até os níveis mais avançados do pensamento, de modo que a abstração empírica sempre é influenciada por abstrações reflexionantes. Piaget (1995) destaca que o processo de abstração reflexionante é realizado por reflexionamentos e reflexões, que se diferem, no momento em que o **reflexionamento** pretende extrair qualidades das coordenações de um patamar inferior para um patamar superior. Behar (2009, p. 133-134) avulta que o reflexionamento está presente em todos os estágios de desenvolvimento do indivíduo, e estabelece:

> Em seu primeiro patamar, o mais elementar, o reflexionamento conduz que um movimento sensório-motor a uma conceituação que o agrega, permitindo conceituar um observável a partir do reflexionamento do observável sobre a ação. O patamar seguinte é o da reconstituição da sequência de ações, reunindo-as em um todo coordenado, sendo que o sujeito pode ou não fazer uma narrativa dessa constituição. O terceiro patamar do reflexionamento é o da comparação, em que o sujeito compara a ação reconstituída com outras, estabelecendo relações e distinguindo o que tem de análogo e de diferente. As possibilidades desse patamar são indefinidas, podendo ocorrer vários graus de reflexões sobre reflexões, ou seja, metarreflexão (pensamento reflexivo), o que permite ao sujeito encontrar as razões da conexão. No segundo e no terceiro patamar tem-se uma generalização que permite um novo

> reflexionamento dos observáveis, mas dessa vez em um nível mais elaborado e enriquecido. [...]. Existe uma interdependência entre o reflexionamento e a reflexão, de modo que quanto mais o reflexionamento evolui, maior a complexidade das reorganizações e das formalizações que derivam desse processo.

A **reflexão** objetiva reorganizar o que foi levado pelo reflexionamento ao patamar superior em torno do que já existia. Ao reorganizar, surge algo novo, uma nova forma (em relação aos conteúdos assimilados); algo que não existia anteriormente. Piaget (1995, p. 274-275) delineia que a reflexão é um "ato mental de reconstrução e reorganização sobre o patamar superior daquilo que foi assim transferido do inferior". Para ele, o reflexionamento e a reflexão dão origem a um processo contínuo em espiral:

> […] Todo reflexionamento de conteúdos (observáveis) supõe a intervenção de uma forma (reflexão), e os conteúdos assim transferidos exigem a construção de novas formas devidas à reflexão. Há, assim, pois uma alternância ininterrupta de reflexionamentos → reflexões → reflexionamentos; e (ou) de conteúdos → formas → conteúdos reelaborados → novas formas, etc., de domínios cada vez mais amplos, sem fim e, sobretudo, sem começo absoluto. (PIAGET, 1995, p.276-277)

Assim, todo reflexionamento de observáveis supõe a intervenção de uma reflexão, desenhando um movimento entre forma e conteúdo que converge para a construção de novas estruturas. Logo, estas são compostas de forma (reflexão) e conteúdo (observáveis), sendo que sua evolução é decorrente da alternância ininterrupta entre reflexionamento e reflexão.

Mas, como Piaget explica as mudanças ocorridas dentro de cada estágio e entre os estágios? Que mecanismo cognitivo garante essas mudanças e transformações ocorridas no decorrer do desenvolvimento cognitivo? Para responder a essas questões, ele se apoia na Teoria da Equilibração, que para criar e explicá-la, retoma dois postulados dos estudos sobre abstração reflexionante:

> **Primeiro Postulado:** todo esquema de assimilação tende a alimentar-se, isto é, a incorporar elementos que lhe são exteriores e compatíveis com sua natureza [...]
>
> **Segundo Postulado:** todo esquema de assimilação é obrigado a se acomodar aos elementos que assimila, isto é, a se modificar em função de suas particularidades, mas, sem com isso, perder sua continuidade [...] (PIAGET, l976, p.14).

No ato de conhecer, o sujeito age sobre o objeto e sua ação se transforma. Todavia, há momentos em que esse resiste aos instrumentos de assimilação disponíveis, levando aquele a se aperfeiçoar ou a se reorganizar, sendo capaz de assimilar objetos cada vez mais complexos. Assimilação e acomodação passam a ser interdependentes e conservam as aquisições novas, conciliando com as antigas. Dessa maneira, conhecer pode tanto transformar o objeto, quanto a si mesmo. É nesse quesito que aparece o segundo postulado, onde o autor afirma a

necessidade de equilíbrio entre ambos.

A partir desses postulados, Piaget define três formas de equilibração:

> 1º) interação fundamental de início entre o sujeito e os objetos, há primeiramente a equilibração entre a assimilação destes esquemas de ações e a acomodação destes últimos aos objetos. Notamos aí, já um começo de conservação mútua, pois o objeto é necessário ao desenrolar da ação e, reciprocamente, é o esquema da assimilação que confere sua significação ao objeto, transformando-o (deslocamento, utilização, etc.) graças a esta ação: assimilação e acomodação;
>
> 2º) uma equilibração assegurar às interações entre os subsistemas. Os subsistemas se constroem comumente em velocidades diferentes, com decolagens temporais mais ou menos importantes: há, pois, aí, razões de desequilíbrios possíveis e a necessidade de uma equilibração. Mas esta é de tipo diferente exterior está exposta à intervenção de múltiplos obstáculos inesperados, devidos à resistência dos objetos, a assimilação recíproca de dois subsistemas válidos e sua acomodação recíproca, cedo ou tarde, serão bem-sucedidas e conduzirão, pois, a uma conservação mútua;
>
> 3º) o equilíbrio progressivo da diferenciação e da integração, logo das relações que unem subsistemas a uma totalidade que as engloba. Esta terceira forma de equilibração não se confunde com a segunda, pois acrescenta uma hierarquia às simples relações entre colaterais. Na verdade, uma totalidade é caracterizada por suas leis próprias de composição, constituindo um ciclo de operações interdependentes e de ordem superior aos caracteres particulares dos subsistemas. (Piaget, l976, p. 15-16).

Percebe-se que os três tipos de equilibrações têm em comum o estar referenciados ao equilíbrio entre a assimilação e a acomodação. La Taille (1997) sublinha que assimilação, acomodação e equilibração descrevem o processo de evolução da inteligência. Os desequilíbrios só terão sentido a partir do momento em que desencadear a saída de determinado estado em busca de uma reequilibração, estabelecendo, então, um progresso. Assim, as reequilibrações representam, não a volta ao estado anterior, mas a modificação e o melhoramento dos esquemas do sujeito.

Faz parte desse processo as regulações, que definem o 'como' da equilibração e das reequilibrações. Uma regulação, nas palavras de Piaget (1976, p. 24), acontece quando "a retomada de A' de uma ação A é modificada pelos resultados desta, logo quando de um efeito contrário dos resultados de A sobre seu novo desenvolvimento A' ". A regulação pode se manifestar por meio de uma correção (*feedback* negativo) ou pelo seu reforço (*feedback* positivo). Piaget (1976, p. 29-30) aponta que o *feedback* negativo consiste em uma correção, ou seja, procura afastar obstáculos, anular ações e modificar esquemas; o *feedback* positivo é um reforço a qualquer negação e generaliza elementos que compõem um esquema de assimilação, remediando uma lacuna. Para o autor, *feedbacks* positivos e negativos funcionam ao mesmo tempo, um reforça e o outro corrige, e são necessários à formação de qualquer

estrutura ou esquema.

Participam desse fluxo as perturbações que Piaget (1976, p. 24) define "como algo que serve de obstáculo a uma assimilação, tal como atingir um objetivo, todas as regulações são, do ponto de vista do sujeito, reações a perturbações", ou seja, a perturbação age sobre o sujeito que se utiliza de regulações para reagir aos primeiros sinais desse desconforto. Piaget distingue duas grandes classes de perturbações:

> A primeira compreende as que se opõem às acomodações: resistências do objeto, obstáculo às assimilações recíprocas de esquemas ou subsistemas, etc. São em resumo as causas de fracassos ou de erros, na medida em que o sujeito se torna consciente disso, e as regulações que lhe correspondem comportam, então, *feedbacks* negativos.
>
> A segunda classe de perturbações, fontes de desequilíbrios, consiste, ao contrário, em lacunas, que deixem as necessidades insatisfeitas e se traduzem pela insuficiente alimentação de um esquema. Mas convém precisar, e isto é essencial, que não é qualquer lacuna que constitui uma perturbação. [...] Em compensação, uma lacuna se torna uma perturbação quando se trata da ausência de um objeto ou das condições de uma situação que seriam necessárias para concluir uma ação, ou ainda da carência de conhecimento que seria indispensável para resolver um problema. A lacuna enquanto perturbação. É, pois, sempre relativa a um esquema de assimilação já ativado, e o tipo de regulação que lhe corresponde comporta então um *feedback* positivo, em prolongamento da atividade assimiladora deste esquema. (PIAGET, 1976, p. 25)

Para Piaget (1976), um fato observável é dependente e condicionado por dados observáveis e coordenações anteriores. Esses dados e coordenações configuram determinadas situações que, dependendo do tipo de perturbação que se crie no sistema cognitivo do sujeito, implicarão em regulações, seja por reforço (*feedback* positivo), seja por correção (*feedback* negativo).

Todavia, Piaget (1976) salienta que toda regulação é uma reação a uma perturbação e a recíproca nem sempre é verdadeira.

> […] não poderíamos falar de regulação quando a perturbação provoca simplesmente uma repetição a ação, sem qualquer mudança, e com a ilusória esperança de ser melhor sucedida (como ocorre frequentemente com a criança); ainda menos quando o obstáculo leva, ao cessar da ação, nem mesmo quando o sujeito, interessado por um aspecto imprevisto da perturbação, empenha sua atividade numa outra direção. Nestes diferentes casos, não se poderia, de fato, falar de uma retomada de A' da ação A com modificação de A' sob o efeito do resultado de A, e na ausência desta regulação não há reequilibração. (PIAGET, 1976, p. 25)

Para haver regulação é necessária a intervenção de um regulador, que é interno do sujeito. Ressalta-se que o equilíbrio proporcionado pelas regulações se torna importante para o processo de evolução dos esquemas.

Conhecer é interpretar, e a qualidade das interpretações depende dos níveis de estrutura da inteligência. Porém, quando o tema é o erro, esta afirmação sofre algumas suposições. Aquele pode ser um indicativo do motivo gerador da falha, sendo possível se chegar a algumas hipóteses (KUTZKE, 2015): ou o indivíduo não está pronto (não possui a maturidade cognitiva) para desenvolver a atividade proposta, ou o indivíduo tem o conhecimento sobre o assunto, mas o utilizou incorretamente para desenvolver a atividade, ou existe uma lacuna de conhecimento em sua aprendizagem.

Cerqueira (2009) destaca três motivos geradores do erro: primeiro, o indivíduo possui a estrutura de pensamento para resolução de determinado problema, detém o conhecimento sobre o assunto, mas utilizou meios inadequados para solucionar a tarefa. É o erro sobre a utilização do seu conhecimento na resolução que, muitas vezes, ocorre pela falta de prática, que se faz necessária na fixação. Segundo o sujeito erra porque o conhecimento que possui não é suficiente, porque existem lacunas em sua estratégia de pensamento, dificultando a assimilação dos dados disponíveis. Na tentativa de resolver o problema, o indivíduo erra e faz suas correções em função da falha, trata-se de um erro construtivo, pois ele vai em busca de novos conhecimentos. Terceiro, o sujeito tem dificuldade de compreender o que é solicitado e erra, pois não tem a estrutura de pensamento necessária para a resolução e não há entendimento do que lhe é pedido, ou seja, o indivíduo não atingiu a maturidade cognitiva necessária para aprender sobre o tema proposto.

La Taille (1997) chama de erro, no campo do conhecimento, dois tipos: aqueles que contradizem os conceitos consolidados pela humanidade (por exemplo: a Terra gira em torno do sol); e, aqueles que, à medida que o indivíduo amadurece, são abandonadas definitivamente (por exemplo: a Lua me segue quando passeio à noite). Tais erros são entendidos de duas formas: um é negativo e o outro, positivo.

É considerado um erro negativo aquele que evidencia uma diferença entre o conhecimento correto e o conhecimento incorreto. Assim, quando se afirma que o Sol gira em torno da Terra, o indivíduo possui um pensamento errado quanto à teoria do universo. A importância da presença desses erros é um diagnóstico a respeito do nível de desenvolvimento da inteligência da pessoa, eles podem dar pistas importantes sobre as reais capacidades de assimilação (LA TAILLE, 1997). O erro positivo está relacionado ao ponto de vista dos indivíduos sobre a representação de seu mundo, que muda com a maturidade.

Demo (2001, p.50) apresenta o erro como um elemento comum na formação do conhecimento, uma vez que "o erro não é um corpo estranho, uma falha na aprendizagem. Ele é essencial, é parte do processo. Ninguém aprende sem errar. O homem tem uma estrutura cerebral ligada ao erro, é intrínseco ao saber-pensar a capacidade de avaliar e refinar, por acerto e erro, até chegar a uma aproximação final."

La Taille (1997) comenta que o erro só terá valor como fonte de enriquecimento se for 'observável' pelo estudante; ou seja, o aluno deve ter o acesso à qualidade do erro (dados observáveis) mediante informações que o façam refletir (reflexionamento e reflexão) e o ajudem a definir estratégias de como chegar no resultado, e não somente saber que errou. Esse retorno ao aluno deve acontecer de forma natural, objetivando trabalhar o erro e disponibilizando elementos em torno dele, para que a regulação, em virtude da perturbação gerada pelo erro negativo, ocorra. De acordo com o autor, um erro é mais produtivo do que um sucesso prematuro. Tal afirmação está baseada na 'sorte' do aluno de acertar a resolução de um problema não pelo conhecimento, mas pelo 'chute'. Se acertar, a tendência será repetir as ações em um outro momento sem refletir os passos realizados para obter o sucesso; caso contrário, a tendência será refletir sobre o erro, sobre o problema e sobre as ações que aplicou para resolvê-lo, tornando o erro uma fonte de tomada de consciência.

De vilão a um valioso aliado da pedagogia, o erro ganhou um lugar de destaque no processo de aprendizagem e desenvolvimento cognitivo. Mas, para isso, La Taille (1997, p.38) enumera três ponderações pedagógicas:

- erro, necessariamente, precisa ser observável pelo aluno, ou seja, não basta saber que errou, ele precisa ter acesso aos detalhes e aos elementos do erro para que a regulação, em virtude do *feedback* negativo, ocorra;
- erro deve ser observável, e isso não depende apenas da tarefa, mas do nível de desenvolvimento do indivíduo. Se o aluno desconhece o assunto que está resolvendo (lacuna de conhecimento), o resultado do erro não terá significado para ele, e o efeito de sua observação será nulo; e
- erro e regulação - a regulação ocorre a partir de uma perturbação e a situação do erro é apenas um dos cenários que a causam. Outra perturbação é decorrente da lacuna de conhecimento. Para La Taille (1997, p.39), o "não saber" é tão possibilitador de desenvolvimento quanto o "saber errado" ou

"acreditar que sabe".

Contudo, o objetivo do aprendizado é sempre atingir o certo. Com isso, o professor deverá sempre encorajar os alunos a tentativas rumo a respostas corretas, todavia, é preciso valorizar os seus erros, e não deixar de dizer: "o que você fez é muito interessante, mas ainda não está correto", como um *feedback* positivo.

4.2 Feedback

É através da comunicação que as pessoas compartilham informações e trocam experiências ao longo dos anos, tornando o ato de se comunicar algo essencial para a existência. Quando o processo de comunicação é estabelecido, há interpretação e entendimento das mensagens, do que se conclui que a comunicação foi bem-sucedida. Em sala de aula, a comunicação é primordial, uma vez que ocorre entre professor↔aluno. Para isso, diferentes formas são utilizadas para estabelecer essa comunicação, sendo uma delas o *feedback*.

Arshavskiy (2019) descreve dois tipos de *feedbacks*, que os indivíduos recebem diariamente, seja por meio de gestos ou por palavras ou por texto escrito, ou seja, formas intrínseca e extrínseca de comunicação. Segundo a autora, o primeiro é considerado um tipo indireto de *feedback,* como o balançar da cabeça do professor sempre que o aluno responder incorretamente. O segundo é um tipo direto de *feedback*, que objetiva comentar o desempenho do aluno diretamente, como "Está correto" ou "Isso está incorreto".

Zeferino et al. (2007) estipulam que o conceito de *feedback* na área educacional se refere à informação passada ao aluno que descreve e/ou discute seu desempenho em determinada situação ou atividade, gerando uma conscientização para a aprendizagem, uma vez que as divergências se sobressaem entre o resultado pretendido e o real, incentivando a mudança. Se a informação for capaz de causar alteração no padrão de desempenho observado, houve um processo de aprendizagem.

Na visão de Williams (2005, p.19), o *feedback* é importante para todos os indivíduos. "É a base de todas as relações interpessoais. É o que determina como as pessoas pensam, como se sentem, como reagem aos outros e, em grande parte, é o que determina como as pessoas encaram suas responsabilidades do dia-a-dia."

De acordo com Abreu-e-Lima & Alves (2011) e Mory (2004), o *feedback* foi incorporado a muitas abordagens de ensino-aprendizagem, desde as primeiras visões do behaviorismo, cognitivismo, passando até pelo construtivismo, e continua sendo parte fundamental do processo, independentemente do modelo de aprendizagem adotado. No behaviorismo, o *feedback* era um mecanismo adotado para reforçar o acerto e descartar respostas incorretas. Segundo Cardoso (2011), nessa época, o erro era percebido como algo negativo, que deveria ser suprimido; as respostas certas deveriam ser reforçadas e recebiam *feedback* positivo. Pela teoria behaviorista, o *feedback* não fornecia meios para correção dos erros, o que limitava sua utilização na aprendizagem.

Em movimentos seguintes, principalmente a partir dos anos 70, com o cognitivismo, o *feedback* passa a ser elemento importante para o conhecimento dos alunos, deixando de ser visto como mero recurso de reforço. Desse jeito, a informação apresentada no *feedback,* sendo um conhecimento novo, passa a ser assimilada e posteriormente acomodada em seus esquemas anteriores. Sem *feedback*, os estudantes possuem mais dificuldades em identificar o conteúdo no qual devem investir mais tempo de estudo, ou não conseguem saber se a solução proposta está satisfatória ou não. É pelo *feedback* que aquele que aprende toma conhecimento de como pensar, agir e focar seus esforços no que realmente interessa, para desenvolver formas mais precisas com a intenção de atingir seus objetivos, porque o "*feedback* contribui com a prática reflexiva, ou capacidade do profissional de rever suas próprias conclusões, raciocínio e decisões" (ZEFERINO et al., 2007, p.177).

Williams (2005, p.52) apresenta quatro tipos de *feedback*: **positivo**, que tem como função básica fortalecer um comportamento que se deseja que se repita, e deve ser aplicado mesmo quando o aluno já está agindo conforme o esperado, o que evita a falta de motivação; **corretivo**, tem como meta mudar um comportamento. O fornecimento deste tipo de *feedback* permite que os alunos avancem em direção ao objetivo da disciplina e informa o nível de domínio que o estudante possui sobre o conteúdo em estudo. Ele mostra quais tópicos os estudantes precisam revisar ou revisitar.

Arshavskiy (2019) comenta que, ao se fornecer o *feedback* corretivo aos alunos, é preciso utilizar um 'tom' amigável e favorável, mesmo que a questão tenha sido respondida incorretamente, o que corrobora a opinião de Williams (2005), que afirma que este tipo de *feedback* precisa muita atenção para não se tornar ofensivo, já que o objetivo de um *feedback*

corretivo não é ofender ou recriminar os alunos, mas promover o aprendizado; **insignificante**, trata-se de um *feedback* vago ou genérico, que acarreta falta de entendimento sobre o seu propósito, não provocando reação alguma ou contrária ao que é esperado; e **ofensivo**, que não orienta, não permite aprendizagem com os erros e não motiva o estudante. Pode gerar conflitos entre professor e aluno, resultando em desmotivação. O tipo de *feedback* utilizado é que determina a resposta obtida pelo aluno.

Além dos quatro tipos de *feedback* apresentados por Williams (2005), Shute (2007) expõe o entendimento de *feedback* formativo que é a informação comunicada ao aluno, para modificar seu pensamento ou comportamento, a fim de melhorar seu aprendizado. O principal objetivo desse *feedback* é aumentar o conhecimento, as habilidades e a compreensão do aluno em algum conteúdo, específico ou geral, como a resolução de problemas. Conforme a autora, esse *feedback* apresenta diferentes funções, tipos de informações, ritmos e constância de apresentação das mensagens e mecanismos cognitivos pelos quais ele pode ser usado pelo aprendiz.

O *feedback* formativo apresenta três mecanismos cognitivos que podem ser utilizados (SHUTE, 2007): primeiro, para indicar uma lacuna entre um nível atual de desempenho comparado a um nível desejado ou meta a ser atingida. Resolver essa lacuna pode motivar níveis mais altos de esforço, ou seja, pode reduzir a incerteza sobre quão bem (ou mal) o aluno está executando uma tarefa. Segundo, pode reduzir a carga cognitiva de um aluno, especialmente dos iniciantes ou de alunos com dificuldades. Esses alunos podem se tornar cognitivamente sobrecarregados durante o aprendizado, devido às demandas de alto desempenho, sendo possível que se beneficiem com o *feedback* de apoio projetado para diminuir a carga cognitiva. Terceiro, o *feedback* pode fornecer informações que podem ser úteis para corrigir erros ou equívocos.

No que diz respeito a efetividade do *feedback*, Zeferino et al. (2007) abordam que ela é maior quando é: assertiva, respeitosa, descritiva, oportuna e específica. Assertiva no sentido de a comunicação ser clara, objetiva e direta, avaliando os impactos e as consequências desse processo e propondo melhorias e mudanças (OLIVEIRA, 2018). Respeitosa, um elemento essencial para o sucesso do *feedback*, professor e aluno devem estar em consonância durante todo o processo. Descritiva, por se tratar de um processo que avalia de forma objetiva, sem julgamentos pessoais. Em relação a ser oportuna, é porque o *feedback* deve ter o momento

certo e local adequado para ser transmitido; e, específica, pois é importante que o professor deixe claras as observações relatadas e especifique, destacadamente, os pontos positivos e negativos.

Com base no exposto, o Quadro 2 foi elaborado resumindo algumas características em torno dos conceitos de *feedback* de cada autor.

Quadro 2 - Resumo sobre características em *feedbacks* encontrados na literatura

Propósito		
Tipos de Informação	Avaliativo	Refere-se ao julgamento da resposta do aluno - se está correta ou errada. (SHUTE, 2007)
	Informativo	Oferece dicas e orientações sobre como chegar ao resultado esperado. (SHUTE, 2007)
Complexidade do Conteúdo	Complexo	São mensagens muito longas ou muito difíceis de entender. (SHUTE, 2007)
	Não Complexo	São mensagens curtas, claras e objetivas. (SHUTE, 2007)
Função	Direta	Informa ao aluno o que precisa ser corrigido ou revisado. (SHUTE, 2007)
	Sugestiva	Fornece comentários e sugestões a fim de orientar os alunos em sua própria revisão. (SHUTE, 2007)
Amplitude	Específico	São mensagens que abordam com mais profundidade o tópico, a resposta e o erro. (SHUTE, 2007)
	Genérico	São mensagens que abordam o erro, abrangentemente, com exemplos ou outro tipo de sugestão global. (SHUTE, 2007)
Direcionamento		
Individual	São mensagens direcionadas a somente um aluno ou ao professor (PYKE & SHERLOCK, 2010)	
Em grupo	São mensagens direcionadas a todos os alunos (PYKE & SHERLOCK, 2010)	
Momento (ritmo e constância)		
Síncrono (imediato)	São mensagens recebidas imediatamente após a realização de uma tarefa. (SHUTE, 2007)	
Assíncrono (adiado)	São mensagens recebidas minutos, horas ou semanas após a realização de uma atividade. (SHUTE, 2007)	
Fonte		
Professor	Mensagens enviadas pelo professor, relatando o progresso dos alunos, encorajando-os na discussão e aprofundamento de conteúdo (PAIVA, 2003).	
Computador (automático)	Retorno de informações às respostas do aluno, oferecido pelo computador durante, ou após, a realização de uma atividade *online* (FILATRO, 2008).	
Aluno	Mensagens enviadas entre os estudantes ou de alunos ao professor. (CARDOSO, 2011)	

Fonte: Adaptado de CARDOSO, 2018

Em função desta pesquisa objetivar e mediar o erro para que auxilie na aprendizagem

do estudante, é preciso identificar as características do *feedback* e seus propósitos para implementar no protótipo. Uma das características importantes é a apresentação dos *feedbacks* avaliativo e informativo, quando o estudante executa uma questão e seu retorno não está em conformidade com o resultado esperado, estabelecido nos casos de teste cadastrados pelo professor. O recurso de *feedback* informativo começará a ser exibido a partir da persistência do erro em mais de três vezes.

Conforme o trabalho de Deitos (2018), que propôs um jogo para auxiliar o aluno do terceiro ano do Ensino Fundamental no aprendizado da escrita na forma de um ditado digital, onde, para cada palavra contida no jogo, um *feedback* está associado à sua correção. Ao errar três vezes, o jogador tem acesso à escrita correta da palavra, e ainda precisa escrevê-la corretamente para que possa prosseguir jogando e, assim, sistematizar a escrita correta. Segundo a autora, essa decisão teve como objetivo estimular o estudante na tentativa da escrita correta, não prejudicando-o ao errar. Essa estratégia foi utilizada no protótipo, na medida em que o ambiente espera o estudante na tentativa de resolução da questão. Além disso, pretende-se direcionar o *feedback* individualmente, com uma amplitude específica para trabalhar com mais profundidade o erro. Outro ponto importante é a constância das mensagens. Para Filatro (2008), em atividades mais complexas, que envolvam habilidades cognitivas e permitam mais de uma solução, a autora recomenda que o *feedback* seja exibido durante a realização da atividade, e não após a sua conclusão. Neste caso, no protótipo, as mensagens serão exibidas imediatamente após a execução da questão, e não após a sua finalização.

Independentemente das características mais indicadas de *feedback*, encontradas na literatura, todos abordam os cuidados mínimos ao elaborá-los com a finalidade de auxiliar os estudantes na aprendizagem. Deve-se ter cuidado com: a linguagem utilizada, que precisa ser simples, organizada, adequada ao contexto; o conteúdo, que pode receber uma abordagem direta e/ou sugestiva e equilibradas de acordo com o desenvolvimento do aluno e das tarefas; e o ritmo e a constância dos *feedbacks*, que implicam considerações sobre o tempo de resposta aos alunos (ABREU-E-LIMA & ALVES, 2011). Além disso, dar *feedback* é um desafio, pois é preciso entender como os indivíduos reagem ao receber um retorno.

4.3 As dificuldades dos alunos nas disciplinas introdutórias de programação

O ensino e a aprendizagem de programação são tarefas complexas. Algumas pesquisas apontam problemas que vão desde a dificuldade dos alunos em compreender os conceitos introdutórios de programação até a falta de motivação dos estudantes em realizar adequadamente a atividade de programação.

A dificuldade na aprendizagem em programação de computadores é objeto de estudo por diversas Instituições de Ensino. Durante as investigações, encontrou-se alguns trabalhos que abordam esse assunto, sendo todos realizados em disciplinas de programação de cursos de ensino superior, e que dão uma dimensão dos problemas enfrentados pelos estudantes. Apesar das pesquisas terem sido realizadas entre 2013 e 2016, os dados mapeados se repetem nos dias atuais.

Sevella et al. (2013) elaboraram um estudo com 20 alunos na disciplina de programação do curso de Ciência da Computação com o objetivo de verificar quais eram as principais barreiras enfrentadas pelos estudantes ao aprenderem os conceitos de programação. A hipótese dos autores era de que as dificuldades que os estudantes enfrentavam, estavam mais relacionadas aos conceitos da linguagem. Depois da análise dos resultados, concluíram que as dificuldades se encontravam na construção dos códigos de programação, e não na parte conceitual como era a hipótese inicial. Durante essa pesquisa, os autores constataram que os participantes tiveram dificuldade para aplicar esses conceitos básicos, tais como: entrada e saída de dados, estruturas condicionais, estruturas de repetição, entre outros.

Lahtinen et al. (2005, p.4) igualmente notaram que “o maior problema dos programadores novatos não é compreensão de conceitos básicos, mas aprender a aplicá-los” e acrescentam: “quanto mais práticas e concretas forem as situações e materiais de aprendizagem, mais o aprendizado acontece. Aprender fazendo deve fazer parte dos estudos o tempo todo.” A ação de aprender fazendo, auxilia na compreensão e na fixação dos conceitos de programação de computadores e ajuda a desenvolver certas habilidades necessárias que só são adquiridas com a prática.

Bosse & Gerosa (2016) realizaram uma pesquisa na Universidade de São Paulo (USP) para identificar padrões de dificuldade relacionados à aprendizagem de como programar. Ela foi realizada com dois públicos: professores e alunos. O primeiro grupo era composto por 16

professores, sendo dez instrutores selecionados aleatoriamente e seis professores que ministravam aulas de programação. O objetivo foi mapear as principais dificuldades dos alunos na visão dos docentes. O segundo grupo continha 34 alunos da Universidade e a coleta de dados foi efetuada por três métodos. Primeiramente, foram aplicadas entrevistas individuais utilizando a técnica *Think Aloud* (que consiste em observar a maneira como os usuários executam tarefas propostas em ambientes controlados). A tarefa para este método foi a resolução de quatro exercícios de programação na linguagem C no *Virtual Programming Lab* (VPL), que é um plugin para o Moodle, desenvolvido pela Universidade de Las Palmas, nas Ilhas Canárias (ULPGC). O áudio e a tela do computador foram capturados para posterior análise.

O segundo método foi fundamentado em relatos de diários. Esse método foi escolhido por possibilitar que informações sobre eventos e experiências fossem obtidas na perspectiva do aluno de maneira espontânea, reduzindo o tempo entre a ocorrência do evento e o momento em que é relatado aos pesquisadores. O terceiro método incluiu uma pesquisa com perguntas específicas sobre possíveis dificuldades encontradas no curso de introdução à programação. Esta pesquisa tinha como objetivo confirmar, quantitativamente, os padrões observados pelo primeiro método. Foram também analisadas as bases de dados com os resultados das avaliações dos alunos nos cursos introdutórios de programação, na USP durante os anos de 2010 a 2014. Das 18.784 inscrições realizadas no período analisado, 30% resultaram em reprovações ou desistências, o que foi constante ao longo dos anos. Foi identificada uma taxa de reprovação maior para os alunos que não eram da área de informática, chegando a 30,3% em comparação com 25,1% dos estudantes da área. Foi descoberto, com esta pesquisa, que mais de 25% dos alunos aprovados já haviam participado duas ou mais vezes da disciplina.

Do ponto de vista dos alunos, os pesquisadores conseguiram identificar algumas dificuldades observadas pelos mesmos e corroboradas pelos escritos nos diários, que foram: erro de sintaxe, problema mais frequente relatado; e falta de entendimento em como declarar variáveis. Foram mencionados problemas de compreensão relacionados a alguns conceitos da linguagem C, o uso da IDE, além de reclamações sobre o idioma das mensagens de erro. Quanto aos erros de sintaxe, as dificuldades foram unânimes em todas as entrevistas: a abertura e fechamento de estruturas com colchetes e ortografia correta dos comandos

(semântica).

Alguns erros que foram dignos de destaque pelos pesquisadores: (a) tentativa de ler os dados de uma matriz sem índice; (b) criação de uma função sem nome, além da declaração incorreta das variáveis para receber os parâmetros, e (c) falta de ponto e vírgula terminando uma estrutura de repetição e seleção que não foi iniciada. O maior desapontamento dos alunos, que foi notado pelos docentes, era quando ocorriam os erros semânticos em comparação aos erros de sintaxe, pois com estes os discentes estavam mais acostumados, mas, quando aconteciam os erros semânticos, eles abandonavam o exercício mais rapidamente, porque percebiam que era preciso de mais tempo para corrigi-los.

Do ponto de vista dos professores, a principal dificuldade citada está relacionada com o raciocínio lógico. Para isso, eles tentaram utilizar o pseudocódigo em sala de aula para explicar rapidamente o conceito, para depois abordar a linguagem de programação. Outro ponto de dificuldade identificado foi com o uso dos operadores - aritméticos, lógicos e relacionais. Os alunos se confundem com precedência. Há uma clara dificuldade em diferenciar os operadores lógicos 'e' e 'ou'. Entre as estruturas de seleção/decisão e controle/repetição, os estudantes citaram que o comando *while* dava a impressão de se ter mais controle sobre a estrutura, e, por isso, preferiam este comando ao invés do comando *for*. Tais informações corroboram os escritos pelos alunos em seus diários. Eles também possuem dificuldades com o encadeamento de comandos de controle e como definir a condição de finalização. Na estrutura de seleção, é difícil visualizar os pares *if..else*, além de que eles misturam conceitos entre estruturas de decisão e de controle. No que tange às matrizes, acontece o ‘esquecimento de colocar o índice’ em relação à posição, e com relação ao conceito de função, não conseguem entender o escopo das variáveis e a importância do valor de retorno.

Os professores acreditam que não há grandes problemas com a passagem de parâmetros, no entanto, quando essa é por referência e a linguagem usada é C, a dificuldade aumenta em função do uso do conhecimento sobre ponteiros para manipular os valores atribuídos às variáveis. Os docentes comentam que existem alguns fatores que contribuem com a adversidade do ensino de programação, como: heterogeneidade dos grupos, baixa participação em sala de aula, baixa frequência, classes muito grandes, desinteresse pelo aprendizado, a linguagem de programação adotada e o trauma dos alunos que repetem a

disciplina.

Gomes et al. (2015a) colaboraram, com seu estudo, na identificação dos principais erros cometidos pelos estudantes, durante a programação com a linguagem C, além de medirem a quantidade de ocorrências. Para tanto, utilizaram uma ferramenta chamada CFacil[1] em uma turma de algoritmos e programação, sendo a amostra de 22 alunos, e o objetivo foi capturar em um arquivo (*log*) as compilações dos exercícios realizados por eles durante um período de três meses e, após a coleta, foi aplicado um questionário com os mesmos estudantes para confirmar os dados. Após a coleta, técnicas de mineração de dados foram usadas, a fim de identificar termos mais relevantes no texto gravado. Durante os experimentos, os autores obtiveram 1.297 compilações com o instrumento de coleta, deste total, 922 (71%) compilações não obtiveram sucesso. Dessa amostragem, foram registrados 1.249 alertas (*warning*) e 2.236 erros (*error*), sendo possível concluir que, para cada compilação errada, havia, em média, um alerta e dois erros. Na percepção de Gomes et a. (2015a), da verificação dos resultados obtidos pela ferramenta CFácil, foi possível reconhecer os seguintes erros:

- falta de fechamento de parênteses e/ou chaves em estruturas como *if, else, printf, scanf*;
- falta de ponto e vírgula no final do comando;
- erro na leitura de strings, ou seja, leitura de uma string com & (e comercial);
- erro ao usar funções de leitura do teclado, como: *fgets()* e *fgetc()*;
- falta de argumentos nas funções *printf()* e *scanf()*; e
- erros por não declaração ou redeclaração de variáveis.

A análise dos dados permitiu constatar que a grande maioria dos erros cometidos pelos alunos foram erros de sintaxe da linguagem C, o que foi confirmado com as respostas do questionário aplicado. A principal dificuldade encontrada pelos discentes foi identificar erros no próprio código, com um percentual de 50%; seguido da dificuldade em construir o programa com 36% e 14%, para compilar o código. As dificuldades na interpretação das mensagens de erros, também foram mapeadas, e apontaram que 36% dos estudantes acharam que a mensagem não era clara e não apontava corretamente para o local do erro; seguido de

1 ferramenta de monitoramento das saídas geradas durante o processo de compilação utilizando o compilador GCC, utilizando a tecnologia de Shell Script.

32% que acharam que essa dificuldade estava relacionada ao idioma da mensagem (inglês) e 32% sinalizam que a organização da mensagem pelo compilador (número da linha, descrição do erro) não era de fácil compreensão.

Segundo os alunos, os erros que mais ocorrem quando eles elaboram um programa são: falta de ponto e vírgula ao final de um comando e erros na declaração de variáveis. Com relação a maiores dificuldades, foram relatados os erros por: falta de *tokens*, como: '{','}', '(', ')', ';' com 49% na CFácil e 54% no questionário; seguido dos erros cometidos em variáveis (não declaração, erros na utilização de tipos, nomes de variáveis em desacordo com regras da linguagem C), com 32% na CFácil e 36% no questionário, e por último, a falta de aspas no início ou fim de sentenças, com 8% na CFácil e 18% no questionário. Um ponto positivo, avaliado na análise dos resultados foi que o erro menos cometido tanto no questionário quanto na ferramenta CFácil está relacionado à declaração de bibliotecas.

Souza et al. (2016) realizaram uma revisão na literatura sobre os problemas e as dificuldades no ensino e na aprendizagem de programação e concluíram que as principais dificuldades investigadas são: aprender os conceitos de programação; aplicação desses durante a construção de programas e a falta de motivação entre os alunos na realização da atividade de programação.

4.4 Sistema Juiz *Online*

Geralmente, a avaliação automática em atividades é realizada em questões de múltipla escolha, haja vista que questões discursivas possuem um grande número de respostas possíveis, sendo computacionalmente quase impossível prevê-las em sua totalidade. Em exercícios de programação de código, é comum haver mais de uma solução para um determinado problema. Portanto, torna-se difícil para o professor avaliar individualmente todas as soluções de um determinado exercício e pontuar de acordo com a quantidade de acertos ou erros. Normalmente, exercícios de programação são avaliados, levando-se em consideração duas medidas (JESUS, 2018): se o algoritmo retorna o resultado esperado; e se a solução está bem escrita.

Os sistemas de juízes *online*, muito utilizados em competições de programação (ALVES & JAQUES, 2014), são capazes de executar os códigos submetidos ao ambiente e

informar, automaticamente, se o programa funcionou corretamente. Todavia, a proposta desses sistemas não possui o recurso de fornecer *feedback* ao usuário, com a finalidade de indicar o tipo de erro e onde ele ocorreu. Conforme Alves & Jaques (2014), em competições de programação, não se pode apontar os problemas encontrados, porque o próprio competidor deverá descobrir os erros no código para submetê-lo novamente. Vislumbrando sua potencialidade na aprendizagem de programação, os juízes *online* foram estudados e adaptados para serem utilizados no contexto escolar, em função de disponibilizar um conjunto de questões para serem resolvidas em um ambiente que se propõe compilar, executar e testar os códigos, fundamentados em dados padronizados para julgar se estão corretos ou não de maneira automática.

Buscando criar um ambiente educacional de aprendizagem, com submissão automática do código-fonte e *feedback* imediato, algumas ferramentas foram desenvolvidas com essas características dentro de Instituições de Ensino e disponibilizadas para seu uso gratuitamente, como por exemplo: The Huxley, CodeBench e URI *Online Judge*.

De modo geral, o processo de avaliação automática desses juízes *online* ocorre da seguinte maneira (CHAVES et al., 2014, p. 240): "o juiz recebe o código-fonte, durante a execução do código, o juiz utiliza dados formatados como a entrada do programa, processa esses dados e realiza a comparação dos resultados obtidos com os resultados esperados, dando uma resposta apropriada com base nessas comparações (certo, errado, erro de compilação ou de execução)". Os dados formatados mencionado por Chaves et al., (2014), são, na maioria dos juízes *online*, um conjunto de casos de teste - não visíveis aos alunos - cadastrados pelo professor para cada questão disponibilizada na ferramenta.

4.5 Teste de Software

A habilidade de desenvolver sistemas para computadores está cada vez mais valiosa para o mercado do século XXI. Tudo o que é consumido no dia a dia é testado de alguma forma, antes de chegar ao mercado, com o software não é diferente. O fato de este não ser algo físico, ou seja, palpável, como é possível testá-lo? Por meio de técnicas e estratégias de teste de software. Braga (2016) assevera que o teste de software é todo procedimento realizado em um sistema que busca determinar se o programa atinge corretamente os requisitos para os quais foi projetado. Quando se realiza um teste de software, é natural aparecerem os problemas,

sendo necessário identificar uma solução para corrigi-los.

Conforme Maldonado et al. (2010), o teste de software envolve, basicamente, quatro etapas: planejamento de testes, projeto de casos de teste, execução e avaliação dos resultados dos testes. Essas atividades devem ser desenvolvidas ao longo do processo de desenvolvimento do sistema, e, em geral, possuem quatro fases de teste (PRESSMAN, 1995, p. 840): unidade, integração, validação e sistema. O **teste de unidade** (unitário) concentra esforços na menor unidade do projeto de software (módulo), buscando identificar erros de lógica e de implementação em cada módulo do software, garantindo que ele funcione adequadamente, como uma unidade. O **teste de integração** é uma atividade sistemática, aplicada durante a integração da estrutura do programa, tencionando descobrir erros associados às interfaces entre os módulos; o objetivo é construir a estrutura de programa que foi determinada pelo projeto. O **teste de validação** garante que o software atenda a todas as exigências funcionais, comportamentais e de desempenho. O **teste de sistema**, realizado após a integração do sistema, verifica se os componentes são compatíveis, se interagem corretamente e se transferem os dados corretamente e no momento certo, utilizando as suas interfaces (SOMMERVILLE, 2011, p. 153).

Em geral, os critérios de teste de software são estabelecidos com alicerce de três técnicas (MALDONADO, 2010): funcional, estrutural e baseada em erros. Essas se diferenciam pela origem da informação utilizada na avaliação e na construção dos conjuntos de casos de teste. Na **técnica funcional**, também conhecida como teste caixa-preta, os critérios e requisitos de teste são estabelecidos pela especificação do software e somente os dados de entrada fornecidos e as respostas produzidas como saída são avaliados, sem se preocupar com os detalhes de implementação. Na **técnica estrutural**, conhecida como teste caixa-branca, os aspectos de implementação são fundamentais na escolha dos casos de teste. O teste estrutural se fundamenta no conhecimento da estrutura interna da implementação do código. Em geral, a maioria dos critérios dessa técnica utiliza uma representação de programa conhecida como grafo de fluxo de controle ou grafo de programa. Na **técnica baseada em erros**, os critérios e requisitos de teste são oriundos do conhecimento sobre erros típicos cometidos no processo de desenvolvimento de software. A ênfase da técnica está nos erros que o programador ou projetista pode cometer durante o desenvolvimento e nas abordagens que podem ser usadas para detectar a sua ocorrência. Maldonado et al. (2010) citam dois

critérios típicos que se concentram em erros: Semeadura de Erros (*Error Seeding*) e Análise de Mutantes (*Mutation Analysis*).

Na área de teste de software, existem muitas terminologias utilizadas no contexto de Engenharia de Software e que provocam alguma confusão. Esse equívoco é apresentado quando se trata dos termos: **defeito** (*fault*), **engano** (*mistake*), **erro** (*error*) e **falha** (*failure*). Apesar de apresentarem a mesma ideia, esses não são sinônimos e são utilizados para caracterizar conceitos diferentes (KOSCHIANSKI & SOARES, 2006). O padrão IEEE Std nº 610.12-1990 (IEEE, 1990) diferencia esses termos como: **Defeito** (*fault*): passo, processo ou definição de dados incorreto, como uma instrução ou comando incorreto; **Engano** (*mistake*): ação humana que produz um resultado incorreto, como uma ação incorreta tomada pelo programador; **Erro** (*error*): diferença entre o valor obtido e o valor esperado, ou seja, qualquer estado intermediário incorreto ou resultado inesperado na execução do programa constitui um erro; e **Falha** (*failure*): produção de uma saída incorreta com relação à especificação.

Nesta pesquisa, os termos engano, defeito e erro são tratados como erro e o termo falha corresponde a um comportamento incorreto do programa.

Maldonado et al. (2010, p. 3) classificam os erros de maneira geral em dois tipos: **erros computacionais**: "o erro provoca uma computação incorreta mas o caminho executado (seqüências de comandos) é igual ao caminho esperado"; e **erros de domínio**: "o caminho efetivamente executado é diferente do caminho esperado, ou seja, um caminho errado é selecionado". Para Braga (2016), toda a falha é originada de um erro e o fato de produzi-los não está relacionado com a capacidade e as habilidades de quem programou o código, nem tão pouco estão relacionadas às tecnologias, recursos e métodos utilizados pelo programador, pois errar faz parte da natureza humana.

Para encontrar os erros no código, existem diversos tipos de testes com suas distintas maneiras de utilizá-los. Nenhum teste é aplicado ao acaso, todos possuem um propósito e devem ser realizados em momentos certos, como o teste unitário que é executado, geralmente, pelo programador após a conclusão da funcionalidade/módulo. A área de teste de software é muito extensa e, por isso, nesta seção serão abordados apenas três conceitos que são: teste funcional (caixa-preta), casos de teste e cobertura de linhas de código, pois são utilizados

neste estudo.

4.5.1 Teste Funcional (Caixa-preta)

O teste de software tem como objetivo principal a identificação de erros em códigos, componentes ou soluções tecnológicas e o seu sucesso está apoiado em um bom planejamento de todas as atividades de teste. Isso para que consiga atingir o seu propósito que é detectar o maior número de defeitos no programa em teste.

Koschianski & Soares (2006) e Pressman (1995) descrevem que o teste caixa-preta, também conhecido como teste funcional, está baseado nas especificações do sistema, que têm casos de teste derivados. Seu funcionamento é simples e busca avaliar o comportamento do código ou componente de software, sem considerar como ele é internamente. Um conjunto de dados de entrada é fornecido, o teste é executado e o resultado obtido é comparado a um resultado esperado e conhecido previamente (MALDONADO et al., 2010). Esse tipo de teste, nas palavras de Koschianski & Soares (2006) e Pressman (1995), serve para identificar problemas tais como: funções incorretas ou omitidas; e erros de interface, nas estruturas de dados, no acesso a banco de dados, de comportamento ou desempenho e de iniciação e término.

Dado que esse tipo de teste serve para avaliar o comportamento de um código sem que se conheça a sua construção, é importante criar casos de teste que sejam capazes de revelar os problemas, muitas vezes, ‘escondidos’ dentro do código. Ele será considerado bem-sucedido quando os resultados obtidos forem iguais aos esperados; caso contrário, o caso de teste identificou um erro. O protótipo, resultado desta pesquisa, utilizará a abordagem de teste de caixa-preta, que tem como propósito garantir o funcionamento do código, sem se ater em como ele foi implementado.

4.5.2 Casos de Teste

Para Koschianski & Soares (2006), um caso de teste é um conjunto condições diferentes para testar o código. Os testes devem abranger o maior número possível de situações, e escolher bem os valores mais representativos para essas situações é uma condição essencial para que contribuam com a verificação da qualidade do programa. Cada cenário de teste será

representado por um conjunto de casos de teste que estabelece o que será testado. Esses determinam quais informações serão empregadas durante os testes e quais serão os resultados esperados, constituindo o conjunto de testes necessários para validar boa parte de situações da solução (BARTIÉ, 2002).

Por exemplo, o programador precisa verificar a igualdade entre duas variáveis como na situação 1:

Situação 1:

```
if ( a == b )
```

Existem dois casos possíveis para testar os dois caminhos para o código da situação 1: um caso em que as variáveis são iguais e outro em que os valores de a e b são diferentes. Entretanto, ao escrever o código, o programador trocou os operadores relacionais escrevendo o código da situação 2:

Situação 2:

```
if ( a >= b )
```

O caso de teste `{a = 7; b = 9}` forneceria o mesmo resultado para as duas situações, neste caso FALSO, sendo incapaz de encontrar o defeito. Porém, ao planejar a bateria de teste, mais um cenário foi criado com os valores `{a = 10; b = 3}` que identificaria o defeito.

Para determinar os cenários de casos de teste, utilizando a abordagem caixa-preta, existem algumas técnicas que auxiliam na identificação da massa de dados que sejam eficientes e eficazes na bateria de teste. Conforme Molinari (2008, p.150-153) e Maldonado et al. (2010, p.5-6), as duas técnicas mais conhecidas são:

- **Particionamento de Equivalência de Classe**: é uma técnica de redução de casos de teste baseada na descoberta de um conjunto de situações (ou classes) que produzem ou geram o mesmo comportamento no código. Pressman (1995, p. 817) descreve que o teste de particionamento de equivalência é um método que divide o domínio de entrada de um programa em classes de dados, dos quais os casos de teste podem ser derivados. Geralmente, as partições são identificadas utilizando a especificação do programa ou a documentação do usuário, bem como pela experiência que prevê as classes de valor de entrada

prováveis de detectar erros.

- **Análise de Valor Limite:** é uma técnica de teste de software utilizada para exercitar os limites do domínio de entrada. Considerada um complemento do Particionamento de Equivalência (ou classes de equivalência), focalizando a seleção de casos de teste nas bordas da classe, ou seja, nos valores próximos às extremidades das classes. Em vez de selecionar qualquer elemento de uma classe de equivalência, a técnica de análise de valor limite leva a seleção de casos de teste nas extremidades da classe. (PRESSMAN, 1995).

Pegando por exemplo, um enunciado de um programa em que é solicitado a leitura de um número inteiro entre 1 a 10 e que números fora deste intervalo deverão gerar uma nova leitura, os casos de teste possíveis, utilizando a técnica de partição de equivalência de classe junto com a técnica de análise de valor limite, são as classes de entrada: Menor que 1, entre 1 e 10, Maior que 10, que poderiam ser: `{0, 1, 5, 10, 11}`, onde 1 e 10 são as fronteiras do programa.

Além das técnicas particionamento de equivalência de classe e análise de valor limite apresentadas, existem outras na literatura para obtenção de casos de teste utilizando a abordagem de teste de caixa-preta como: técnica de análise de domínio, técnica por tabela de decisão, técnica de transição de estado, técnica de grafo de causa-efeito, entre outras. A partir das combinações de técnicas, um conjunto de cenários possíveis podem ser elaborados antes de iniciar a bateria de teste.

É importante ressaltar que as técnicas de teste devem ser vistas como complementares. Maldonado et al. (2010) comentam que as técnicas e critérios de teste fornecem ao programador uma abordagem metodizada e fundamentada, além de auxiliarem na avaliação da qualidade e na adequação da atividade de teste. Mas, como saber se o código foi suficientemente testado? Pressman (1995, p.841) responde que não existe uma resposta definitiva para essa pergunta, porém, uma possível solução para este questionamento é que jamais se completa a atividade de teste, porque esta é transferida do testador para o cliente/usuário. Toda vez que o usuário executar o programa, este será testado com um novo conjunto de dados de teste. Transportando essa ideia para o contexto escolar, quando a execução dos testes realizados pelo aluno se esgotar, a carga será transferida para o professor realizar novos testes com seus conjuntos de dados.

4.5.3 Cobertura de linhas de código

A cobertura de linhas de código (chamada *code coverage)* é uma medida que apresenta a quantidade que um código-fonte que foi exercitado por um conjunto de testes, de acordo com critérios pré-definidos, que são medidos pelo número de linhas, as quais são acionadas sempre que um determinado conjunto de casos de teste é executado. O grande objetivo nesse tipo de cobertura de teste é alcançar 100% da execução do código-fonte, isto é, exercitar as linhas do algoritmo pelo menos uma vez durante a execução dos testes (BARTIÉ, 2002).

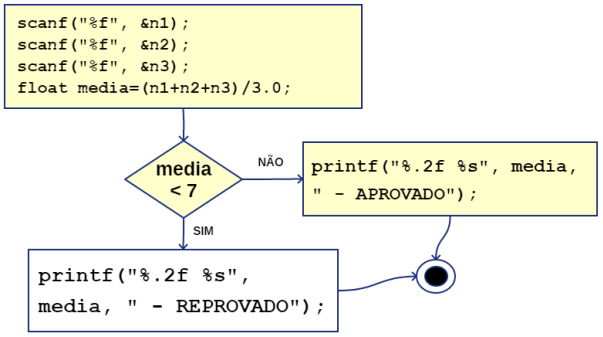
De acordo com Farias (2015), a cobertura de código serve para detectar partes da aplicação que não estão sendo testadas adequadamente ou não são usadas, oferecendo uma visão mais clara de quanto o código-fonte foi realmente executado durante os testes. A visualização desta cobertura ajudará o aluno a descobrir se existem linhas supérfluas, ou não cobertas por testes, que podem ser removidas ou que necessitem de mais casos de teste para exercitar os caminhos não cobertos. Esse tipo de métrica não tenciona avaliar a eficiência ou a qualidade da bateria de testes, mas identificar possíveis problemas no código-fonte avaliado.

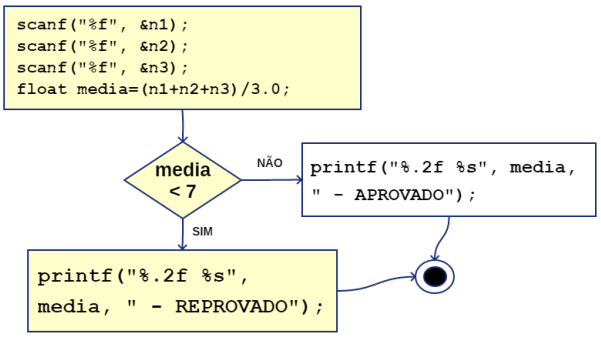
Inicialmente, a tarefa de encontrar os casos de teste certos parece ser uma tarefa simples, porque, com alguns casos de testes é possível cobrir 65% do código-fonte. Isso acontece porque, segundo Bartié (2002), um único caso de teste exercitará boa parte das linhas do código. Porém, para cobrir 100% deste, será necessária uma extensa massa de dados, exigindo um grande esforço para atingir o objetivo do teste.

No exemplo do Quadro 3, é possível visualizar um exemplo de cobertura de linhas de código em um algoritmo que calcula a média do aluno a partir da leitura de três notas e imprime na tela se o aluno está aprovado ou reprovado. Os caminhos percorridos no código-fonte, de acordo com os dados de entrada, estão destacados na cor verde.

Quadro 3 - Exercitando e Analisando a cobertura de linhas de código

Passo	Código-fonte	Grafo/Execução

1	`#include <stdio.h>`
	
	`int main(void) {`
	
	`float n1, n2, n3, media;`
	`scanf("%f", &n1);`
	`scanf("%f", &n2);`
	`scanf("%f", &n3);`
	`media=(n1+n2+n3)/3.0;`
	
	`if(media<7)`
	`printf("%.2f %s", media, "`
	`- REPROVADO");`
	`else`
	`printf("%.2f %s", media, "`
	`- APROVADO");`
	
	`return 0;`
	}
		
		Caso de Teste 1: { 5.0, 7.0 e 10.0 }

2	`#include <stdio.h>`
	
	`int main(void) {`
	
	`float n1, n2, n3, media;`
	`scanf("%f", &n1);`
	`scanf("%f", &n2);`
	`scanf("%f", &n3);`
	`media=(n1+n2+n3)/3.0;`
	
	`if(media<7)`
	`printf("%.2f %s", media, "`
	`- REPROVADO");`
	`else`
	`printf("%.2f %s", media, "`
	`- APROVADO");`
	
	`return 0;`
	}
		
		Caso de Teste 2: { 5.0, 5.0 e 5.0 }

Fonte: Autora

Este tipo de visualização será apresentado ao aluno após uma execução do código-fonte no protótipo. A próxima seção apresenta os trabalhos relacionados expondo as características sobre juízes *online*.

5 TRABALHOS RELACIONADOS

Existe uma quantidade substancial de pesquisas no ensino de programação de computadores com a finalidade tornar as linguagens e os ambientes de programação mais compreensíveis, além de minimizar as dificuldades encontradas pelos estudantes iniciantes nos cursos de Computação. No Brasil, o interesse se manifesta, com inúmeros artigos apresentados nos principais eventos da comunidade brasileira de ensino em computação: o Simpósio Brasileiro de Informática na Educação (SBIE) e o Congresso Brasileiro de Informática na Educação (CBIE)/Workshop de Informática na Escola (WIE).

Para realizar as buscas dos trabalhos correlatos, foram utilizadas as bases de pesquisa: Catálogo da Capes[2], Portal brasileiro de publicações científicas em acesso aberto (OASISBR)[3], Biblioteca Digital Brasileira de Teses e Dissertações (BDTD)[4], os repositórios do: Simpósio Brasileiro de Informática na Educação[5] (SBIE), Congresso Brasileiro de Informática na Educação[6] (CBIE), Anais do Workshop de Informática na Escola[7] (WIE) e as coleções de artigos do Workshop sobre Educação em Computação (WEI) disponíveis na internet. Para esta pesquisa, foram considerados trabalhos nos idiomas português e inglês, publicados no período de 2010 a 2019.

Quadro 4 - Strings de busca para a seleção dos artigos

Português	("juiz on*" OR "avaliação automática" OR "feedback") AND ("introdução à programação" OR "ensino de programação" OR "linguagem de programação")
Inglês	("online judge" OR "automated assessment" OR "feedback") AND ("introduction to programming" OR "programming teaching" OR "programming language")

Fonte: Autora

Os critérios de inclusão e exclusão utilizados para a seleção dos artigos são apresentados no Quadro 5. Uma vez que esta pesquisa tem como base elaborar uma estratégia de *feedback* e aplicá-lo em ambientes que disponibilizam ao usuário um conjunto de exercícios do tipo questão 'problema/solução com implementação algorítmica', ou seja, discursivas para resolução e fixação dos conceitos introdutórios em programação, buscou-se

2 <http://catalogodeteses.capes.gov.br/catalogo-teses/#!/>
3 <http://oasisbr.ibict.br/vufind/>
4 <http://bdtd.ibict.br/vufind/>
5 <http://www.br-ie.org/pub/index.php/sbie/index>
6 <http://www.br-ie.org/pub/index.php/wcbie/index>
7 <http://www.br-ie.org/pub/index.php/wie/issue/view/179>

pesquisar trabalhos que abordam, com maior detalhamento, o funcionamento dos juízes *online* pelas características de compilar, executar e testar os códigos automaticamente. Sabe-se que os ambientes de juízes *online* na sua concepção original não permitem a exibição de *feedbacks* aos competidores em torneios de programação, pois faz parte das regras do evento (ALVES & JAQUES, 2014). Todavia, esses tipos de ambientes estão sendo estudados e propostos para serem utilizados no contexto escolar, sendo adaptados para melhor atender a aprendizagem em programação de computadores.

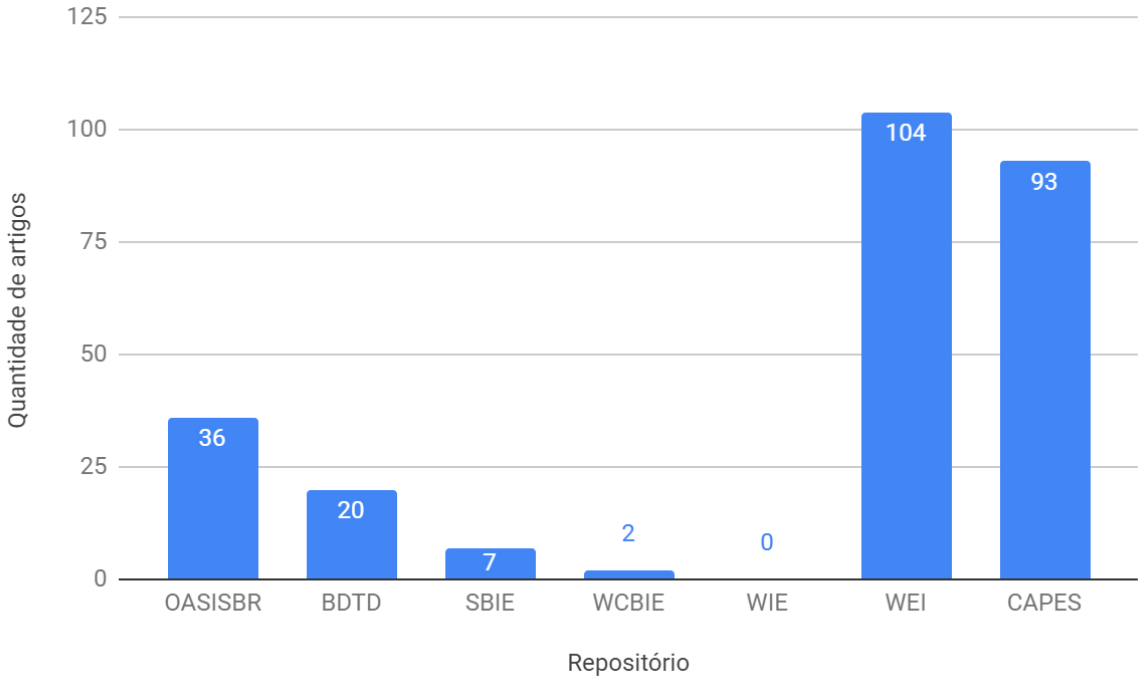
Quadro 5 - Critérios de Inclusão e Exclusão dos trabalhos

Critérios	**ID**	**Descrição**
Exclusão	E1	Texto completo não estar disponível gratuitamente para leitura
	E2	Sem identificação do ano de publicação
	E3	Escrito em idioma diferente do Português/Inglês
	E4	Trabalho não apresentar nenhum dos temas: juiz *online*, *feedback* automático/personalizado, ambiente para o ensino de programação de computadores ou aprendizagem por *feedback*
	E5	Não ser um capítulo de livro, ou artigo de periódico, ou artigo de conferência, ou dissertação de mestrado, ou tese de doutorado
	E6	Não direcionado à aprendizagem da Linguagem de Programação C
	E7	Ambiente Desktop
	E8	Antes de 2010
Inclusão	I1	Apresentar um juiz *online*, abordando as características da ferramenta.
	I2	Apresentar os resultados da pesquisa com o uso do juiz *online* no ensino da programação de computadores
	I3	Apresentar tipos de *feedback* utilizados no juiz *online*
	I4	Tratar sobre ensino-aprendizagem em programação de computadores com a linguagem de programação C em um juiz *online*

Fonte: Autora

Utilizando a *string* de pesquisa mostrada no Quadro 4, chega-se ao gráfico apresentado na Figura 3, que exibe a quantidade de artigos encontrados com a busca realizada em cada repositório de artigos.

Figura 3 - Quantidade de artigos por repositório filtrados pelo ano de publicação



Fonte: Autora

Sobre o total de 262 trabalhos encontrados, realizou-se uma seleção preliminar dos trabalhos mediante a leitura do resumo de cada um. Dos artigos analisados, 252 foram excluídos por atenderem os critérios de exclusão: E4, E5, E6, E7, E8 e os dez materiais selecionados, exibidos no Quadro 6, passaram pela fase de análise mais detalhada. A mesma pesquisa em inglês foi realizada nas bases de pesquisa: IEEEXplorer Digital Library, ACM Digital Library, Springer, Science Direct e Scopus.

Quadro 6 - Trabalhos incluídos conforme critérios de inclusão

Critério	Referências
I1	(PAES et al., 2013)[8]; (SANTOS & RIBEIRO, 2011)[9]; (SELIVON et al., 2015)[10]; (DAGOSTINI et al., 2018)[11];
I2	(JESUS, 2018)[12]; (RIBEIRO et al., 2018)[13]; (DAGOSTINI et al., 2017)[14]; (CARVALHO et al.,

8 PAES, Rodrigo de Barros; MALAQUIAS, Romero; GUIMARÃES, Márcio; ALMEIDA, Hyggo. (2013). **Ferramenta para a Avaliação de Aprendizado de Alunos em Programação de Computadores.**

9 SANTOS, Joanna C. S.; RIBEIRO, Admilson R. L. (2011). **JOnline: proposta preliminar de um juiz online didático para o ensino de programação**.

10 SELIVON, Michele; BEZ, Jean Luca; TONIN, Neilor A. (2015). **URI Online Judge Academic: Integração e Consolidação da Ferramenta no Processo de Ensino/Aprendizagem**.

11 DAGOSTINI, Jessica; LIMA, Marcos Vinicius de Moura; BEZ, Jean Luca; TONIN, Neilor. (2018). **URI Online Judge Blocks: Construindo Soluções em uma Plataforma Online de Programação**.

12 JESUS, Galileu Santos de. (2018). **Uma abordagem para auxiliar a correção de erros de programadores iniciantes**.

	2016)[15]; (JESUS et al., 2018)[16]; (SELIVON et al., 2015)
I3	(SANTOS & RIBEIRO, 2011); (RIBEIRO et al., 2018);
I4	(CARVALHO et al., 2016); (SELIVON et al., 2018)

Fonte: Autora

Dentre tantas ferramentas de apoio à prática da programação com finalidades de submissão, execução e avaliação de exercícios de maneira automatizada, selecionou-se cinco trabalhos em função de sua relevância no meio acadêmico: (CAMPOS & FERREIRA, 2004); (SANTOS & RIBEIRO, 2011); (SELIVON et al., 2015); (PAES et al., 2013) e (CARVALHO et al., 2016). Os trabalhos citados passaram por um estudo mais aprofundado das características dos ambientes propostos e sua contribuição como ferramenta auxiliadora da aprendizagem.

A linguagem de programação C é oficialmente utilizada na disciplina SSI042-Linguagem de Programação I do curso STSI - alvo deste trabalho -, sendo de suma importância a observação das características dos ambientes que cobrem o desenvolvimento de exercícios com *feedback* neste tipo de linguagem. Optou-se por procurar ferramentas desenvolvidas com aporte bibliográfico voltado ao ensino-aprendizagem de programação de computadores, apesar de existirem na internet outros ambientes com características semelhantes e com acesso gratuito, como por exemplo HackerRank[17] e Coderbyte[18], os quais são voltados para competições e concurso para seleção de novos programadores a vagas de emprego. As ferramentas propostas nos trabalhos escolhidos são: BOCA (*Brazilian Online Contest Administrator*), JOnline, URI Online Judge/Academic, The Huxley e CodeBench por oferecerem um ambiente com exercícios para programar na linguagem C e são utilizados em ambiente educacionais.

13 RIBEIRO, Ralph Breno; FERNANDES, David; CARVALHO, Leandro Silva Galvão de; OLIVEIRA, Elaine. (2018). **Gamificação de um Sistema de Juiz Online para Motivar Alunos em Disciplina de Programação Introdutória**.

14 DAGOSTINI, Jessica; LIMA, Marcos Vinicius de Moura; BUCIOR, Lucas; TONIN, Neilor; BEZ, Jean Luca. (2017). **Incentivando a Aprendizagem de Algoritmos Através do URI Online Judge Forum 2.0**.

15 CARVALHO, Leandro S. G.; OLIVEIRA, David B. F.; GADELHA, Bruno F. (2016). **Juiz online como ferramenta de apoio a uma metodologia de ensino híbrido em programação**.

16 JESUS, Galileu Santos de; SANTOS, Kleber; CONCEIÇÃO, Jaine; RIBEIRO, Elisalvo; COSTA NETO, Alberto. (2018). **Avaliação de uma abordagem para auxiliar a correção de erros de aprendizes de programação**.

17 <https://www.hackerrank.com/>

18 <https://coderbyte.com/>

5.1 BOCA *Online Contest Administrator* (2004)

Campos & Ferreira (2004) apresentaram a ferramenta intitulada **BOCA**, que, atualmente, é a plataforma usada nas competições de programação promovidas pela Sociedade Brasileira de Computação (SBC). Ela foi desenvolvida na linguagem PHP utilizando o banco de dados relacional PostgreSQL para armazenar os dados e controlar a concorrência. Com uma interface totalmente web e *open source*[19] sob licença GNU *Public License* 3, o **BOCA** é um juiz *online* desenvolvido com propósito principal de suportar competições de programação, por intermédio da submissão e avaliação automática de códigos-fonte desenvolvidos pelos usuários nas linguagens: C, C++ ou Java.

Quadro 7 - Tipos de retorno do Juiz *Online* BOCA

Resposta	**Descrição**
Yes	O juiz *online* aprovou o código-fonte.
NO: Incorrect Output	Também conhecido como *Wrong Answer*. Indica que o código-fonte respondeu incorretamente a algum(ns) dos testes aplicados.
NO: Time-limit Exceeded	A execução do código-fonte excedeu o tempo permitido pelo juiz *Online.*
NO: Runtime Error	Durante o teste, ocorreu um erro de execução (causado pelo código-fonte), que pode ter sido provocado quando houve acesso a posições irregulares de memória ou estouro dos limites da máquina, por exemplo.
NO: Compilation Error	O código-fonte possui erros de sintaxe.
NO: Output Format Error	Conhecido como *Presentation Error*, indica que a saída do código-fonte não segue a especificação exigida para a questão, apesar do "resultado" estar correto.

Fonte: SILVA & CAMPOS, 2006

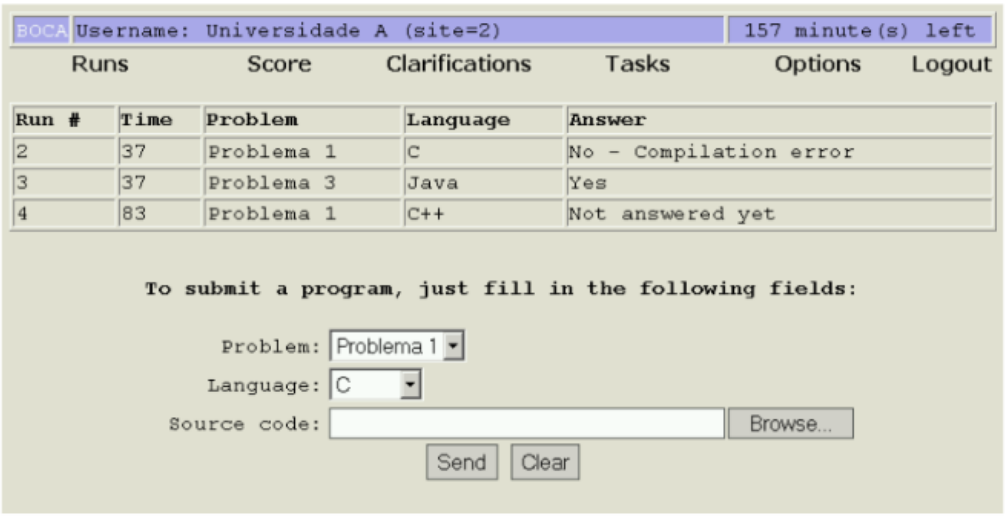
Por ser um juiz *online* com características somente competitiva, o sistema apenas sinaliza se o programa está funcionando corretamente ou não, conforme Quadro 7, não apontando onde o erro se encontra nem sua causa. Além disso, permite ao usuário realizar perguntas sobre suas dúvidas de forma individual ou coletiva e encaminhar ao juiz cadastrado para a competição. O ambiente possui um recurso similar a um chat/fórum de discussões, que funciona de maneira dirigida e controlada. O **BOCA** está dividido em cinco partes, de acordo com a característica de cada usuário que utilizará o sistema: time, juiz, administrador, staff e placar. Cada uma dessas partes possui uma interface própria, de acordo com as necessidades e responsabilidades de cada perfil do usuário no sistema.

Perfil Time: é interface simples onde os usuários conseguem enviar os programas para correção (*Runs*), alterar informações pessoais, submeter dúvidas aos juízes

19 <https://www.ime.usp.br/~cassio/boca/>

(*Clarifications*), chamar assistência dos *staffs* e solicitar impressão dos códigos-fonte (*Tasks*) e verificar o placar da competição (*Score*). No momento de submeter o arquivo-fonte (*Runs*), para correção dos juízes, o time escolhe a linguagem que programou o desafio e para qual problema ele é destinado. Na mesma janela de submissões, o time poderá verificar o andamento das correções de submissões enviadas por ele, como mostra na Figura 4.

Figura 4 - Interface do time



Fonte: CAMPOS & FERREIRA, 2004

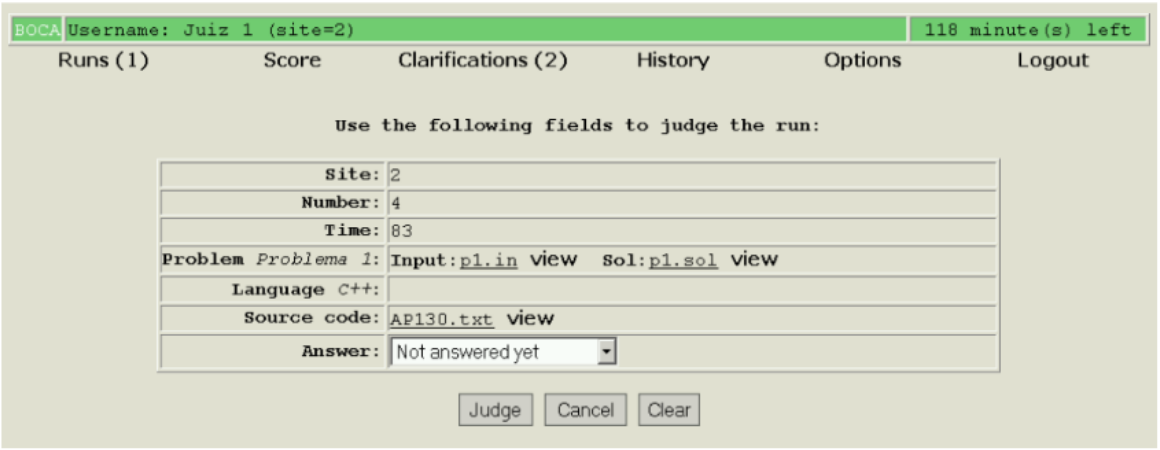
Perfil Administrador: por essa interface é que são configuradas as informações sobre a competição, as linguagens de programação que serão permitidas e as questões da prova. É nela que são cadastrados times, juízes, placares, staff e administradores; é onde a prova é definida, iniciada e interrompida; e por ela se pode ter acesso aos registros de tudo que está ocorrendo no sistema.

Perfil Staff: são as pessoas responsáveis em dar o suporte aos usuários na utilização do sistema, tais como: controlar o acesso e a movimentação dos times no local da prova; imprimir arquivos solicitados pelos times e entregar a impressão; disponibilizar balões para cada time que acerta um problema (esses são utilizados como 'placar visual' durante as competições); e dar assistência a times com problemas de hardware e/ou software.

Perfil Juiz: é a interface que permite aos juízes responderem perguntas e dúvidas de forma individual ou coletiva aos usuários, além de possibilitar que um juiz crie perguntas globais para serem respondidas a todos. O juiz tem a chance de analisar e corrigir os programas enviados pelos times, definindo no sistema qual a resposta que o time receberá

para sua tentativa. As respostas possíveis podem ser configuradas para cada competição, como por exemplo: certo, saída errada, saída mal formatada, erro de compilação, erro em tempo de execução, entre outras (Quadro 7). O juiz pode acompanhar o placar da competição em seu computador ou alterar seus dados cadastrais, como nome, senha, etc. No item *Runs*, o juiz acompanha as submissões encaminhadas pelos times e que ainda não foram completamente processadas. Aparecem nesta janela os códigos que estão sendo julgados por outros juízes, pelo próprio, e ainda aqueles que não começaram a ser julgados. Na opção *Clarifications,* o juiz poderá incluir uma dúvida para ser respondida por ele mesmo ou por outro juiz, a fim de publicar a todos os participantes algum detalhe das questões e/ou enunciados que não ficaram claros após a produção do conjunto de problemas disponível para a competição. Os juízes têm à disposição a opção *History*, onde podem acompanhar todas as respostas de perguntas passadas e todos os códigos-fonte julgados por eles.

Figura 5 - Interface do juiz



Fonte: CAMPOS & FERREIRA, 2004

Placar: local onde são apresentados os resultados, em tempo real, da competição às pessoas locais que estão diretamente envolvidas com o certame e por pessoas localizadas fora do ambiente da prova.

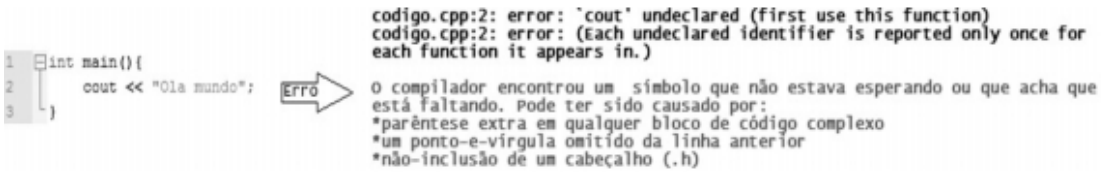
5.2 JOnline (2011)

O trabalho de Santos & Ribeiro (2011) descreve JOnline, um juiz *online* que permite a submissão de códigos nas linguagens de programação em C e C++ e que possui como características a apresentação de dicas na língua portuguesa para corrigir erros de compilação do código-fonte enviado, a revelação de casos de testes que verificam a corretude dos

resultados gerados, a organização de problemas por assunto e o grau de dificuldade e a votação do nível de dificuldade dos problemas pelos usuários, além da programação colaborativa.

A ajuda aos usuários na correção dos erros de execução e de lógica é realizada por meio da exibição dos casos de testes (entradas) que podem gerar resultados incompatíveis com a resposta esperada ou com as falhas na execução normal do programa. A funcionalidade que analisa o resultado dos casos de teste, objetiva facilitar que o usuário encontre seus erros dentro do código-fonte, mediante um *feedback* dos possíveis problemas. Os erros podem ser classificados em três tipos: compilação, lógica e execução. Os erros gerados são processados pelo JOnline e as sugestões de como solucioná-los são apresentados aos usuários.

Figura 6 - Apresentação de dicas a partir de erro no JOnline



Fonte: SANTOS & RIBEIRO, 2011

O ambiente possibilita a programação colaborativa, permitindo compartilhar, entre um grupo de usuários, o código-fonte, facilitando a edição, compilação, o teste e a execução deste código por todos os integrantes. A comunicação entre os participantes é simplificada com a utilização de uma ferramenta *chat* dentro do ambiente, para que esses usuários possam discutir sobre a resolução do exercício.

O JOnline considera dois tipos de usuários: professores e alunos. Essa distinção caracteriza as funcionalidades que cada tipo de usuário pode acessar no ambiente. Sendo assim, ao professor é disponibilizada a obtenção de relatórios de submissão para um determinado problema ou usuário, a definição das entradas de teste e saídas esperadas para os problemas, além do gerenciamento de problemas e de alunos, entre outras. Ao aluno é fornecida a possibilidade de submissão de código-fonte e resolução de problemas aplicando a abordagem colaborativa e a visualização de resultados.

Para atender os requisitos do ambiente, o desenvolvimento foi dividido em três camadas (SANTOS & RIBEIRO, 2011): gerenciamento de conteúdo e usuários, processamento da solicitação e análise e obtenção de dados.

- **camada de gerenciamento de conteúdo e usuários:** é responsável pelas atividades de gerenciamento do ambiente. Ela abrange o banco de dados do sistema e contém uma subcamada que pretende apresentar o conteúdo de maneira mais fácil ao usuário;
- **camada de processamento da solicitação:** é a que realiza a comunicação entre a camada de gerenciamento de conteúdo e usuários com a camada de análise e obtenção de dados; e
- **camada de análise e obtenção de dados:** é a que contém a lógica de processamento do juiz *online*. Nela são realizadas as operações de compilação/ execução de programas, obtenção de erros, entre outras.

A ferramenta possibilita aos usuários acessar problemas, obter documentos PDF contendo o enunciado dos problemas nela cadastrados, submeter código-fonte, acessar tutoriais de uso do ambiente e participar de fóruns.

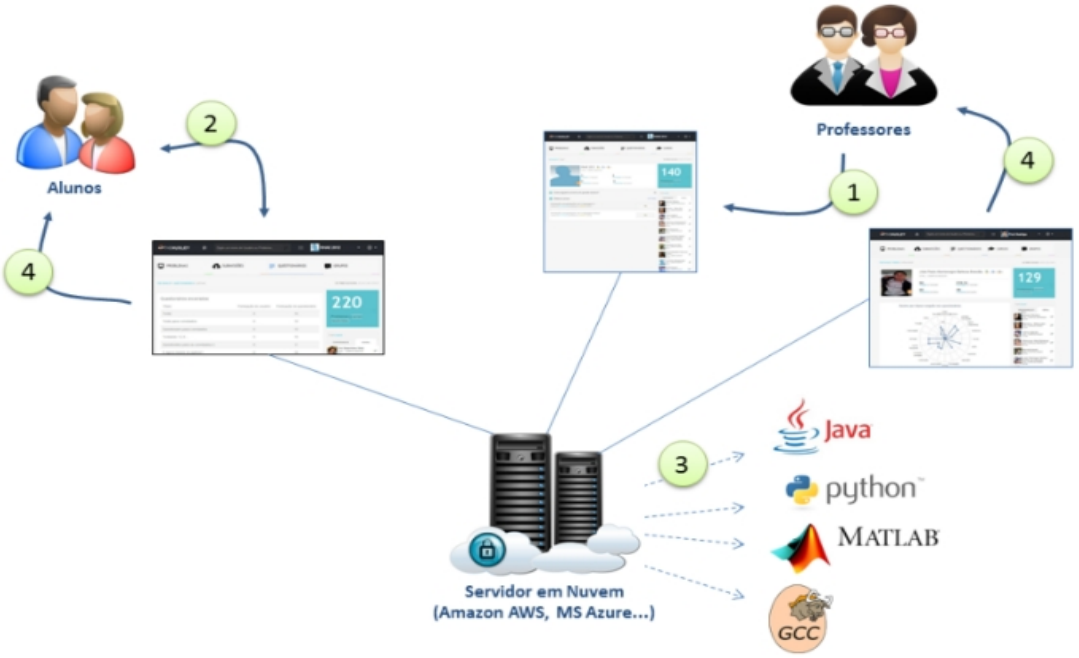
5.3 The Huxley (2013)

Desenvolvido na Universidade Federal de Alagoas (UFAL), o juiz *online* The Huxley foi proposto no trabalho de Paes et al. (2013), os quais permite aos alunos submeterem código nas linguagens de programação C, C++, Python ou Pascal como respostas a problemas de programação disponíveis em um banco de questões. Com a análise sintática do código submetido e de testes de aceitação, o aluno é imediatamente informado se seu código é uma solução válida para o problema que tentou solucionar. No The Huxley, o professor tem uma visão do desempenho de seus alunos, incluindo a quantidade de problemas resolvidos, porcentagem de acertos/erros, tipos de problemas com mais erros, detecção de plágio e erros específicos de cada aluno. Dentro do ambiente, professores e alunos possuem diferentes panoramas, todos focados no conceito de painel, que permite uma visualização geral do status de exercícios, avaliações e conteúdo na página inicial do sistema.

O processo de funcionamento do juiz *online*, Figura 7, inicia com a criação de questionários e avaliações, feitas pelo professor na interface do The Huxley. Esse poderá criar novos exercícios, caso necessite, juntamente com os testes de aceitação do resultado (1). Após a disponibilização das questões aos alunos, esses acessam os exercícios definidos, desenvolvem os problemas computacionalmente, com os códigos-fonte e os submetem via

interface da ferramenta (2). Para cada código desenvolvido, The Huxley compila e testa o programa, empregando a linguagem de programação selecionada para resolução do problema, exibindo e armazenando o resultado para consulta posterior tanto pelo aluno, quanto pelo professor (3). O resultado é visualizado pelos professores em forma de planilhas e gráficos, que podem atribuir nota diretamente com base na avaliação do sistema. Os alunos também recebem o resultado, para que possam submeter novamente a solução, caso haja algum problema de compilação ou nos testes de aceitação (4).

Figura 7 - Processo de funcionamento do juiz online The Huxley



Fonte: PAES et al., 2013

Após a autenticação, o ambiente disponibiliza aos usuários três tipos de desafios: resolução de problemas através de **algoritmos**, **múltipla escolha** e **complete o código**.

Na opção **Algoritmos**, os desafios são listados por nível de dificuldade (1 - iniciante, 2 - fácil, 3 - médio, 4 - avançado e 5 - expert), juntamente com o título do problema e o tópico que ele trata, como: decisão, repetição, etc., e o status da questão, informando se esta foi resolvida, não resolvida ou se encontra com erro/problema. Ao entrar no exercício, são apresentadas ao usuário cinco abas: a descrição do problema, enviar resposta (o usuário precisará resolver o problema por meio de um algoritmo utilizando uma linguagem selecionada), submissões (lista das que foram realizadas pelo usuário para a questão, com os dados de: data/hora, resultado da correção, linguagem de programação, o tempo de execução

e a opção de visualizar o código avaliado), oráculo (dica), estatísticas (informações referentes à quantidade de submissões enviadas e de questões corretas, o número de tentativas e as questões resolvidas globalmente). Na aba Enviar Resposta, ele poderá escolher a linguagem de programação de sua preferência dentre as opções: C(GCC), CPP(G++), Python2, Python3 ou Pascal.

A resolução da questão sempre será realizada pelo editor disponível na página, caso o aluno tenha a solução em um arquivo separado, este deverá ser enviado ao servidor do The Huxley pela opção 'Enviar arquivo de código', o sistema o lerá e carregará o código na área do editor. Optando por usar o editor, dependendo da linguagem de programação selecionada, um esqueleto básico é apresentado ao usuário, a fim de iniciar a resolução do exercício, incluindo algumas bibliotecas-padrão necessárias para tratamento de entrada/saída de dados, strings, cálculos matemáticos e conversões. Sendo uma solução que exija entrada de dados (uso do *scanf*), o usuário deverá selecionar a opção 'Entrada Customizada', para informar os valores, caso contrário um erro será exibido na área de saída. O botão 'Executar Código' aciona o compilador do juiz *online*, passando o código implementado e as entradas informadas, compilando e executando o código e exibindo o resultado na área de saída. Ocorrendo um erro de sintaxe durante a compilação, a mensagem do compilador em inglês é exibida na área de saída. O botão 'Enviar Resposta' submete ao juiz apenas o código desenvolvido, que é validado com os dados esperados para a questão. Não estando de acordo, uma mensagem de *feedback* é exibida ao usuário juntamente com a quantidade de tentativas realizadas. Exemplo: Submissão #2: O programa não produziu a saída esperada.

Na opção de **Múltipla Escolha**, o The Huxley apresenta uma questão com quatro alternativas, sendo apenas uma a correta. Nessa tela, o usuário poderá tirar dúvidas antes de submeter a questão para correção. Na opção **Complete o Código,** os desafios são listados por nível de dificuldade (1 - iniciante, 2 - fácil, 3 - médio, 4 - avançado e 5 - expert), juntamente com o título do problema, o tópico que o problema trata (decisão, repetição, etc.) e o status da questão informando: resolvida, não resolvida ou problema não resolvido, semelhante à opção **Algoritmos**. Ao entrar na questão, o sujeito recebe uma descrição do problema e um código base que precisa ser complementado com a lógica faltante.

Um outro recurso interessante é a visualização de um ranking, chamado de *topCoder*, o qual lista os alunos com maior quantidade de exercícios resolvidos e as medalhas para

aqueles que conquistaram o maior índice Huxley. Os estudantes ganham pontos quando eles enviam submissões corretas, ou seja, quando um estudante envia um código-fonte que produz os resultados esperados para um problema, ele ganha os pontos correspondentes aos níveis de dificuldade do problema resolvido. Assim, se um estudante resolveu um problema de nível 7, ele ganha sete pontos. Dessa forma, quanto mais difícil é o problema, mais pontos ele ganha. A soma de todos os pontos obtidos compõe o *topCoder*. Os dez maiores *topCoders* de cada turma aparecem com destaque na página inicial da turma, em contrapartida, as dez maiores pontuações de todas as turmas aparecem na página inicial da ferramenta, visível para todos os usuários. Além disso, o The Huxley permite a realização de provas dentro do ambiente como uma Tarefa. Neste caso, o professor define uma avaliação que ficará disponível aos alunos durante o tempo preestabelecido para a realização da prova. Após a expiração do tempo, a avaliação fica indisponível no The Huxley, e o professor consegue exportar uma planilha com as notas depois da correção automática ter sido realizada pelo juiz *online*.

Nos trabalhos de Jesus et al. (2018) e Jesus (2018), foi proposto um aprimorando do juiz *online* para melhorar a capacidade de produzir mensagens de *feedback* que facilitem sua compreensão pelos estudantes iniciantes em programação, orientando-os sobre os erros de sintaxe apresentados ao realizar uma submissão ao juiz *online* na linguagem Python. A integração com o The Huxley foi executada pela interface gráfica, onde, ao clicar no *link* 'ver saída amigável' é feita a requisição a um *webservice* passando os parâmetros necessários e obtendo uma *string* como retorno, contendo o resultado do processamento da requisição.

5.4 CodeBench (2016)

Implementado pela Universidade Federal do Amazonas (UFAM), o CodeBench é um juiz *online* com a função de automatizar a correção dos exercícios de programação e foi abordado no trabalho de Carvalho et al. (2016). No ambiente, os professores disponibilizam exercícios de programação aos alunos nas linguagens C, C++, Python 3, Java, Haskell, Lua 5.3 e Prolog, que por sua vez os codificam e os submetem pela interface do sistema. Assim que o código é submetido, o CodeBench informa instantaneamente se a solução está de acordo ou não. Para julgar a corretude do código submetido pelo aluno, o CodeBench segue dois passos principais (CARVALHO et al., 2016):

1. **Análise sintática do código:** o juiz online verifica se o código submetido

possui algum erro sintático, de acordo com a gramática da linguagem de programação proposta pela disciplina. Caso haja algum problema sintático com o código, o aluno é imediatamente informado sobre a natureza do problema, bem como a linha onde o problema se encontra; e

2. **Análise lógica do código:** o juiz *online* verifica se o código desenvolvido pelo aluno soluciona corretamente o problema proposto pelo professor com a aplicação de casos de testes cadastrados pelo docente ao criar a questão. Cada caso de teste possui dois valores: um valor de entrada, passado como entrada para o programa do aluno; e um valor de saída, que é a saída correta para o valor de entrada informado.

Caso o sistema identifique que o código-solução está incorreto, seja por erros de sintaxe ou de lógica, o aluno poderá rever seu código e submetê-lo novamente, tantas vezes quantas forem necessárias, até o prazo definido pelo professor. Para Carvalho et al. (2016, p. 145), a "política de ressubmissão de códigos incentiva o aluno a identificar e solucionar os erros por si próprio, ao contrário da abordagem tradicional, em que o aluno tem que esperar pela correção do professor, sem oportunidade de identificar os erros e corrigir seus códigos. Tal abordagem mimetiza o desenvolvimento de código na vida profissional, em que os programadores precisam identificar os erros de seus códigos e solucioná-los".

Em fevereiro de 2019, o sistema tinha um banco de questões com mais de 2.800 exercícios de programação, que podem ser escolhidos pelos professores durante a criação de trabalhos em sala de aula, e conta com três instituições parceiras cadastradas. Sempre que um novo exercício é cadastrado no sistema, ele passa a integrar o banco de questões, ficando disponível para todos os professores cadastrados no CodeBench. Além disso, o ambiente possui recurso de detecção de plágio, caixa de e-mails que pode ser usada para troca de mensagens entre os usuários do sistema e materiais didáticos para auxiliar os alunos em seu processo de aprendizagem.

O ambiente dispõe de uma IDE que os alunos podem utilizar para resolver os exercícios. Caso prefiram outra ferramenta, o aluno poderá submeter, a qualquer momento, os códigos pela interface do CodeBench. Adicionalmente, este dispõe de uma interface com a qual o professor pode conferir os códigos submetidos por seus alunos, verificando quais casos de teste passaram ou não. O registro para o uso da ferramenta é por autoinscrição, isto é,

qualquer pessoa poderá se inscrever e se matricular em uma turma, haja vista que o CodeBench não interage com o sistema de controle acadêmico da Universidade. No entanto, o registro de professores e tutores só poderá realizado mediante autorização do administrador do sistema.

Ribeiro (2018) apresenta a versão gamificada do CodeBench, a qual utiliza os elementos de gamificação: desafios, competição, *badges*, *feedback*, recompensas, estado de vitória, combate, *ranking*, pontos e bens virtuais. Tudo começa com uma história em um reinado distante e, antes de iniciar a resolução dos exercícios propostos na disciplina, o aluno deverá escolher um avatar e as armas que utilizará durante o desafio. Conforme explica o autor, o sistema de pontuação adotado pelo CodeBench segue o seguinte método: no momento em que o sistema verifica que o código enviado está correto, ele sorteia uma carta de baralho para o estudante. Essa carta representa alguns elementos chamados de 'sorte' para o jogo e que podem ser configuráveis pelo professor. Tais elementos podem ser força ou posição no mapa, por exemplo. Caso o estudante consiga obter nota máxima na lista de exercícios, o seu avatar ganha uma medalha (*badge*), representada por uma arma mais forte.

5.5 URI Online Judge (2011) e o módulo Academic (2015)

O URI *Online Judge*, apresentado no trabalho de Selivon et al. (2015), é um projeto que vem sendo desenvolvido na Universidade Regional Integrada (URI) - Campus de Erechim, desde 2011. A ferramenta, além de características como correção automática em tempo real, fórum e aceitação de soluções em diferentes linguagens de programação (C, C++, Java, Python), dispõe de um *toolkit* integrado para testar entradas e saídas adicionais, facilitando a compreensão do problema a ser resolvido; problemas separados por categorias e por níveis de dificuldade; aplicação de conceitos de gamificação pelo sistema de recompensa por *badges* e *ranks*; visualização das linhas do código-fonte com erro, quando o usuário recebe como resposta um erro de compilação (*Compilation Error*); visualização de detalhes sobre erros encontrados durante a execução da solução (*Runtime Error*); e indicação do percentual de casos de teste que falharam, quando o usuário submete uma solução incorreta para julgamento.

No ano de 2013, foi adicionado ao URI *Online Judge* o módulo Academic que tem acesso diferenciado para professores, possibilitando o controle das atividades propostas aos

estudantes tanto em horário de aula quanto em atividades extraclasse. O novo ambiente integrado ao portal, passou a permitir o controle de listas de exercícios e lista de estudantes com o acompanhamento em tempo real pelo professor, bem como a determinação do intervalo válido de datas, para resolução de uma lista de exercício. Desta forma, resoluções feitas antes da data de início ou após a data final da lista não são contabilizadas e nem exibidas ao professor. É possível que ele confira, por exemplo, o histórico de submissões de cada estudante, para cada problema.

O URI *Online Judge* e o módulo Academic estão integrados e possuem duas interfaces distintas: uma para estudantes e outra para professores. Para utilizar o módulo como professor, o usuário deve enviar uma solicitação à equipe URI Online Judge que autoriza o cadastro de professores vinculados a algumas instituições de ensino. Ao logar no Academic, o professor encontra um ambiente preparado para facilitar e organizar seu trabalho na utilização do portal como ferramenta de ensino. Nele, o docente registra suas disciplinas e os exercícios que comporão listas a serem liberadas para resolução dos estudantes. As disciplinas criadas podem ser visualizadas, editadas ou excluídas. É importante destacar que, durante a criação dessas listas, o professor tem a possibilidade de estabelecer a linguagem que deverá ser utilizada para a resolução do exercício; escrever instruções que julgar importantes para a orientação dos estudantes e delimitar o prazo de conclusão das atividades. Para compor cada lista, o docente tem à sua disposição um conjunto de problemas disponíveis no acervo do URI *Online Judge*. Após a conclusão da criação das listas, o professor convida os estudantes de sua turma, pelo sistema, a fim de iniciar seus estudos. Assim que o convite for aceito, o professor pode visualizar o avanço de cada estudante dentro da ferramenta.

A visualização do módulo pelos estudantes é feita pela seleção da opção Academic. Para ter acesso às listas de exercícios ele precisa aceitar o convite enviado pelo mestre por meio do sistema. Ao aceitar, terá acesso às disciplinas onde estarão as listas de exercícios que precisará resolver e uma barra de progresso indica o quanto uma tarefa foi realizada. As listas, visíveis apenas aos estudantes convidados pelo professor, apresentarão, entre outras informações, a identificação dos problemas, número de submissões, nível de dificuldade, orientações que foram passadas pelo professor da disciplina e prazo de conclusão com contagem regressiva do tempo. Além disso, o portal URI *Online Judge* disponibiliza um fórum integrado ao ambiente organizado por categorias.

3.6 Requisitos de um Juiz *Online*

Cada juiz *online* apresenta características próprias, desenvolvidas de acordo com a necessidade da Instituição de Ensino. Umas trazem os conceitos de gamificação na resolução dos exercícios dentro do ambiente e outras não. Algumas possuem uma preocupação na identificação do plágio nas respostas dos alunos, o que não acontece em outras. No entanto, todas possuem funcionalidades semelhantes que são exibidas, resumidamente, no Quadro 8.

Quadro 8 - Comparativo entre os juízes *online*

	URI	**The Huxley**	**CodeBench**	**BOCA**	**JOnline**
Ref. (Ano)	Selivon et al. (2015)	Paes et al. (2013)	Carvalho et al. (2016)	Campos & Ferreira (2004)	Santos & Ribeiro (2011)
Instituição Ensino	URI	UFAL	UFAM	USP	UFS
Avaliação Automática do código-fonte	S	S	S	S	S
Múltiplas submissões para mesma questão	S	S	S	S	S
Feedback automático avaliativo	S	S	S	S	S
Feedback automático informativo	S	N	N	N	S
Lista de Exercícios para exercitar os conceitos de programação	S	S	S	N	S
Histórico das execuções	S	S	S	S	N
Ambiente Próprio	S	S	S	S	S
Área do Professor	S	S	S	S	S
Área do Aluno	S	S	S	S	S
Disponível para acesso ONLINE	S	S	S	S	S
Exercícios classificados por Níveis de dificuldades (fácil, médio e difícil)	S	S	S	S	S
Exercícios classificados por tópicos de ensino (Estrutura de Controle, Variáveis e Tipos de dados, etc)	S	S	S	S	S
Identifica erros lógicos e sintáticos	S	S	S	S	S

(semânticos e léxicos)					
Trabalha com Turma, Disciplina	S	S	S	S	S
Definição das entradas de teste e saídas esperadas para os problemas pelo professor	S	S	S	S	S
Gamificado	S	S	S	N	N
Detecção de plágio	S	S	S	S	N

Fonte: Autora

Tendo como base essas características, identifica-se os requisitos funcionais básicos de um juiz *online,* que é permitir:

- a exibição de questões, podendo escolher o nível de dificuldade, tópico em estudo, linguagem de programação e disciplina;
- o cadastro de caso de teste, elaborado pelo professor, e que será executado sobre o código solução do aluno para identificar o erro lógico;
- a listagem de disciplinas e turmas;
- a inscrição de usuários do sistema, estabelecendo níveis de acesso e vinculando-os a uma ou mais disciplinas/turmas;
- a relação de questões, vinculando-a a uma disciplina/turma;
- a criação de perfis de acesso e permissões para vincular ao usuário. Todas as permissões da ferramenta serão liberadas por perfil de acesso que, inicialmente, conterá: administrador, professor e aluno;
- o cadastro de tópicos abordados nas questões, tais como: estrutura de iteração e de decisão, array, matrizes, etc.
- a análise de plágio da resolução submetida ao aluno no ambiente, informando-o que existe um código muito semelhante no sistema;
- que o aluno possa realizar múltiplas submissões do código para mesma questão até a sua conclusão;
- a identificação dos erros lógicos e sintáticos (semânticos e léxicos) ao executar uma questão;
- acompanhamento do professor sobre os progressos e andamentos das resoluções das questões realizadas pelos alunos;

- elaboração de relatório de desempenho ao aluno, para que este tenha uma visão de seu progresso dentro da disciplina;
- a disponibilização de um espaço para aluno e professor, os quais apresentarão informações relevantes para cada tipo de perfil; e
- o acesso ao ambiente via browser.

Foi observado, durante as análises dos juízes *online* que, em todas as ferramentas, o *feedback* apresentado aos usuários está mais relacionado ao *feedback* avaliativo do que ao informativo, isto é, preocupam-se em apresentar se o retorno da execução está correto ou não, sem muita explicação sobre o erro, com exceção do JOnline e URI que disponibilizam mais detalhes sobre o erro ocorrido em português. Das ferramentas analisadas, apenas o BOCA está disponível para *download* e instalação em um servidor. Porém, o juiz o*nline* BOCA não apresenta *feedbacks* aos usuários, uma vez que foi desenvolvido para competições em programação, sendo a única informação disponível se o algoritmo falhou ou não, ou seja, apresenta um *feedback* avaliativo. As demais ferramentas analisadas, necessitam de autorização para o seu uso pelo professor (cadastrar turma e alunos), a fim de utilizá-lo em sala de aula, o que levaria tempo para estabelecer o convênio entre as Instituições de Ensino.

Com isso, optou-se em desenvolver um juiz *online* com a finalidade de criar um ambiente controlado, onde fosse possível capturar os erros encontrados na execução das questões, e sobre tais erros apresentar um *feedback* avaliativo e informativo, tendo como base para sua criação a estratégia de *feedback*, dando mais autonomia no manuseio da ferramenta. Esse *feedback* busca dar visibilidade ao erro, tornando-o observável pelo aluno, contribuindo, assim, em seu entendimento e, consequentemente, na resolução do desequilíbrio cognitivo do estudante, na medida que esse novo conhecimento (dados em torno do erro) passa a ser assimilado em suas estruturas. Os requisitos levantados pela avaliação dos juízes *online* serviram de base para a criação dos requisitos do protótipo, que é apresentado com mais detalhes no **Apêndice I**.

A criação do *feedback* não pode ser feita de qualquer jeito, dado que é necessário criar uma orientação para o professor como forma de sistematização do levantamento das informações que comporão a recomendação ao seu registro. Com isso, a próxima seção apresenta uma estratégia de *feedback,* com a finalidade de padronizar o mapeamento e o registro dos erros pelo professor, na intenção de automatizá-lo.

6 ESTRATÉGIA DE FEEDBACK

Esta seção apresenta uma estratégia de *feedback* para propor uma maneira sistemática de mapear e registrar os erros de sintaxe e as falhas lógicas para que estes possam ser automatizados. Todavia, na seção 4 foram apresentados os conceitos em torno do tema *feedback*, apresentando uma visão de *feedback* formativo proposta por Shute (2007), o qual, segundo a autora, é a informação comunicada ao aluno que se destina a modificar seu pensamento ou comportamento, com a intenção de melhorar seu aprendizado. Ao exibir um *feedback* ao estudante, não basta simplesmente informar que o que foi feito está certo ou errado, precisa ir além, para que o erro (*feedback* negativo) produza um efeito significativo na aprendizagem.

Para Shute (2007), existem duas maneiras de transmitir a informação ao discente, podendo ser através de um *feedback* avaliativo e/ou informativo. O primeiro será um simples julgamento da resposta sobre o que o aluno fez — se está correta ou errada — não indicando **o quê** e nem **o porquê** errou; ao contrário do segundo, que oferece dicas e orientações sobre como chegar ao resultado esperado.

Buscando informações sobre o tema no Design Instrucional, Filatro (2008) comenta que, dependendo do tipo de questão, o *feedback* pode ser automatizado e exibido para o estudante após a correção de uma questão ou ao final de um bloco de perguntas. No entanto, para atividades mais complexas que envolvam habilidades cognitivas, que permitam mais de uma solução, como resolução de problemas (questão problema/solução com implementação), a autora recomenda que o *feedback* seja exibido durante a realização da atividade, e não após a sua conclusão, isto é, *feedback* instantâneo (síncrono).

Dependendo do tipo de aprendizagem, Filatro (2008, p.130) enumera oito estratégias que podem ser utilizadas no *feedback* durante os estudos em ambiente eletrônico: (1) indicar se a resposta está correta ou incorreta, sem informações extras; (2) indicar se a resposta está correta ou incorreta, explicando o motivo; (3) fornecer elementos para que o próprio estudante conclua que sua resposta está correta ou não, com o porquê; (4) apontar estratégias apropriadas para o encaminhamento de uma questão *online*, sem informar se o aluno está correto ou não; (5) mostrar ao aluno as consequências de suas respostas, especialmente com o uso de jogos ou simulações, nos quais cada ação é seguida por um *feedback* do sistema; (6) oferecer informação cumulativa sobre o seu progresso durante uma atividade - por exemplo,

informar sobre os padrões de erros mais cometidos ou quão próximo o aluno está de alcançar o objetivo proposto; (7) registrar, em formato de imagem ou vídeo, demonstrações de aprendizagem, que devem ser observadas pelos alunos, a fim de verificar passo a passo os efeitos de cada ação; e, por último, oferecer atividades extras, para que o aluno possa aplicar o *feedback* recebido em novas situações.

Shute (2007, p.2) avulta que, em resposta a uma ação do estudante (realização de uma tarefa, questionamento etc.), com o objetivo de facilitar a aprendizagem e melhorar seu desempenho em ambiente digital, o *feedback* poderá ser fornecido por mensagens, áudios, vídeos, imagens ou outros recursos digitais. Hattie e Timperley (2007, p. 86) consideram um *feedback* eficaz aquele em que é possível responder a três questões (CAPTIVO, 2018, p. 10) (ORNELAS, 2018, p. 12):

I. **Para onde eu vou (*Where am I going:* What are the goals)?** que objetivos se pretende ou o que é suposto fazer;

II. **Como eu vou (*How am I going:* What progress is being made toward the goal)?** quais progressos estão sendo realizados em direção ao objetivo; e

III. **Quais são os próximos passos (*Where to next:* What activities need to be undertaken to make better progress)?** quais atividades são necessárias para progredir na aprendizagem.

As respostas a essas perguntas "[...] melhoram a aprendizagem quando não há uma discrepância entre o que é entendido e o que se destina a ser compreendido" (Hattie & Timperley, 2007, p. 102). Segundo Hattie & Timperley (2007), as três perguntas estão relacionadas com o *feed up, feed back* e *feed forward*, respectivamente. Essas perguntas dependem do nível de *feedback* em que se encontram, e são classificadas, pelos autores, em quatro níveis, que estão delineados no Quadro 9.

Quadro 9 - Os quatro níveis de *feedback*

4º nível →aluno (Self: personal evaluations and affect (usually positive) about the learner) Relaciona-se com as características pessoais dos estudantes. O *feedback* fornecido inclui elogios sobre o que o aluno desenvolveu na tarefa. Este tipo de *feedback* motiva os alunos a continuarem se esforçando e fortalecem a confiança e a autoestima.
3º nível →Auto-regulação (Self-regulation: self-monitoring, directing and regulating of actions) Incide sobre a regulação das tarefas, em que se desenvolve a autoavaliação e a confiança no envolvimento da atividade. Para o aluno poder se avaliar, é importante que o professor produza um *feedback* que incorpore

indicações do que o aluno deve melhorar e compreender. Este *feedback* influencia as competências de autorregulação de tal forma que os alunos são incentivados ou informados a melhorar a tarefa.
2º nível →processo (Process: the main process needed to understand/perform tasks) Direciona-se para o processo e compreensão das tarefas. Esse feedback diz respeito a informações sobre as relações no ambiente, as relações percebidas por uma pessoa e as relações entre o ambiente e as percepções da pessoa. Com isso, o professor, durante este nível, deve produzir um *feedback* que englobe indicações que requerem a compressão do aluno, para este completar a tarefa. Deve ser um *feedback* que consiga ir ao encontro dos objetivos da própria tarefa.
1º nível →tarefa (Task: how well tasks are understood/performed) Relaciona-se com o desempenho do aluno durante a realização das tarefas. Para tal, o professor precisa fornecer um *feedback* acessível, que recomende o que deve ser melhorado e o que está correto e incorreto, mas sem mostrar o erro. Esse tipo de feedback é chamado de *feedback* corretivo ou conhecimento de resultados, e pode estar relacionado à correção, organização, comportamento ou algum outro critério relacionado à realização da tarefa.

Fonte: HATTIE & TIMPERLEY, 2007, p.88-97

A estratégia proposta nesta seção tende a trabalhar nos 1º e 2º níveis sugeridos por Hattie & Timperley (2007), mas, com enfoque na abordagem do estudo do erro produzido durante o aprendizado em programação de computadores. Cabe salientar que o *feedback* cadastrado no protótipo é elaborado sobre os erros encontrados pelo compilador, e os erros de lógica pela avaliação dos casos de teste aplicados no código do aluno. À medida que Hattie & Timperley (2007) utilizam a estratégia para avaliar o desempenho dos alunos, o plano proposto nesta seção intenciona abordar o erro de maneira diferenciada, buscando seu entendimento pelo aluno.

Apesar de a importância do *feedback* ser apresentada em todas as ações realizadas pelo estudante, com mensagens apropriadas de acordo com o resultado obtido (acerto ou falha), é indispensável que as mensagens tenham valor para o estudante e sirvam como fonte de formação de novos conceitos. Não obstante, é necessário estabelecer um padrão de registro desses *feedbacks*, que possa ser seguido pelo professor e que cause o efeito de orientação ao aluno, auxiliando na regulação acionada pelo *feedback* negativo (perturbação).

Com isso, para exemplificar o uso dessa estratégia sobre o erro, selecionou-se dois exercícios aplicados em um simulado em sala de aula, antes da primeira prova, na disciplina de Linguagem de Programação I do curso STSI do IFRS *Campus* Porto Alegre. Cada exercício trabalha alguns conteúdos aprendidos durante o semestre, com assuntos que versam sobre: variáveis e tipos de dados, operadores, estrutura de decisão e estrutura de controle. As duas questões do simulado são listadas no Quadro 10.

Quadro 10 - Lista de exercícios base para a estratégia de *feedback*

Nº	Enunciado	Habilidades
Q1	Faça um algoritmo que leia 10 valores inteiros, com estrutura de repetição [1 ponto]. Se o valor for ímpar, apresentar a quantidade de divisores [1 ponto], se o valor for par, apresentar o quadrado [1 ponto]. Uso obrigatório de estrutura de repetição.	Variáveis e tipos de dados, Operadores relacionais e aritméticos, Estruturas de Decisão e Controle
Q2	Em uma eleição para Miss RS existem quatro candidatas. Os votos são informados através de códigos. Os dados utilizados para a contagem dos votos obedecem à seguinte codificação: ● 11 = Candidata 1 ● 22 = Candidata 2 ● 33 = Candidata 3 ● 44 = Candidata 4 ● 77 = voto em branco ● Qualquer número diferente de 0 = Voto Nulo Elabore um algoritmo que leia os códigos da candidata que receberá o voto, até a eleição ser finalizada com o voto 0. Escreva no final: ● Total de votos para cada candidata ● Percentual de VOTOS VÁLIDOS* de cada candidata ● Total de votos nulos ● Total de votos em branco * Votos Válidos = Total de votos subtraídos os votos NULOS.	Variáveis e tipos de dados, Estruturas de Decisão e Controle

Fonte: Questões do simulado da disciplina LPI do curso STSI disponível no Moodle

6.1 Proposta de Estratégia de *Feedback*

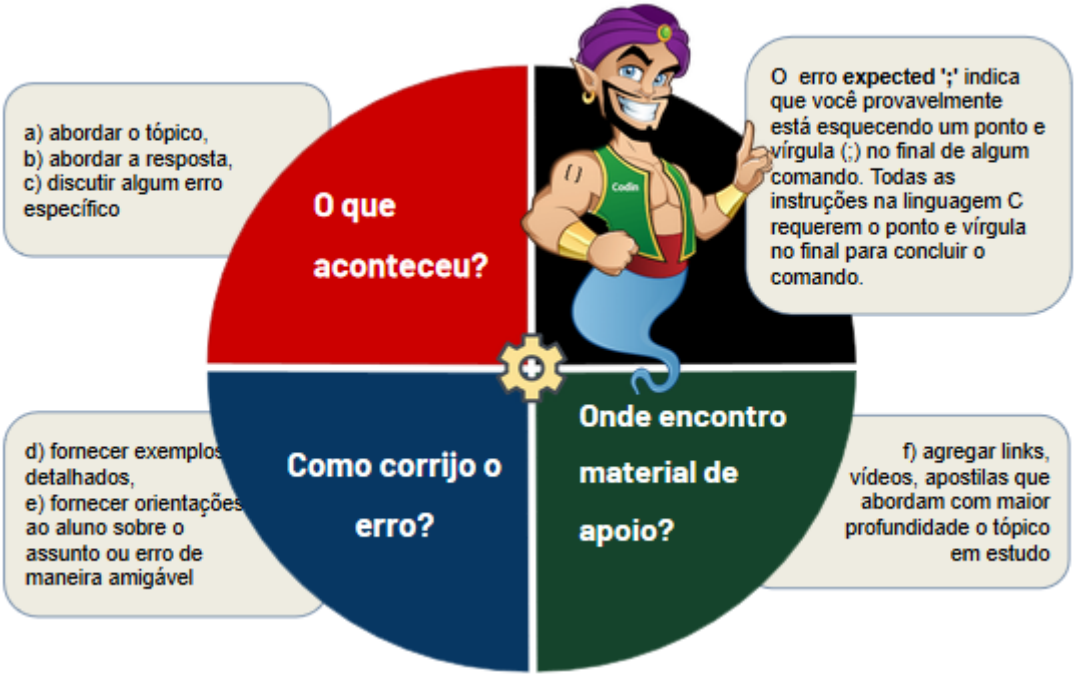
Como tornar o erro perceptível ao estudante? De acordo com Shute (2007), com o *feedback* formativo é possível traçar diversas abordagens (Figura 8): versar sobre o tópico, explanar a resposta, discutir algum erro específico, fornecer exemplos detalhados e expor orientações ao aluno sobre o assunto ou erro de maneira amigável. Com isso, agrupou-se as características mais semelhantes, classificando e depois transformando-as em questões, criando, assim, os questionamentos:

I. **O que aconteceu?** espaço para o professor explicar o erro e realizar uma breve explanação sobre o tópico que ocorreu o problema, fornecendo elementos para que o próprio estudante conclua que sua resposta está incorreta;

II. **Como corrijo o erro?** espaço para o docente apresentar exemplos, dicas com os comandos mais apropriados para resolver a questão e orientações do que os alunos precisam ter em mente para corrigir o erro e finalizar a tarefa;

III. **Onde encontro material de apoio?** espaço para o professor agregar links, vídeos, apostilas que abordam com maior profundidade o tópico em estudo. Este item é opcional, já que existem situações nas quais não há necessidade de apresentar material de apoio pela simplicidade do erro. Cita-se nessa categoria o erro *expected ';' before '...'*. que ocorre quando o aluno esquece o *token* ponto e vírgula (;) no final do comando. Respondendo as duas primeiras perguntas da estratégia, é suficiente para explicar e exemplificar o uso deste *token* no código.

Figura 8 - O que? Como? e Onde? da Estratégia de *Feedback*



Fonte: Autora

As respostas a essas questões disponibilizará ao aluno um guia por onde começar a sua investigação sobre o erro, desencadeando sua reflexão antes de partir para o ajuste da falha no código. É importante observar que as informações apresentadas no *feedback,* de acordo com Shute (2007) e Abreu-e-Lima & Alves (2011), devem mostrar uma linguagem clara, de fácil entendimento e o conteúdo escrito não deve ser extenso (complexidade), além de ser organizado e adequado ao contexto (OLIVEIRA, 2018).

Os códigos-fonte utilizados nos exemplos foram reproduzidos de erros reais ocorridos em sala de aula e identificados durante a observação da pesquisadora. Esses códigos servirão de suporte para aplicação da estratégia definida nesta seção, onde as questões 1 e 2 foram

executadas na ferramenta DevC++ e os erros apresentados relacionados aos erros sintático e lógico, respectivamente.

Tomando como exemplo a Q2, apresentada no Quadro 10, o aluno precisará realizar algumas operações necessárias para chegar ao resultado definido no enunciado da questão. No entanto, ao compilar a solução proposta por ele, ocorre um erro inesperado durante a execução do código-fonte no arquivo **main.c**, na ferramenta DevC++. O erro *[Error] ld returned 1 exit status* aponta para a linha 25 do arquivo **Makefile.win** (que é um arquivo interno do DevC++) e contém o seguinte código: **$(CC) $(LINKOBJ) -o $(BIN) $(LIBS)**. Esta mensagem deixa o estudante confuso, pois o erro ocorreu em outro arquivo. O algoritmo e o erro do compilador são apresentados na Figura 9.

Figura 9 - [Error] ld returned 1 exit status

```c
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int i, voto, cand1 = 0, cand2 = 0, cand3 = 0, cand4 = 0, brancos = 0, nulos = 0, validos;

    for(i = 10; i > 0; i--) {
        scanf(voto);

        switch(voto) {
            case 11: cand1++; break;
            case 22: cand1++; break;
            case 33: cand1++; break;
            case 44: cand1++; break;
            case 77: brancos++; break;
            default: nulos++;
        }
    }

    validos = (cand1 + cand2 + cand3 + cand4 + brancos);

    print("Candidata 1: %d - %d%% validos\n", cand1, (cand1 * 100) / validos);
    print("Candidata 2: %d - %d%% validos\n", cand2, (cand2 * 100) / validos);
    print("Candidata 3: %d - %d%% validos\n", cand3, (cand3 * 100) / validos);
    print("Candidata 4: %d - %d%% validos\n", cand4, (cand4 * 100) / validos);

    return 0;

}//main
```

Unidade	Mensagem
C:\Users\Usuario\Documents\main.o	main.c:(.text+0xf4): undefined reference to `print'
C:\Users\Usuario\Documents\main.o	main.c:(.text+0x114): undefined reference to `print'
C:\Users\Usuario\Documents\main.o	main.c:(.text+0x134): undefined reference to `print'
C:\Users\Usuario\Documents\collect2.exe	[Error] ld returned 1 exit status
C:\Users\Usuario\Documents\Makefile.win	recipe for target 'Projeto1.exe' failed

Fonte: Autora

A mensagem exibida pelo compilador GCC não é clara para concluir o motivo do erro. No entanto, o comando **print()**, de fato, não existe na linguagem de programação C e é um erro clássico de linkagem, quando o compilador não encontra a função nas bibliotecas declaradas no início do programa. Como propor um *feedback* para o erro? Seguindo a

estratégia mencionada no início desta seção, o *feedback* deverá responder a três perguntas.

Quadro 11 - Proposta de *feedback* para o [Error] ld returned 1 exit status

(i)o que aconteceu?
O erro ***ld returned 1 exit status*** ocorre quando um programa usa uma função ou variável que não está definida em nenhum dos arquivos ou bibliotecas de objetos fornecidos ao compilador. Também, pode ser causado por uma biblioteca ausente ou pelo uso incorreto do nome ou função.

(ii)como corrijo o erro?
Veja o exemplo:

```
int main(void)
{
 int num;
 scanf("%d", &num);

 if(num > 0) {
  print("maior que zero");
 }
 return 0;
}
```

O erro ocorreu porque o comando **print()** não existe na linguagem C, mas **printf()** sim, cuja sintaxe é:

```
printf("Conversão/Formato do argumento", argumentos);
```

Onde:

<Conversão/Formato do argumento> como as mensagens serão exibidas

<argumentos> pode conter variáveis, expressões aritméticas ou valores fixos

(iii)onde encontro material de apoio extra?
No nosso curso no moodle na aula sobre variáveis possui mais informações sobre o seu uso e os especificadores de conversão/formatadores. Você pode acessar, também, o livro digital C Completo e Total da Makron Books.

Fonte: Autora

Continuando a exemplificação, utiliza-se a Q1 do Quadro 10, na qual o aluno precisa elaborar um algoritmo que leia dez valores inteiros, empregando uma estrutura de repetição. Se o valor for ímpar, deverá apresentar a quantidade de divisores, se o valor for par, apresentar o seu quadrado. Os erros sintáticos, léxicos e semânticos são facilmente identificados pelos compiladores das linguagens de programação, porém, os erros lógicos são mais complexos de se detectar, principalmente, quando os códigos são extensos. Para

identificar os erros lógicos é preciso que seja criado um conjunto de casos de teste com suas entradas e saídas esperadas.

A atividade de elaborar os casos de testes é crucial para cobrir o máximo de situações possíveis no algoritmo, com a finalidade de conhecer as falhas de lógica no código. Um ponto primordial na atividade de teste é a avaliação da qualidade de um determinado conjunto de casos de teste utilizado para testar um programa. É inviável utilizar todo o domínio de dados de entrada para avaliar se o que foi implementado foi realmente solicitado no enunciado da questão. Maldonado et al. (2010) discutem que um teste bem sucedido é aquele que consegue determinar casos de teste, para aqueles que o programa em teste falhe.

Para conduzir e avaliar a qualidade da atividade de teste, existem as técnicas de teste: funcional, estrutural e baseada em erros, como comentado na seção 4. Molinari (2008) apresenta duas técnicas de teste funcional que auxiliam na identificação dos dados que comporão os casos de teste. Para exemplificar, será aplicada a técnica de **Teste de Equivalência de Classe**, a fim de identificar os conjuntos de dados de cada classe de equivalência. O **Teste de Valor Limite** busca identificar valores fronteiriços entre as classes, por ser muito comum a ocorrência de problemas nas extremidades limítrofes, caso que acontece, principalmente, quando estruturas condicionais são utilizadas nos programas. Na Q1, não será necessário usar essa técnica, uma vez que não há necessidade deste controle. Em função de que o enunciado não solicita a validação da entrada de dados para aceitar somente números inteiros, não haverá um conjunto de teste para verificar essa condição. Os casos de teste com as entradas e resultado esperado para a Q1 são apresentados no Quadro 12.

Quadro 12 - Casos de Teste elaborado para a Q1 do Quadro 10

Técnica	**Entrada**	**Resultado Esperado**
Técnica de Teste de Equivalência de Classe	{2,4,6,8,10,12,14,16,18,20} Equivalência de Classe par	Quadrado: 4, Quadrado: 16, Quadrado: 36, Quadrado: 64, Quadrado: 100, Quadrado: 144, Quadrado: 256, Quadrado: 324, Quadrado: 400
	{1,3,5,7,9,11,13,15,17,19} Equivalência de Classe ímpar	Divisor: 1, Divisores: 2, Divisores: 2, Divisores: 2, Divisores: 3, Divisores: 2, Divisores: 2, Divisores: 4, Divisores: 2, Divisores: 2
	{2,3,6,8,5,7,9,10,15,1} Equivalência de Classe mista	Quadrado: 4, Divisores: 2, Quadrado: 36, Quadrado: 64, Divisores: 2, Divisores: 2, Divisores: 3, Quadrado:

		100, Divisores: 4, Divisor: 1

Fonte: Autora

Durante as execuções da bateria de teste, o caso `{2,4,6,8,10,12,14,16,18,20}` passou sem erros, ainda assim, os casos `{1,3,5,7,9,11,13,15,17,19}` e `{2,3,6,8,5,7,9,10,15,1}` não passaram no teste quando aplicados no código apresentado na Figura 10. O erro está na variável **qtdDiv**, que foi colocada dentro do comando **for()** e não é reinicializada fora do laço. Isso provoca a adição de todos os divisores na variável.

Figura 10 - Algoritmo desenvolvido para a Q2 do Quadro 10

```c
#include <stdio.h>
#include <stdlib.h>

int main() {
    int n, i, div, qtdDiv = 0;

    for(i = 0; i < 10; i++) {
        //printf("Informe um valor inteiro: ");
        scanf("%d", &n);

        if(n % 2 != 0) {
            for(div = 1; div <= n; div++){
                if(n % div == 0) {
                    qtdDiv++;
                }
            }

            printf("Quantidade de divisores: %d\n", qtdDiv);
        }

        if(n % 2 == 0) {
            printf("Quadrado: %d\n", n * n);
        }
    }

    return 0;
}
```

```
2
Quadrado: 4
3
Quantidade de divisores: 2
6
Quadrado: 36
8
Quadrado: 64
5
Quantidade de divisores: 4
7
Quantidade de divisores: 6
9
Quantidade de divisores: 9
10
Quadrado: 100
15
Quantidade de divisores: 13
1
Quantidade de divisores: 14
```

Fonte: Autora

Como propor um *feedback* para esse tipo de erro? Para cada caso de teste, é possível criar recomendações específicas, de acordo com a equivalência de classe, pela qual alguns casos podem passar e outros não, assim como todos os casos de testes podem falhar. A sugestão é criar uma recomendação mais genérica para as falhas com casos de testes que tenham valores mistos; e recomendações mais específicas para os casos de teste com valores direcionados, já que é possível se realizar uma bateria de testes de modo mais particular.

Quadro 13 - Proposta de *feedback* para o erro da Q2 do Quadro 10 - Classe eq. mista

```
(i)o que aconteceu?
O resultado obtido da execução do caso de teste não coincidiu com o
resultado esperado. Para resolver esta questão, você precisará utilizar os
comandos de controle e decisão, além dos operadores relacionais e
```

aritméticos.

Atente-se a alguns detalhes de implementação:

- Como saber se um número é par? neste caso, você precisa utilizar o operador **módulo (%)**, que retorna o resultado da operação, podendo ser 0 ou 1. Caso seja 0, significa que o valor é divisível, caso contrário, não.
- Como saber quantos divisores o número **ímpar** possui? Primeiramente, é necessário identificar quais números são divisíveis por ele. Por exemplo, o número 9 é divisível pelos números: **1, 3 e 9**. Para fazer isso dinamicamente, é preciso utilizar uma estrutura de repetição, como o comando for junto com o **operador %**, comentado anteriormente.
- Não esqueça de utilizar uma variável contadora que some o número de divisores, e esse contador deverá ser inicializado a cada número ímpar.

(ii)como corrijo o erro?
Além das dicas apresentadas, verifique também as seguintes situações no seu código:

- Tipo de dado está adequado na variável?
- Entrada está correta, ou seja, foi utilizado o "&" na variável?
- Cálculo está correto?

Tais situações, também, podem ocorrer problemas na execução do código. Como exercício, outra opção interessante é realizar a depuração do código manualmente.

(iii)onde encontro material de apoio extra?
Maiores informações sobre os comandos while, if e operadores acesse o nosso curso no Moodle nas aulas de: **Sentença de iteração - Laço (FOR WHILE DO/WHILE)** e **Operadores e Sentença de Decisão (IF).**

Fonte: Autora

Por ser mais genérico, o *feedback* para o caso de teste com valores mistos (Quadro 13), é interessante abordar os possíveis problemas no formato de tópicos, para não sobrecarregar de informações na tela do computador, seguindo as orientações de Shute (2007), Abreu-e-Lima & Alves (2011) e Oliveira (2018). Cada equivalência de classe visa exercitar caminhos diferentes no código-fonte. Com isso, o caso de teste {2,4,6,8,10,12,14,16,18,20}, que contempla a equivalência de classe para números pares, apresentado no Quadro 10, busca percorrer o caminho que realiza o quadrado de um número par, e em caso de falha, uma sugestão de *feedback* é apresentada no Quadro 14, que tem o intuito de explicar como chegar ao resultado.

Quadro 14 - Proposta de *feedback* para o erro da Q2 do Quadro 10 - Classe eq. par

(i)o que aconteceu?
O resultado obtido da execução do caso de teste não coincidiu com o resultado esperado. Para resolver esta questão, você precisará utilizar os comandos de controle e decisão, além dos operadores relacionais e aritméticos.

O caso de teste {2,4,6,8,10,12,14,16,18,20} busca verificar se o código está calculando corretamente o quadrado do número. Mas, para identificar os números pares, é preciso utilizar o **operador módulo (%)**, que retorna como resultado da operação 0 ou 1.

Veja o seguinte exemplo: **if(num%2 == 0)**, verifica se num é par. É igual a 0, porque todo resto da divisão de números pares dividido por 2, SEMPRE será 0.

(ii)como corrijo o erro?
Além das dicas apresentadas, verifique também as seguintes situações no seu código:

- Tipo de dado está adequado na variável?
- Entrada está correta, ou seja, foi utilizado o "&" na variável?
- Cálculo está correto?

Tais situações, também, podem ocorrer problemas na execução do código. Como exercício, outra opção interessante é realizar a depuração do código manualmente.

(iii)onde encontro material de apoio extra?
Maiores informações sobre os comandos while, if e operadores, acesse o nosso curso no Moodle nas aulas de: **Sentença de iteração - Laço (FOR WHILE DO/WHILE)** e **Operadores e Sentença de Decisão (IF).**

Fonte: Autora

O mesmo procedimento é realizado para a equivalência de classe para números ímpares, ao utilizar o caso de teste {1,3,5,7,9,11,13,15,17,19} como entradas, e em caso de falha, um *feedback* apresentado no Quadro 15 é exibido ao aluno.

Quadro 15 - Proposta de *feedback* para o erro da Q2 do Quadro 10 - Classe eq. ímpar

(i)o que aconteceu?
O resultado obtido da execução do caso de teste não coincidiu com o resultado esperado. Para resolver esta questão, você precisará utilizar os comandos de controle e decisão, além dos operadores relacionais e aritméticos.

O caso de teste {1,3,5,7,9,11,13,15,17,19} busca verificar se o código está calculando corretamente a quantidade de divisores do número.

Como saber quantos divisores o número ímpar possui? Primeiramente, é necessário identificar quais números são divisíveis por ele. Por exemplo, o número 9 é divisível pelos números: **1**, **3** e **9**. Para fazer isso dinamicamente, é preciso utilizar uma estrutura de repetição (for) junto com o **operador** %. Por quê? para avaliar se o número é divisível pelo

índice da estrutura de repetição.

- OBSERVAÇÃO: Não esqueça de utilizar uma variável contadora que some o número de divisores e esse contador deverá ser inicializado a cada número ímpar.

(ii)como corrijo o erro?
Além das dicas apresentadas, verifique também as seguintes situações no seu código:

- Tipo de dado está adequado na variável?
- Entrada está correta, ou seja, foi utilizado o "&" na variável?
- Cálculo está correto?

Tais situações, também, podem ocorrer problemas na execução do código. Como exercício, outra opção interessante é realizar a depuração do código manualmente.

(iii)onde encontro material de apoio extra?
Maiores informações sobre os comandos while, if e operadores acesse o nosso curso no Moodle nas aulas de: **Sentença de iteração - Laço (FOR WHILE DO/WHILE)** e **Operadores e Sentença de Decisão (IF).**

Fonte: Autora

Para saber qual *feedback* exibir, de acordo com o erro gerado, é necessário realizar um levantamento dos erros praticados pelos alunos nas disciplinas iniciais em programação. Os erros exibidos pelo compilador são relativamente fáceis de se estabelecer um *feedback* apropriado, dado que a maioria dos erros são conhecidos e publicamente disponibilizados para consulta. Porém, para criar os *feedbacks* para o erro lógico, o professor precisará analisar as situações possíveis, a fim de fornecer informações sobre o erro, que estejam de acordo com o possível problema. Por exemplo, se, durante a bateria de teste do docente todos os casos de teste falharem, uma das prováveis conclusões é desconfiar que pode ser um erro de cálculo em uma variável utilizada por um comando de decisão no código ou a ausência do operador *&* no comando de *scanf()*. Não obstante, caso ocorra o erro em apenas um caso de teste é possível existir uma falha na utilização do operador relacional ou lógico, ou até mesmo em uma variável contadora. Cada situação precisará ser analisada e pensada pelo mestre antes de elaborar o conjunto de teste para o exercício e seus *feedbacks*. Apesar disso, destaca-se dois grandes desafios na construção das recomendações: saber se o conteúdo do *feedback* é suficiente e esclarecedor para ajudar o aluno na atividade; e se a quantidade de informação no *feedback* está adequada e não está ocasionando um efeito contrário. A única maneira de driblar esses desafios é ouvindo os alunos. A próxima seção trata sobre o protótipo desenvolvido para este estudo, mostrando suas principais funcionalidades.

7 O PROTÓTIPO - CODEIN'PLAY

Com base na análise das ferramentas selecionadas na literatura, descritas na seção 5, identificou-se os requisitos comuns entre os juízes *online* e para eles se aponta oportunidades de melhoria, para a proposição da ferramenta chamada de Codein'play.

O Codein'play é um juiz *online* disponibilizado em um ambiente WEB com *design* responsivo para os alunos iniciantes em programação de computadores, para orientá-los na aprendizagem do ensino da programação estruturada com a linguagem C. Seu objetivo é proporcionar um panorama diferenciado dos erros que ocorrem durante a resolução de exercícios. Além da validação automática do código, o sistema emite *feedbacks* com orientações sobre os erros gerados pela solução do aluno, permitindo que este consiga evoluir na correção de seu programa. Todos os *feedbacks* são cadastrados pelo professor em uma área específica dentro da aplicação, seguindo uma estratégia de *feedback* cujo conteúdo será norteado a partir da resposta de três questões (o que aconteceu? como corrijo? onde encontro material de apoio?) relativas ao erro.

7.1 Os Requisitos Funcionais da Ferramenta

O desenvolvimento da ferramenta buscou atender aos requisitos descritos nesta seção, que foram definidos em função da análise das características de ferramentas selecionadas nos trabalhos relacionados. No Quadro 16, são apresentadas algumas funcionalidades implementadas na ferramenta, associando seu respectivo requisito funcional.

Quadro 16 - Funcionalidades do Codein'play

Funcionalidade	**RF#**	**Professor**	**Aluno**	**Administrador**
Gerenciar Eventos (acompanhar as entregas pelos alunos)	03	x		
Gerenciar Turmas e Alunos	02	x		
Gerenciar Questões (casos de teste, dicas, enunciado, tópicos, esqueleto inicial do código-fonte, etc.)	04	x		
Gerenciar Usuários (cadastrar usuário e aprovar inscrições)	11	x		x
Gerenciar Comandos Proibidos	12	x		
Gerenciar Erros e Recomendações	14	x		
Visualizar o Histórico das Execuções	15	x	x	
Desenvolver Questões (executar, testar com os casos de teste do professor, cobertura de código)	09	x	x	
Compartilhar e Realizar *download* do código	15	x	x	
Possibilitar filtragem de questões	20	x	x	
Configurar o Ambiente	13			x

Gerenciar Alertas de mensagens do compilador não encontradas na base	19	x		
Visualizar LOG das ações realizadas pelo usuário	18			x
Emitir Relatório de Erros cometidos pelos alunos	07	x		
Exibir Relatório de Número de Execuções (acertos/erros) por questão/aluno	07	x		
Apresentar Relatório de Tentativas de acertos por questão/aluno	07	x		
Consultar Relatório de Desempenho por questão e tópico	08	x	x	
Emitir Relatório de Histórico de Execuções por questão/aluno	07	x		
Gerenciar Perfil de Acesso	01			x

Fonte: Autora

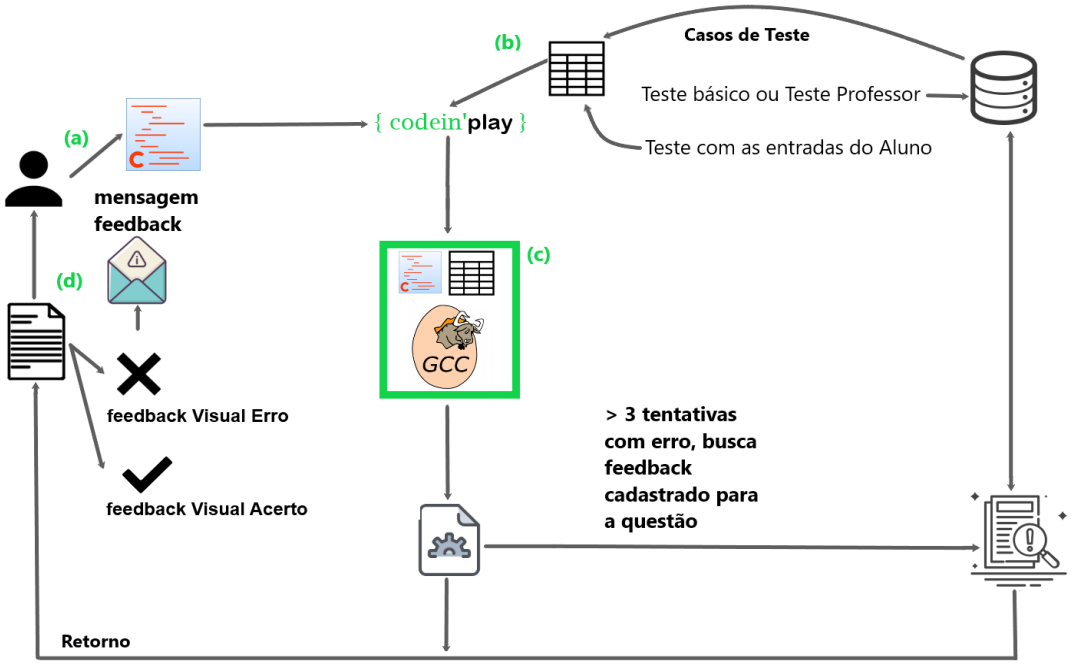
A descrição completa dos Requisitos Funcionais, o Modelo Relacional e o Script de criação do banco de dados e tabelas do Codein'play estão no **Apêndice I**.

7.2 Descrição da Ferramenta

O processo de funcionamento da ferramenta, Figura 11, inicia com a execução do código-fonte pelo aluno **(a)**. Após esta ação, sendo ela uma execução com dados de caso de teste básico ou caso de teste do professor, o Codein'play busca, no banco de dados, os testes cadastrados pelo professor para o exercício que o aluno está tentando resolver. Na ferramenta, será possível testar o código de três maneiras: executando o teste básico sugerido pelo professor, ou, o aluno informar as entradas de dados para a questão, ou, executar a bateria de casos de teste definidos pelo docente. Em todos os casos é realizado o teste de caixa-preta, quando são inseridos dados conhecidos, coletando as saídas geradas pelo programa e comparando com as saídas esperadas, com exceção do teste com os dados informados pelo aluno, no qual o resultado esperado é desconhecido no sistema previamente **(b)**.

Os dados de testes e o código são reunidos e todos os testes são executados um por um e, a cada execução, a ferramenta compara a resposta gerada pelo programa do aluno com as respostas esperadas, definidas pelo professor **(c)**. Havendo falha na execução, a mensagem de erro gerada pelo compilador é exibida ao aluno no idioma inglês, com a opção de tradução para o português. Persistindo o erro mais de três vezes, uma mensagem de *feedback* é exibida ao aluno, contendo orientações para que este consiga resolver o problema e prossiga no exercício **(d)**. Caso contrário, o programa é considerado correto e um *feedback* avaliativo positivo é exibido ao aluno.

Figura 11 - Processo de funcionamento Codein'play



Fonte: Autora

Neste projeto algumas ferramentas foram usadas com a finalidade de criar o ambiente e construir suas telas, sendo uma delas o aplicativo *open-source* Pencil[20] da empresa Evolus, que é um software de prototipagem que permite a criação de *mockups* de forma rápida e fácil. Para desenvolver o ambiente, foram utilizadas as tecnologias: PHP versão 7.2.19, framework CodeIgniter versão 3.1.10, jQuery, HTML 4, CSS, framework Bootstrap 4.0 e o banco de dados MariaDB versão 10.3.17. O editor utilizado na área de programa foi a biblioteca *open-source* ACE[21], configurada para mostrar a sintaxe da linguagem de programação C destacando em colorido os comandos da linguagem (*Syntax highlighting*). Outras características da biblioteca são: a facilidade em indentar os comandos no código-fonte e a opção de habilitar a numeração das linhas que será visualizada dentro do IDE, entre outras possibilidades. No que tange à infraestrutura, a ferramenta foi hospedada em uma máquina virtual linux Ubuntu versão 19.04 de 1GB de memória RAM e 16GB de HD, em um Servidor Intel(R) Xeon(R) de 2.40GHz no IFRS, rodando em um Servidor Web de código aberto NGINX versão 1.15.9.

Para executar o código-fonte na linguagem de programação C, o Codein'play integrou os comandos do compilador GCC e o pacote GCOV (cobertura de teste) que se encontram instalados na máquina virtual, juntamente com a ferramenta Codein'play. Esta integração é

20 <https://pencil.evolus.vn/>
21 <https://ace.c9.io/>

realizada mediante a chamada dos comandos do compilador, por meio de linha de comando dentro do *framework* PHP, armazenando seu retorno.

7.3 Controle de acesso de usuários

No ambiente, existem, inicialmente, três tipos de usuários: aluno, professor e administrador. O software possui um controle de acesso com login e senha e, depois da validação, a ferramenta identifica o perfil do usuário logado e carrega as informações de acordo com o tipo de acesso permitido.

O cadastro de usuário poderá ser realizado no sistema de três formas: em lote, a partir de um arquivo com as extensões .txt ou .csv, seguindo a configuração estabelecida para a leitura dos dados; individualmente, informando os dados do aluno manualmente; ou, com a inscrição feita pelo próprio aluno no ambiente, esta deverá ser aprovada pelo professor.

Ao autenticar-se na ferramenta, pelo número de matrícula fornecida pelo IFRS, o usuário é conduzido à tela principal do ambiente de acordo com o seu perfil de acesso. Nela, conterá a lista de questões e eventos cadastrados para a turma relacionados à linguagem de programação C e o relatório de desempenho de todas as suas execuções realizadas na ferramenta.

7.4 As Questões no Codein'play

No sistema, o aluno poderá resolver as questões de duas maneiras: mediante uma atividade agendada por um evento, ou, solucionar os exercícios avulsos, utilizando o link Questões, localizado no menu. Todas as execuções das questões são armazenadas em banco de dados para posterior consulta pelo professor e pelo aluno. Na tela que exibe as questões do menu Questões, o estudante poderá filtrar por: nível de dificuldade, tópico, disciplina e evento.

Para isso, o professor precisará cadastrar as questões que farão parte do banco de questões para serem utilizadas em eventos ou disponibilizá-las para resolução em separado, ou avulsas.

7.4.1 As Questões no Codein'play - Administração

O tipo de questão do Codein'play será ‘Questão problema/solução com implementação’ na linguagem de programação C, além disso, nada impede de o professor criar questões nas quais o aluno precise complementar o código, ou que o forcem a encontrar o erro dentro de um trecho de código.

Uma questão conterá campos que servirão de apoio no momento em que o aluno tentar resolvê-la, sendo uma delas a dica. Na tela de cadastro de questões, o professor poderá classificar a questão pelo nível de dificuldade (Fácil, Médio e Difícil), e apresentar um esqueleto inicial para o aluno adicionar seu algoritmo. O campo ‘Quantidade de inputs na questão’ indica a quantidade de objetos *Input Text* distribuídos na área de entrada (*scanf*) dentro da ferramenta, isto é, os campos nos quais os alunos entrarão com os dados para testar seus códigos. Adicionalmente, poderá habilitar a alteração da questão por outro professor, que por *default* somente o Autor da questão poderá realizá-la.

O campo ‘Solução Modelo’ conterá a solução proposta pelo docente para a questão, o objetivo é publicá-la após o aluno finalizar a resolução (clicar no botão Enviar e Finalizar da questão). Este campo também poderá ser utilizado com o propósito de verificar, no futuro, se a solução implementada pelo aluno se aproxima da solução cadastrada pelo docente. Esta funcionalidade não foi implementada na ferramenta, sendo uma melhoria do juiz *online*.

Após salvar o cadastro inicial, o professor deverá informar outros dados importantes para o correto funcionamento dentro do ambiente, como: o tópico no qual a questão está diretamente relacionada e algum recurso de exceção, que é o valor que finalizará o algoritmo. Por exemplo: em um algoritmo de votação, o número ‘0’ encerrará o pleito, ou seja, sairá do *loop* e imprimirá os dados coletados na tela. Neste caso, o professor deverá informar o valor que encerra o *script*. Ao registrar essa informação, o sistema adicionará mais um campo na área de entrada de dados (*scanf),* com o valor no final de todos os campos *inputs* sem ter a opção de o aluno alterar.

Outro recurso disponibilizado na ferramenta é a funcionalidade ‘Abortar Execução’, que, ao ser acionada, abortará imediatamente a execução do algoritmo, o qual acidentalmente, entrou em *loop* infinito. Problema muito comum quando o estudante está aprendendo o uso das estruturas de controle. Por último, o mestre deverá cadastrar os casos de teste para a

questão, que são de dois tipos: Básico e Professor, que se diferem na maneira como são executados.

O **teste básico** está divulgado no enunciado da questão e é uma sugestão do professor ao aluno de um teste em que são conhecidas as entradas e saídas e é possível saber se o resultado obtido é o mesmo do esperado. Os valores das entradas do teste básico definidos pelo professor são carregados nos campos de entrada de dados (*scanf*) para facilitar a execução do código. Se este teste passar, não é um indicativo de corretude, já que somente um caminho de teste é realizado. Durante a execução deste teste, o aluno poderá visualizar a cobertura do código percorrida pelo compilador dentro de seu código-fonte, em decorrência das entradas informadas.

O **teste do professor** é uma bateria de testes, que tende a cobrir a maior quantidade de situações dentro do código do aluno. Os testes são executados um a um, e o resultado obtido é avaliado com o esperado, que é cadastrado pelo professor. No final, as execuções são exibidas ao estudante com o retorno se o teste passou ou não.

Figura 12 - Cadastro dos casos de teste da questão

Fonte: Autora

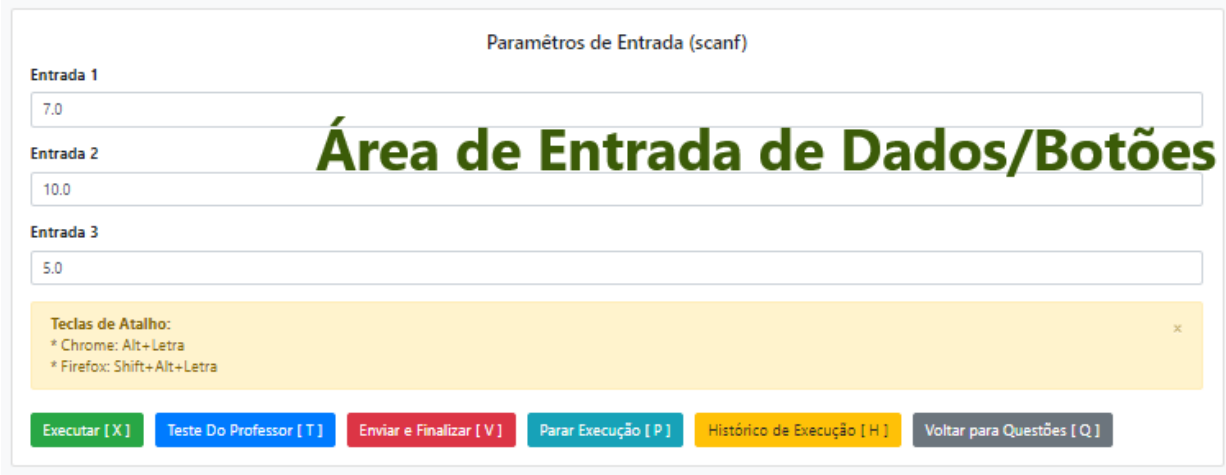
Para analisar a corretude do código, ao cadastrar um caso de teste na questão, o professor deverá informar quais os elementos (*tokens*) que o resultado do compilador deve, obrigatoriamente (**E**) ou opcionalmente (**OU**), possuir. Assim, cada resposta do aluno é submetida a uma validação, onde um conjunto de *tokens* é identificado dentro do resultado obtido, após a execução do código-fonte. Se o docente cadastrar no caso de teste o *token* com a opção **OU**, o sistema buscará no resultado do compilador, pelo menos, uma ocorrência do

token cadastrado; sendo a opção **E** acompanhada do cadastro do *token*, o ambiente tentará achar todos os *tokens* no resultado obtido. Se nenhum *token* for encontrado ou for parcialmente achado neste conjunto, existe um erro na resposta.

7.4.2 As Questões no Codein'play - Visão do Aluno

Ao selecionar uma questão, será exibido ao aluno quatro áreas, conforme exibidas na Figura 13.

Figura 13 - Questão no Codein'play



Fonte: Autora

Essas áreas possuem funções específicas na questão, as quais são:

- **área de enunciado:** local que contém o enunciado da questão, valores do teste inicial (básico) - que é uma sugestão válida que constará nos campos de

entrada de dados -, dica do professor para resolver o problema proposto;

- **área de programa:** IDE em que o aluno desenvolverá a solução para a questão;
- **área de entrada de dados/botões:** para informar os valores de entrada (*scanf)* e apresentar os botões Executar, Enviar e Finalizar, Teste do Professor, Parar execução, Histórico de Execução e Voltar para as Questões, descritos no Quadro 17; e
- **área de resultado**: local em que exibirá o resultado do teste junto com a cobertura do código.

Na área de programa, o sistema verificará se existe uma solução submetida ou executada pelo aluno, ao menos uma vez, para a questão selecionada. Se encontrar, o sistema carregará a última execução dentro da área de programa; caso contrário, carregará o conteúdo inserido no campo 'esqueleto inicial do código', informado pelo professor durante o cadastro da questão.

Quadro 17 - Botões disponíveis na questão

BOTÕES	FUNÇÃO
Executar [X]	chama o compilador GCC passando os dados de entrada informados na área de entrada (*scanf*), gravando a execução no banco de dados. Enquanto a questão estiver com o status 'em desenvolvimento', o aluno poderá alterar e executar o código quantas vezes forem necessárias. Caso ocorra algum erro de sintaxe, semântico e lexical, a mensagem do compilador é exibida na área de resultado; caso contrário o código é compilado e a cobertura do teste (GCOV) é exibida juntamente com o resultado. Neste momento, a ferramenta analisa o erro de lógica no código, se o teste executado for básico; senão este tipo de erro não é avaliado. A ferramenta, entendendo que o aluno está executando o código mais de três vezes e o erro persistir, exibirá uma mensagem de *feedback*, previamente, mapeada pelo professor, a fim de orientar e ajudar o discente na identificação e na correção do problema. Tecla de atalho para executar o código: Alt+X ou Shift+Alt+X
Enviar e Finalizar [F]	salva o código no banco de dados, informando ao sistema que a solução da questão foi entregue, procedimento que desabilita a opção de alteração. Uma janela de confirmação é exibida para o usuário, antes de fechar a questão. Tecla de atalho para finalizar a questão: Alt+F ou Shift+Alt+F
Teste do professor [T]	salva a execução no banco de dados e chama o compilador GCC, passando os casos de teste informados pelo professor. Caso ocorra algum erro de sintaxe, semântico, lexical e lógico, a mensagem do compilador é exibida na área de resultado. A avaliação do erro lógico do código do aluno é realizada por meio de uma avaliação da saída obtida com o retorno do compilador. Se ela não for a mesma definida pelo professor, o sistema entenderá que há um erro lógico e um *feedback* será exibido ao

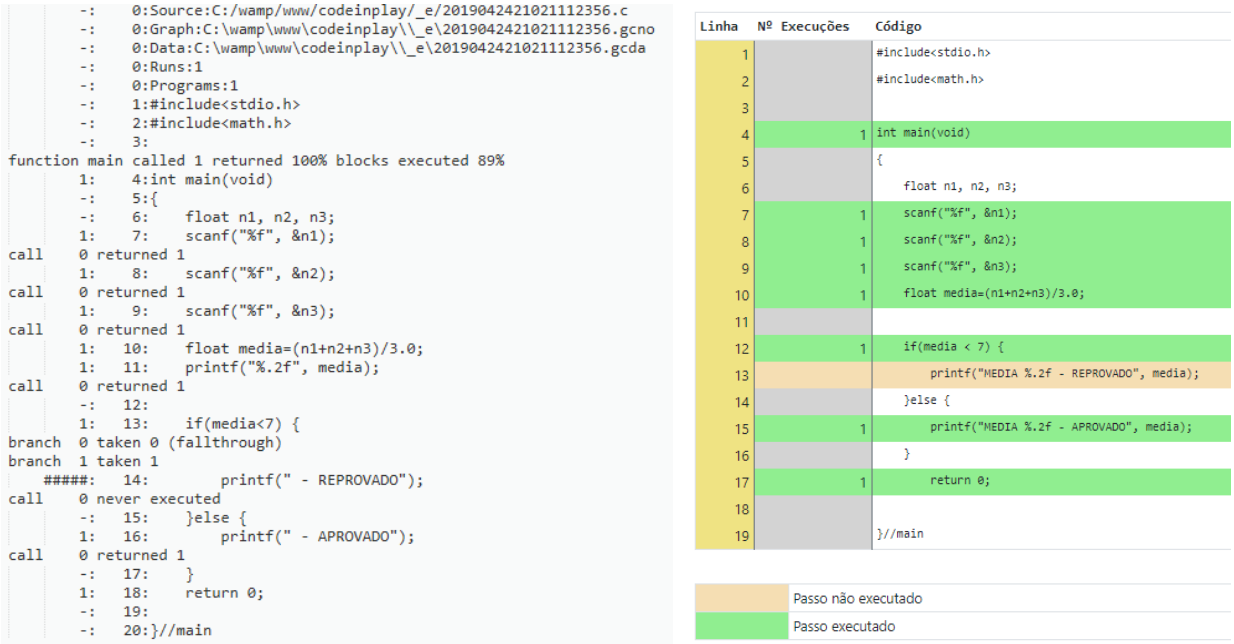
	aluno, quando a falha ocorrer mais de três vezes. Tecla de atalho para executar o Teste do professor no código: Alt+T ou Shift+Alt+T
Histórico Execução [H]	exibe na área de resultados todas as execuções gravadas para a questão, juntamente com os dados de entrada, resultado da execução, *feedback* exibido para o aluno, a opção de *download* do arquivo na extensão .c e o compartilhamento do código-fonte que produziu o histórico com um colega. Tecla de atalho para exibir o Histórico da questão: Alt+H ou Shift+Alt+H
Voltar para Questões [Q]	retorna para a listagem geral de questões disponível no menu Questões Tecla de atalho para voltar à lista de questões da ferramenta: Alt+Q ou Shift+Alt+Q
Parar Execução [P]	interrompe a execução do código Tecla de atalho para interromper a execução: Alt+P ou Shift+Alt+P

Fonte: Autora

7.5 Cobertura de linhas de código

Estando sem erros de sintaxe, o Codein'play compilará e executará o código desenvolvido pelo aluno com as entradas informadas, depois de o aluno clicar sobre o botão ‘Executar’, localizado na **Área de entrada de dados/botões** da questão, exibindo o resultado do compilador e a cobertura de linhas de código, tencionando mostrar ao estudante os caminhos que o compilador percorreu dentro do algoritmo. A cobertura de linhas de código é uma medida que apresenta a quantidade de linhas que são acionadas, sempre que um determinado conjunto de casos de teste é realizado. Para isso, o GNU *Compiler Collection* (GCC) dispõe de um pacote chamado GCOV, que se trata de uma ferramenta de análise de cobertura do código-fonte que fornece informações sobre a frequência com que um programa executa segmentos de código por instruções passo a passo.

Figura 14 - Exemplo de um arquivo .gcov para o cálculo da média



```
        -:    0:Source:C:/wamp/www/codeinplay/_e/2019042421021112356.c
        -:    0:Graph:C:\wamp\www\codeinplay\\_e\2019042421021112356.gcno
        -:    0:Data:C:\wamp\www\codeinplay\\_e\2019042421021112356.gcda
        -:    0:Runs:1
        -:    0:Programs:1
        -:    1:#include<stdio.h>
        -:    2:#include<math.h>
        -:    3:
function main called 1 returned 100% blocks executed 89%
        1:    4:int main(void)
        -:    5:{
        -:    6:    float n1, n2, n3;
        1:    7:    scanf("%f", &n1);
call    0 returned 1
        1:    8:    scanf("%f", &n2);
call    0 returned 1
        1:    9:    scanf("%f", &n3);
call    0 returned 1
        1:   10:    float media=(n1+n2+n3)/3.0;
        1:   11:    printf("%.2f", media);
call    0 returned 1
        -:   12:
        1:   13:    if(media<7) {
branch  0 taken 0 (fallthrough)
branch  1 taken 1
    #####:   14:        printf(" - REPROVADO");
call    0 never executed
        -:   15:    }else {
        1:   16:        printf(" - APROVADO");
call    0 returned 1
        -:   17:    }
        1:   18:    return 0;
        -:   19:
        -:   20:}//main
```

Linha	Nº Execuções	Código
1		#include<stdio.h>
2		#include<math.h>
3		
4	1	int main(void)
5		{
6		float n1, n2, n3;
7	1	scanf("%f", &n1);
8	1	scanf("%f", &n2);
9	1	scanf("%f", &n3);
10	1	float media=(n1+n2+n3)/3.0;
11		
12	1	if(media < 7) {
13		printf("MEDIA %.2f - REPROVADO", media);
14		}else {
15	1	printf("MEDIA %.2f - APROVADO", media);
16		}
17	1	return 0;
18		
19		}//main

Passo não executado

Passo executado

Fonte: Autora

Duas opções devem ser configuradas na chamada do compilador GCC, para habilitar a análise de cobertura durante a compilação: ***-fprofile-arcs*** e ***-ftest-covegare***. A primeira registra as estatísticas de ramificação, e a segunda salva a contagem de execuções das linhas. Após a execução do código-fonte, na linguagem de programação C, dois arquivos são criados com extensões .gcda (armazena informações de cobertura do programa executado) e .gcno (armazena informações sobre os blocos do programa e suas respectivas linhas de comando). Ao chamar o comando gcov, os arquivos .gcda e .gcno são lidos e, só então, é criado um arquivo com extensão .gcov, produzindo a listagem anotada do código-fonte, semelhante ao da Figura 14 (lado esquerdo). Cada linha do código-fonte é prefixada com a quantidade de vezes em que foi realizada, sendo as linhas que não foram executadas prefixadas com cinco caracteres cerquilha (#####). A leitura do arquivo .gcov gerará uma visualização ao aluno semelhante à Figura 14 (lado direito).

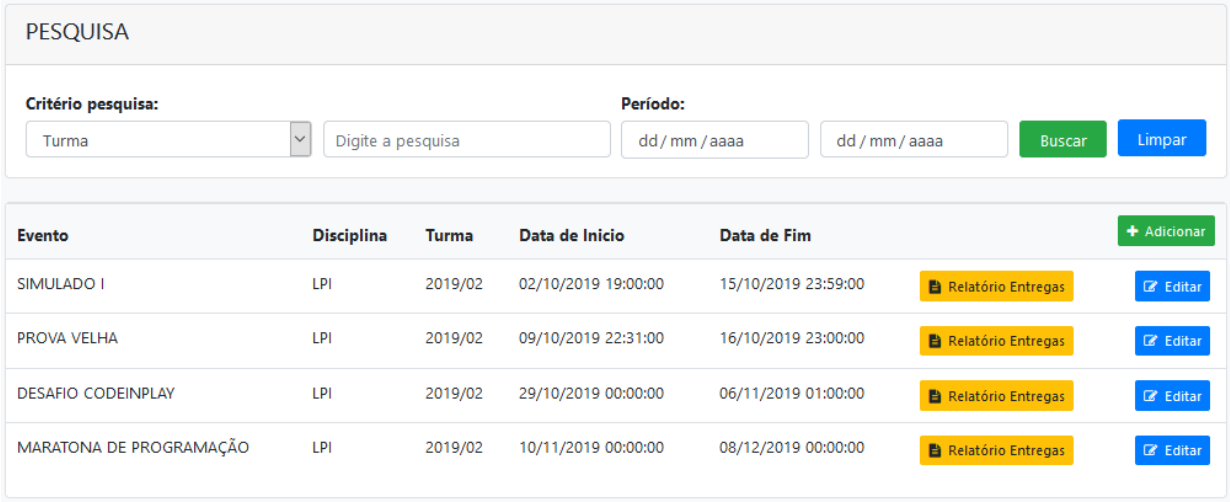
7.6 Gerenciando Eventos no Codein'play

Um evento é uma atividade proposta pelo professor à turma com data de início e fim para acontecer, tais como: simulados, lista de exercícios, provas, desafios, entre outras opções. As questões relacionadas ao evento poderão, ou não, aparecer na listagem geral de questões, e o professor poderá configurá-la para que não sejam exibidas as mensagens de *feedback* e o

botão ‘teste do professor’.

Considerando que uma questão poderá ser utilizada em mais de um evento, o sistema verificará se existe uma solução submetida ou feita pelo aluno para a questão dentro do evento. Sendo o primeiro acesso à questão, o sistema carregará o esqueleto inicial do código informado no cadastro feito pelo professor, caso contrário, o sistema carrega a última execução realizada pelo aluno dentro da área do programa.

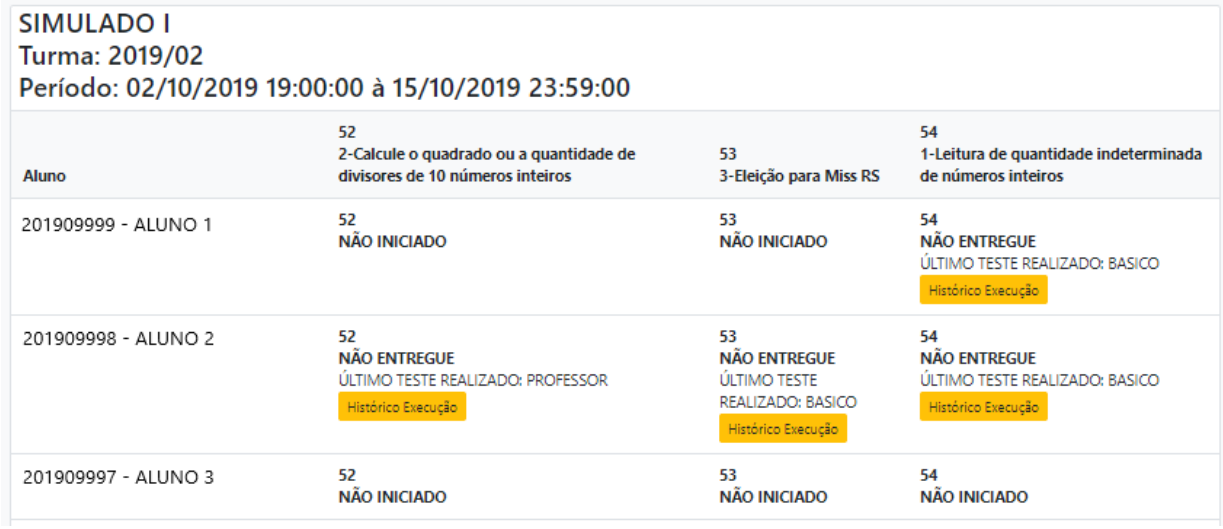
Figura 15 - Lista de Eventos



Fonte: Autora

Na tela de eventos, o docente poderá acompanhar as entregas realizadas pelos alunos por meio de um relatório. Para cada questão, este apresenta três *status*: Resolvido, Não entregue e Não iniciado. As questões não finalizadas estarão com o estado **Não Entregue** e as que não possuírem uma execução na base de dados, estarão com o estado **Não Iniciado**. Para aquelas entregues, estará indicado o estado **Resolvido** junto com a informação se contém erros ou não. Disponibilizará, também, a opção de o professor visualizar o histórico da execução e o último teste realizado no código pelo aluno.

Figura 16 - Visualizando as entregas dos alunos



SIMULADO I
Turma: 2019/02
Período: 02/10/2019 19:00:00 à 15/10/2019 23:59:00

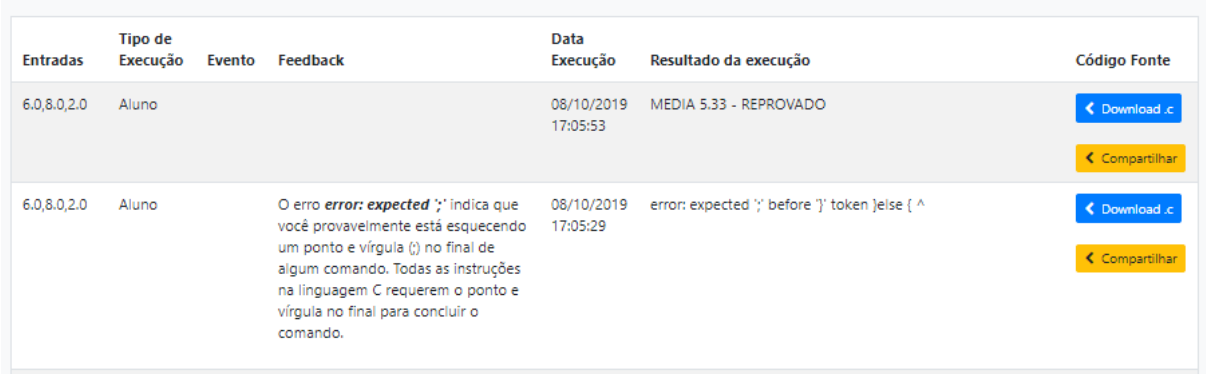
Aluno	52 2-Calcule o quadrado ou a quantidade de divisores de 10 números inteiros	53 3-Eleição para Miss RS	54 1-Leitura de quantidade indeterminada de números inteiros
201909999 - ALUNO 1	52 NÃO INICIADO	53 NÃO INICIADO	54 NÃO ENTREGUE ÚLTIMO TESTE REALIZADO: BASICO Histórico Execução
201909998 - ALUNO 2	52 NÃO ENTREGUE ÚLTIMO TESTE REALIZADO: PROFESSOR Histórico Execução	53 NÃO ENTREGUE ÚLTIMO TESTE REALIZADO: BASICO Histórico Execução	54 NÃO ENTREGUE ÚLTIMO TESTE REALIZADO: BASICO Histórico Execução
201909997 - ALUNO 3	52 NÃO INICIADO	53 NÃO INICIADO	54 NÃO INICIADO

Fonte: Autora

7.7 Histórico de Execução do código-fonte

A qualquer momento, o aluno e o professor poderão consultar o histórico de execução da questão. O sistema disponibiliza duas maneiras de visualizá-lo: na questão pelo botão 'Histórico Execução', ou pelo menu 'Histórico da Execução' (disponível somente para o professor). Na visualização do histórico da execução com o botão na questão, o aluno terá as informações: entradas testadas, data e tipo de execução (teste do aluno, do professor e básico), *feedback* exibido, resultado da execução do compilador e *download* do código-fonte na extensão .c versionado, em outras palavras, a versão do código da execução e o *link* para compartilhar a solução com um colega.

Figura 17 - Botão Histórico da execução da questão - Visão do Aluno



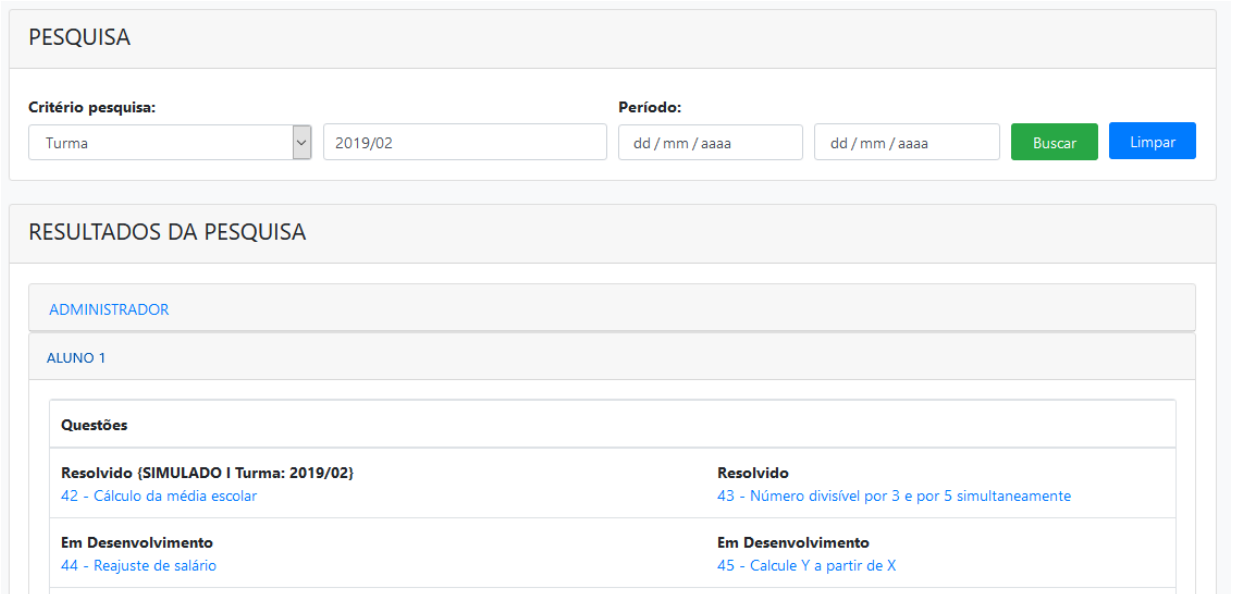
Entradas	Tipo de Execução	Evento	Feedback	Data Execução	Resultado da execução	Código Fonte
6.0,8.0,2.0	Aluno			08/10/2019 17:05:53	MEDIA 5.33 - REPROVADO	Download .c Compartilhar
6.0,8.0,2.0	Aluno		O erro ***error: expected ';'*** indica que você provavelmente está esquecendo um ponto e vírgula (;) no final de algum comando. Todas as instruções na linguagem C requerem o ponto e vírgula no final para concluir o comando.	08/10/2019 17:05:29	error: expected ';' before '}' token }else { ^	Download .c Compartilhar

Fonte: Autora

Na visão do mestre, o sistema apresentará alguns filtros adicionais, para realizar uma

busca mais assertiva das submissões realizadas pelos alunos da turma. No resultado da pesquisa do histórico das execuções, aparecerá, em cada questão, o estado em que ela se encontra. O estado 'Resolvido' indica que o estudante Enviou e Finalizou a solução, informando ao professor que não pretende realizar nenhum tipo de alteração. 'Em Desenvolvimento' mostra que o estudante executou ou testou o código com os casos de teste do professor, pelo menos uma vez, e que pretende retornar ao código-fonte para continuar o desenvolvimento, ou seja, a solução ainda não está totalmente resolvida. Ao clicar sobre o título da questão, o professor terá acesso ao histórico da execução com as seguintes informações: entradas utilizadas, teste executado (professor, básico e aluno), data da execução, resultado obtido e *feedback* que foi exibido ao aluno, juntamente com a opção de visualizar o código-fonte que gerou a linha no banco de dados e que, posteriormente, compôs o histórico.

Figura 18 - Histórico de Execução - Visão do Professor



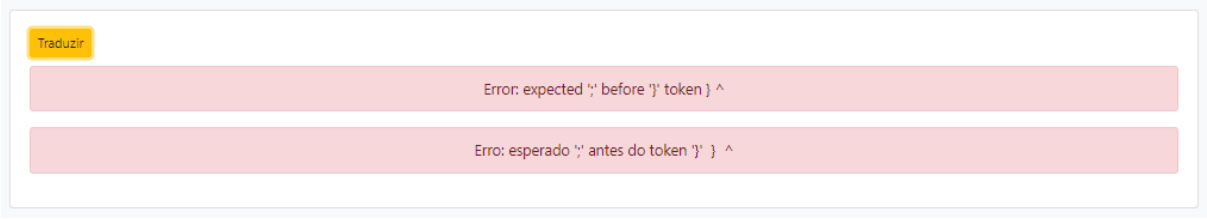
Fonte: Autora

7.8 As mensagens do compilador no Codein'play

Sabe-se que umas das dificuldades relatadas pelos alunos iniciantes em computação é o problema na interpretação das mensagens exibidas pelo compilador, por estarem no idioma inglês (GOMES, 2015b). Em função disso, no Codein'play, as mensagens do compilador serão exibidas tanto no idioma inglês quanto em português. Assim, o aluno começa a se familiarizar com os termos no idioma americano. Contudo, para viabilizar a transposição

entre os idiomas, foi utilizada a *Application Programming Interface* (API) de tradução PHP Google Translate Free[22], que foi instalada via Composer para dentro do *framework* Codeigniter do projeto.

Figura 19 - Traduzindo os erros do compilador



Fonte: Autora

7.9 Os *feedbacks* no Codein'play

O *feedback* é um elemento importante na aprendizagem, pois, quando feito no momento certo, reforça os conceitos estudados e auxilia o aluno no entendimento de seus erros. Nesse sentido, o Codein'play apresenta mensagens de *feedbacks* instantaneamente, no momento em que o sistema identifica que o aluno está tentando resolver a questão e está encontrando dificuldades - Figura 20. Estas são capturadas pela ocorrência de um erro sendo repetido mais de três vezes na questão.

O cadastro de *feedback* poderá ser feito pelo professor sob duas formas: pela associação de uma mensagem do compilador, ou mediante a vinculação com um caso de teste cadastrado na questão que será solucionada e executada pelo aluno.

Para vincular uma mensagem do compilador a um *feedback*, antes, o professor deverá cadastrar a raiz do erro em uma tela específica destinada aos erros do compilador, pois existem infinitas possibilidades de o mesmo erro aparecer em situações diferentes. Por exemplo: o erro de sintaxe ***error: expected ';'*** ocorre sempre que o compilador identifica a ausência do ponto e vírgula (;) no final de um comando. A mensagem deste erro varia muito em função do comando que antecede a falta dele, que pode ser após um *return*, *printf*, *scanf*, atribuição de valor a uma variável, entre outras situações. Após o cadastro da raiz do erro, quando ocorrer a falha, o sistema verificará se não existe uma raiz do erro cadastrada na base de dados com alguma recomendação. Se houver, exibe-a ao aluno; caso contrário, registra o erro que não possui um *feedback* associado e alerta o professor, na tela principal do sistema,

22 <https://github.com/statickidz/php-google-translate-free>

que existe um erro que precisa ser cadastrado.

Para associar um caso de teste a um *feedback*, primeiramente, o docente deverá cadastrar o caso de teste na questão, e, após, em tela específica, relacioná-lo a um *feedback* explicando a falha do resultado. Em ambos os casos, o professor deverá seguir a estratégia de *feedback* apresentada na seção 6, e, baseando-se na recomendação apresentada, o aluno avalia qual problema necessita ser corrigido em seu código. Todos os *feedbacks* exibidos são registrados em banco de dados para posterior consulta no histórico da execução do código-fonte.

Figura 20 - Exibindo *feedback* ao aluno

Fonte: Autora

7.10 Cadastro de Comandos Proibidos

Essa funcionalidade é muito importante dentro do ambiente para não permitir uma fragilidade do sistema. Sempre que um comando possa ser identificado como perigoso, é necessário adicioná-lo na lista de comandos proibidos de serem compilados e executados pela ferramenta. O sistema verificará, antes de executar o código do aluno, se não existe um comando ou biblioteca proibidos no algoritmo. Caso encontre, comenta a linha, não

permitindo sua execução.

7.11 Cadastro de Turmas

O professor poderá cadastrar as turmas com os seus alunos para utilizar em eventos, mas esses precisam estar cadastrados na base de usuários para serem carregados no cadastro da turma.

7.12 Relatórios

Como meio de acompanhar o aluno, a ferramenta disponibilizará um conjunto de relatórios para o professor observar o progresso dos estudantes. Fazem parte desse conjunto:

- lista de submissões/não submissões em eventos dos alunos matriculados na turma;
- quantidade de erros ocorridos por tipo de erro;
- quantidade de acertos por tentativas por questão e por aluno, separados por até três tentativas, quatro tentativas e mais do que quatro; e
- relatório com a quantidade de execuções, de acertos e erros por aluno e por questão.

Para o aluno, será disponibilizado um relatório com o seu desempenho geral por tópico e por questão para que possa ver onde estão as suas dificuldades. Esta seção se propôs apresentar a ferramenta Codein'play desenvolvida para validar a estratégia de *feedback* apresentada na seção 6. O Resultado e a Análise dos Dados desta pesquisa são apresentados na próxima seção.

8 RESULTADOS E ANÁLISES DOS DADOS

Esta seção apresenta os resultados das pesquisas e das observações realizadas nas turmas durante o estudo. A investigação utilizou uma abordagem quantitativa para análise dos dados, utilizando a avaliação das respostas dos questionários; e uma abordagem qualitativa mediante observação em sala de aula. A seção está dividida em quatro subseções: a primeira apresenta a análise das dificuldades dos alunos identificadas nas respostas do questionário do **Apêndice A**; a segunda destaca as observações realizadas em sala de aula; a terceira traz a análise dos códigos-fontes das provas I e II entregues pelos alunos; e a quarta expõe a análise das respostas do questionário do **Apêndice C**, sobre o uso da ferramenta Codein'play.

8.1 As dificuldades dos alunos do IFRS na disciplina LPI curso STSI

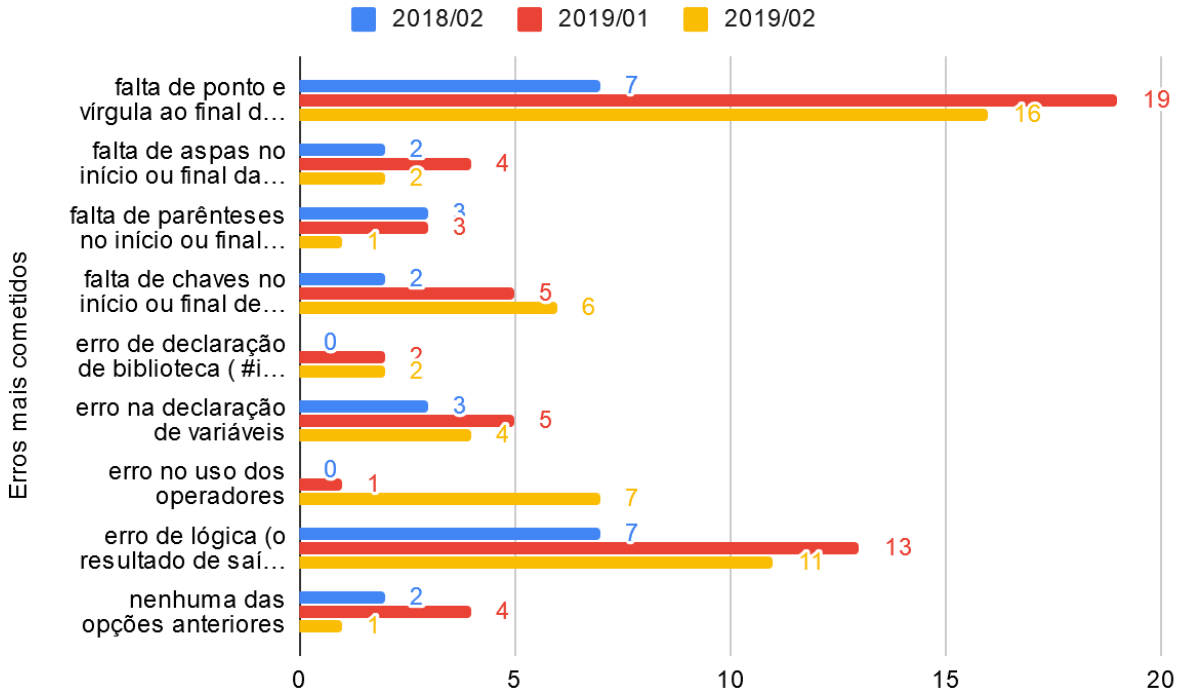
Foi aplicado um questionário (**Apêndice A**) na turma de Linguagem de Programação I do IFRS, campus Porto Alegre nos semestres 2018/02 (manhã), 2019/01 (noite) e 2019/02 (manhã), abordando a percepção dos alunos sobre os erros mais cometidos durante as aulas em programação de computadores, juntamente com o seu sentimento perante o erro e as dificuldades encontradas na aprendizagem da disciplina.

Participaram da primeira pesquisa 12 alunos; na segunda, 30; e, na terceira, 22, tendo o questionário sido aplicado em momentos diferentes. A primeira turma, respondeu no mês de outubro de 2018, antes da segunda avaliação. As segunda e terceira turmas tiveram os questionários aplicados logo após a primeira avaliação, no mês de junho de 2019 e outubro de 2019, nesta ordem. O primeiro questionário, aplicado ao primeiro grupo, foi disponibilizado *online* para os alunos responderem de maneira voluntária. A pesquisa foi realizada após a desistência de parte dos alunos da turma, que tinha inicialmente cerca de 40 estudantes. Com os segundo e terceiro grupos, os alunos responderam o questionário impresso e foram convidados pelo professor a participar da pesquisa, sendo de comum acordo a todos os presentes na turma.

Dos discentes pesquisados em 2018/02, nove eram do sexo masculino e três do sexo feminino, com faixa etária entre 18 e 41 anos. Na turma de 2019/01, 22 eram do sexo masculino e oito do feminino, com faixa etária entre 18 e 43 anos. Na turma de 2019/02, 14 eram do sexo masculino e oito do feminino, na faixa etária entre 19 e 50 anos.

As respostas da pesquisa estão descritas no **Apêndice B**; e, em linhas gerais, os alunos relataram que possuem muita dificuldade em entender o enunciado do exercício e iniciar/propor uma solução algorítmica para a questão. Apontaram algumas dificuldades com a sintaxe dos comandos da linguagem de programação C, principalmente com uso dos *tokens* ponto e vírgula (;) e a abertura/fechamento das chaves ({}), junto com o erro lógico e problemas na declaração das variáveis. Há relatos de confusão na utilização dos operadores (aritméticos, lógicos e relacionais), conforme mostra a Figura 21.

Figura 21 - Os erros mais cometidos na percepção dos alunos



Fonte: Resultado questionário do Apêndice A

Outro fator interessante foi que, mesmo ocorrendo os erros durante a resolução dos exercícios, a maioria dos alunos se sente encorajada a encontrar os problemas e corrigi-los, tornando o erro, de certa forma, motivacional, o que pode ser observado na narrativa do aluno: "*declarar a variável como =0 me frustra um pouco, eu sei que para evitar lixo e tudo mais, porem n entendo o porquê disso sendo que programas anteriores não precisavam, pele menos melhor do que usar getch() para limpar o lixo na memória, meus erros de logica, são frutos de minha própria incompetência devo buscar mais conhecimento para corrigi-los, estes erros não me desanimam só me deixam com raiva e desafiado a resolver mas o lixo de memória ainda não compreendi do que se trata*.". A grande maioria dos participantes acha

importante analisar os erros cometidos, tendo se tornado uma prática investigá-los sempre que ocorrem. Mas, a preferência para a reflexão sobre eles costuma ser a sós ou com um colega mais experiente. A opção de procurar o professor para compartilhar o problema, não é muito aplicada, segundo os alunos.

Quanto à ocorrência e ao entendimento da mensagem de erro, a maioria dos estudantes possui como prática ler a mensagem antes de corrigir a falha no código; todavia, existe uma parcela de discentes que acha a mensagem confusa, com apresentação e organização da mensagem de difícil compreensão; e a mensagem estar em inglês dificulta o entendimento. Dentre as narrativas dos questionários do **Apêndice A**, surgiram alguns comentários sobre a ferramenta DevC++, adotada na disciplina, em torno da formatação das mensagens exibidas pelo compilador e da sintaxe dos comandos: "*DevC++ é um compilador que não exibe mensagens de erro claras; preciso revisar conceitos de matemática; a sintaxe não é muito amigável*." e "*DevC++ pouco intuitivo*", apesar da maioria dos alunos informar que compreende perfeitamente o conteúdo das mensagens de erro do compilador - **Q6 do Apêndice B**.

Com base nisso, foi aplicado o questionário do **Apêndice D** para obter mais informações a respeito das ferramentas IDEs utilizadas em sala de aula. 96,4% dos alunos utilizam o DevC++, para resolver os exercícios propostos na disciplina. Destes, 75% sinalizaram que o DevC++ é uma ferramenta simples de utilizar e que facilita na resolução dos exercícios. 78% responderam que ela ajuda na localização dos erros, empatando com 42,9% cada, alguns alunos acham que não ajuda a compreender e conhecer o significado do erro, e outros acham que sim. Porém, para conseguir compreender o significado do erro o estudante precisa ser mais experiente, conforme o relato do aluno: "*Ajuda a compreender, mas não para leigos ou quem ainda tem dificuldade*".

Com relação aos erros, alguns alunos confirmam dificuldades na sintaxe, entendimento/função dos comandos e no problema de identificar o erro de lógica no código, conforme relatam: "*Tenho dificuldades em escrever as sintaxes corretas e também tenho dificuldade na lógica dos exercícios*."; "*A minha maior dificuldade é diferenciar um bug do dev com a lógica.*"; "*Construções das operações lógicas"; "Dificuldade com for, while que sinto que estou parada nesse conteúdo.*".

Em outros casos, fica evidente a dificuldade na resolução dos exercícios, em função da base fraca em matemática, levantada por Borges (2000) e Giraffa (2003), mediante a narrativa do aluno: "*Tenho apenas dificuldade quando a questão faz uso de perguntas matemáticas, pois não lembro, por fazer muitos anos que não uso*.". Apareceu, também, como uma barreira na aprendizagem, a falta de disponibilidade de tempo para se dedicar aos estudos: "*A dificuldade maior é o tempo para ficar na frente do PC treinando*."; e "*Pratico pouco, na primeira dificuldade desanimo*."; e a ausência de motivação para fazer os exercícios propostos no AVA Moodle da disciplina, como expões o aluno: "*Não tenho motivação para fazer os exercícios*." o que dificulta o seu aprendizado.

Conhecendo as dificuldades na aprendizagem em programação de computadores relatados nos trabalhos citados anteriormente e na pesquisa aplicada nas turmas 2018/02, 2019/01 e 2019/02, organizou-se a análise dos códigos-fonte das provas (I e II) produzidos pelos estudantes da disciplina de Linguagem de Programação I, com a finalidade de identificar os erros cometidos pelos discentes e mapeá-los. O processo utilizado será apresentado na subseção 8.3.

8.2 Observações em sala de aula LPI e as atividades no Codein'play

Durante os semestres de 2019/01(noite) e 2019/02 (manhã), as aulas foram acompanhadas pela pesquisadora com o intuito de observar o desenvolvimento dos estudantes e identificar as suas dificuldades no aprendizado da programação da disciplina LPI. No decorrer das aulas, pode-se verificar que alguns alunos não conseguiam acompanhar as explicações e, muitas vezes, se omitiam, permitindo o avanço do conteúdo.

Os estudantes tinham acesso ao conteúdo apresentado em aula, além de materiais extras, como atividades e fóruns de discussão, por meio da página da disciplina disponibilizada no AVA Moodle. Assim, em cada aula, o professor liberava o acesso aos slides que seriam trabalhados, os exercícios práticos de programação e algumas respostas de atividades da aula anterior. Logo, as aulas eram divididas em três momentos: correção de exercícios da aula anterior, ensino do novo conteúdo e momento para esclarecer dúvidas. Na primeira parte, o professor realizava a correção de alguns desses exercícios e sanava dúvidas dos estudantes, quando os mesmos pediam, e, durante esse momento, pode-se observar que poucos alunos faziam questionamentos, e o docente tinha que pedir constantemente a

participação deles. Na segunda parte, o professor passava o novo conteúdo, com alguns exemplos práticos (códigos e trechos de códigos). E, por fim, era dado o espaço para esclarecer as dúvidas e contar com o auxílio do professor. Mas, a maioria ia embora, e os que ficavam dificilmente pediam ajuda ou resolviam suas dúvidas. Foi observada a falta de interesse de alguns estudantes na disciplina durante a exposição do conteúdo e, quando os exercícios eram corrigidos pelo professor, muitos ficavam acessando, pelo computador do laboratório de informática, conteúdo no *facebook* e *whatsapp*, outros acessavam sites diversos, e alguns ficavam jogando. Toda semana o professor publicava uma lista de exercícios, contemplando o conteúdo visto em aula, com prazo de dez dias para a entrega da resolução.

Ao longo do semestre de 2019/2, foram propostas atividades gamificadas (jogo dos dez erros sobre código-fonte, jogo da memória, quiz sobre a linguagem de programação C, entre outros) pelo mestrando Márcio Fabiano de Carvalho, contemplando o conteúdo aprendido. A maioria dos estudantes participava deste momento para esclarecer as dúvidas sobre o conteúdo abordado.

Além das listas de exercícios oferecidas no AVA Moodle, foram disponibilizadas algumas questões no Codein'play para resolução extraclasse. Constatou-se uma baixa adesão à resolução dos exercícios utilizando o juiz *online*, na turma de 2019/2, em função da não obrigatoriedade de sua utilização. Nesse semestre, foram realizadas três atividades no juiz *online* em momentos distintos: Simulado, Desafio Codein'play e Maratona de Programação, que pontuaram no Terceiro Torneio Mágico, trabalho de Mestrado em Informática na Educação do mestrando Márcio Fabiano de Carvalho. As questões utilizadas nas três atividades estão no **Apêndice H**.

O **Simulado** ocorreu de 2 a 15 de outubro de 2019 e foi uma atividade individual, na qual o aluno precisava resolver três questões, cujos exercícios trabalhavam os conteúdos dados em aula antes da primeira prova, tais como variáveis e tipos de dados, operadores, estrutura de decisão e de controle, e que foi iniciado em sala de aula e finalizado em casa. O objetivo desta prática foi revisar os conteúdos e exercitá-los antes da primeira avaliação. O simulado foi a primeira atividade utilizando o juiz *online* em sala de aula e participaram **24** alunos. De acordo com os dados retirados da base de dados do juiz *online*, foram **695 execuções**, com **491 (71%)** compilações sem sucesso, sendo: **50** com erros de sintaxe; **90**

com *warnings*, **101** com erros de sintaxe e *warnings*, **157** com erros lógicos e **93** execuções entraram em *loop* infinito. Em função da quantidade elevada de *loops* infinitos, ocorreram problemas de lentidão no uso da ferramenta. Para corrigir esse contratempo, foi necessário reduzir o percentual de processamento ocupado na CPU, e mudar a prioridade para cada compilação e execução do código do aluno.

Além disso, implementou-se uma rotina residente no servidor que roda a cada 15 segundos, a fim de remover os processos travados. Esses procedimentos minimizaram o problema de lentidão para o Desafio Codein'play. Como a atividade iniciou em sala de aula, notou-se que os alunos ficaram um pouco confusos para autenticar na ferramenta e para chegar até a lista de questões do Simulado, conseguindo depois de uma breve explanação da pesquisadora. Após a lentidão no uso da ferramenta, alguns alunos abandonaram o ambiente, e resolveram as três questões do Simulado no DevC++, disponibilizando a solução no AVA Moodle. Percebeu-se, também, que a funcionalidade de teste automático foi bem aceito pelos estudantes e, em função disso, uma parcela de alunos resolvia as questões no DevC++ e após compilar, copiava a solução e colava no Codein'play, com a intenção de aplicar os testes cadastrados pelo professor para o exercício, gerando histórico para posterior consulta. No entanto, o que mais chamou atenção dos estudantes foi o resultado imediato quanto à corretude ou não do código após a execução.

O **Desafio Codein'play** ocorreu de 30 de outubro a 6 de novembro de 2019 e foi uma atividade em grupo de até três integrantes. As equipes precisavam fazer a inscrição em um formulário criado no *Forms Group* para a atividade, era preciso informar o nome da equipe e seus participantes. O Desafio foi composto de dois problemas de fácil solução algorítmica, mas que demandavam entender o enunciado e traçar uma estratégia para a resolução. Para esses dois problemas, as equipes precisavam criar o algoritmo solução, realizar o planejamento de como resolver as questões - PLEA (**Anexo C**) -, e registrar em vídeo a explicação de como as equipes resolveram as questões, postando o *link* no tópico do fórum aberto para o evento no AVA Moodle utilizado na disciplina. Este evento foi premiado até o 3º lugar: para o 1º lugar, a premiação foi a camiseta Codein'play, os 2º e 3º lugares ganharam chocolates da marca Cacau Show. O evento obteve **seis inscrições** e somente **quatro equipes** participaram do Desafio, realizando todas as tarefas propostas. Houve uma inscrição que continha apenas um integrante, que entregou os códigos e o PLEA, mas não gravou os vídeos

explicando a estratégia da solução por não se sentir confortável com essa atividade. Foi solicitado que apenas um integrante das equipes entregasse a solução, o que de fato fizeram. Apesar deste evento prever uma semana para a entrega final, as equipes finalizaram as atividades em quatro horas, iniciando à meia-noite do dia 30 de outubro de 2019. O objetivo do Desafio foi provocar os alunos a pensarem e planejarem em uma solução colaborativamente entre os integrantes, ao mesmo tempo em que estimulava a competição entre equipes. Contudo, não houve restrição de uso de tecnologia de comunicação, deixando os times livres para escolher a ferramenta de comunicação que melhor os ajudasse nas atividades, e a grande maioria utilizou o *whatsapp* como recurso de troca de mensagens entre eles. A única restrição imposta foi o uso da ferramenta Codein'play para resolver os dois problemas e o uso do AVA Moodle para responder o PLEA e para fazer a postagem do *link* do vídeo. Conforme os dados retirados da base de dados do juiz *online*, foram **253 execuções**, com **87 (34%)** compilações com falhas, sendo: **24** com erros de sintaxe; **19** com *warnings*; **três** com erros de sintaxe e *warnings*; **39** com erros lógicos e **duas** execuções que entraram em *loop* infinito.

A última atividade proposta na disciplina foi a **Maratona de Programação** que aconteceu de 10 de novembro a 3 de dezembro de 2019, quando foi solicitado aos alunos que não atingiram a média 7,0 na primeira prova (nove alunos), que a fizesse para acrescentar pontos à nota, apesar de estar acessível a todos os alunos. A cada semana eram disponibilizadas cinco questões com prazo de resolução para os próximos sete dias. O primeiro conjunto de questões abordou o uso de variáveis e tipos de dados (03), manipulação de strings (01) e estruturas de Decisão (01). O segundo conjunto contemplou o conhecimento em funções (01), matrizes e vetores (01), estrutura de decisão (02) e estrutura de controle (01). E, por último, foram publicados exercícios que trataram sobre ponteiros (02), matrizes e vetores (01), estrutura de controle (01) e estrutura de decisão (01), totalizando 15 questões diversificadas, que exigiam conhecimento de todo o conteúdo visto em sala de aula até a segunda prova.

O objetivo da Maratona de Programação foi revisar os conteúdos e exercitá-los, de maneira que servisse como um grande simulado antes da segunda prova. Participaram, inicialmente, da Maratona 13 alunos e, segundo os dados retirados da base de dados do Codein'play, foram **1.819 execuções**, com **1.412 (78%)** compilações com erros, sendo: **212**

com erros de sintaxe; **379** com *warnings*; **299** com erros de sintaxe e *warnings*; **497** com erros lógicos; e **25** execuções que entraram em *loop* infinito. Dos estudantes participantes da Maratona, apenas **seis** finalizaram as 15 questões.

Descobre-se que sobre as **1.990 (71%)** execuções com falhas ocorridas entre as três atividades propostas no juiz *online*, em um total de **2.767 execuções**, ocorreram **2.692** alertas (*warning*) e **2.593** erros (*error*), ou seja, cada compilação com falha continha, em média, um alerta e um erro. Um cenário quase semelhante com os estudos de Gomes et al. (2015a), que concluíram que cada compilação errada continha em média um alerta e dois erros. Da verificação dos resultados obtidos pela ferramenta Codein'play, foi possível identificar as seguintes falhas:

- chamada de comando inexistente ou biblioteca não declarada, gerando erro de linkagem;
- falta de fechamento de parênteses e/ou chaves em estruturas como *if().. else*;
- uso equivocado do *else* (o comando *if()* não estava iniciando a condição);
- falta de ponto e vírgula no final do comando;
- uso incorreto dos operadores relacionais;
- erro na leitura de variáveis com *scanf()* (uso equivocado do &);
- uso incorreto das funções *printf() e fgets(),* como por exemplo: incompatibilidade de formatação com as variáveis do programa, falta de argumentos e quantidade de parênteses insuficiente para operações;
- uso inadequado de funções; e
- erros por não declaração, redeclaração de variáveis ou conflito de tipos de dados.

A análise dos dados permitiu confirmar que a grande maioria dos erros cometidos pelos alunos são erros de sintaxe da linguagem C. Gomes et al. (2015a) identificaram que a ausência do ponto e vírgula é o erro mais cometido pelos alunos durante sua pesquisa, no entanto, de acordo com a análise das execuções na base de dados do Codein'play, percebeu-se uma grande quantidade de erros relacionados à linkagem, ou seja, o uso de comandos inexistentes nas bibliotecas declaradas ou a ausência de sua declaração, seguido de erros gerados pelo uso inadequado de variáveis. No Quadro 18, são apresentados os dez erros mais cometidos pelos alunos na ordem de quantidade de ocorrência.

Quadro 18 - Os 10 erros mais cometidos pelos alunos registrados na ferramenta

	Erros	**Tópico**
01	error: ld returned 1 exit status	Linkagem
02	error: '<variável>' undeclared error: array type has incomplete element type error: assignment to expression with array type error: conflicting types error: invalid type argument of unary '*' (have 'int')	Variáveis
03	error: expected ';' before ...	Ponto e Vírgula
04	error: expected '(' before <variável> error: 'else' without a previous 'if'	sintaxe *if()*
05	error: expected ')' before ';' token printf error: expected '=', ',', ';', 'asm' or '__attribute__' before 'printf' error: expected declaration specifiers or '...' before string constant printf error: expected expression before ')' token printf error: lvalue required as left operand of assignment printf	sintaxe *printf()*
06	error: expected identifier or '(' before '{' error: invalid use of void expression error: too few arguments to function	Funções
07	error: expected expression before ‘token’ error: lvalue required as left operand of assignment	Operadores Relacionais
08	error: expected expression before '%' token	sintaxe *scanf()*
09	error: expected declaration or statement at end of input }	Chaves
10	error: case label not within a switch statement	sintaxe *switch..case*

Fonte: Base de dados do Codein’play

Foi possível identificar que a média de tentativas de chegar a uma solução correta das 20 questões foi de sete execuções. Calculou-se, também, a média de tentativas até o acerto por evento (atividade), chegando aos seguintes resultados: Maratona (8,64), Simulado (6,05) e Desafio (3,47). Com essas médias, foi possível verificar que uma parcela dos alunos teve como recurso o *feedback* informativo, que é exibido quando o erro ocorre mais de três vezes e, mesmo assim, aparentemente, continuavam errando. Em função disso, levanta-se um questionamento com o resultado desta pesquisa: será que os alunos possuem uma ansiedade tão grande para obter respostas imediatas (acertaram/erraram) e estão menos preocupados com os processos que levam ao acerto ou ao erro? Durante o acompanhamento de uma parte do Simulado em sala de aula, constatou-se que os alunos estavam mais interessados no *feedback* avaliativo do que no *feedback* informativo. A função do *feedback* avaliativo é indicar ao aluno se o resultado esperado coincide com o resultado obtido pelo compilador, ou seja, se errou ou acertou.

8.3 Análise dos códigos-fontes dos alunos de LPI e os erros mapeados

Para selecionar os códigos a serem analisados, optou-se em coletar os códigos-fonte das provas com notas inferiores a 7.0, ficando distribuídos como expostos no Quadro 19.

Quadro 19 - Quantidade de Códigos das provas I e II

Semestre	**Prova I** **Códigos Analisados**	**Prova II** **Códigos Analisados**	**Total** **Códigos Analisados**
2018/02	28 (11 alunos)	18 (7 alunos)	46
2019/01	27 (9 alunos)	15 (6 alunos)	42
2019/02	17 (7 alunos)	9 (3 alunos)	26
Totais:	**72**	**42**	**114**

Fonte: Autora

As provas eram compostas por três questões, abordando, no primeiro teste, o uso das estruturas de decisão e controle, variáveis, entrada e saída de dados; e, na segunda prova, testaram os conhecimentos dos alunos na utilização de estruturas de decisão e controle, matrizes, ponteiros, variáveis, bem como entrada/saída de dados.

O procedimento de análise iniciou com a execução do programa, sem avaliar sua corretude; dos **114** códigos entregues e executados, **38** não compilaram e os erros apresentados variam desde uso incorreto de variáveis (erro de declaração, inicialização e utilização), até a falta do ponto e vírgula após o comando. Na utilização de estrutura de decisão, por exemplo, chama a atenção a dificuldade na visualização dos pares *if..else*, sendo utilizado o comando *else* solto sem atrelar ao comando *if*. Observa-se, também, o uso inadequado das chaves {} na composição do bloco de comando. Outro ponto interessante, no mesmo trecho de código, é o uso exagerado do ponto e vírgula, onde o aluno 'reafirma' sua utilização antes e depois do comando.

Com as matrizes, houve o esquecimento de associar o índice à variável da matriz. Vê-se o mesmo equívoco na utilização do operador módulo % (cuja variável que se pretende comparar não está presente), quando o aluno tenta verificar se o valor armazenado na **LinhaxColuna** da matriz é um número ímpar. Sobre o uso de função, a dificuldade está em entender como defini-la. Alguns alunos criam funções dentro da função main(). Outra questão importante está no uso incorreto das estruturas de controle. Dos códigos analisados, alguns entraram em *loop* infinito em função do estudante esquecer de incrementar a variável de controle da iteração ou declarar uma variável e não a utilizar na iteração.

Da análise dos resultados coletados foi possível identificar as seguintes dificuldades:

- uso incorreto do comando *for(), do..while() e while()*, resultando em *loop* infinito;
- uso inadequado/falta do ponto e vírgula (;) e das chaves ({}) nos fechamentos dos blocos;
- uso incorreto das funções *printf()* e *scanf()*;
- erros por não declaração, redeclaração, inicialização e não saber utilizar as variáveis;
- erro na declaração de biblioteca no início do programa;
- uso incorreto do comentário de linha //;
- uso incorreto dos operadores; e
- códigos sem indentação, dificultando a leitura.

A análise dos dados permitiu constatar que a grande maioria dos erros cometidos pelos alunos foram de sintaxe da linguagem C. Todavia, a principal dificuldade encontrada pelos discentes, mostrada nos códigos, foi entender o enunciado da questão e propor uma solução que atendesse. Muitos códigos entregues, com nota inferior a 7.0, não tinham relação com o enunciado da questão, dando a impressão de que o aluno não sabia como iniciar a resolução, e 'entregar qualquer coisa' era melhor do que não entregar. Assim, os dados coletados nos questionários e na análise dos códigos dos alunos são confirmados com os relatos nos trabalhos de (GOMES et al., 2015a) e (BOSSE & GEROSA, 2016). Percebe-se uma correlação entre as respostas dos questionários do **Apêndice A** com os erros encontrados nos códigos-fontes avaliados.

Desta análise, chega-se a um conjunto de **24 erros/alertas** identificados e mapeados, que serviram de base inicial ao juiz *online*. Agregado a isso, foram reconhecidos **17 erros/alertas** e possíveis causas divulgadas nos sites da Digital Fanatics[23] (GCC *Error Messages*) e Linuxtopia[24] (*Compiler error messages*), chegando a um total de **41 erros/alertas** iniciais com seus *feedbacks* (**Apêndice E**). O juiz *online* conta com o recurso de registrar os erros e alertas, que ocorrem durante a execução das questões e que não possuem um *feedback* associado. Atualmente, o Codein'play possui **89 erros/alertas** mapeados com

23 <http://digitalfanatics.org/resources/gcc-error-messages/> acesso em: 15 ago. 2019

24 <https://www.linuxtopia.org/online_books/an_introduction_to_gcc/gccintro_94.html> acesso em: 15 ago. 2019

seus *feedbacks*, e a Maratona de Programação foi a atividade em que mais erros de sintaxe e alertas foram capturados, com **25** novas mensagens cadastradas. Isso pode ter ocorrido em função da quantidade de questões disponibilizadas e devido ao nível de dificuldade dos exercícios. O Simulado gerou **23** novas mensagens e o desafio não gerou mensagens extras.

8.4 Análise das avaliações realizadas pelos alunos da ferramenta Codein'play

A avaliação da ferramenta ocorreu em dois momentos: outubro de 2019, após a utilização do Codein'play com o Simulado, e em novembro de 2019, durante a Maratona de Programação. O questionário objetivou avaliar as características da ferramenta como: usabilidade, conteúdo das recomendações dos *feedbacks* informativos, cobertura do código, enunciado das questões, execução automática do código e automatização dos testes, mensagens de erro em português (tradução) e, principalmente, se a ferramenta auxiliou no entendimento sobre o erro.

Quanto à usabilidade e à facilidade na seleção dos exercícios de acordo com o conhecimento do aluno, mais de 70% dos estudantes concordaram que a interface da ferramenta é fácil de utilizar, o que confirma com o relato de um aluno: "*Achei uma ferramenta simples e intuitiva. Gostaria de algo assim disponível para fazer exercícios e praticar*". No entanto, houve uma fala sobre a posição confusa dos botões na questão "*coloração e posicionamento dos botões um pouco confusa*"; a necessidade de aumento de contraste da tela para realçar os elementos da interface: "*Aumentar o contraste das cores para identificar os elementos*." e a possibilidade de uso de teclas de atalho para as funcionalidades disponibilizadas nos botões, na área de entrada de dados/botões: "*Atalhos de teclado ajudariam :)*" e "*Senti falta de poder usar instruções condicionais sem o uso das chaves e das teclas de atalho.*". As teclas de atalho foram implementadas após a atividade Simulado.

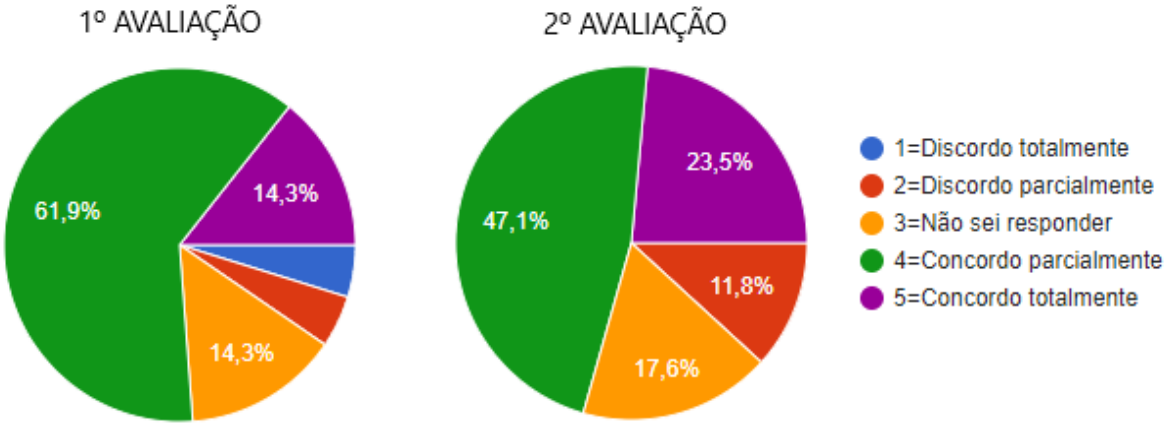
Com relação a redação dos enunciados, houve uma preocupação em deixar o texto o mais claro possível, não dando margem a diversas interpretações sobre o que deveria ser feito. Conforme os dados coletados nas duas avaliações, mais de 76% das respostas dos alunos concordam que os enunciados ajudaram no entendimento do que precisava ser feito, o que é reforçado com o relato de um aluno "*O enunciado dos exercícios também são claros e ajudam na resolução*".

Além disso, percebeu-se que os estudantes gostaram de três funcionalidades: execução

automática do código, aplicando o teste de caixa-preta; exibição da cobertura do código, mostrando por onde o compilador passou no código-fonte, durante a execução dos testes; e o *feedback* avaliativo, o qual informa se o código está correto ou não. Assim, o aluno conseguia visualizar instantaneamente se a solução estava de acordo ou não.

Com relação aos *feedbacks*, com 73,4% - primeira avaliação: 76,2% (61,9% e 14,3%) e segunda avaliação: 70,6% (47,1% e 23,5%) - dos estudantes responderam que as recomendações auxiliaram no entendimento e conhecimento sobre o erro. De acordo com eles, o *feedback* informativo ajudou a identificar qual era o erro e onde provavelmente ocorria, no entanto, o *feedback* avaliativo ajudou a enxergar se estava tudo certo com os diferentes tipos de testes (básico e professor). Outro quesito relacionado ao *feedback* informativo é que a recomendação sobre o erro foi suficiente para encontrar a falha. Conforme observado na Figura 22, uma parcela de 15,9% (14,3% e 17,6% - primeira e segunda avaliações, respectivamente) dos alunos não souberam responder se o *feedback* auxiliou no entendimento do erro e com 10,7% (primeira avaliação: 9,6%; segunda avaliação: 11,8%) discordaram da afirmação.

Figura 22 - A ferramenta auxiliou no entendimento e conhecimento sobre o erro

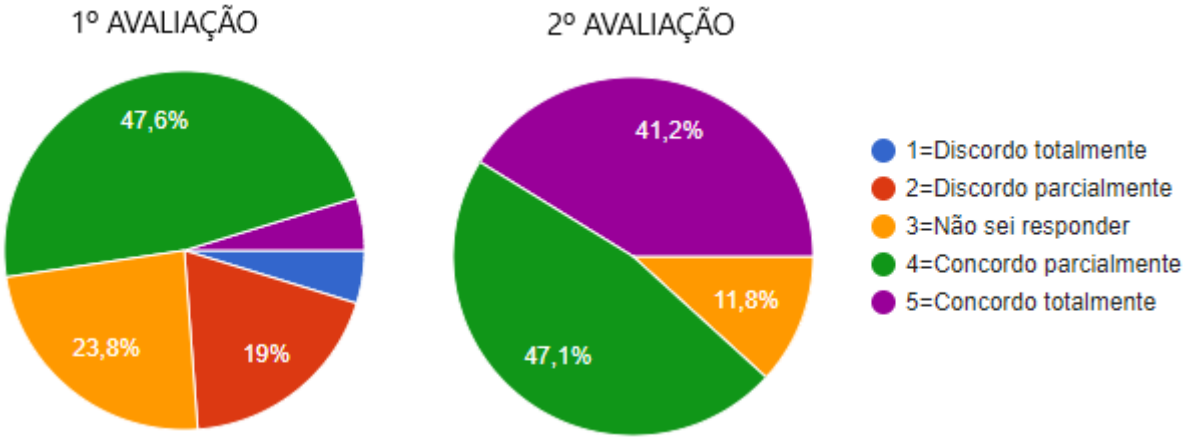


Fonte: Resultado questionário do Apêndice C

Quando perguntados se o *feedback* gerou uma reflexão antes de corrigir o código, com 70,3% - primeira avaliação: 52,4% (47,6% e 4,8%) e segunda avaliação: 88,3% (47,1% e 41,2%) - dos alunos concordaram que ele acarretou uma reflexão sobre o erro e isso possibilitou identificá-lo e corrigi-lo adequadamente. Apesar disso, de acordo com a Figura 23, uma parcela de 17,8% (23,8% e 11,8% - primeira e segunda avaliações) não souberam responder e 23,8% (19% e 4,8%) dos alunos que responderam a primeira pesquisa, discordam

dessa afirmação. A diferença de opiniões entre as opções discordo e não souberam responder, nas duas avaliações, podem ter sido ocasionadas pelos problemas técnicos ocorridos no início do uso da ferramenta.

Figura 23 - *Feedback* gerou reflexão sobre o erro



Fonte: Resultado questionário do Apêndice C

Dois pontos interessantes na pesquisa chamaram a atenção: a irrelevância do ambiente estar disponível na internet e, a dispensabilidade das mensagens estarem em português. Este item confere com as respostas dos alunos respondidas no questionário do **Apêndice A**, onde 50% informaram que entendem perfeitamente as mensagens exibidas pelo compilador GCC do DevC++.

De um modo geral, a ferramenta foi avaliada positivamente pelos alunos com mais de 50% de aceitação e, de certa forma, o ambiente ajudou na resolução dos exercícios e no entendimento sobre o erro. As respostas desta pesquisa são apresentadas no **Apêndice D**.

9 CONSIDERAÇÕES FINAIS

Muitas atividades foram realizadas durante o período desta pesquisa, das quais, cita-se: revisão bibliográfica buscando conhecimentos nas áreas da educação e informática, que atendesse os objetivos propostos; análise documental da instituição de ensino e da disciplina de linguagem de programação I, a fim de situar o contexto da pesquisa; e observações e aplicação de questionários em sala de aula, com a finalidade de conhecer os alunos e entender suas dificuldades na arte da programação na disciplina inicial do curso de computação. Além disso, foi necessário investigar como o uso de ferramentas de avaliação automática de exercícios poderia ser útil ao professor de programação. Para atingir o objetivo delineado, a pesquisa contou com diversas fases: revisão da literatura sobre ferramentas de avaliação automática; identificação de características comuns e obrigatórias entre as ferramentas encontradas, e a definição de um conjunto de requisitos funcionais em torno dessas ferramentas, que serviram de subsídio para elaborar os requisitos do protótipo, implementação do ambiente e avaliação junto aos alunos.

Aliado a isso, foi necessário pesquisar sobre *feedback* e suas características. A finalidade foi identificar quais informações são pertinentes apresentar ao aluno para que o *feedback* seja eficaz, tornando o erro observável pelo estudante. Para adequar os *feedbacks* na ferramenta, propôs-se a criação de uma estratégia visando sistematizar e padronizar o processo de mapeamento e registro dos erros identificados nos códigos dos alunos, tencionando sua automatização; e a construção do ambiente chamado Codein'play com as mesmas funcionalidades dos juízes *online*. A intenção do juiz *online,* primeiramente, foi validar a estratégia de *feedback* elaborada nesta investigação, pretendendo proporcionar um olhar diferenciado dos erros que ocorrem durante a resolução de exercícios, estimulando a aprendizagem. No entanto, a ferramenta foi muito além do esperado, como pode ser observado durante as atividades propostas, apresentadas na seção 8.

Um fato interessante observado na análise das ferramentas foi que, durante a revisão da literatura sobre os juízes *online*, revelou-se que a maioria delas utiliza o tipo de questão de submissão de código com avaliação automática, algumas fornecem relatórios de desempenho para alunos e professores, mas nenhuma possui a exibição da cobertura do código, informando ao aluno os caminhos percorridos pelo compilador em seu código, e o *feedback* informativo explicando em detalhes o erro ocorrido após a compilação. Essas duas

características foram implementadas no Codein'play, tendo em face a importância de o aluno acompanhar o que está acontecendo em seu código e porque está errando.

Na fase de validação do ambiente, este foi avaliado pelos alunos matriculados na disciplina LPI do curso STSI do IFRS Campus Porto Alegre, no segundo semestre de 2019, em três momentos distintos. Essa aferição teve como objetivo analisar o grau de concordância dos estudantes sobre os *feedbacks* informativos mapeados e identificar se as recomendações auxiliaram no entendimento sobre o erro. Buscou-se validar os requisitos da ferramenta, identificando pontos fortes e possíveis melhorias. Os resultados obtidos mostram a contribuição do Codein'play em proporcionar um ambiente que auxilia o estudante na resolução dos exercícios de programação. Conforme os resultados desta pesquisa, a ferramenta foi avaliada positivamente com mais de 50% de aceitação, concluindo, que o *feedback* gerou uma reflexão sobre o erro, possibilitando sua identificação e correção adequada.

Com um percentual acima de 70%, os alunos concordaram que a ferramenta auxiliou no entendimento do erro de alguma forma, contribuindo na assimilação do conteúdo. Porém, de acordo com a média de tentativas de acerto das 20 questões, aparentemente, há indícios de que o *feedback* informativo não surtiu o efeito esperado, haja vista a quantidade de execuções acima de três tentativas, momento em que o *feedback* informativo começa a ser exibido e mesmo assim, aparentemente, continuavam errando. Durante o acompanhamento de uma parte do Simulado em sala de aula, constatou-se que os alunos estavam mais interessados no *feedback* avaliativo do que no *feedback* informativo. A função desse é indicar ao aluno se o resultado esperado coincide com o obtido pelo compilador, ou seja, se errou ou acertou. Em função disso, levanta-se um questionamento com esse resultado: será que os alunos possuem uma ansiedade tão grande para obter respostas imediatas (acertaram/erraram) e estão menos preocupados com os processos que levam ao acerto ou ao erro?

Um outro dado relevante está relacionado à reprovação na disciplina. Dos 13 alunos que participaram da Maratona de Programação, seis precisavam recuperar nota e sete não precisavam. Do total de alunos que fizeram as questões (parcialmente ou totalmente) no juiz *online*, dez estudantes foram aprovados, de uma total de 23 aprovações na disciplina. Dos seis alunos que precisavam recuperar nota para serem aprovados, quatro fizeram todas as questões da Maratona, e destes, apenas três foram aprovados na disciplina. Dos sete alunos que não

precisavam recuperar nota, seis tiveram um desempenho melhor ou igual à primeira avaliação. O juiz *online* começou a ser utilizado a partir da atividade de Simulado que antecedeu a primeira prova e a evasão/abandono na disciplina chegou a 38,64% no segundo semestre de 2019. Contudo, evidencia-se que, apesar de a ferramenta estar disponível para auxiliar na aprendizagem desde um pouco depois do início do semestre, ela não garantiu a permanência dos alunos na disciplina. Por outro lado, ela auxiliou nos estudos daqueles que não atingiram a média mínima na primeira prova e, ainda, permaneceram na turma, mediante o exercício das questões propostas na Maratona. A aquisição desse novo conhecimento, pela prática, foi observado na melhora das notas na segunda prova. Com isso, vê-se indícios que usar o juiz *online* como meio não evitou a evasão/abandono escolar, mas contribuiu, junto com outros recursos aplicados em sala de aula (atividades gamificadas e aulas expositivas), no aumento do número de aprovações dos estudantes que continuaram matriculados, mesmo apresentando dificuldades. Observa-se uma diminuição na reprovação da disciplina de 6,82% no segundo semestre de 2019, comparando com o mesmo período no ano de 2018.

O Codein'play está em constante melhoria, dessa forma, como trabalhos futuros, o juiz *online* requer uma continuidade em seu desenvolvimento, de modo a aprimorar suas funcionalidades e *layout*. Assim, é necessário ajustar o cadastro dos casos de teste, permitindo validar múltiplas saídas; desenhar um novo *template*, com cores e disposições mais apropriadas para os componentes (botões); estender a ferramenta, para outras linguagens de programação utilizada no IFRS; aprimorar a IDE, para sinalizar o erro de sintaxe antes de executar o código; aperfeiçoar a ferramenta, para permitir o uso de endereço de memória útil na aprendizagem sobre ponteiros; e implementar um sistema de recomendação e um tutor virtual, que recomendem, automaticamente, materiais de apoio ao aluno sobre os tópicos que ele mais erra dentro do ambiente. O tutor ficaria vinculado ao *feedback* informativo, dando a impressão ao aluno de que o objeto animado o está orientando.

Acredita-se que esta pesquisa contribuiu na integração de conceitos oriundos de outras áreas de conhecimento, dentro do campo da Informática da Educação, haja vista a sua interdisciplinaridade. Nesse sentido, pode-se citar como exemplo o uso de conceitos sobre o teste de software, padrões de desenvolvimento, e levantamento e documentação de requisitos advindos da área da Informática. O mesmo aconteceu com a área da Educação na busca de concepções em torno da formação do conhecimento e dos mecanismos ao redor do erro. Não

se pode deixar de mencionar os *feedbacks*, muito trabalhados neste estudo e presentes em diversas áreas de conhecimento. Observou-se que o uso deles é uma boa alternativa no aprendizado de disciplinas iniciais de programação de computadores, em conjunto com um ambiente que se propõe a compilar, testar e executar os códigos automaticamente. Todavia, o uso desses ambientes não substitui o professor em sala de aula, mas o auxilia na correção imediata de atividades, quando estas puderem ser automatizadas.

REFERÊNCIAS

ABREU, Luiz Carlos de; OLIVEIRA, Márcio Alves de; CARVALHO, Tatiana Dias de; MARTINS, Sonia R.; GALLO, Paulo Rogério; REIS, Alberto O. A. (2010). **A EPISTEMOLOGIA GENÉTICA DE PIAGET E O CONSTRUTIVISMO**. Rev. Bras. Crescimento Desenvolvimento Humano. 2010; 20(2): 361-366

ABREU-E-LIMA, Denise Martins de; ALVES, Mario Nunes. (2011) **O FEEDBACK SUA IMPORTÂNCIA NO PROCESSO DE TUTORIA A DISTÂNCIA**. Pro-Posições, Campinas. maio/ago. 2011, vol.22, n.2, pp.189-205.

ALVES, Fábio P.; JAQUES, Patrícia. **UM AMBIENTE VIRTUAL COM FEEDBACK PERSONALIZADO PARA APOIO A DISCIPLINAS DE PROGRAMAÇÃO**. 3º Congresso Brasileiro de Informática na Educação (CBIE 2014). Workshops (WCBIE 2014). 2014

AMBROSIO, Ana Paula; ALMEIDA, Leandro S.; MACEDO, Joaquim; SANTOS, Alexandre; FRANCO, Amanda H. (2011) **PROGRAMAÇÃO DE COMPUTADORES: COMPREENDER AS DIFICULDADES DE APRENDIZAGEM DOS ALUNOS**. Revista Galego-Portuguesa de Psicoloxía e Educación. Vol. 19 (nº1), Ano 16º-2011 ISSN:1138-1663

ANDRADE, Raul. **INVESTIGANDO O FEEDBACK DOS ALUNOS SOBRE ASPECTOS QUALITATIVOS DO CÓDIGO: UM ESTUDO DE CASO**. VII Congresso Brasileiro de Informática na Educação (CBIE 2018). Anais dos Workshops do VII Congresso Brasileiro de Informática na Educação (WCBIE 2018). 2018

ANZAI, Y.; SIMON, H. A. **THE THEORY OF LEARNING BY DOING.** Carnegie-Mellon University, Pittsburgh. 1979

ARSHAVSKIY, Marina. (2019) **DESIGN INSTRUCIONAL PARA E-LEARNING: GUIA ESSENCIAL PARA CRIAR CURSOS DE E-LEARNING BEM-SUCEDIDOS**. Tradução de Jander Temístocles de Oliveira. 1º. ed.: Babelcube Books, 2019. 580 p.

AZENHA, Maria da Graça. **CONSTRUTIVISMO: DE PIAGET A EMILIA FERREIRO**. 7. ed. São Paulo, SP: Ática, 2001. 112 p. (Princípios)

BARTIÉ, Alexandre. (2002) **GARANTIA DE QUALIDADE DE SOFTWARE**. Rio de Janeiro, RJ: Campus, 2002

BECKER, Fernando (2014) **ABSTRAÇÃO PSEUDO-EMPÍRICA E REFLEXIONANTE: SIGNIFICADO EPISTEMOLÓGICO E EDUCACIONAL**. Schème. Revista Eletrônica de Psicologia e Epistemologia Genética. Vol. 6. Número Especial. Novembro/2014

BEHAR, Patrícia Alejandra. (2009) **MODELOS PEDAGÓGICOS EM EDUCAÇÃO A DISTÂNCIA**. Porto Alegre: Artmed, 2009

BORGES, Marcos Augusto F. (2000) **AVALIAÇÃO DE UMA METODOLOGIA**

ALTERNATIVA PARA A APRENDIZAGEM EM PROGRAMAÇÃO. Workshop de Educação em Computação, Congresso anual da SBC, Curitiba, PR. 2000.

BOSSE, Yorah; GEROSA, Marco Aurélio. (2016) **WHY IS PROGRAMMING SO DIFFICULT TO LEARN? PATTERNS OF DIFFICULTIES RELATED TO PROGRAMMING LEARNING**. ACM SIGSOFT Software Engineering Notes. November 2016 Volume 41 Number 6

BRAGA, Pedro Henrique Cacique. (2016) **TESTES DE SOFTWARE**. São Paulo: Pearson Education do Brasil, 2016

BRASIL (2008). Casa Civil. **LEI Nº 11.892, DE 29 DE DEZEMBRO DE 2008**. Institui a Rede Federal de Educação Profissional, Científica e Tecnológica, cria os Institutos Federais de Educação, Ciência e Tecnologia, e dá outras providências. Disponível em: <http://www.planalto.gov.br/ccivil_03/_ato2007-2010/2008/lei/l11892.htm> Acesso em: 09 jul. 2019

BRASSCOM (2015). **O MERCADO DE PROFISSIONAIS DE TECNOLOGIA DA INFORMAÇÃO E COMUNICAÇÃO NO BRASIL: UMA ANÁLISE DO PERÍODO DE 2006 A 2013**. Disponível em: https://brasscom.org.br/wp-content/uploads/2017/08/estudo_profissionais_tic_brasil_2006_2013.pdf. Acesso em: 11 jan. 2019

CAMPOS, C. P.; FERREIRA, C. E., (2004) **BOCA: UM SISTEMA DE APOIO A COMPETIÇÕES DE PROGRAMAÇÃO**. Workshop de Educação em Computação, Anais do Congresso da SBC, Salvador, BA. Disponível em: <https://www.ime.usp.br/~cassio/boca/campos-ferreira-wei2004.pdf> Acesso em: 11 jan. 2019

CAPTIVO, Maria Tavares Manso. (2018). **O CONTRIBUTO DO FEEDBACK ESCRITO NA APRENDIZAGEM MATEMÁTICA DE ALUNOS DO 12.º ANO DE ESCOLARIDADE**. Mestrado em Ensino de Matemática. Relatório da Prática de Ensino Supervisionada. Universidade de Lisboa, 2018. Disponível em: <https://repositorio.ul.pt/bitstream/10451/35358/1/ulfpie053114_tm.pdf> Acesso em: 02 set. 2019

CARDOSO, Ana Carolina Simões. (2018). **O FEEDBACK ALUNO-ALUNO EM UM AMBIENTE VIRTUAL DE APRENDIZAGEM**. Trab. linguist. apl., Campinas , v. 57, n. 1, p. 383-409, Apr. 2018 . Disponível em: <http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0103-18132018000100383&lng=en&nrm=iso>. Acesso em: 14 out. 2019. https://doi.org/10.1590/010318138647945235301.

CARDOSO, Ana Carolina Simões. (2011). **FEEDBACK EM CONTEXTOS DE ENSINO-APRENDIZAGEM ON-LINE**. Linguagens e Diálogos, v. 2, p. 17. Disponível em: <https://docplayer.com.br/4328328-Feedback-em-contextos-de-ensino-aprendizagem-on-line.html>

Acesso em: 28 jul. 2019

CERQUEIRA, Magali. (2009). **O PAPEL DO ERRO NA PERSPECTIVA PIAGETIANA**. Disponível em:<http://webfoliomagalicerqueira.blogspot.com/2009/11/o-papel-do-erro-na-perspectiva.html> Acesso em: 14 out. 2019

CHAVES, J. O. M. ; CASTRO, A. F.; LIMA, R. W.; LIMA, M. V. A.; FERREIRA, K. H. A. (2014). **MOJO: UMA FERRAMENTA PARA INTEGRAR JUÍZES ONLINE AO MOODLE NO APOIO AO ENSINO E APRENDIZAGEM DE PROGRAMAÇÃO**. HOLOS, Ano 30, Vol. 5. 2014

DEITOS, Fernanda Nunes. (2018) **JOGO DE DITADO DIGITAL: O ERRO COMO PARTE DO PROCESSO DE APRENDIZAGEM**. Dissertação de Mestrado. Instituto Federal do Rio Grande do Sul - IFRS, Porto Alegre/RS. 2018

DEMO, Pedro E. (2001) **É ERRANDO QUE A GENTE APRENDE**. Nova Escola. São Paulo, n.144, pp.49-51, ago. 2001.

FARIAS, Igor Sodré. (2015) **USO DE COBERTURA DE CÓDIGO NO TESTE EXPLORATÓRIO**. Trabalho de conclusão de curso em Ciências da Computação - Universidade Estadual do Sudoeste da Bahia, Vitória da Conquista, 2015.

FILATRO, A. (2008). **DESIGN INSTRUCIONAL NA PRÁTICA**. São Paulo: Pearson Education do Brasil.

FONSECA, J. J. S. (2002). **METODOLOGIA DA PESQUISA CIENTÍFICA**. Fortaleza: UEC, 2002.

FORBELLONE, André L. V; EBERSPÄCHER, Henri F. (2005) **LÓGICA DE PROGRAMAÇÃO: A CONSTRUÇÃO DE ALGORITMOS E ESTRUTURAS DE DADOS**. 3ª ed. - São Paulo: Prentice Hall, 2005

GIL, Antônio C. (2007) **COMO ELABORAR PROJETOS DE PESQUISA**. 4. ed. São Paulo: Atlas, 2007.

GIRAFFA, Lucia; MARCZAK, Sabrina; ALMEIDA, Gláucio. (2003) **O ENSINO DE ALGORITMOS E PROGRAMAÇÃO MEDIADO POR UM AMBIENTE WEB**. Congresso Nacional da Sociedade Brasileira de Computação. Campinas, SP. 2003.

GOMES, Marina S.; BECKER, Liliane; HUEGEL, César; GESTARO, Lucas; CAMARGO, Alex; AMARAL, Érico. (2015a) **CFACIL - UMA FERRAMENTA DE APOIO AO PROCESSO DE ENSINO-APRENDIZAGEM DE ALGORITMOS E PROGRAMAÇÃO. RECONHECENDO ERROS**. Disponível em: <https://www.researchgate.net/publication/303864599>. Acesso em: 13 out.2018

GOMES, Marina S.; BECKER, GESTARO, Lucas, AMARAL, Érico, TAROUCO, Liane (2015b). **UM ESTUDO SOBRE ERROS EM PROGRAMAÇÃO: RECONHECENDO**

AS DIFICULDADES DE PROGRAMADORES INICIANTES. Anais dos Workshops do IV Congresso Brasileiro de Informática na Educação (CBIE 2015). Disponível em: <https://www.researchgate.net/publication/300237355_Um_estudo_sobre_erros_em_programacao_-_Reconhecendo_as_dificuldades_de_programadores_iniciantes>. Acesso em: 13 out.2018

HATTIE, J. & TIMPERLEY, H. (2007). **THE POWER OF FEEDBACK**. Review of Educational Research, 77 (1), 81-112

HOSHINO, Edna Ayako. (2004). **CAPÍTULO 8 - RECUPERAÇÃO DE ERROS E OUTROS TIPOS DE ANALISADORES SINTÁTICOS**. Notas de aulas da disciplina de Projeto e Implementação de Linguagens da Universidade Federal de Mato Grosso do Sul (UFMS). Disponível em: <http://www.facom.ufms.br/~ricardo/Courses/CompilerI-2009/Materials/cap8.pdf> Acesso em: 12 ago. 2018

IEEE (1990). IEEE Standard Glossary of Software Engineering Terminology. **IEEE Std 610.12-1990 (Revision and Redesignation of IEEE Std 792-1983)**. 1990. Disponível em: <http://www.mit.jyu.fi/ope/kurssit/TIES462/Materiaalit/IEEE_SoftwareEngGlossary.pdf> Acesso em: 31 ago. 2019

IFRS (2015). **ORGANIZAÇÃO DIDÁTICA**. Aprovada pelo Conselho Superior do IFRS, conforme Resolução nº 046, de 08 de maio de 2015 e alterada pelas Resoluções nº 071, de 25 de outubro de 2016 e nº 086, de 17 de outubro de 2017. Disponível em: <https://ifrs.edu.br/wp-content/uploads/2017/07/OD-Alterada-Publica%C3%A7%C3%A3o-Portal-1.pdf>. Acesso em: 21 nov. 2019.

IFRS (2018a). **PROJETO PEDAGÓGICO DO CURSO SUPERIOR DE TECNOLOGIA EM SISTEMAS PARA INTERNET**. Disponível em: <http://www.poa.ifrs.edu.br/images/Cursos/Superiores/Tecnologia_Sistemas_Internet/Curso_Sistemas_para_Internet_PPC_2018-07-13.pdf>. Acesso em: 21 nov. 2019.

IFRS (2018b). **PLANO ESTRATÉGICO DE PERMANÊNCIA E ÊXITO DOS ESTUDANTES DO INSTITUTO FEDERAL DO RIO GRANDE DO SUL**. Aprovado pelo Conselho Superior, conforme Resolução nº 064, de 23 de outubro de 2018. Disponível em: <https://ifrs.edu.br/wp-content/uploads/2018/10/Resolucao_064_18_Aprovar_Plano_Estrategico_Completo.pdf>. Acesso em: 10 jun. 2019

INEP (2017). **SINOPSES ESTATÍSTICAS DA EDUCAÇÃO SUPERIOR – GRADUAÇÃO**. Disponível em: <http://portal.inep.gov.br/web/guest/sinopses-estatisticas-da-educacao-superior>. Acesso em: 11 jan. 2019

JENKINS, T. (2002). **ON THE DIFFICULTY OF LEARNING TO PROGRAM**. In: Proceedings of 3rd Annual LTSN_ICS, Conference Loughborough University.

JESUS, Galileu Santos de. (2018) **UMA ABORDAGEM PARA AUXILIAR A CORREÇÃO DE ERROS DE PROGRAMADORES INICIANTES**. Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação, Universidade Federal de Sergipe (UFS), 2018.

KOSCHIANSKI, André; SOARES, Michel dos Santos. (2006) **QUALIDADE DE SOFTWARE: APRENDA AS METODOLOGIAS E TÉCNICAS MAIS MODERNAS PARA O DESENVOLVIMENTO DE SOFTWARE**. São Paulo, SP: Novatec, 2006

KUTZKE, Alexander Robert (2015). **INFORMÁTICA EDUCACIONAL E A MEDIAÇÃO DO ERRO NA EDUCAÇÃO: UM ESTUDO TEÓRICO-CRÍTICO E UMA PROPOSTA DE INSTRUMENTO COMPUTACIONAL**. Tese de Doutorado. UFP - Curitiba. Disponível em: <http://hdl.handle.net/1884/40907> Acesso em: 18 out. 2018

LA TAILLE, Y. de. (1997) **O ERRO NA PERSPECTIVA PIAGETIANA**. In: AQUINO, J. G. (Org.) Erro e fracasso na escola: alternativas teóricas e práticas. São Paulo: Summus, 1997 p. 25- 45.

LAHTINEN, E.; ALA-MUTKA, K.; JÄRVINEN, H.-M. (2005) **A STUDY OF THE DIFFICULTIES OF NOVICE PROGRAMMERS**. Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education - ITiCSE '05, 2005.

MALDONADO, José C.; VINCENZI, Auri M. R.; BARBOSA, Ellen F.; SOUZA, Simone do Rocio S. de; DELAMARO, Márcio E. (2010). **ASPECTOS TEÓRICOS E EMPÍRICOS DE TESTE DE COBERTURA DE SOFTWARE**. Laboratório de Engenharia de Software. Material Didático. ICMC - USP. Disponível em: <http://www.labes.icmc.usp.br/site/content/aspectos-te%C3%B3ricos-e-emp%C3%ADricos-de-teste-de-cobertura-de-software> Acesso em: 29 ago. 2019

MOLINARI, Leonardo (2008). **TESTES DE SOFTWARE: PRODUZINDO SISTEMAS MELHORES E MAIS CONFIÁVEIS**. 1. ed. São Paulo, SP: Érica, 2008.

MORY, Edna Holland (2004). **FEEDBACK RESEARCH REVISITED**. In D. H. Jonassen (Ed.), Handbook of research on educational communications and technology (pp. 745-783). Mahwah, NJ, US: Lawrence Erlbaum Associates Publishers.

NASCIMENTO, Adaiane Késsila Mota. (2012). **APRENDIZAGEM INFANTIL: O ERRO NA VISÃO CONSTRUTIVISTA**. Site Webartigos. Disponível em: <https://www.webartigos.com/artigos/aprendizagem-infantil-o-erro-na-visao-construtivista/93152> Acesso em:14 out. 2019

OLIVEIRA, Cristiana Filipa Ferreira. (2018). **A UTILIZAÇÃO DO FEEDBACK COMO INSTRUMENTO DE APRENDIZAGEM**. Instituto Politécnico de Setúbal. Escola Superior de Educação. Relatório do Projeto de Investigação, 2018. Disponível em: <https://comum.rcaap.pt/bitstream/10400.26/25525/1/Relat%c3%b3rio%20do%20projeto

%20de%20investiga%c3%a7%c3%a3o.pdf> Acesso em: 02 set. 2019.

ORNELAS, Anabela Gomes. (2018). **FEEDBACK E RESOLUÇÃO DE PROBLEMAS DE MATEMÁTICA: UMA EXPERIÊNCIA COM ALUNOS DO 4.º ANO**. Instituto Politécnico de Setúbal. Escola Superior de Educação. Relatório do Projeto de Investigação, 2018. Disponível em: <https://comum.rcaap.pt/bitstream/10400.26/25527/1/relat %C3%B3rio_de_investiga%C3%A7%C3%A3o.pdf> Acesso em: 02 set. 2019.

PACHECO, E. M.; PEREIRA, CALDAS, L. A.; SOBRINHO, M. D. (2010). **INSTITUTOS FEDERAIS DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA: LIMITES E POSSIBILIDADES**. Linhas Críticas, Brasília, DF, v. 16, n. 30, p. 71-88, 2010.

PACHECO, E. (Organizador) (2011). **OS INSTITUTOS FEDERAIS: UMA REVOLUÇÃO NA EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA**. Brasília: Moderna, 2011

PAES, R.B.; MALAQUIAS, R.; GUIMARÃES, M.; ALMEIDA, H. (2013). **FERRAMENTA PARA A AVALIAÇÃO DE APRENDIZADO DE ALUNOS EM PROGRAMAÇÃO DE COMPUTADORES**. In Anais dos Workshops do Congresso Brasileiro de Informática na Educação (Vol. 2, No. 1).

PAIVA, V. L. M. O. (2003) **FEEDBACK EM AMBIENTE VIRTUAL. IN: LEFFA, V. (ORG.) INTERAÇÃO NA APRENDIZAGEM DAS LÍNGUAS**. Pelotas: EDUCAT. Disponível em: <www.veramenezes.com/feedback.htm>. Acesso em: 28 jul. 2019

PIAGET, Jean. (1967). **PSICOLOGIA DA INTELIGÊNCIA**. 2. ed. Rio de Janeiro: Fundo de Cultura, 1967.

PIAGET, Jean. (1976). **A EQUILIBRAÇÃO DAS ESTRUTURAS COGNITIVAS - PROBLEMA CENTRAL DO DESENVOLVIMENTO**. Trad. Profa. Dra. Marion Merlone dos Santos Penna. Rio de Janeiro: Zahar, 1976.

PIAGET, Jean. (1983). **A EPISTEMOLOGIA GENÉTICA/ SABEDORIA E ILUSÕES DA FILOSOFIA; PROBLEMAS DE PSICOLOGIA GENÉTICA**. In: Piaget. Traduções de Nathanael C. Caixeiro, Zilda Abujamra Daeir, Célia E. A. Di Pier. 2ª ed. São Paulo: Abril Cultural, 1983. (Os pensadores)

PIAGET, Jean. (1995) **ABSTRAÇÃO REFLEXIONANTE: RELAÇÕES LÓGICO-ARITMÉTICAS E ORDEM DAS RELAÇÕES ESPACIAIS**. Trad. Fernando Becker e Petronilha Beatriz Gonçalves da Silva. Porto Alegre : Artes Médicas, 1995.

PIAGET, Jean. (1999). **SEIS ESTUDOS DE PSICOLOGIA**. 24. ed. Rio de Janeiro, RJ: Forence Universitária, 1999. 136 p. ISBN 85-218-0246-3

PIMENTEL, E. NIZAM, O. (2008). **ENSINO DE ALGORITMOS BASEADO NA APRENDIZAGEM SIGNIFICATIVA UTILIZANDO O AMBIENTE DE**

AVALIAÇÃO NETEDU. In Anais do XXXVIII Congresso da Sociedade Brasileira da Computação: Workshop sobre educação em computação.

PRESSMAN, Roger S. **ENGENHARIA DE SOFTWARE**. São Paulo, SP: Makron Books, 1995. xxxii, 1056 p.

PRODANOV, Cleber Cristiano; FREITAS, Ernani Cesar de. **METODOLOGIA DO TRABALHO CIENTÍFICO: MÉTODOS E TÉCNICAS DA PESQUISA E DO TRABALHO ACADÊMICO**. 2. ed. Novo Hamburgo, RS: Feevale, 2013. 276 p.

PYKE, J. & SHERLOCK, J. (2010). **A CLOSER LOOK AT INSTRUCTOR-STUDENT FEEDBACK ONLINE: A CASE STUDY ANALYSIS OF THE TYPES AND FREQUENCY**. MERLOT Journal of Online Learning and teaching, vol. 6, no. 1. Disponível em: <http://jolt.merlot.org/vol6no1/pyke_0310.pdf> . Acesso em: 28 jul. 2019

SANTOS, Joanna C. S.; RIBEIRO, Admilson. (2011). **JONLINE: PROPOSTA PRELIMINAR DE UM JUIZ ONLINE DIDÁTICO PARA O ENSINO DE PROGRAMAÇÃO**. Anais do XXII SBIE - XVII WIE. 2011

SILVA, Ulisses F. F. da; CAMPOS, Cassio P. de. (2006). BOCA ONLINE CONTEST ADMINISTRATOR - SISTEMA DE SUBMISSÃO: **MANUAL DE REFERÊNCIA PARA AS EQUIPES.** 2006. Disponível em: <http://maratona.ime.usp.br/manualBOCA.html> Acesso em: 04 ago. 2019

SOUZA, Draylson Micael; BATISTA, Marisa Helena da Silva; BARBOSA, Ellen Francine. (2016) **PROBLEMAS E DIFICULDADES NO ENSINO E NA APRENDIZAGEM DE PROGRAMAÇÃO: UM MAPEAMENTO SISTEMÁTICO**. Revista Brasileira de Informática na Educação, Volume 24, Número 1, 2016

SEBER, Maria da Glória. (1997). **PIAGET: O DIÁLOGO COM A CRIANÇA E O DESENVOLVIMENTO DO RACIOCÍNIO**. 1. ed. São Paulo, SP: Scipione, 1997. 245 p. (Pensamento e ação no magistério)

SEVELLA, Pranay Kumar; LEE, Young; YANG, Jeong. (2013) **DETERMINING THE BARRIERS FACED BY NOVICE PROGRAMMERS**. INTERNATIONAL JOURNAL OF SOFTWARE ENGINEERING (IJSE), Vol. 4. p. 10, 2013.

SHUTE, V. J. (2007) **FOCUS ON FORMATIVE FEEDBACK**. Review of Educational Research, Princeton, v. 1, n. 78, p. 153-189, 2008. Disponível em: <http://www.ets.org/Media/Research/pdf/RR-07-11.pdf>. Acesso em: 27 abr. 2011.

SOFTEX (2018). **SOFTEX: BRASIL VAI PERDER R$ 167 BILHÕES NOS PRÓXIMOS CINCO ANOS POR FALTA DE PROFISSIONAIS DE TICS**. 09-10-2018. Disponível em: http://www.agenciatelebrasil.org.br/Noticias/Softex%3A-Brasil-vai-perder-R%24-167-bilhoes-nos-proximos-cinco-anos-por-falta-de-profissionais-de-TICs-144.html?UserActiveTemplate=site. Acesso em: 11 jan. 2019

SOMMERVILLE, Ian. (2011). **ENGENHARIA DE SOFTWARE**. 9. ed. São Paulo, SP: Pearson, 2011. xiii, 529 p.

TEIXEIRA, Adriano Canabarro; FURINI, Caroline da Silva; CAIMI, Flávia Eloisa. (2018). **A EQUILIBRAÇÃO MAJORANTE EM CRIANÇAS DE EDUCAÇÃO INFANTIL: UM ESTUDO DE CASO**. Rev. FAEEBA - Ed. e Contemp., Salvador, v. 27, n. 53, p. 251-270, set./dez. 2018. Disponível em: <https://www.researchgate.net/publication/330001471_A_EQUILIBRACAO_MAJORANTE_EM_CRIANCAS_DE_EDUCACAO_INFANTIL_um_estudo_de_caso> Acesso em: 08 jan. 2020

VIEGAS, Thaís Ramos. (2017) **CONSPROG: UMA PROPOSTA PEDAGÓGICA PARA O ENSINO-APRENDIZAGEM DE PROGRAMAÇÃO**. Dissertação de Mestrado. Instituto Federal do Rio Grande do Sul - IFRS, Porto Alegre/RS. 2017

WILLIAMS, Richard Leonard. (2005) **PRECISO SABER SE ESTOU INDO BEM!: UMA HISTÓRIA SOBRE A IMPORTÂNCIA DE DAR E RECEBER FEEDBACK**. 2. ed. Rio de Janeiro, RJ: Sextante, 2005.

ZEFERINO, Angélica Maria Bicudo; DOMINGUES, Rosângela Curvo Leite; AMARAL, Eliana. (2007) **FEEDBACK COMO ESTRATÉGIA DE APRENDIZADO NO ENSINO MÉDICO**. Rev. bras. educ. med. 2007, vol.31, n.2, pp.176-179.

APÊNDICE A

QUESTIONÁRIO INICIAL

Caros(as) alunos(as),

O objetivo deste questionário é obter informações a respeito das dificuldades encontradas durante a resolução dos exercícios propostos na disciplina de Linguagem de Programação I, bem como identificar o seu entendimento e sentimento sobre os erros ocorridos durante a execução do programa, além de conhecer os erros mais comuns.

Agradeço desde já a tua participação.

Idade: _____ Data: _____ Gênero: [ ] M [ ] F

1. Você possuía algum conhecimento de programação antes de cursar a disciplina?

[ ] sim, qual linguagem?_____ [ ] não

Das questões 2 à 4, selecione apenas 1 opção.

2. Qual a maior dificuldade encontrada nas atividades de programação da disciplina:

(a) entender o que está sendo pedido no enunciado

(b) compilar o código (não encontrar erros de compilação)

(c) identificar e entender os erros no código

(d) executar corretamente o programa (atende ao enunciado da questão)

(e) não tenho dificuldade

3. Ao programar com a linguagem C tenho mais dificuldades em:

(a) propor a lógica para resolução do programa

(b) utilizar a sintaxe correta (escrever os comandos da forma correta)

(c) manter a ordem das palavras do comando (troco o comando if por fi, por exemplo)

(d) respeitar as regras da linguagem C (sempre atribuo um valor em variável de tipo diferente, por exemplo: um valor float (flutuante) em uma variável do tipo int (inteiro))

(e) não tenho dificuldade

4. Ao compilar um determinado programa, quando ocorre um erro você:

(a) faz uma leitura da mensagem de erro e avalia os comandos no código

(b) vai direto ao código e tenta achar o erro sem ler a mensagem com detalhe

(c) recompila o programa mais uma vez na esperança que não ocorra o erro novamente, senão vai ter que buscar o erro no código

(d) descarta o programa e reescreve o código novamente

(e) nenhuma das opções anteriores

5. Com relação aos meus erros que mais acontecem. Escolha até 3:

[ ] falta de ponto e vírgula ao final do comando (;)

[ ] falta de aspas no início ou final da sentença ("")

[ ] falta de parênteses no início ou final do comando (())

[ ] falta de chaves no início ou final de um bloco ({})

[ ] erro de declaração de biblioteca (#include)

[ ] erro na declaração de variáveis

[ ] erro no uso dos operadores

[ ] erro de lógica (o resultado de saída da execução do programa nunca é o esperado)

Das questões 6 à 12, selecione apenas 1 opção.

6. Durante a compilação de um programa aparece uma mensagem de erro e você:

(a) não entende o que diz a mensagem pelo fato do idioma da mensagem ser em inglês

(b) a mensagem não é clara e não aponta corretamente para o local do erro e isso o deixa perdido

(c) a apresentação e organização da mensagem não é fácil de entender

(d) compreendo perfeitamente a mensagem de erro

7. Quando ao executar o programa ocorre um resultado inesperado na saída, você:

(a) testa o programa novamente com outros valores de entrada sem revisar o código

(b) aborta a execução do programa e revisa a lógica do código

(c) pede ajuda a um colega mais experiente ou ao professor

(d) não sei o que fazer

8. Quando ocorre um erro no programa, me sinto:

(a) fracassado e desanimado

(b) desafiado a procurar o problema no código

(c) irritado, porque fiz tudo certo

(d) não sinto nada

1=Não é importante, 2=Pouco importante, 3=Não sei responder, 4=Muito importante, e 5=Extremamente importante

9. O quão importante é, para você, a análise de respostas erradas (erros) durante sua aprendizagem?

Não é importante ① ② ③ ④ ⑤ Extremamente importante

1=Nunca, 2=Raramente, 3=Não sei responder, 4=Muitas vezes, e 5=Sempre

10. Com que frequência você costuma analisar seus próprios erros nas atividades em sala de aula?

Nunca ① ② ③ ④ ⑤ Sempre analiso meus erros

11. De que maneiras você costuma analisar suas respostas erradas?

(a) Não costumo analisar minhas respostas erradas

(b) Compartilho o problema com o professor

(c) Reflito sozinho

(d) Geralmente pergunto aos colegas

1=Não é importante, 2=Pouco importante, 3=Não sei responder, 4=Muito importante, e 5=Extremamente importante

12. Na sua opinião, quão relevante seria uma ferramenta computacional para facilitar a análise de suas respostas?

Não é importante ① ② ③ ④ ⑤ Extremamente importante

Comentários: (sinta-se à vontade em escrever sobre as dificuldades encontradas na aprendizagem de programação)

APÊNDICE B

RESPOSTAS DO QUESTIONÁRIO INICIAL

Este questionário foi aplicado nas turmas de LPI de 2018/02, 2019/01 e 2019/02 e tem como objetivo principal investigar a percepção dos alunos sobre os erros mais cometidos durante as aulas em programação de computadores, assim como entender o seu sentimento perante o erro e as dificuldades encontradas na aprendizagem da disciplina. Participaram desta pesquisa 64 alunos, que, de maneira voluntária, preencheram 12 questões fechadas e uma questão aberta, sendo esta opcional.

Uma das primeiras perguntas aplicados no questionário foi se os pesquisados possuíam algum conhecimento em programação antes de cursar a disciplina. Destes, oito alunos responderam que sim, e as linguagens de seus conhecimentos eram: DELPHI, VB, C, JAVA, UNITY, PYTHON, C++, HTML, CSS E JAVASCRIPT, na turma de 2018/02. Na turma de 2019/01, 16 alunos possuíam conhecimentos nas mesmas linguagens citadas, acrescentando a elas: PHP e PERL. Na turma de 2019/02, nove alunos responderam que sim, listando as mesmas linguagens mencionadas nos anos anteriores, acrescentando: Shell Script.

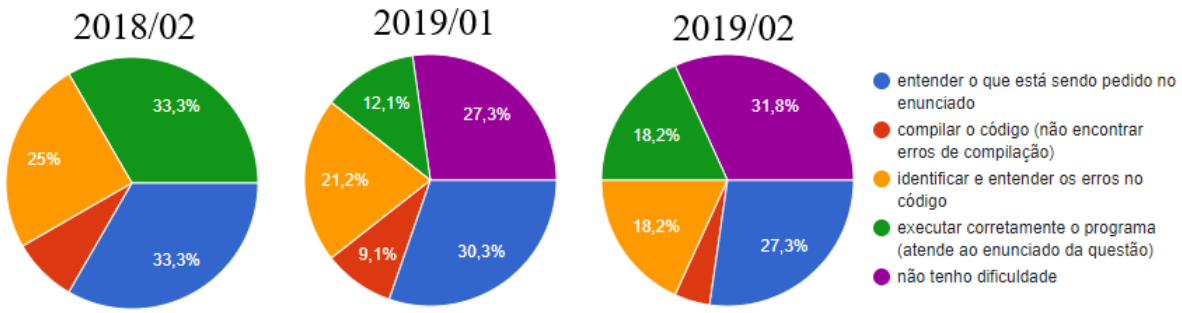
Q2 - Qual a maior dificuldade encontrada nas atividades de programação da disciplina?
Dos resultados obtidos com a questão 2, que trata da relação das dificuldades encontradas nas atividades de programação da disciplina pelos estudantes, foi relatado que,

Turma 2018/02: 33,3% estão relacionados à dificuldade de entender o que está sendo pedido no enunciado - o que reflete nos obstáculos encontrados no desenvolvimento de um programa e, consequentemente, na ocorrência dos erros -; 33,3%, aos problemas em executar corretamente os programas; 25%, à dificuldade para identificar os erros no programa; e 8,3%, à dificuldade para compilar o código

Turma 2019/01: 33,3% estão relacionados à dificuldade de entender o que está sendo pedido no enunciado da questão; 23,3%, à dificuldade para identificar e entender os erros nos códigos; 13,3%, à inexequibilidade para executar corretamente o programa (atende ao enunciado da questão); e 10% acham complexo compilar o código, ou seja, não encontrar erros de compilação. Porém, 30% dos respondentes afirmaram não possuir nenhuma impossibilidade na atividade de programação.

Turma 2019/02: 31,8% disseram que não possuíam dificuldade; 27,3% citaram a dificuldade de entender o que está sendo pedido no enunciado da questão; 18,2% para cada uma das dificuldades: identificar e entender os erros no código; e executar corretamente o programa (atende ao enunciado da questão); e 4,5% relataram terem problema para compilar o código (não encontrar erros de compilação).

Figura 24 - Maior dificuldade encontrada nas atividades de programação da disciplina



Fonte: Resultado questionário do Apêndice A

Q3 - Ao programar com a linguagem C tenho mais dificuldades em:

Esta questão buscava identificar as dificuldades, tendo como base os quatro tipos de erros apresentados por Hoshino (2004). Na questão 3, segundo os alunos, os maiores problemas encontrados ao programar com a linguagem C estão em:

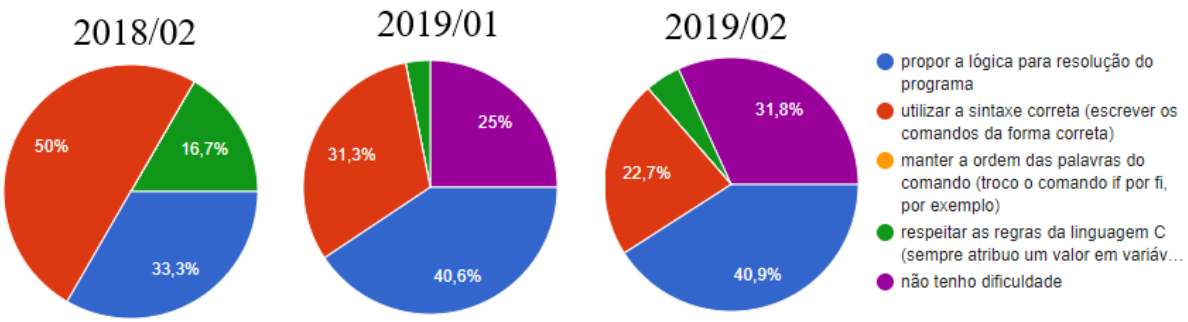
Turma 2018/02: utilizar a sintaxe correta dos comandos (escrever os comandos da forma correta) (50%); dificuldade em propor uma lógica para resolução do problema (33,3%); e respeitar as regras da linguagem C (16,7%) (atribuir um valor em variável de tipo diferente, por exemplo: um valor *float* (flutuante) em uma variável do tipo int (inteiro)). A opção da dificuldade em escrever o comando na ordem das palavras (trocar o comando *if* por *fi,* por exemplo) não foi selecionada.

Turma 2019/01: propor uma lógica para resolução do programa (40,6%); dificuldade de utilizar a sintaxe correta dos comandos (31,3%); dificuldade em respeitar as regras da linguagem C (3,1%), isto é, o aluno sempre atribui um valor em variável de tipos diferentes, por exemplo: um valor *float* (flutuante) em uma variável do tipo int (inteiro). No entanto, 25% responderam não possuir dificuldade durante a prática da programação e a opção da dificuldade em escrever o comando na ordem das palavras (trocar o comando *if* por *fi,* por

exemplo) não foi selecionada.

Turma 2019/02: propor uma lógica para resolução do programa (40,9%); dificuldade de utilizar a sintaxe correta dos comandos (22,7%); e problemas em respeitar as regras da linguagem C (4,5%). No entanto, 31,8% relataram não possuir dificuldade ao programar com a linguagem de programação C. A opção da dificuldade em escrever o comando na ordem das palavras (trocar o comando *if* por *fi,* por exemplo) não foi selecionada.

Figura 25 - Tenho mais dificuldades ao programar com a linguagem C



Fonte: Resultado questionário do Apêndice A

Q4 - Ao compilar um determinado programa, ocorre um erro e você:

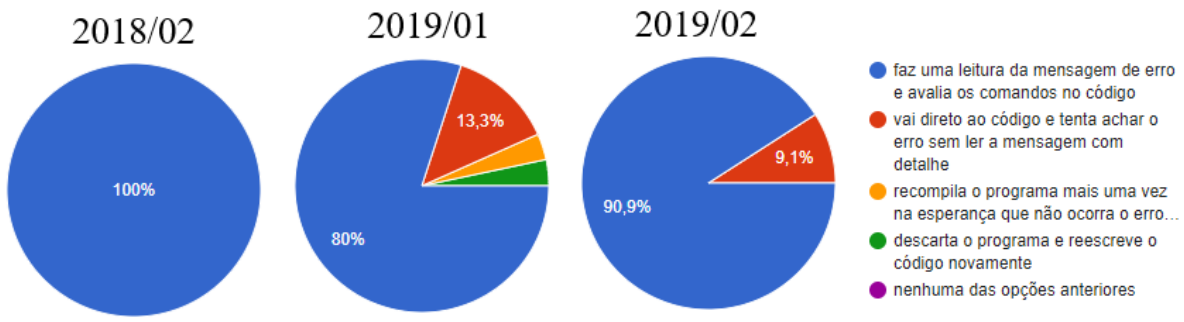
Com a questão 4, foi perguntada qual a ação tomada pelo estudante quando uma mensagem de erro era exibida:

Turma 2018/02: 100% dos alunos responderam que fazem a leitura da mensagem de erro e avaliam os comandos no programa, a fim de corrigir o problema.

Turma 2019/01: 80% dos alunos responderam que, primeiramente, leem a mensagem de erro para depois avaliar os comandos no código; 13,3% dos estudantes olham direto o código e tentam achar o erro, sem antes analisar, mais detalhadamente, a mensagem; e 3,3% para cada uma das opções: recompila o programa mais uma vez, na esperança de que o erro não ocorra novamente, e descarta o programa e reescreve o código novamente.

Turma 2019/02: 90,9% dos alunos responderam que, primeiro, leem a mensagem de erro e depois avaliam os comandos no código; 9,1% dos respondentes visualizam o código e tentam achar o erro sem analisar a mensagem detalhadamente.

Figura 26 - O que você faz quando ocorre um erro?



Fonte: Resultado questionário do Apêndice A

Q5 - Com relação aos meus erros que mais acontecem. Escolha até 3:

Quanto aos erros mais cometidos, tratado na questão 5, foi solicitado aos alunos que selecionassem até três erros mais cometidos durante a programação. Dos resultados:

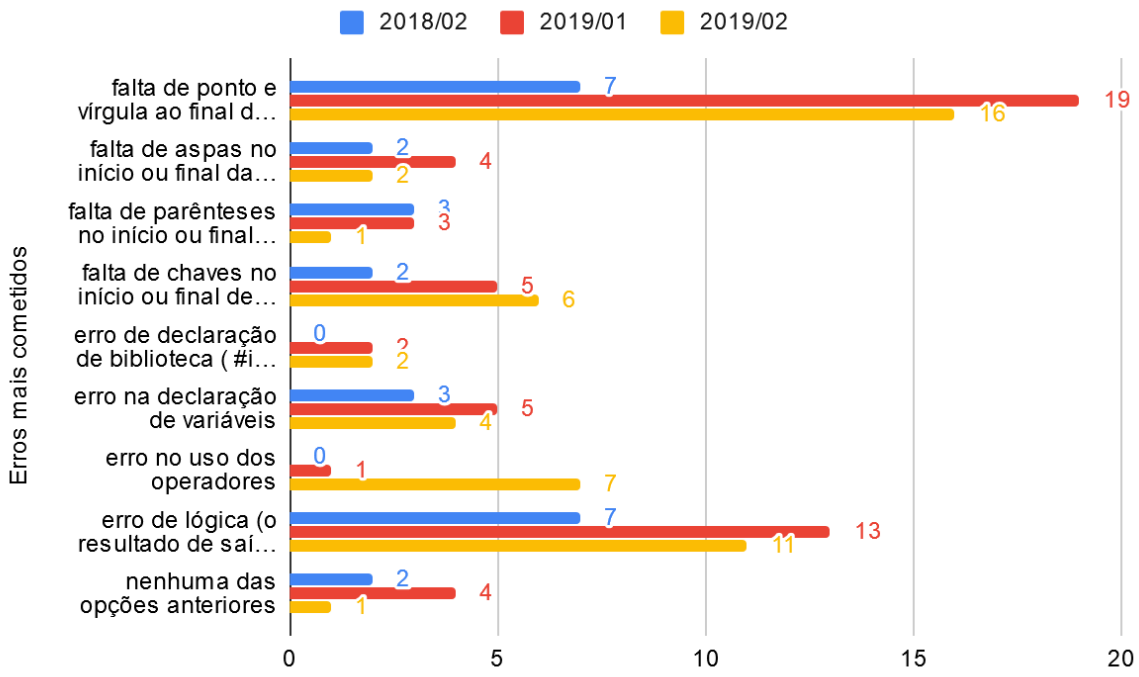
Turma 2018/02: 58,3% responderam as opções selecionadas de falta de ponto e vírgula ao final do comando (;) e o mesmo percentual disse erro de lógica (o resultado de saída da execução do programa nunca é o esperado); 25% citaram o erro na declaração de variáveis; 25% salientaram falta de parênteses no início ou final do comando (()); e empatadas com 16,7% estão as opções com os erros relacionados aos tokens da linguagem: falta de aspas no início ou final da sentença ("") e falta de chaves no início ou final de um bloco ({}).

Turma 2019/01: 63,3% indicaram a opção de falta de ponto e vírgula ao final do comando (;); 43,3% apontaram o erro de lógica (o resultado de saída da execução do programa nunca é o esperado) – ficando em primeiro e segundo lugares. Empatadas com 16,7%, estão as opções: falta de chaves no início ou final de um bloco ({}) e erro na declaração de variáveis. Com 13,3% está a opção de falta de aspas no início ou final da sentença (""); e com 6,7%, erro na declaração das bibliotecas. Neste questionário, houve um relato por escrito de erros em programas por confundir a utilização dos operadores lógicos AND e OR. Não havia esta opção entre as oferecidas como resposta na questão.

Turma 2019/02: 72,7% selecionaram a opção de falta de ponto e vírgula ao final do comando (;); 50%, o erro de lógica (o resultado de saída da execução do programa nunca é o esperado), são esses os primeiro e segundo lugares. Com 31,8% está a opção do uso incorreto dos operadores; com 27,3%, a opção de falta de aspas no início ou final da sentença (""); e com 18,2% , o uso incorreto na declaração de variáveis. As opções falta de chaves no início

ou final de um bloco ({}) e erro na declaração das bibliotecas, aparecem com o mesmo percentual de 9,1%. Por último, com 4,5%, aparece o erro por falta de parênteses no início ou no final do comando (()).

Figura 27 - Os erros mais cometidos na percepção dos alunos



Fonte: Resultado questionário do Apêndice A

Q6 - Durante a compilação de um programa aparece uma mensagem de erro e você:
Com relação às mensagens de erros durante a compilação, explorada na questão 6, das respostas obtidas:

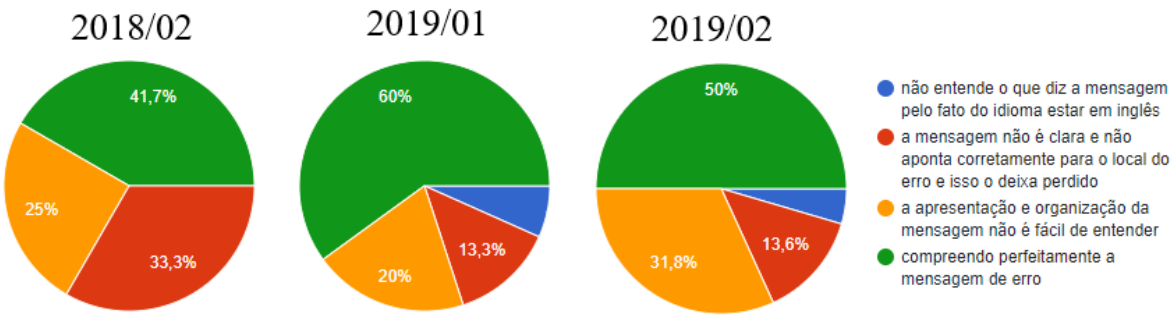
Turma 2018/02: 41,7% dos estudantes responderam que conseguem compreender perfeitamente a mensagem de erro; 33,3% disseram que não entendem as mensagens de erro porque não são claras e não apontam corretamente para o local do erro, e isso os deixa perdidos; e 25% reclamaram que a organização da mensagem não é de entendimento fácil. Um fator interessante nesta questão foi que ninguém relatou ter dificuldade na interpretação da mensagem pelo fato dela estar no idioma inglês.

Turma 2019/01: 60% dos estudantes disseram que conseguem compreender perfeitamente a mensagem de erro; 20% reclamaram que a organização da mensagem não é de entendimento fácil; 13,3% disseram que não entendem as mensagens de erro porque não são claras e não

apontam corretamente para o local do erro, e isso os deixa perdidos; e 6,7% relataram que não entendem as mensagens pelo fato do idioma estar em inglês.

Turma 2019/02: 50% dos estudantes informaram que conseguem compreender perfeitamente a mensagem de erro; 31,8% apontaram que a organização da mensagem não é de entendimento fácil; 13,6% destacaram que não entendem as mensagens de erro porque não são claras e não apontam corretamente para o local do erro, e isso os deixa perdidos; e 4,5%, relataram não entender as mensagens pelo fato do idioma estar em inglês.

Figura 28 - Qual o seu entendimento das mensagens de erro do compilador?



Fonte: Resultado questionário do Apêndice A

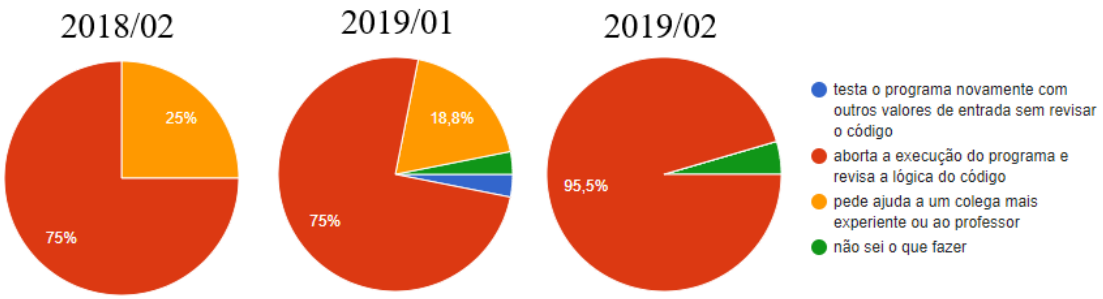
Q7 - Quando ao executar o programa ocorre um resultado inesperado na saída, você:
Quanto aos resultados inesperados na saída da execução de um programa, abordada na questão 7, e que caracterizam erros lógicos. Tem-se das respostas dos discentes,

Turma 2018/02: 75% disseram que abortam a execução do programa e revisam a lógica do código; e 25% pedem ajuda a um colega mais experiente ou ao professor.

Turma 2019/01: 80% afirmaram que abortam a execução do programa e revisam a lógica do código; 20% pedem ajuda a um colega mais experiente ou ao professor; e 3,3% para cada uma das seguintes opções: testam o programa com outros valores de entrada e não sabem o que fazer.

Turma 2019/02: 95,5% declararam que abortam a execução do programa e revisam a lógica do código; e 4,5% não sabem o que fazer.

Figura 29 - O que acontece quando ocorre um erro inesperado?



Fonte: Resultado questionário do Apêndice A

Q8 - Quando ocorre um erro no programa, me sinto:

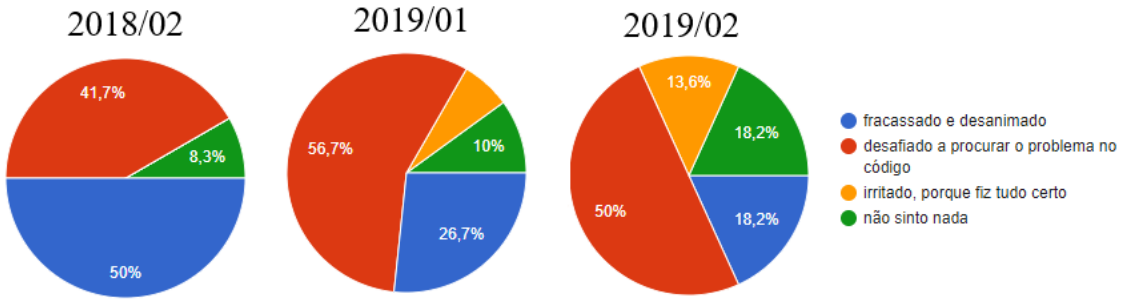
Com relação ao sentimento perante o erro, perguntada na questão 8:

Turma 2018/02: 50% dos estudantes se sentem fracassados e desanimados com o resultado; 41,7% se sentem desafiados a procurar o problema no código; e 8,3% ficam indiferentes perante o erro.

Turma 2019/01: 56,7% se sentem desafiados a procurar o problema no código; 26,7% se sentem fracassados e desanimados com o resultado; 6,7% relataram que ficaram irritados com o resultado, afinal fizeram tudo certo (na sua percepção); e 10% ficaram indiferentes perante o erro, pois ele faz parte da aprendizagem.

Turma 2019/02: 50% se sentem desafiados a procurar o problema no código; 18,2% empatadas as duas opções: fracassado e desanimado com o resultado e não sentirem nada perante o erro; 13,6% relataram que ficaram irritados com o resultado, afinal fizeram tudo certo (na sua percepção).

Figura 30 - Qual o sentimento perante o erro



Fonte: Resultado questionário do Apêndice A

Q9 - O quão importante é, para você, a análise de respostas erradas (erros) durante sua aprendizagem?

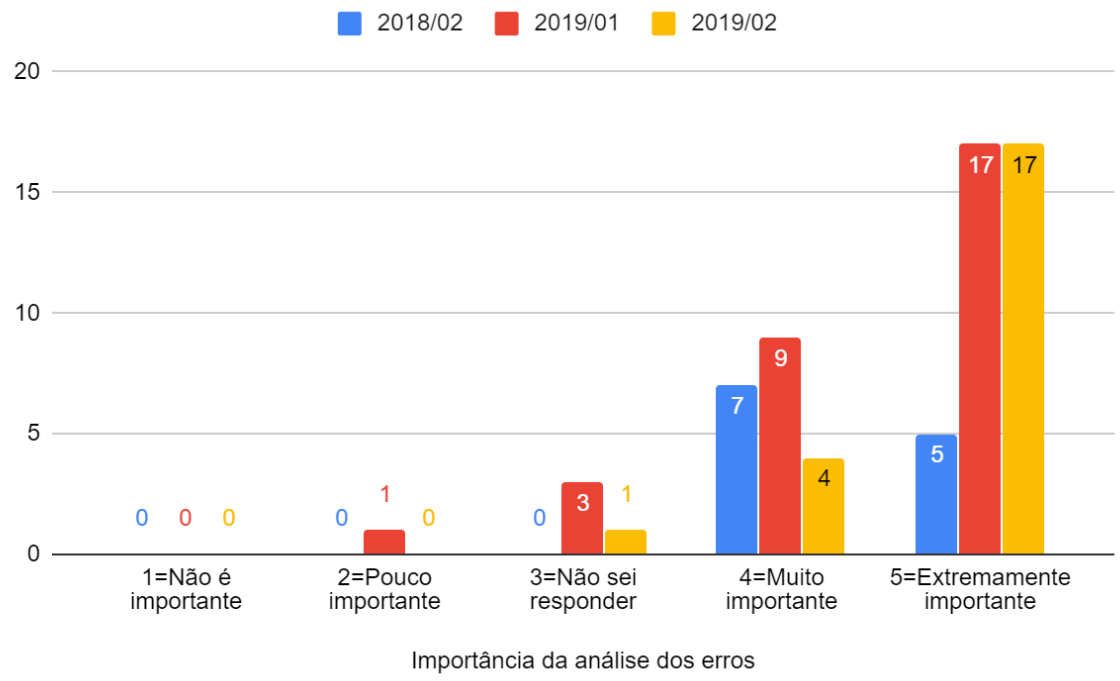
Quanto a importância em analisar os erros, indagação da questão 9:

Turma 2018/02: 58,3% dos alunos avaliaram ser muito importante; e 41,7% acharam extremamente importante, confirmando ser uma boa prática pelos alunos.

Turma 2019/01: 56,7% dos estudantes informaram ser extremamente importante; 30% acharam muito importante, confirmando ser uma boa prática pelos alunos; 10% não souberam responder; e 3,3% acham pouco importante.

Turma 2019/02: 77,33% dos discentes avaliaram ser extremamente importante; 18,2% acharam muito importante, confirmando ser uma boa prática pelos alunos; e 4,5% não souberam responder.

Figura 31 - Importância da análise das respostas erradas pelos alunos



Fonte: Resultado questionário do Apêndice A

Q10 - Com que frequência você costuma analisar seus próprios erros nas atividades em sala de aula?

Ainda nesta linha de análise dos erros e em resposta à questão 10, que trata da frequência com

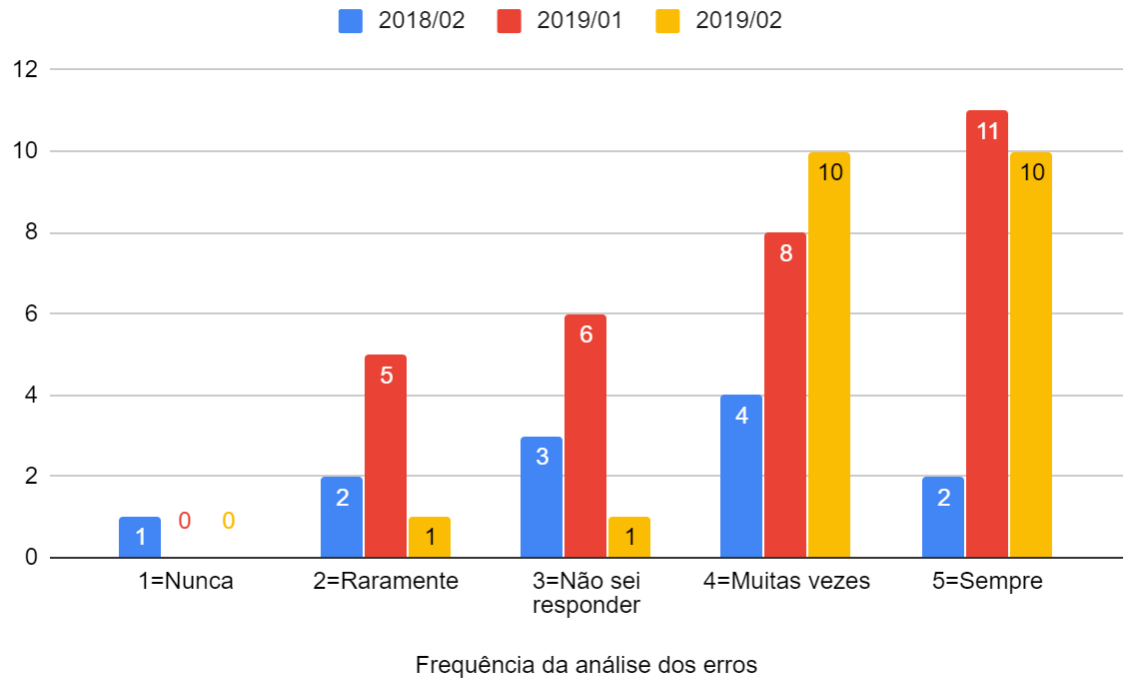
que os alunos costumam analisar seus erros nas atividades de programação em sala de aula:

Turma 2018/02: 33,3% responderam que muitas vezes analisam os erros; 16,7% sempre analisam os erros; 25% não souberam responder; 16,7% responderam que raramente analisam seus erros cometidos, apesar de terem respondido ser importante a prática de analisar as falhas ocorridas durante a programação; e 8,3% responderam que nunca analisam as respostas erradas.

Turma 2019/01: 36,7% afirmaram que sempre analisam os erros; 26,7% alegaram que, muitas vezes, analisam os erros; 20% não souberam responder; e 16,7% responderam que raramente analisam seus erros cometidos, apesar de terem respondido ser importante a prática de analisar as falhas ocorridas durante a programação. Não houve resposta para a opção nunca analisam seus erros nas atividades escolares.

Turma 2019/02: 45,5% alegaram que sempre analisam os erros; 45,5% frisaram que, muitas vezes, analisam os erros; 4,5% não souberam responder; e 4,5% responderam que raramente analisam seus erros cometidos, apesar de terem respondido ser importante a prática de analisar as falhas ocorridas durante a programação. Não houve resposta para a opção nunca analisam seus erros nas atividades escolares.

Figura 32 - Frequência da análise dos erros pelos alunos



Fonte: Resultado questionário do Apêndice A

Q11 - De que maneiras você costuma analisar suas respostas erradas?

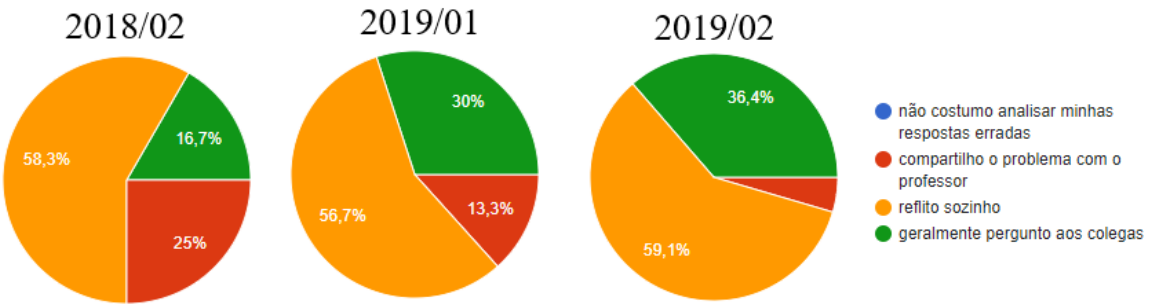
Se avaliar é importante na visão dos discentes, perguntou-se, na questão 11, de que maneira, então, eles costumavam analisar as respostas erradas:

Turma 2018/02: 58,3% responderam que refletem sozinhos as possíveis causas geradoras dos erros; 25% informaram que compartilham o problema com o professor; e 16,7% geralmente perguntam aos colegas.

Turma 2019/01: 56,7% informaram que refletem sozinhos as possíveis causas geradoras dos erros; 30% geralmente perguntam aos colegas; e 13,3% responderam compartilhar o problema com o professor.

Turma 2019/02: 59,1% responderam que refletem sozinhos as possíveis causas geradoras dos erros; 36,4% geralmente perguntam aos colegas; e 4,5% responderam compartilhar o problema com o professor.

Figura 33 - De que maneiras você costuma analisar suas respostas erradas?



Fonte: Resultado questionário do Apêndice A

Q12 - Na sua opinião, quão relevante seria uma ferramenta computacional que ajudasse na resolução dos exercícios de programação?

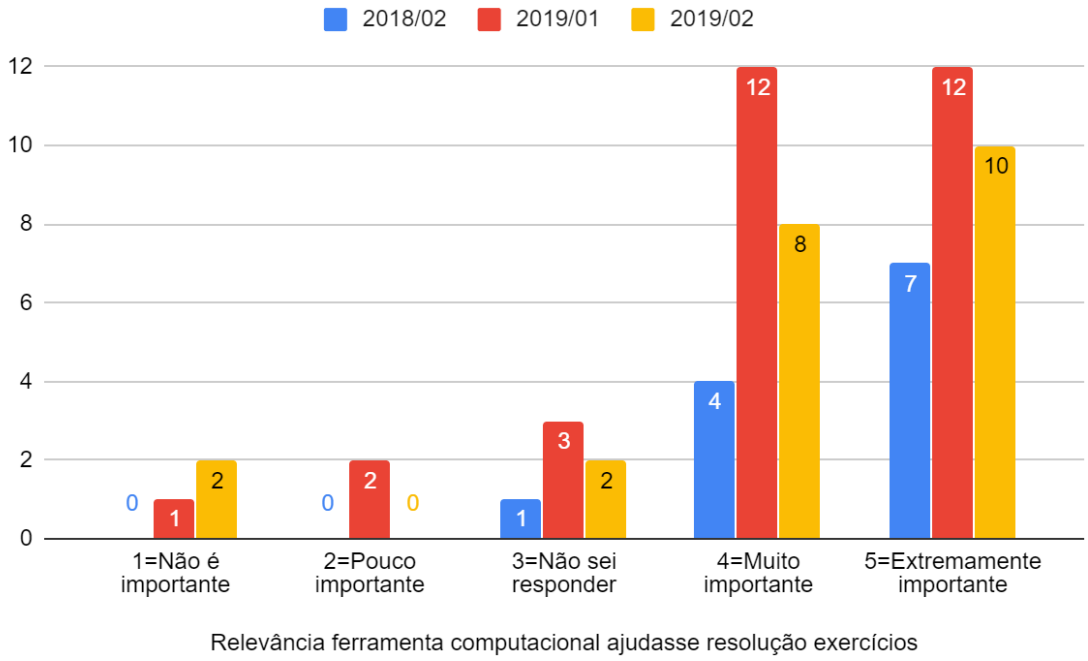
Na questão 12, foi perguntado aos discentes o quão relevante seria uma ferramenta computacional que ajudasse na resolução dos exercícios de programação e retornasse respostas indicando os problemas encontrados nos códigos e nas recomendações de materiais de estudos para reforço. Das respostas:

Turma 2018/02: 58,3% disseram achar extremamente importante e que ajudaria na aprendizagem pelo ambiente; 33,3% acharam ser muito importante; e 8,3% não souberam responder.

Turma 2019/01: 40% responderam achar extremamente importante e o mesmo percentual disse ser muito importante a criação de uma ferramenta que ajudasse na aprendizagem pelo ambiente; 10% não souberam responder; 6,7% acham pouco importante; e 3,3% relataram não ter importância.

Turma 2019/02: 45,5% evidenciaram achar extremamente importante e que ajudaria na aprendizagem pelo ambiente; 36,4% acharam ser muito importante; 9,1% não souberam responder; e 9,1% não acham importante.

Figura 34 - Relevância de uma ferramenta computacional que ajudasse na resolução dos exercícios



Fonte: Resultado questionário do Apêndice A

APÊNDICE C

QUESTIONÁRIO DE AVALIAÇÃO FERRAMENTA

Caros(as) alunos(as),

O objetivo deste questionário é avaliar as contribuições da ferramenta Codein'play no seu aprendizado durante a disciplina de Linguagem de Programação I. **Não precisa colocar o seu nome. Selecione apenas 1 opção para cada questão.**

Idade: _____ Data: _____ Gênero: [ ] M [ ] F

Com relação a ferramenta: escolha uma das opções 1=Discordo totalmente, 2= Discordo parcialmente, 3=Não sei responder, 4=Concordo parcialmente, e 5=Concordo totalmente

1. Você considera que o uso do Codein'play facilitou a resolução dos exercícios?

Discordo totalmente ① ② ③ ④ ⑤ Concordo totalmente

2. O *feedback* recebido pela ferramenta ajudou na reflexão dos erros e conseguiu localizá-los e corrigi-los com facilidade.

Discordo totalmente ① ② ③ ④ ⑤ Concordo totalmente

3. Consegui assimilar o conteúdo da disciplina através das mensagens de *feedback* recebidas através da ferramenta.

Discordo totalmente ① ② ③ ④ ⑤ Concordo totalmente

4. Consegui entender o que o enunciado dos exercícios pediam para fazer.

Discordo totalmente ① ② ③ ④ ⑤ Concordo totalmente

5. A ferramenta me ajudou na digitação da sintaxe correta dos comandos.

Discordo totalmente ① ② ③ ④ ⑤ Concordo totalmente

6. A ferramenta me ajudou a compreender e conhecer o significado o erro.

Discordo totalmente ① ② ③ ④ ⑤ Concordo totalmente

7. A ferramenta é simples e posso selecionar o exercício com o nível de complexidade proporcional ao meu conhecimento.

Discordo totalmente ① ② ③ ④ ⑤ Concordo totalmente

Quais funcionalidades disponíveis na ferramenta você considerou mais relevante para o seu aprendizado.

1=Irrelevante, 2=Pouco relevante, 3=Não sei responder, 4=Muito relevante, e 5=Extremamente relevante

8. Em relação a exibição da cobertura de teste quando entro com meus dados, quão relevante você a considerou?

Irrelevante ① ② ③ ④ ⑤ Extremamente relevante

9. Em relação execução do meu código com o teste do professor, quão relevante você a considerou?

Irrelevante ① ② ③ ④ ⑤ Extremamente relevante

10. Em relação ao ambiente estar disponível na internet, quão relevante você a considerou?

Irrelevante ① ② ③ ④ ⑤ Extremamente relevante

11. Em relação às recomendações fornecidas nos *feedbacks*, quão relevante você a considerou?

Irrelevante ① ② ③ ④ ⑤ Extremamente relevante

12. Em relação às mensagens de erro em português, quão relevante você a considerou?

Irrelevante ① ② ③ ④ ⑤ Extremamente relevante

Comentários: (sinta-se à vontade em escrever sobre as percepções e ajustes identificado na utilização da ferramenta Codein'play)

APÊNDICE D

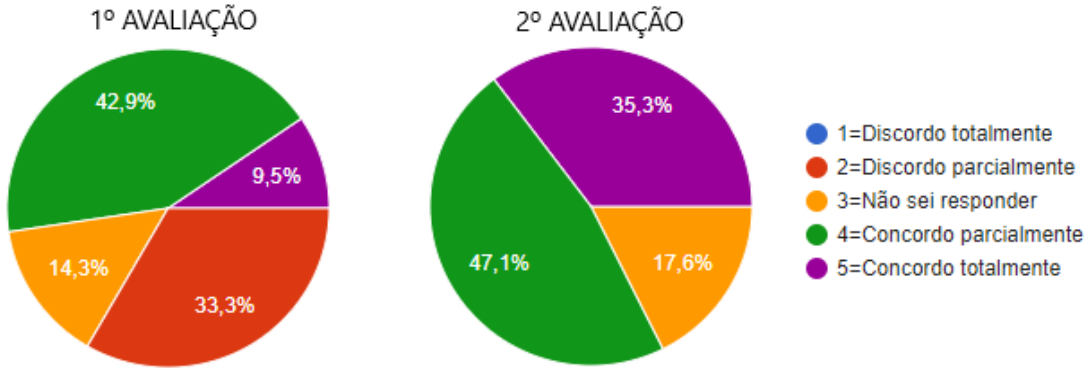
RESPOSTA DO QUESTIONÁRIO DE AVALIAÇÃO FERRAMENTA

O questionário do **Apêndice C** foi aplicado na turma 2019/02 de LPI em dois momentos. A primeira avaliação ocorreu após a utilização do Codein'play com a atividade do Simulado no mês de outubro, e a segunda avaliação aconteceu durante o uso da ferramenta com a Maratona de Programação no mês de novembro. Entre as duas atividades, ocorreu o Desafio Codein'play, que serviu de prática e subsídios para avaliar a ferramenta. Participaram da primeira avaliação 21 alunos, e na segunda avaliação 17 alunos, que responderam 12 questões fechadas e uma aberta, esta opcional. A ideia principal para justificar a aplicação do mesmo questionário em momentos distintos, foi verificar se houve mudança de opinião após o uso mais contínuo da ferramenta e o resultado é apresentado neste apêndice.

Q1-Você considera que o uso do Codein'play facilitou a resolução dos exercícios?

Esta questão objetiva verificar se a ferramenta, de um modo geral, facilitou na resolução dos exercícios, seja pela clareza do enunciado do exercício ou pelo fornecimento de dicas e *feedbacks* informativos sobre os erros ocorridos durante as execuções. Mais de 50% (52,4% na 1ª avaliação e 82,4% na 2ª avaliação) dos alunos concordam que o ambiente ajudou de alguma forma na resolução dos exercícios. Esta questão teve uma avaliação negativa na primeira aplicação do questionário (33,3%) face aos problemas de instabilidade da ferramenta, que apresentou lentidão em função da quantidade de códigos em *loop* infinito.

Figura 35 - A ferramenta facilitou a resolução dos exercícios

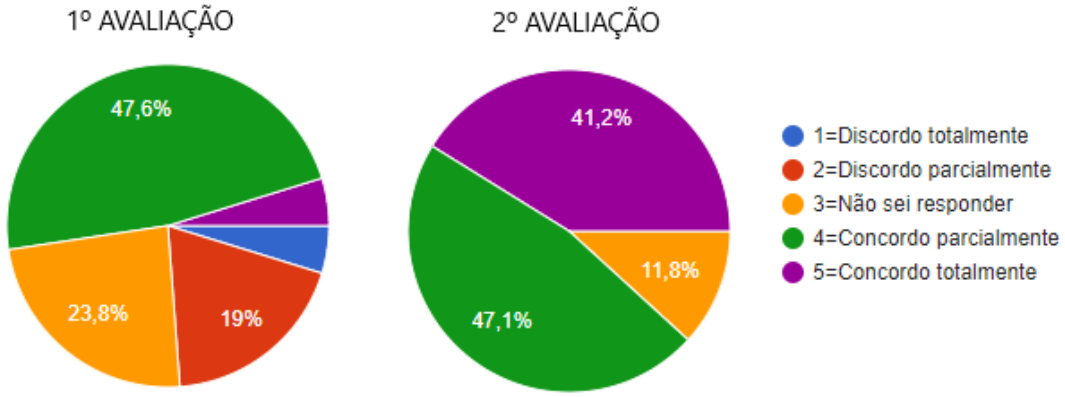


Fonte: Resultado questionário do Apêndice C

Q2-O *feedback* recebido pela ferramenta ajudou na reflexão dos erros e conseguiu localizá-los e corrigi-los com facilidade

Esta questão visa identificar se a explicação sobre o erro despertou uma reflexão sobre ele e sabendo o que aconteceu, o aluno conseguiu corrigir o problema. Mais de 50% (52,4% na 1ª avaliação e 88,3% na 2ª avaliação) dos alunos concordaram que o *feedback* gerou uma reflexão e isso possibilitou identificá-lo e corrigi-lo adequadamente.

Figura 36 - Feedback gerou reflexão sobre o erro

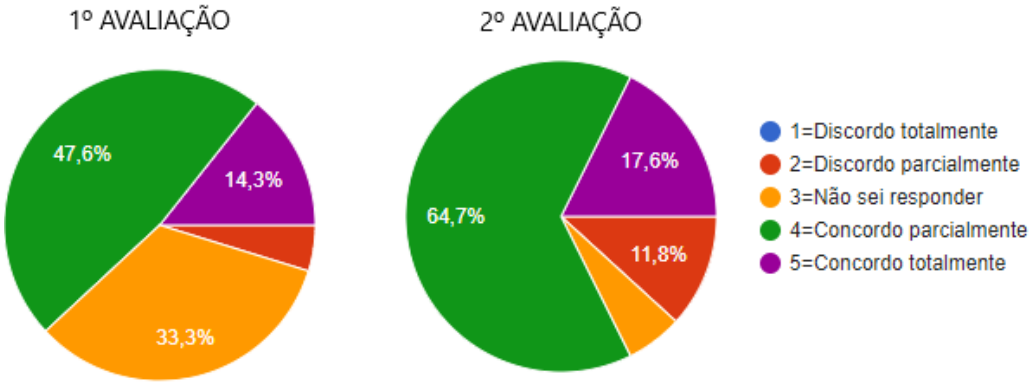


Fonte: Resultado questionário do Apêndice C

Q3-Consegui assimilar o conteúdo da disciplina através das mensagens de *feedback* recebidas através da ferramenta.

Esta questão objetiva identificar se o *feedback* ajudou o aluno na assimilação do conteúdo que ocorreu o erro, ou seja, a recomendação foi clara e esclarecedora sobre o tópico de dificuldade. Mais de 60% (61,9% e 82,3%) dos discentes concordaram que as mensagens de *feedback* auxiliaram no entendimento do conteúdo em dificuldade ao apresentarem exemplos e sugestões de materiais extras, seguindo as perguntas da estratégia de *feedback*.

Figura 37 - Consegui assimilar o conteúdo através dos *feedbacks*

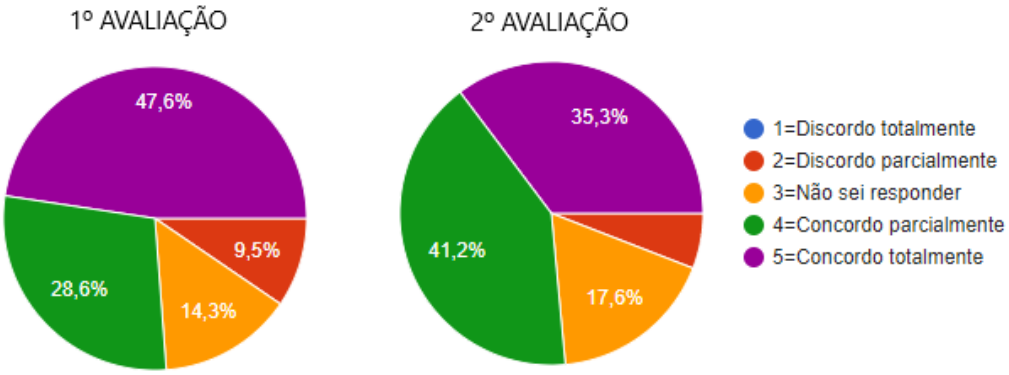


Fonte: Resultado questionário do Apêndice C

Q4-Consegui entender o que o enunciado dos exercícios pedia para fazer.

Este item avalia a escrita dos enunciados das questões. Na redação dos enunciados, houve uma preocupação em deixar o texto o mais claro possível, não dando margem a diversas interpretações sobre o que deveria ser feito. Conforme os dados coletados nas duas avaliações, mais de 76% (76,2% na 1º avaliação e 76,5% na 2º avaliação) dos estudantes informaram que os enunciados ajudaram no entendimento do que precisava fazer, e foi reforçado com o relato de um aluno "*O enunciado dos exercícios também são claros e ajudam na resolução*".

Figura 38 - Os enunciados ajudaram no entendimento do que precisava ser feito

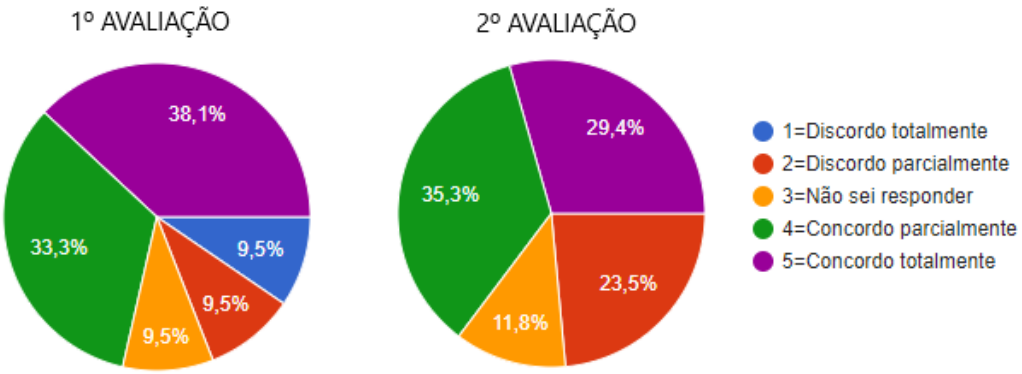


Fonte: Resultado questionário do Apêndice C

Q5-A ferramenta me ajudou na digitação da sintaxe correta dos comandos.

Esta questão avalia a IDE (ace) e seus recursos disponibilizados na ferramenta Codein'play. Após o Simulado, foi desabilitada a propriedade "*enableLiveAutocompletion*" da IDE, na qual o *plugin* sugeria e autocompletava o comando na medida que o usuário digitava, por sugestão de um aluno que relatou: "*Ao digitar uma palavra, o programa autocompletava e para evitar, precisava apertar o ESC, o que atrapalhou na digitação, pois muitas vezes não era a palavra esperada. Caso fosse teclado espaço, ou enter, a palavra se autocompletava com a sugestão.*", optou-se, em acordo com o estudante, deixar o funcionamento da área de programa semelhante ao recurso disponível no DevC++, ou seja, as teclas ctrl+espaço sugerem o comando digitado. Essa alteração refletiu nas atividades do Desafio e da Maratona. Mais de 60% (71,4% na 1º avaliação e 64,7% na 2º avaliação) dos discentes concordaram que a ferramenta, de certa forma, auxiliou na sintaxe dos comandos.

Figura 39 - A IDE da ferramenta ajudou na digitação correta da sintaxe dos comandos



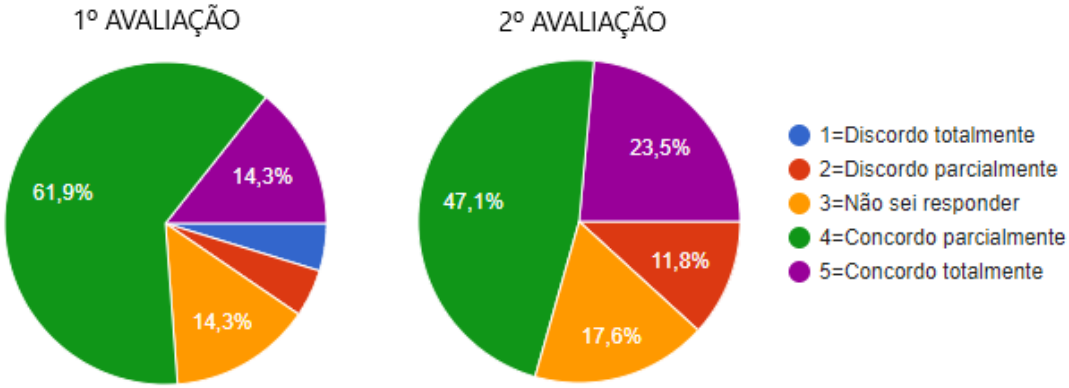
Fonte: Resultado questionário do Apêndice C

Outro aluno aprovou a ferramenta e relatou "*Gostei muito da interface e dos Snippets disponíveis.*"

Q6-A ferramenta me ajudou a compreender e conhecer o significado o erro.

Esta questão objetiva saber se o ambiente contribuiu no entendimento sobre erro, seja pela opção de tradução do erro, ou pelos *feedbacks* informativos exibidos. Esta questão é a mais importante da pesquisa, em função deste trabalho pretender mediar o erro e o ambiente ser a via para isso. O resultado deste estudo foi que mais de 70% (76,2% na 1ª avaliação e 70,6% na 2ª avaliação) dos estudantes alegaram que a ferramenta auxiliou no entendimento do erro de alguma forma.

Figura 40 - A ferramenta auxiliou no entendimento e conhecimento sobre o erro



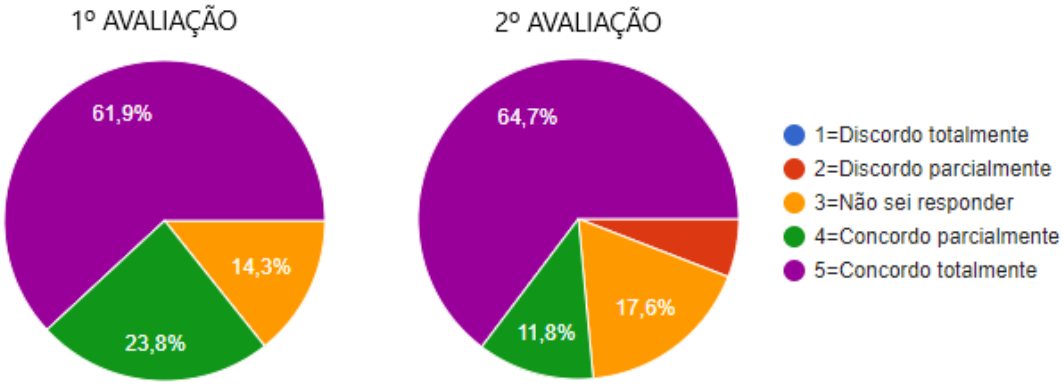
Fonte: Resultado questionário do Apêndice C

Q7-A ferramenta é simples e posso selecionar o exercício com o nível de complexidade proporcional ao meu conhecimento.

Esta questão avalia a usabilidade da ferramenta e a facilidade de os alunos selecionarem os

exercícios de acordo com a seu conhecimento. Mais de 70% (85,7% na 1ª avaliação e 76,5 na 2ª avaliação) dos alunos concordam que a interface da ferramenta é fácil de utilizar, o que confirma com o relato de um aluno: "*Achei uma ferramenta simples e intuitiva. Gostaria de algo assim disponível para fazer exercícios e praticar*". No entanto, houve uma relato sobre a posição confusa dos botões na questão: "*coloração e posicionamento dos botões um pouco confusa*"; a necessidade de aumento de contraste da tela para realçar os elementos da interface: "*Aumentar o contraste das cores para identificar os elementos*."; e a possibilidade de uso de teclas de atalho para as funcionalidades disponibilizadas nos botões na área de entrada de dados/botões: "*Atalhos de teclado ajudariam :)*" e "*Senti falta de poder usar instruções condicionais sem o uso das chaves e das teclas de atalho.*". As teclas de atalho foram implementadas após a atividade de Simulado.

Figura 41 - Simplicidade e facilidade da ferramenta

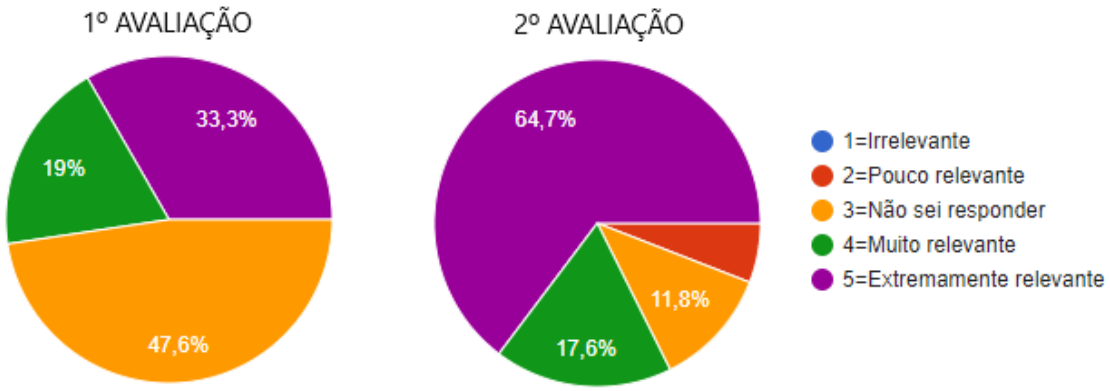


Fonte: Resultado questionário do Apêndice C

Q8-Em relação a exibição da cobertura de teste quando entro com meus dados, quão relevante você a considerou?

Esta questão mede a relevância de exibir a cobertura do código para o aluno, mostrando por onde o compilador passou destacando as linhas executadas e as não executadas, de acordo com os dados de entrada informados. Na 1ª avaliação, os alunos não entenderam e não sabiam o que era uma cobertura de código, tendo sido explicado pelo professor o conceito e os pontos positivos ao se realizar a análise dos caminhos percorridos pelo compilador para que o código possa ser refatorado, isto é, melhorado, removendo trechos de instruções desnecessários. Essa explicação refletiu na avaliação desta questão na segunda aplicação do questionário, quando mais de 80% acharam relevante este recurso e relataram: "*A demonstração das linhas executadas também é muito útil*".

Figura 42 - A cobertura de código é relevante para o aprendizado

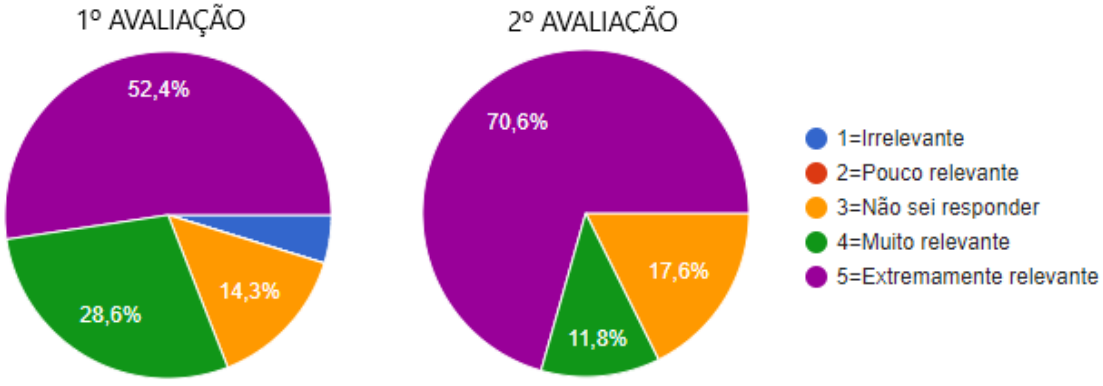


Fonte: Resultado questionário do Apêndice C

Q9-Em relação execução do meu código com o teste do professor, quão relevante você a considerou?

Esta questão aborda a facilidade de execução dos testes automaticamente, o que, segundo os relatos dos alunos: "*A ferramenta no geral é muito boa, principalmente a parte dos testes.*" e "*Os testes são muito bons*", foi uma das funcionalidades que mais agradou. Mais de 80% (81% na 1ª avaliação e 82,4% na 2ª avaliação) dos estudantes a acharam relevante na ferramenta. Assim, eles se concentram em solucionar o problema e os testes cadastrados pelo professor é que irão avaliar se o resultado obtido do código está de acordo com o resultado esperado pelo professor.

Figura 43 - Relevância dos testes automáticos



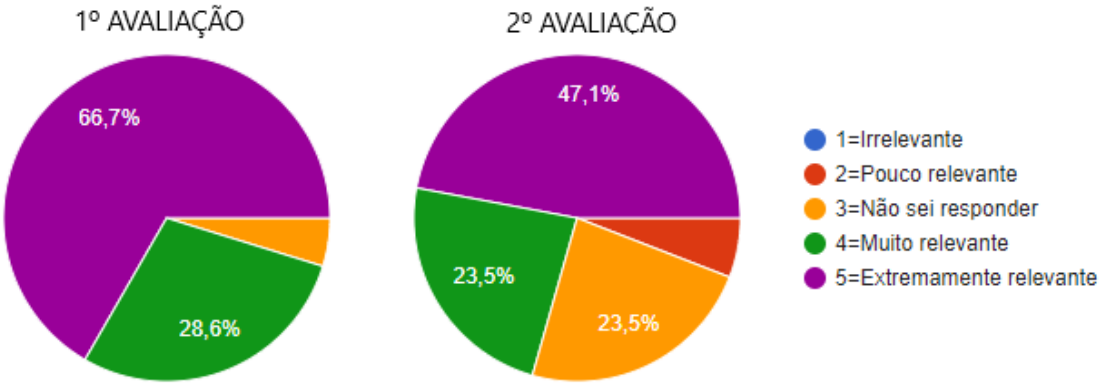
Fonte: Resultado questionário do Apêndice C

Q10-Em relação ao ambiente estar disponível na internet, quão relevante você a considerou?

Esta questão avalia a relevância do ambiente estar *online*. Mais de 70% (95,3% na 1ª

avaliação e 70,6% na 2ª avaliação) dos respondentes acharam essencial a ferramenta estar disponível para acessar de qualquer lugar e em qualquer dispositivo (ambiente responsivo). No entanto, aumentou o percentual de alunos que não souberam responder da 1ª avaliação para a 2ª avaliação, talvez, porque a IDE DevC++ seja gratuita e de fácil instalação, tornando este quesito irrelevante para o aprendizado.

Figura 44 - Relevância do ambiente estar *online*

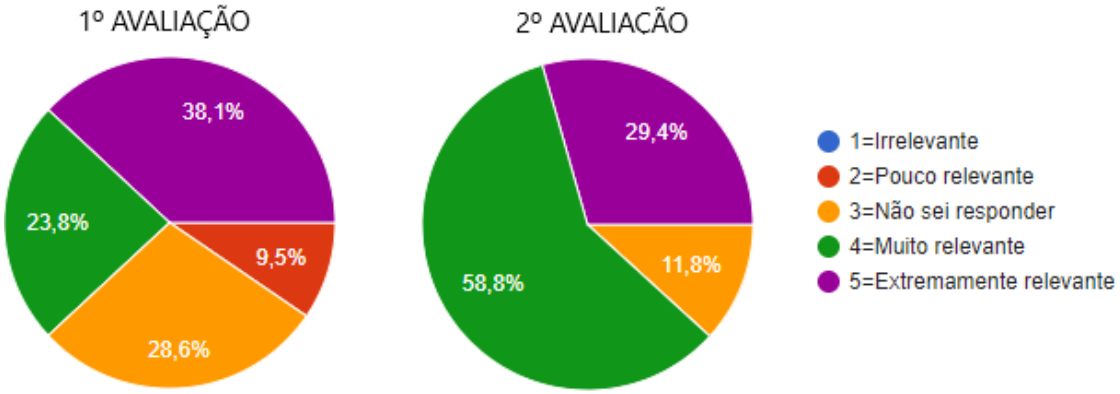


Fonte: Resultado questionário do Apêndice C

Q11-Em relação às recomendações fornecidas nos *feedbacks*, quão relevante você a considerou?

Esta questão objetiva saber se o *feedback* informativo ajudou no entendimento sobre erro e despertou uma reflexão sobre ele antes de realizar a correção da falha no código. É a mais importante da pesquisa, pois este estudo pretende mediar o erro e o ambiente é via para isso. Relembrando, com a intenção de elaborar os *feedbacks* para as execuções que falharam, foi empregada uma estratégia de *feedback* para sistematizar e padronizar o mapeamento e os registros dos erros e suas recomendações. Porém, para se chegar a eles e, consequentemente, às suas recomendações, fez-se um levantamento dos erros mais cometidos através da análise dos códigos-fontes entregues pelos alunos, conforme descrito na seção 8.3. Apesar de mais de 75% (61,9% na 1ª avaliação e 88,2% na 2ª avaliação) dos alunos acharem relevantes as informações apresentadas nos *feedbacks*, houve relato de que a recomendação não se encaixava no erro apresentado: "*Na explicação dos erros as vezes ficava um pouco confuso a interpretação do erro no meu código com a explicação*", o que de fato pode ocorrer, uma vez que alguns erros são seguidos de *warnings*. A ferramenta exibe todas as recomendações encontradas na base de dados que estejam relacionadas com as ocorrências de erro sintático e *warnings* deparados pelo compilador.

Figura 45 - Relevância sobre as recomendações fornecidas nos *feedbacks*

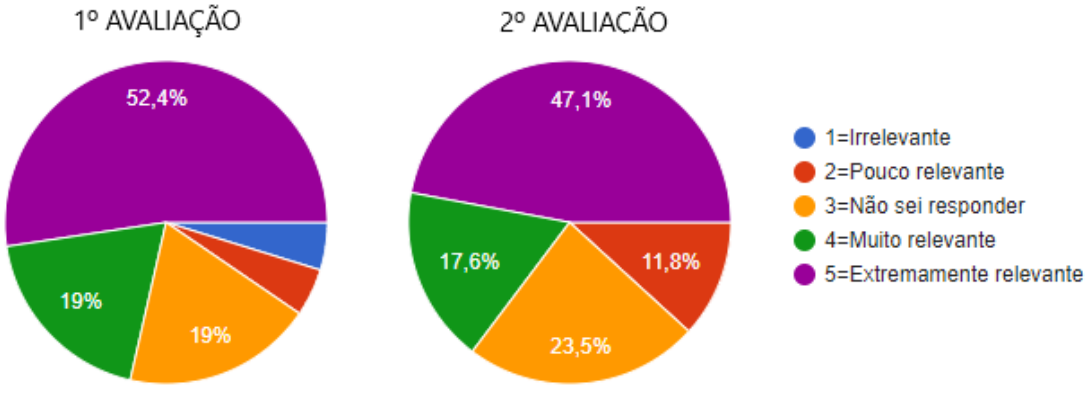


Fonte: Resultado questionário do Apêndice C

Q12-Em relação às mensagens de erro em português, quão relevante você a considerou?

Esta questão visa avaliar a tradução das mensagens emitidas pelo compilador. Mais de 60% (71,4% na 1ª avaliação e 64,7% na 2ª avaliação) dos alunos afirmaram ser relevante a funcionalidade de tradução das mensagens do compilador. No entanto, houve um aumento de 4,5% entre a 1ª e a 2ª avaliação, talvez em função da tradução ficar confusa em alguns momentos, conforme o relato de um aluno: "*Às vezes o feedback vinham traduzidos para o português de forma literal. Em alguns momentos a mensagem ficou confusa.*". Outra hipótese é que, conforme a pesquisa do **Apêndice B**, 50% dos alunos informaram que entendem perfeitamente as mensagens exibidas pelo compilador, tornando este quesito irrelevante para o aprendizado.

Figura 46 - Relevância da tradução das mensagens do compilador



Fonte: Resultado questionário do Apêndice C

A tradução da mensagem do compilador do inglês para o português é feita por meio de

uma API de tradução PHP *Google Translate* instalado no projeto, o que, realmente, pode ficar estranho à primeira vista.

APÊNDICE E

MAPEAMENTO DOS ERROS

O mapeamento dos erros ocorreu de três maneiras: através da análise dos códigos das provas dos alunos que não atingiram nota superior a 7,0 nos semestres de 2018/02, 2019/01 e 2019/02; por meio de informações divulgadas nos sites relacionados a erros do compilador GCC; e mediante o recurso que registra os erros e alertas no Codein'play, que ocorrem durante a execução da questão e que não possui um *feedback* associado. O Quadro 20 apresenta todas as mensagens cadastradas no juiz *online* levantadas até 29 de Novembro de 2019.

Quadro 20 - Erros e alertas (*warnings*) cadastrados na ferramenta

	Lista de erros e alertas *(warnings)*	
01	Mensagem:	**expected ';'**
	Origem:	Código do aluno
	Tipo:	Error
	Feedback:	O erro ***expected ';'*** indica que você provavelmente está esquecendo um ponto e vírgula (;) no final de algum comando. Todas as instruções na linguagem C requerem o ponto e vírgula no final para concluir o comando.
02	Mensagem:	**expected expression before**
	Origem:	Código do aluno
	Tipo:	Error
	Feedback:	O erro ***expected expression before " token*** ocorre quando algo está faltando para completar a sintaxe correta do comando, tais como: um operando ou uma variável. Veja os exemplos: if (ret ==) o operando direito de == não foi informado. printf("\n NUMERO DA SALA %d",) a variável está faltando Revise a sintaxe dos comandos utilizados no seu código e identifique se algum operando ou variável não esteja faltando.
03	Mensagem:	**parse error at end of input**
	Origem:	Sites
	Tipo:	Error
	Feedback:	O erro ***parse error at end of input*** indica que provavelmente tem uma chave de fechamento ausente (}) em algum lugar de seu programa.

		Veja o exemplo:
		
		`int main (void) {`
		`  if (1) {`
		`   printf ("Hello World!\n");`
		`    return 0; /* chave final não fechada */`
		`}`
		
		O comando if não teve suas chaves fechadas. Revise o código para identificar uma situação semelhante.
		
		Dica: Realize a indentação dos comandos para que o código fique visivelmente organizado.

04	Mensagem:	**expected declaration or statement at end of input**
	Origem:	Código do aluno/sites
	Tipo:	Error

	Feedback:	O erro ***expected declaration or statement at end of input*** indica que provavelmente tem uma chave de fechamento ausente (}) em algum lugar de seu programa.
		
		Veja o exemplo:
		
		`int main (void) {`
		`  if (1) {`
		`   printf ("Hello World!\n");`
		`    return 0; /* no closing brace */`
		`}`
		
		O comando if não teve suas chaves fechadas. Revise o código para identificar uma situação semelhante.
		
		Dica: Realize a indentação dos comandos para que o código fique visivelmente organizado.

05	Mensagem:	**ld returned 1 exit status**
	Origem:	Código do aluno
	Tipo:	Error
	Feedback:	O erro ***ld returned 1 exit status*** ocorre quando um programa usa uma função ou variável que não está definida em nenhum dos arquivos ou bibliotecas de objetos fornecidos ao compilador. Também, pode ser causado por uma biblioteca ausente ou pelo uso incorreto do nome ou função. Veja o exemplo:

		int main(void)
		{
		  int num;
		  scanf("%d", &num);
		
		  if(num > 0) {
		  print("maior que zero");
		  }
		  return 0;
		}
		O erro ocorreu porque o comando print não existe na linguagem C, mas printf() sim, cuja sintaxe é:
		
		printf("Conversão/Formato do argumento",argumentos);
		
		Onde:
		
		<Conversão/Formato do argumento> como as mensagens que serão exibidas
		
		<argumentos> pode conter identificadores, expressões aritméticas ou lógicas e valores fixos
		
		No nosso curso no moodle na aula sobre variáveis possui mais informações sobre o seu uso e declaração. Você pode acessar, também, o livro digital C Completo e Total da Makron Books.

06	Mensagem:	**expected constructor, destructor, or type conversion before**
	Origem:	Sites
	Tipo:	Error

	Feedback:	O erro ***expected constructor, destructor, or type conversion before*** ocorre quando há uma chamada de um comando dentro de uma função. Em C, não é possível adicionar código executável no corpo de um cabeçalho ou arquivo de implementação (.c, .cpp, .cxx, etc ...). Em vez disso, deve-se adicioná-lo dentro de uma função, como na função main().
		
		Veja o exemplo:
		
		int main(void) {
		  int soma;
		  soma = 1+2;
		}
		printf("A soma eh: %d", soma);
		
		Observe que o comando prinft() está fora da função main(). Verifique em seu código se não há uma situação semelhante.

07	Mensagem:	**undefined reference to**
	Origem:	Código do aluno

	Tipo:	Error

	Feedback:	O erro de ***undefined reference*** significa que você tem uma referência a um nome (função, variável, constante, etc.) em seu programa que o compilador não consegue encontrar em nenhuma biblioteca que compõe o projeto.
		
		Veja o exemplo:
		
		`int main(void) {`
		` int soma;`
		` soma = 1+2;`
		` print("A soma eh: %d ", soma);`
		`}`
		
		O erro ocorreu porque o comando print não existe na linguagem C, mas printf(), cuja sintaxe é:
		
		`printf("expressão de controle",argumentos);`
		
		Mais informações sobre o uso do printf() acesse o material de aula no Moodle.

08	Mensagem:	**undeclared**
	Origem:	Código do aluno
	Tipo:	Error

	Feedback:	O erro ***undeclared (first use in this function)*** ocorreu em função do uso de uma variável desconhecida. Na linguagem C, as variáveis devem ser declaradas antes de serem utilizadas. A mensagem de erro exibida indica que o compilador encontrou um nome de variável que não tem uma declaração correspondente. Pode ser causado por uma declaração inexistente ou por um erro de digitação no nome. Lembre-se que na Linguagem C os nomes de variáveis fazem distinção entre maiúsculas e minúsculas, ou seja, são Case Sensitive. Portanto, "idade" e "Idade", apesar de terem o mesmo nome, são variáveis diferentes e ocupam espaços de memória distintos.
		
		Para declarar uma variável você deverá informar primeiro o tipo de dado e depois o nome. No decorrer do programa utilize sempre estas mesmas variáveis. Atente-se que o nome da variável não pode ter espaços e deve possuir até 32 caracteres.
		
		Veja o exemplo:
		
		`int main(void) {`
		` float nota1, nota2;`
		` scanf("%f", &nota1);`
		` scanf("%f", &nota2);`
		` printf(“A media eh %.2f ”, (nota+nota2)/2);`

		} Observe que a variável nota não existe nas declarações de variáveis, mas nota1. No nosso curso no moodle na aula sobre variáveis possui mais informações sobre o seu uso e declaração. Você pode acessar, também, o livro digital C Completo e Total da Makron Books. Para entretenimento, assista o vídeo no youtube Variáveis e Tipos de Dados.
09	Mensagem:	**expected 'while' before**
	Origem:	Código do aluno
	Tipo:	Error
	Feedback:	O erro ***expected 'while' before*** ocorreu porque o comando do..while não está completo, sendo o do a parte faltante. Veja o exemplo: A sintaxe do comando 'do..while' é: do { statements } while (condition); verifique se você não aconteceu algo semelhante no seu código. No nosso curso no moodle na aula sobre Sentença de iteração - Laço (FOR WHILE DO/WHILE) possui mais informações sobre o seu uso e declaração. Você pode acessar, também, o livro digital C Completo e Total da Makron Books.
10	Mensagem:	**invalid operands of type float and float to binary operator ***
	Origem:	Código do aluno
	Tipo:	Error
	Feedback:	O ***erro invalid operands of types float and float to binary operator %*** ocorreu porque o operador de módulo (%) não pode ser utilizado com variáveis de tipo float e double. Ele é usado para obter o resto da divisão, quando o tipo inteiro x é dividido por y. Verifique o tipo de variável que você está utilizando para realizar a operação com o operador módulo.
11	Mensagem:	**was not declared in this scope**
	Origem:	Código do aluno
	Tipo:	Error

	Feedback:	O erro ***was not declared in this scope*** ocorreu porque provavelmente está sendo utiliza um comando ou uma variável que não está declarada dentro da função. Veja o exemplo: `int voto, cA = 0, cB = 0, Cc = 0,Dc = 0; br = 0, nul = 0;` Observe que existe um ponto e vírgula depois da variável Dc = 0 quebrando a declaração das variáveis. Verifique se não há um ponto e vírgula no lugar da vírgula na linha em que é feita a declaração das variáveis, ou então, analise se o comando que você está utilizando existe e está sintaticamente correto.
12	Mensagem:	**too few arguments to function**
	Origem:	Código do aluno
	Tipo:	Error
	Feedback:	O ***erro too few arguments to function*** significa que o número de argumentos reais (os argumentos que serão fornecidos ao chamar a função) não é igual ao número de argumentos formais (os argumentos que são declarados durante a definição de uma função) ou existe uma vírgula finalizando a lista de argumentos. Veja o exemplo: `int menor (int a, int b, int c, int d,) {}` Observe que no final da lista de argumentos da função menor() existe uma vírgula indicando que outro argumento virá a seguir. Verifique se uma situação semelhante não está ocorrendo em seu código.
13	Mensagem:	**character constant too long**
	Origem:	Sites
	Tipo:	Error

	Feedback:	O erro character constant too long ocorre quando se utiliza aspas simples (') para representar uma cadeia de caracteres, ou seja, forem usadas para incluir mais de um caracter. Em C, as cadeias de caracteres devem estar entre aspas duplas (""). Quando se utiliza aspas simples ('), o C espera encontrar um único caractere (ou um caractere de escape, como por exemplo: '\n').
		
		Veja o exemplo:
		
		`#include <stdio.h>`
		`int main (void)`

		{
		printf ('Hello World!\n'); /* wrong quotes */
		return 0;
		}
		
		Observe que foi utilizado as aspas simples (') dentro do comando printf() para imprimir uma cadeia de caracteres. Verifique se esta situação está acontecendo em seu código.

14	Mensagem:	**initialization makes integer from pointer without a cast**
	Origem:	Sites
	Tipo:	Error

	Feedback:	O erro ***initialization makes integer from pointer without a cast*** indica um uso indevido de um ponteiro em um contexto inteiro. Tecnicamente, é possível conversão entre tipos inteiro e ponteiro, mas isso raramente é necessário fora dos aplicativos no nível do sistema. Com mais freqüência, esse aviso é o resultado do uso de um ponteiro sem desmarcá-lo (por exemplo, escrevendo int i = p em vez de int i = * p). Esse erro também pode ocorrer com os tipos char e char *, pois char é um tipo inteiro.
		
		Veja os exemplos:
		
		int main (void)
		{
		char c = "\n"; /* incorrect */
		return 0;
		}
		
		A variável c tem o tipo char, enquanto a string "\n" é avaliada como um ponteiro const char * (para uma região de 2 bytes da memória que contém o valor ASCII para nova linha seguida por um byte zero '\0'). O código ASCII para nova linha deve ser utilizando como char c = '\n' com aspas simples. Erros semelhantes ocorrem, também, com o uso incorreto do NULL.
		
		int main (void)
		{
		int i = NULL; /* incorrect */
		return 0;
		}
		
		Em C, o NULL é definida como ((void *) 0) em 'stdlib.h' e deve ser usada apenas com ponteiros.

15	Mensagem:	**multiple types in one declaration**
	Origem:	Sites
	Tipo:	Error
	Feedback:	O erro ***multiple types in one declaration*** ocorre porque toda declaração

		de variável deve individual, em sua própria linha, com um comentário explicativo sobre o papel da variável. Declarar várias variáveis em uma única declaração pode causar confusão em relação aos tipos das variáveis e seus valores iniciais. Se mais de uma variável for declarada em uma declaração, deve-se tomar cuidado para que o tipo e o valor inicial da variável sejam manipulados corretamente. Veja o exemplo: int float x; Observe que nesta declaração o tipo de dados int não possui uma variável associada. Em C, para declarar uma variável você deverá informar primeiro o tipo de dado e depois o nome. No decorrer do programa utilize sempre estas mesmas variáveis. Atente-se que o nome da variável não pode ter espaços e deve possuir até 32 caracteres. Sintaxe: tipo_de_dados <nome_da_variável>; Exemplo de declaração de variável do tipo inteiro: int contador; onde: ● int é o tipo da variável (inteiro) ● contador é o nome da variável. Os tipos de dados mais comuns em linguagem C, são: ● int: armazena valores numéricos inteiros. ● char: armazena caracteres. ● float: armazena números com ponto flutuante (reais) com precisão simples. ● double: armazena números com ponto flutuante, com precisão dupla, ou seja normalmente possui o dobro da capacidade de uma variável do tipo float. Verifique se não existem declarações de variáveis com tipos diferentes na mesma linha. Mais informações sobre declaração de variáveis, acesse o nosso curso no Moodle na aula sobre Variáveis e o post do *Eduardo Casavella*.
16	Mensagem:	**unterminated string of character constant**
	Origem:	Sites
	Tipo:	Error
	Feedback:	O erro ***unterminated string or character constant*** é causado por uma abertura de string ou por uma citação de caractere que não possui o fechamento correspondente. As aspas devem ocorrer em pares

		correspondentes, ou seja, as aspas simples 'a' para caracteres ou aspas duplas "aaa" para seqüências de caracteres.
		
		Veja o exemplo:
		
		`int main (void)`
		`{`
		`printf ("Hello World!\n); /* no closing quote */`
		`return 0;`
		`}`
		
		Observe que a sequência de caracteres Hello World dentro do comando printf() não foi fechada. Verifique se esta situação está ocorrendo em seu código.

17	Mensagem:	**implicit declaration of function**
	Origem:	Sites
	Tipo:	Error

	Feedback:	O erro ***implicit declaration of function*** indica que a função utilizada no código não possui uma biblioteca inicializada.
		
		Veja o exemplo:
		
		`int main (void)`
		`{`
		`printf ("Hello World!\n"); /* no header */`
		`return 0;`
		`}`
		
		O arquivo de cabeçalho do sistema 'stdio.h' não está incluído, e com isso, o comando de printf() não está declarado. O programa precisa de uma linha inicial #include no início do código.
		
		Maiores informações sobre as funções e bibliotecas na linguagem C acesse o documento Bibliotecas Standardizadas. (http://www.di.ubi.pt/~operativos/praticos/pdf/6-standardlibs.pdf)

18	Mensagem:	**dereferencing pointer to incomplete type**
	Origem:	Sites
	Tipo:	Error
	Feedback:	O erro ***dereferencing pointer to incomplete type*** ocorre quando um programa tenta acessar os elementos de struct através de um ponteiro sem o layout da estrutura sendo declarado primeiro. Em C, é possível declarar ponteiros para estruturas antes de declarar seu layout de estrutura, desde que os ponteiros não sejam desreferenciados. Isso é conhecido como declaração de encaminhamento. Veja o exemplo:

		struct btree * data;
		int main (void)
		{
		data->size = 0; /* incomplete type */
		return 0;
		}
		
		Observe que data->size não pertence a estrutura. Mais informações sobre struct na linguagem C, acesse o post do *Eduardo Casavella*

19	Mensagem:	**unknown escape sequence**
	Origem:	Sites
	Tipo:	Error

	Feedback:	O erro ***unknown escape sequence*** é causado por um uso incorreto do caractere de escape em uma seqüência de caracteres. Sequências de escape válidas são:
		
		● \n newline
		● \t tab
		● \b backspace
		● \r retorno de carro
		● \f feed de formulário
		● \v guia vertical
		● \um alerta (sino)
		
		As combinações \\, \ ', \ "e \? Podem ser usadas para caracteres individuais. As seqüências de escape também podem usar códigos octal \ 0 - \ 377 e códigos hexadecimais \ 0x00 - \ 0xFF.
		
		Veja o exemplo:
		
		#include <stdio.h>
		int main (void)
		{
		printf ("HELLO WORLD!\N");
		return 0;
		}
		
		Observe que o escape \N não é um escape válido. Verifique se esta situação não está ocorrendo em seu código.

20	Mensagem:	**suggest parentheses around assignment used as truth value**
	Origem:	Sites
	Tipo:	Error
	Feedback:	O erro ***suggest parentheses around assignment used as truth value*** destaca um erro potencialmente grave, usando o operador de atribuição '=' em vez do operador de comparação '==' no teste de uma instrução condicional ou outra expressão lógica. Embora o operador de atribuição possa ser usado como parte de um valor lógico, esse raramente é o

<table>
<tr><td></td><td></td><td>comportamento pretendido.

Veja o exemplo:

<pre>#include <stdio.h>
int main (void)
{
 int i = 0;
 if (i = 1) { /* = should be == */
  printf ("unexpected result\n");
 }
 return 0;
}</pre>
Observe que o operador '=' foi utilizado equivocadamente no comando de decisão if para verificar o valor da variável i. Para corrigir este erro, utilize o operador de comparação '=='.
Na linguagem C, existem os operadores relacionais que diferem a sua utilização, conforme mostra a tabela abaixo:

<table><tr><th>Símbolo</th><th>Operador</th><th>Exemplo</th><th>Significado</th></tr><tr><td>></td><td>Maior que</td><td>x > y</td><td>x é maior que y?</td></tr><tr><td>>=</td><td>Maior ou igual</td><td>x >= y</td><td>x é maior ou igual a y?</td></tr><tr><td><</td><td>Menor que</td><td>x < y</td><td>x é menor que y?</td></tr><tr><td><=</td><td>Menor ou igual</td><td>x <= y</td><td>x é menor ou igual a y?</td></tr><tr><td>==</td><td>Igualdade</td><td>x == y</td><td>x é igual a y?</td></tr><tr><td>!=</td><td>Diferente de</td><td>x != y</td><td>x é diferente de y?</td></tr></table>
Maiores informações sobre sua utilização, acesse a aula Operadores do nosso curso no Moodle.</td></tr>
<tr><td>21</td><td>Mensagem:</td><td>control reaches end of non-void function</td></tr>
<tr><td></td><td>Origem:</td><td>Sites</td></tr>
<tr><td></td><td>Tipo:</td><td>Error</td></tr>
<tr><td></td><td>Feedback:</td><td>O erro control reaches end of non-void function ocorreu porque uma função que foi declarada com um tipo de retorno, não obteve um valor retornado.

Veja o exemplo:

<pre>#include <stdio.h>
int display (const char * str)
{
 printf ("%s\n", str);</pre></td></tr>
</table>

		} Observe que a função display() foi definida que retornará um inteiro, no entanto, é executado um printf(). Neste caso, a função deveria ter sido declarada como nula (void), pois funções declaradas nulas não precisam de retornos. Maiores informações sobre declaração de funções, acesse o post do *Eduardo Casavella*
22	Mensagem:	**unused variable**
	Origem:	Código do aluno/sites
	Tipo:	Error

	Feedback:	O alerta ***unused variable*** indica que uma variável foi declarada como uma variável local ou nos parâmetros de uma função, mas não foi usada em nenhum lugar. Uma variável não usada pode ser o resultado de um erro de programação, como acidentalmente usando o nome de uma variável diferente no lugar da variável desejada.
		
		Veja o exemplo:
		
		int foo (int k, char * p)
		{
		int i, j;
		j = k;
		return j;
		}
		
		Observe que a variável p não é utilizada dentro da função foo().

23	Mensagem:	**unused parameter**
	Origem:	Sites
	Tipo:	Error

	Feedback:	O alerta ***unused parameter*** indica que uma variável foi declarada como uma variável local ou nos parâmetros de uma função, mas não foi usada em nenhum lugar. Uma variável não usada pode ser o resultado de um erro de programação, como acidentalmente usando o nome de uma variável diferente no lugar da variável desejada.
		
		Veja o exemplo:
		
		int foo (int k, char * p)
		{
		int i, j;
		j = k;
		return j;
		}
		
		Observe que a variável p não é utilizada dentro da função foo().

<table>
<tr><td rowspan="4">24</td><td>Mensagem:</td><td>no such file or directory</td></tr>
<tr><td>Origem:</td><td>Sites</td></tr>
<tr><td>Tipo:</td><td>Error</td></tr>
<tr><td>Feedback:</td><td>O erro No such file or directory ocorre quando o GCC não consegue encontrar um arquivo solicitado em seu caminho de pesquisa. O arquivo pode ter sido especificado na linha de comando ou com uma instrução #include do pré-processador. O nome do arquivo foi digitado incorretamente ou o diretório do arquivo precisa ser adicionado ao caminho de inclusão ou ao caminho do link.

Veja o exemplo:

<pre>#include <stdoi.h> /* incorrect */
int main (void)
{
 printf ("Hello World!\n");
 return 0;
}</pre>

Observe que o arquivo digitado no #include está incorreto.
Além da biblioteca padrão, outras bibliotecas foram desenvolvidas para incorporar outras funcionalidades específicas.

Arquivos de Cabeçalho do ANSI C
<table>
<tr><th>Nome do arquivo</th><th>Descrição do arquivo de cabeçalho</th></tr>
<tr><td><assert.h></td><td>Implementa ajuda na detecção de erros em versões de depuração de programas.</td></tr>
<tr><td><complex.h></td><td>Trata da manipulação de números complexos. – Até aqui</td></tr>
<tr><td><ctype.h></td><td>Funções para conversão de maiúsculas, minúsculas e outros tratamentos de caracteres.</td></tr>
<tr><td><errno.h></td><td>Teste de códigos de erro reportados pelas funções de bibliotecas.</td></tr>
<tr><td><fenv.h></td><td>Define várias funções e macros para tratar de exceções em variáveis do tipo ponto flutuante.</td></tr>
<tr><td><float.h></td><td>Define limites e precisão de variáveis de ponto flutuante.</td></tr>
<tr><td><inttypes.h></td><td>Trata de conversão precisa entre tipos inteiros.</td></tr>
</table>
</td></tr>
</table>

		<iso646.h> Adiciona a possibilidade de programação usando a codificação de caracteres¨de acordo com a ISO646. <limits.h> Constantes de propriedades específicas de implementação da biblioteca de tipos inteiros, como a faixa de números que pode ser representada (_MIN, _MAX). <locale.h> Especifica constantes de acordo com a localização específica, como moeda, data, etc. <math.h> Funções matemáticas comuns em computação. <setjmp.h> Define as macros setjmp e longjmp, para saídas não locais e tratamento de exceções. <signal.h> Implementa definições para receber e fazer o tratamento de sinais. <stdarg.h> Acesso dos argumentos passados para funções com parâmetro variável. <stdbool.h> Trata da definição para tipo de dados booleano. <stdint.h> Padrões de definição de tipos de dados inteiros. <stddef.h> Padrões de definições de tipos. <stdio.h> Tratamento de entrada/saída. <stdlib.h> Implementa funções para diversas operações, incluindo conversão, alocação de memória, controle de processo, funções de busca e ordenação. <string.h> Tratamento de strings. <tgmath.h> Implementa facilidades para utilização de funções matemáticas. <time.h> Trata de tipos de data e hora. <wchar.h> Tratamento de caracteres para suportar diversas línguas. <wctype.h> Contém funções para classificação de caracteres *wide*. Fonte: http://linguagemc.com.br/a-biblioteca-padrao-da-linguagem-c/
25	Mensagem:	**#include nested too deeply**

	Origem:	Sites
	Tipo:	Error

	Feedback:	O erro *#include nested too deeply* ocorre quando o pré-processador encontra muitas diretivas #include aninhadas. Geralmente, é causado por dois ou mais arquivos tentando incluir um ao outro, levando a uma recursão infinita.
		
		Veja o exemplo:
		
		`/* foo.h */`
		`#include "bar.h"`
		`...`
		`/* bar.h */`
		`#include "foo.h"`
		`int main(void) { ... }`
		
		Observe que foo.h inclui chamada de bar.h, o que acontece em bar.h derivando uma recursividade infinita de chamadas. Analise se em seu código não está ocorrendo algo semelhante.

26	Mensagem:	**this file includes at least one deprecated or antiquated header**
	Origem:	Sites
	Tipo:	Warning

	Feedback:	O aviso *This file includes at least one deprecated or antiquated header* ocorreu porque está sendo utilizado um cabeçalho antigo do padrão ANSI. Os cabeçalhos antigos importam suas funções para o namespace global - std::namespace. Observe que os arquivos de cabeçalho de estilo antigo ainda são suportados, portanto, essa mensagem é apenas um aviso e os programas existentes continuarão a ser compilados. A mensagem é gerada por uma diretiva #warning nos arquivos de cabeçalho antigos e não pelo próprio pré-processador.
		
		Veja o exemplo:
		
		`#include <iostream.h> /* old style */`
		`int main (void)`
		`{`
		`cout << "Hello World!\n";`
		`return 0;`
		`}`
		
		A chamada iostream.h está depreciada, apesar de por ser utilizada. Para corrigir o problema, o comando cout está definido no std::namespace, neste caso é necessário utilizar std::cout em vez de somente cout. Ao invés de utilizar o comando cout, troque para o comando printf() da biblioteca stdio.h.

27	Mensagem:	**file not recognized**
	Origem:	Sites

<table>
<tr><td></td><td>Tipo:</td><td>Error</td></tr>
<tr><td></td><td>Feedback:</td><td>O erro file not recognized ocorre quando a extensão do arquivo compilado é diferente de .c. O GCC usa a extensão de um arquivo, como .c ou .cc, para determinar seu conteúdo. Se a extensão estiver faltando ou for diferente do padrão da linguagem, o GCC não consegue reconhecer o tipo de arquivo e apresentará esse erro.</td></tr>
<tr><td>28</td><td>Mensagem:</td><td>expected identifier or ‘(‘ before</td></tr>
<tr><td></td><td>Origem:</td><td>Código do aluno</td></tr>
<tr><td></td><td>Tipo:</td><td>Error</td></tr>
<tr><td></td><td>Feedback:</td><td>O erro expected identifier or '(' before ocorre por diversas razões:

● existe alguma linha comentada que continha a abertura de uma chave e deixou seu fechamento descomentado (Exemplo 1);
● não finalizou a sequência de comandos com um ponto e vírgula, mas com uma vírgula (Exemplo 2), ou,
● foi colocado um ponto e vírgula no cabeçalho de alguma função (Exemplo 3).

Veja os exemplo

Exemplo 1:

--------- Fechamento de chaves de um comando if que está comentado
//if(num!=0) {
maior=num;
menor=num;
}

Exemplo 2:

--------- declaração de variáveis finalizando com vírgula
int voto, voto_a=0, voto_b=0,Voto_br=0, voto_nulo=0,

Exemplo 3:

--------- função com ponto e vírgula no final dos ()
int *pont2(int a, int b, int c, int d, int e, int f, int menor);{

return &menor;
}

Verifique se algumas dessas situações podem estar ocorrendo no seu código.</td></tr>
<tr><td>29</td><td>Mensagem:</td><td>‘else’ without a previous ‘if’</td></tr>
<tr><td></td><td>Origem:</td><td>Código do aluno</td></tr>
</table>

	Tipo:	Error

	Feedback:	O erro ***'else' without a previous 'if'*** ocorre quando existe um desbalanceamento entre as chaves dos comandos. Toda chave aberta deverá ter um correspondente de fechamento.
		
		Veja o exemplo:
		
		`if(num%2==0){`
		`;printf("\nDobro de %d: %d\n", num, num*2);`
		`}else{`
		`divisores=0;`
		`{ /*error*/`
		`if(num%n==0){divisores++;}`
		`printf("\nDivisores de %d: %d\n",num, cont);`
		`}else{`
		`if (num==0);`
		`;printf("\nDigite outro número:\n");`
		`;scanf ("%d", &num);`
		`}`
		
		Observe no exemplo que existe mais chaves que abrem do que fecham. Para facilitar o entendimento do código, faça a identação dos comandos; assim facilitará encontrar o problema.

30	Mensagem:	**stray '\' in program**
	Origem:	Código do aluno
	Tipo:	Error

	Feedback:	O erro *stray '\' in program* ocorre quando a barra que insere comentário não é a correta. Os comentários de um programa devem ser colocados entre /* e */ ou pela sequência das barras //. Quaisquer textos colocados entre estes dois símbolos(/**/ e //) serão ignorados pelo compilador.
		
		Quando utilizá-los:
		
		● Se pretender escrever um texto ou um grupo de linhas como comentário deve-se iniciar o comentário com /* e terminar com */
		● Se pretender apenas escrever uma linha, pode-se iniciar o comentário com // seguido do texto.
		
		Veja o exemplo:
		
		`/* Isto é um comentário */`
		`int i;  //índice do vetor de saída <- comentário de uma linha iniciado por //`
		`float soma; /* Soma dos valores pagos */`
		`char letra; /* este é um comentário de`
		`02 linhas */`

31	Mensagem:	**a label can only be part of a statement and a declaration is not a**

		statement
	Origem:	Código do aluno
	Tipo:	Error

	Feedback:	O erro ***a label can only be part of a statement and a declaration is not a statement*** ocorre quando há uma variável sendo utilizada erroneamente.
		
		Veja o exemplo:
		
		`switch(codigo){ //inserindo o código para a votação`
		`  case 111: voto_a++;`
		`  break;`
		`  case 222: voto_b++;`
		`  break;`
		`  case 555: Voto em branco++; /*ERROR*/`
		`  break;`
		
		`  default: nul++;`
		`}`
		
		Observe que foi utilizado como variável Voto em branco++ que não é válido na linguagem C.
		
		Em C, para declarar uma variável você deverá informar primeiro o tipo de dado e depois o nome. No decorrer do programa utilize sempre estas mesmas variáveis. Atente-se que o nome da variável não pode ter espaços e deve possuir até 32 caracteres.
		Sintaxe:
		
		`tipo_de_dados nome_da_variável;`
		
		Exemplo de declaração de variável do tipo inteiro: int contador;
		
		onde:
		
		● int é o tipo da variável (inteiro)
		● contador é o nome da variável.
		
		Os tipos de dados mais comuns em linguagem C, são:
		
		● int: armazena valores numéricos inteiros.
		● char: armazena caracteres.
		● float: armazena números com ponto flutuante (reais) com precisão simples.
		● double: armazena números com ponto flutuante, com precisão dupla, ou seja normalmente possui o dobro da capacidade de uma variável do tipo float.
		
		Verifique se não existem declarações de variáveis com tipos diferentes na mesma linha. Mais informações sobre declaração de variáveis, acesse o nosso curso no Moodle na aula sobre Variáveis e o post do *Eduardo*

<table>
<tr><td></td><td></td><td>Casavella.</td></tr>
<tr><td rowspan="4">32</td><td>Mensagem:</td><td>format ‘%d’ expects argument of type ‘int *’, but argument 2 has type ‘int’</td></tr>
<tr><td>Origem:</td><td>Código do aluno</td></tr>
<tr><td>Tipo:</td><td>Warning</td></tr>
<tr><td>Feedback:</td><td>O alerta format '%d' expects argument of type 'int *', but argument 2 has type 'int' ocorre quando é esquecido o & na variável de leitura como scanf("%d", voto) em vez de scanf("%d", &voto).

Veja o exemplo:

<pre>
int main(void)
{
 //Escreva seu código aqui.
 int voto,c1=0,c2=0,c3=0,c4=0,br=0,nul=0,validos;

 do{
  scanf("%d",voto); /*ALERTA*/
  switch(voto){
    case 11: c1++;break;
    case 42: c2++;break;
    case 63: c3++;break;
    case 74: c4++;break;
    case 77: br++; break;
    case 0: break;
    default: nul++;
  }
 } while(voto!=0);

 validos=c1+c2+c3+c4+br;

 printf("Governador A: %d votos - %.1f%%",c1,(c1*100.0/validos));
 printf("\n Governador B: %d votos - %.1f%%",c2,(c2*100.0/validos));
 printf("\n Governador C: %d votos - %.1f%%",c3,(c3*100.0/validos));
 printf("\n Governador D: %d votos - %.1f%%",c4,(c4*100.0/validos));
 printf("\n Brancos: %d votos",br);
 printf("\n Nulos: %d votos",nul);

 return 0;

}//main
</pre>
Observe que a leitura da variável voto não foi realizada com o &.</td></tr>
<tr><td rowspan="4">33</td><td>Mensagem:</td><td>format ‘%f’ expects argument of type ‘float *’, but argument 2 has type ‘double’</td></tr>
<tr><td>Origem:</td><td>Código do aluno</td></tr>
<tr><td>Tipo:</td><td>Warning</td></tr>
<tr><td>Feedback:</td><td>O alerta format '%f' expects argument of type 'float *', but argument 2</td></tr>
</table>

		has type 'double' ocorre quando o compilador não reconhece o segundo argumento.
		
		Veja o exemplo:
		
		`int main(void)`
		`{`
		`  float salario, salarioFinal;`
		`  scanf("%f", salario); /* ALERTA*/`
		
		`  if(salario > 300) {`
		`  salarioFinal = salario * 1.5;`
		
		`  } else {`
		
		`  salarioFinal = salario * 1.3;`
		`  }`
		
		`  printf("%.2f", salarioFinal);`
		
		`  return 0;`
		`}//main`
		
		Observe que a na leitura da variável salario pelo scanf() foi esquecido o &. Toda variável possui um endereço de memória associado a ela, que se refere ao local da memória onde os dados ficarão armazenados. Por isso, que o segundo argumento da função de biblioteca scanf() é o endereço da variável que deve receber o valor lido do teclado.

34	Mensagem:	**assignment makes integer from pointer without a cast**
	Origem:	Código do aluno
	Tipo:	Error

	Feedback:	O erro ***assignment makes integer from pointer without a cast*** ocorre quando é atribuído uma sequência de caracteres (string) a uma variável.
		
		Veja o exemplo:
		
		`#include <stdio.h>`
		`#include <string.h>`
		`int main(void){`
		` char name[10];`
		` name = "hello";`
		` return 0;`
		`}`
		
		A atribuição de valores a strings não se faz com o operador '=' mas sim com funções como a strcpy.
		Sintaxe:
		
		`strcpy(string_destino, string_origem);`

		Maiores informações do uso deste comando, acesse o blog do *Eduardo Casavella* sobre A biblioteca string.h
35	Mensagem:	**assignment to expression with array type**
	Origem:	Código do aluno
	Tipo:	Error

	Feedback:	O erro ***assignment to expression with array type*** ocorre quando é atribuído uma sequência de caracteres (string) a uma variável.
		
		Veja o exemplo:
		
		`#include <stdio.h>`
		`#include <string.h>`
		`int main(void){`
		`char name[10];`
		`name = "hello";`
		`return 0;`
		`}`
		
		A atribuição de valores a strings não se faz com o operador '=' mas sim com funções como a strcpy.
		Sintaxe:
		
		`strcpy(string_destino, string_origem);`
		
		Maiores informações do uso deste comando, acesse o blog do *Eduardo Casavella* sobre A biblioteca string.h

36	Mensagem:	**format '%d' expects argument of type 'int *', but argument 2 has type 'float *'**
	Origem:	Código do aluno
	Tipo:	Warning

	Feedback:	O alerta ***format '%d' expects argument of type 'int *', but argument 2 has type 'float *'*** ocorre quando uma variável foi declarada com um tipo e na leitura foi utilizada outra.
		
		Veja o exemplo:
		
		`#include<stdio.h>`
		`int main(void)`
		`{`
		`float salario, salarioFinal;`
		`scanf("%d", &salario); /* ALERTA*/`
		
		`if(salario > 300) {`
		`salarioFinal = salario * 1.5;`
		
		`} else {`

<table>
<tr><td></td><td></td><td>

```
  salarioFinal = salario * 1.3;
 }

 printf("%.2f", salarioFinal);

 return 0;
}//main
```

Observe que a na leitura da variável pelo scanf() foi utilizado "%d" ao invés de "%f".</td></tr>
<tr><td>37</td><td>Mensagem:</td><td>format ‘%f’ expects argument of type ‘double *’, but argument 3 has type ‘int’</td></tr>
<tr><td></td><td>Origem:</td><td>Código do aluno</td></tr>
<tr><td></td><td>Tipo:</td><td>Warning</td></tr>
<tr><td></td><td>Feedback:</td><td>O alerta format '%f' expects argument of type 'double', but argument 3 has type 'int' ocorreu porque foi definida a saída de uma variável com tipo diferente ao especificado como formato do argumento.

Veja o exemplo:


```
#include<stdio.h>
int main(void)
{
 //Escreva seu código aqui.
 int voto,c1=0,c2=0,c3=0,c4=0,br=0,nul=0,validos;

 do{
  scanf("%d",&voto);
        switch(voto){
                case 51: c1++;break;
                case 72: c2++;break;
                case 83: c3++;break;
                case 44: c4++;break;
                case 77: br++; break;
                case 0: break;
                default: nul++;
        }
 } while(voto!=0);

 validos=c1+c2+c3+c4+br;

 printf("GovernadoA: %d votos - %.1f%%",c1,c1); /*ALERTA*/
 printf("\n GovernadoB: %d votos - %.1f%%",c2,(c2*100.0/validos));
 printf("\n GovernadoC: %d votos - %.1f%%",c3,(c3*100.0/validos));
 printf("\n GovernadoD: %d votos - %.1f%%",c4,(c4*100.0/validos));
 printf("\n Brancos: %d votos",br);
 printf("\n Nulos: %d votos",nul);

 return 0;
}//main
```
</td></tr>
</table>

<table>
<tr><td></td><td></td><td>Observe que a na saída printf("GovernadoA: %d votos - %.1f% %",c1,c1); espera-se um float e na verdade a segunda variável c1 é um inteiro.</td></tr>
<tr><td rowspan="4">38</td><td>Mensagem:</td><td>format ‘%f’ expects argument of type ‘float *’, but argument 2 has type ‘int *’</td></tr>
<tr><td>Origem:</td><td>Código do aluno</td></tr>
<tr><td>Tipo:</td><td>Warning</td></tr>
<tr><td>Feedback:</td><td>O alerta format '%f' expects argument of type 'float *', but argument 2 has type 'int *' ocorre quando a variável é definida de um tipo e na leitura é utilizado outro formato de argumento.

Veja o exemplo:

<pre>
int main(void)
{
 //Escreva seu código aqui.
 int voto,c1=0,c2=0,c3=0,c4=0,br=0,nul=0,validos;

 do{
  scanf("%f",&voto); /*ALERTA*/
        switch(voto){
                case 11: c1++;break;
                case 42: c2++;break;
                case 63: c3++;break;
                case 74: c4++;break;
                case 77: br++; break;
                case 0: break;
                default: nul++;
        }
 } while(voto!=0);

 validos=c1+c2+c3+c4+br;

 printf("Governador A: %d votos - %.1f%%",c1,(c1*100.0/validos));
 printf("\n Governador B: %d votos - %.1f%%",c2,(c2*100.0/validos));
 printf("\n Governador C: %d votos - %.1f%%",c3,(c3*100.0/validos));
 printf("\n Governador D: %d votos - %.1f%%",c4,(c4*100.0/validos));
 printf("\n Brancos: %d votos",br);
 printf("\n Nulos: %d votos",nul);

 return 0;

}//main
</pre>

Observe que a leitura da variável voto foi utilizado o formato de argumento %f no lugar do %d.</td></tr>
<tr><td>39</td><td>Mensagem:</td><td>format ‘%f’ expects arguments of type ‘double’, but argument ‘ has type ‘int’</td></tr>
</table>

<table>
<tr><td></td><td>Origem:</td><td>Código do aluno</td></tr>
<tr><td></td><td>Tipo:</td><td>Warning</td></tr>
<tr><td></td><td>Feedback:</td><td>O alerta format '%f' expects argument of type 'double', but argument 2 has type 'int' ocorre quando a variável é definida de um tipo e na saída é utilizada com outro formato de argumento.

Veja o exemplo:

<pre>
int main(void)
{
 //Escreva seu código aqui.
 int voto,c1=0,c2=0,c3=0,c4=0,br=0,nul=0,validos;

 do{
  scanf("%d",&voto);
        switch(voto){
                case 11: c1++;break;
                case 42: c2++;break;
                case 63: c3++;break;
                case 74: c4++;break;
                case 77: br++; break;
                case 0: break;
                default: nul++;
        }
 } while(voto!=0);

 validos=c1+c2+c3+c4+br;

 printf("Governador A: %.1f votos - %.1f%%",c1,(c1*100.0/validos)); /
ALERTA/
 printf("\n Governador B: %d votos - %.1f%%",c2,(c2*100.0/validos));
 printf("\n Governador C: %d votos - %.1f%%",c3,(c3*100.0/validos));
 printf("\n Governador D: %d votos - %.1f%%",c4,(c4*100.0/validos));
 printf("\n Brancos: %d votos",br);
 printf("\n Nulos: %d votos",nul);

 return 0;

}//main
</pre>

Observe que a saída de quantidades de votos do Governador A é de tipo inteiro, mas no comando printf("Governador A: %f votos - %.1f%%",c1, (c1*100.0/validos)); foi utilizado %f.</td></tr>
<tr><td>40</td><td>Mensagem:</td><td>expected ‘)’ before</td></tr>
<tr><td></td><td>Origem:</td><td>Código do aluno</td></tr>
<tr><td></td><td>Tipo:</td><td>Error</td></tr>
<tr><td></td><td>Feedback:</td><td>O erro expected ')' before ocorre quando um parêntese aberto não possui o correspondente de fechamento.</td></tr>
<tr><td>41</td><td>Mensagem:</td><td>is used uninitialized in this function</td></tr>
</table>

<table>
<tr><td></td><td>Origem:</td><td>Simulado</td></tr>
<tr><td></td><td>Tipo:</td><td>Warning</td></tr>
<tr><td></td><td>Feedback:</td><td>O alerta variable is used uninitialized in this function ocorre quando uma variável está sendo utilizada em um comando printf(), não foi inicializada e nem utilizada na resolução da questão.

Veja o exemplo:

<pre>int numero, soma, impar, par;
scanf("%d", &numero);
if(numero % 2 == 1){
  soma = numero + impar;
}
printf("%d", par); /*ALERTA*/</pre>
Observe que a variável não possui valor para ser utilizada no printf().</td></tr>
<tr><td>42</td><td>Mensagem:</td><td>passing argument 1 of scanf makes pointer from integer without a cast</td></tr>
<tr><td></td><td>Origem:</td><td>Simulado</td></tr>
<tr><td></td><td>Tipo:</td><td>Warning</td></tr>
<tr><td></td><td>Feedback:</td><td>O alerta passing argument 1 of scanf makes pointer from integer without a cast ocorre quando o comando scanf() não segue a sua sintaxe correta.

Sintaxe:

scanf(“expressão de controle”, lista de argumentos);

<table>
<tr><th>Formato</th><th>Tipo de Dados</th></tr>
<tr><td>%c</td><td>char</td></tr>
<tr><td>%d</td><td>int</td></tr>
<tr><td>%f</td><td>float</td></tr>
<tr><td>%s</td><td>char[]</td></tr>
</table>
Veja o exemplo:

<pre>int idade; float nota;

scanf("%d", &idade);
scanf("%f", &nota);</pre>
Observe que o formato deve SEMPRE vir entre duas aspas (") e acompanha o & junto ao nome da variável.</td></tr>
<tr><td>43</td><td>Mensagem:</td><td>multi-character character constant</td></tr>
</table>

<table>
<tr><td></td><td>Origem:</td><td>Simulado</td></tr>
<tr><td></td><td>Tipo:</td><td>Warning</td></tr>
<tr><td></td><td>Feedback:</td><td>O alerta multi-character character constant ocorre quando o comando não segue a sua sintaxe correta.

Veja o exemplo:

<pre>int voto
switch (voto){
 case '11': c1++; break;

}</pre>
Observe que a variável voto é inteira, mas no case ela é testada como char.</td></tr>
<tr><td>44</td><td>Mensagem:</td><td>expected identifier before & token scanf</td></tr>
<tr><td></td><td>Origem:</td><td>Simulado</td></tr>
<tr><td></td><td>Tipo:</td><td>Error</td></tr>
<tr><td></td><td>Feedback:</td><td>O erro expected identifier before & token scanf ocorre quando o comando scanf() não segue a sua sintaxe correta.

Sintaxe:

scanf(“expressão de controle”, lista de argumentos);

<table><tr><td>Formato</td><td>Tipo de Dados</td></tr><tr><td>%c</td><td>char</td></tr><tr><td>%d</td><td>int</td></tr><tr><td>%f</td><td>float</td></tr><tr><td>%s</td><td>char[]</td></tr></table>
Veja o exemplo:

<pre>int idade; float nota;
scanf("%d", &idade);
scanf("%f", &nota);</pre>
Observe que o formato deve SEMPRE vir entre duas aspas (") e acompanha o & junto ao nome da variável. O & só não será utilizado no scanf() quando a variável for do tipo char. Outro ponto importante é o separador entre os parâmetros da função que é a vírgula (,).</td></tr>
<tr><td>45</td><td>Mensagem:</td><td>format %i expects argument of type int *, but argument 2 has types int</td></tr>
</table>

	Origem:	Simulado
	Tipo:	Warning
	Feedback:	O ***alerta format %i expects argument of type int *, but argument 2 has type int*** ocorre quando uma variável foi declarada com um tipo e na leitura foi utilizada outra. O %i não é mais sinônimo de %d. O que %i faz é interpretar o valor digitado como hexadecimal, se iniciar por 0x ou 0X; como octal, se iniciar-se por 0; ou como decimal, caso nenhuma dessas condições seja verificada. Neste caso, a melhor alternativa é trocar o %i pelo %d.
46	Mensagem:	**expected statement before] token scanf**
	Origem:	Simulado
	Tipo:	Error
	Feedback:	O erro *expected statement before] token scanf* ocorreu o caracter] foi colocado equivocamente após um scanf(). Veja o exemplo: `scanf("%d",&sala);]` Verifique no seu código se não há algo parecido.
47	Mensagem:	**invalid type argument of unary ***
	Origem:	Simulado
	Tipo:	Error

	Feedback:	O erro ***invalid type argument of unary * (have double)*** ocorre quando são utilizados os operadores =*, =+, =-, =/. Esses operadores não possuem suporte no padrão ANSI, mas os operadores *=, +=, -= e /= são muito utilizados em expressões matemáticas como atalhos para as operações aritméticas mais comuns.
		
		Veja o exemplo1:
		
		`num += 1; // equivale a num = num + 1;`
		`num -= 1; // equivale a num = num – 1;`
		`num *= 2; // equivale a num = num * 2;`
		`num /= 2;// equivale a num = num / 2;`
		
		Outra situação é quando se utiliza o operador * para trabalhar com ponteiros, mas a variável definida não é um ponteiro.
		
		Veja o exemplo2:
		
		`int pa,pb,pc;`
		`printf("Em ordem crescente:%d,%d,%d\n",pa,pb,*pc);`

		Observe que a variável pc não é um ponteiro, no entanto, está tentando imprimir o conteúdo de pc utilizando-o como ponteiro.
48	Mensagem:	**set but not used**
	Origem:	Simulado
	Tipo:	Warning
	Feedback:	O alerta ***variable " set but not used*** informa que existe uma variável que está sendo declarada, mas não é utilizada no código. Avalie se essa variável é necessária no programa.
49	Mensagem:	**self-comparison always evaluates to false**
	Origem:	Simulado
	Tipo:	Warning
	Feedback:	O alerta ***self-comparison always evaluates to false*** informa que existe uma comparação com a mesma variável dentro do programa. Veja o exemplo : if(num > num) {...} Avalie se não está ocorrendo algo semelhante no seu código.
50	Mensagem:	**format %d expects a matching int argument**
	Origem:	Simulado
	Tipo:	Warning
	Feedback:	O alerta ***format %d expects a matching int argument*** ocorre quando o comando prinft() espera um argumento inteiro correspondente a formatação informada. Veja o exemplo: printf("Divisores: %d, divisores"); Observe que a variável divisores foi colocada dentro das aspas e não fora como recomenda a sintaxe do comando. Veja outro exemplo: printf("\nCandidata 1: %d votos - %d%% válidos",c1); Neste caso, foi informado ao comando printf() que haveria 2 duas saídas %d, no entanto, apenas uma é apresentada (variável c1). Veja outro exemplo: printf("\nQuantidade Total %d\n"); Observe que foi informado que será impresso uma variável inteira, no entanto, essa variável não foi informada. Só para relembrar, a sintaxe do comando printf() é: printf("Conversão/Formato do argumento",argumentos);

		Atenção: A quantidade de formatadores deverá ser na mesma quantidade de variáveis que serão impressas.
51	Mensagem:	**may be used uninitialized in this function**
	Origem:	Simulado
	Tipo:	Warning

	Feedback:	O alerta ***variable may be used uninitialized in this function*** ocorre quando uma variável está sendo utilizada em uma operação e não foi inicializada.
		
		Veja o exemplo:
		
		`int numero, soma, impar;`
		`scanf("%d", &numero);`
		`if(numero % 2 == 1){`
		`  soma = numero + impar;`
		`}`
		
		Observe que a variável não possui valor para ser utilizada no decorrer do código.

52	Mensagem:	**too many arguments for format**
	Origem:	Simulado
	Tipo:	Warning

	Feedback:	O alerta ***too many arguments for format*** ocorre quando existe mais argumentos do que formatadores na função, ou a formatação está ausente.
		
		Veja os exemplos:
		
		`printf("\nMaior: %i Menor: %i Pares: %i Soma Impares",maior,menor,qtd,soma);`
		`printf("\n Votos validos: ", ((a/soma)*100));`
		
		Observe que no primeiro exemplo existem 4 variáveis (maior,menor,qtd,soma) para 3 formatadores (%i); e no último, falta a formatação da operação ((a/soma)*100).

53	Mensagem:	**statement with no effect**
	Origem:	Simulado
	Tipo:	Warning
	Feedback:	O alerta ***statement with no effect*** ocorre quando há troca entre os operadores de atribuição (=) e igualdade (==). Veja o exemplo: `int par==0;` O operador correto é usar o operador de atribuição (=).

54	Mensagem:	**unknown conversion type character**
	Origem:	Simulado
	Tipo:	Warning
	Feedback:	O alerta ***unknown conversion type character '' in format*** ocorre quando há algum problema na Conversão/Formato do argumento no comando printf(). Veja o exemplo: printf("\nCandidata 1: %d Votos, %.1f% Validos", a , ((a/soma)*100)); Aparentemente, parece estar correto formato do argumento, mas para o GCC o formato %.1f% V é o que é interepretado. Para formatar a saída do printf() com o caracter %, deve-se configurar utilizando %%: printf("Candidata 1: %d votos - %.1f%%",c1,(c1*100.0/validos));
55	Mensagem:	**called object a is not a function or function pointer**
	Origem:	Simulado
	Tipo:	Error
	Feedback:	O erro ***called object a is not a function or function pointer*** ocorre quando um comando não segue a sintaxe correta. Veja o exemplo: printf("\nCandidata 1: %d Votos, %d% Validos", a ((a/soma)*100)); Observe que a ((a/soma)*100) faltou uma vírgula para separar os argumentos. O GCC interpretou como uma chamada de uma função a[...] que não existe no programa. Este erro também ocorre com o uso de ponteiros.
56	Mensagem:	**overflow in conversion from long int to int changes value**
	Origem:	Simulado
	Tipo:	Warning
	Feedback:	O alerta *overflow in conversion from long int to int changes value from ''* ocorre quando é atribuido um valor maior suportado pelo tipo de dados. O tipo int possui um range -32.768 a 32.767 e qualquer atribuição maior que esse valor estoura a capacidade da variável. Veja o exemplo: menor=666666666666;
57	Mensagem:	**suggest parentheses around '&&' within**
	Origem:	Simulado
	Tipo:	Warning
	Feedback:	O alerta ***suggest parentheses around '&&' within '||'*** ocorre quando o compilador identifica o uso de parênteses na operação para respeitar a

precedência de operadores das condições.

Veja o exemplo:

```
if (numero != 0 && numero < menor || menor == 0){
        menor = numero;
}
```

Pelas regras de precedência de operadores:

Operadores	Descrição	Associatividade
::	Resolução de escopo	esquerda para direita
++ -- () [] . ->	Incremento e decremento pós-fixo Parênteses (chamada de função) Elemento de arranjo Seleção de elemento por identificador Seleção de elemento por ponteiro	esquerda para direita
++ -- + - ! ~ (*tipo*) * & sizeof new [] delete []	Incremento e decremento prefixo Adição e subtração unária *Não* lógico e complemento Conversão de tipo de dado Desreferência Referência (endereço de elemento) tamanho de elemento Alocação dinâmica de memória Desalocação dinâmica de memória	direita para esquerda

		.* ->*	Ponteiro para membro	esquerda para direita
		* / %	Multiplicação, divisão, e módulo (resto)	
		+ -	Adição e subtração	
		<< >>	Deslocamento de bits à esquerda e à direita	
		< <= > >=	“menor que” e “menor ou igual que” “maior que” e “maior ou igual que”	
		== !=	“Igual a” e “diferente de ”	
		&	*E* para bits	
		^	*Ou exclusivo* para bits	
		|	*Ou* para bits	
		&&	*E* lógico	
		||	*Ou* lógico	
		c?t:f	Condição ternária	direita para esquerda
		= += -= *= /= %= <<= >>= &= ^= |=	Atribuição Atribuição por adição ou subtração Atribuição por multiplicação, divisão ou módulo (resto)	

		Atribuição por deslocamento de bits Atribuição por operações lógicas
		throw — Lançamento de exceção — ---
		, — Vírgula — esquerda para direita
58	Mensagem:	**expected identifier before , token**
	Origem:	Simulado
	Tipo:	Error
	Feedback:	O erro ***expected identifier before , token*** ocorre quando a sintaxe do comando não está de acordo. Veja o exemplo: printf("Nulos: %i".,vn); Observe que há um ponto (.) entre os argumentos do comando prinft(). A sintaxe do printf(): printf("Conversão/Formato do argumento",argumentos); Onde: <Conversão/Formato do argumento> como as mensagens que serão exibidas <argumentos> pode conter identificadores, expressões aritméticas ou lógicas e valores fixos
59	Mensagem:	**spurius trailing % in format**
	Origem:	Simulado
	Tipo:	Warning
	Feedback:	O alerta ***spurious trailing % in format*** ocorre quando a sintaxe do printf() não está de acordo na sua formatação. Veja o exemplo: printf("\n Votos validos: %.0f %", validos); Aparentemente, parece estar correto formato do argumento, mas para o GCC o formato % é o que é interpretado. Para formatar a saída do printf() com o caracter %, deve-se configurar utilizando %%: printf("Candidata 1: %d votos - %.1f%%",c1,(c1*100.0/validos));
60	Mensagem:	**format %d expects argument of type int, but argument 2 has type double**

<table>
<tr><td></td><td>Origem:</td><td>Simulado</td></tr>
<tr><td></td><td>Tipo:</td><td>Warning</td></tr>
<tr><td></td><td>Feedback:</td><td>O alerta format '%d' expects argument of type int, but argument 2 has type double ocorre quando o compilador não reconhece o segundo argumento.

Veja o exemplo:

<pre>printf("\n\nCandidata 2: %d Votos", b);
printf("\n Votos validos: %d",(((b*1.0)/soma)*100));</pre>
Observe que no printf() foi formatado como %d, no entanto, o resultado da operação é um float.</td></tr>
<tr><td>61</td><td>Mensagem:</td><td>expected identifier before) token</td></tr>
<tr><td></td><td>Origem:</td><td>Simulado</td></tr>
<tr><td></td><td>Tipo:</td><td>Error</td></tr>
<tr><td></td><td>Feedback:</td><td>O erro expected identifier before) token ocorre quando o uso dos parênteses não está de acordo com a sintaxe.

Veja o exemplo:

<pre>if (num != 0 (&&) num < menor (||) menor == 0){
        menor = num;
}</pre>
Observe que foi colocado parênteses separando os operadores lógicos.

Pelas regras de precedência de operadores:

<table>
<tr><th>Operadores</th><th>Descrição</th><th>Associatividade</th></tr>
<tr><td>::</td><td>Resolução de escopo</td><td>esquerda para direita</td></tr>
<tr><td>++ --
()
[]
.
-></td><td>Incremento e decremento pós-fixo
Parênteses (chamada de função)
Elemento de arranjo
Seleção de elemento por identificador
Seleção de elemento por ponteiro</td><td>esquerda para direita</td></tr>
</table>
</td></tr>
</table>

		++ -- + - ! ~ (*tipo*) * & sizeof new [] delete []	Incremento e decremento prefixo Adição e subtração unária *Não* lógico e complemento Conversão de tipo de dado Desreferência Referência (endereço de elemento) tamanho de elemento Alocação dinâmica de memória Desalocação dinâmica de memória	direita para esquerda
		.* ->*	Ponteiro para membro	esquerda para direita
		* / %	Multiplicação, divisão, e módulo (resto)	
		+ -	Adição e subtração	
		<< >>	Deslocamento de bits à esquerda e à direita	
		< <= > >=	“menor que” e “menor ou igual que” “maior que” e “maior ou igual que”	
		== !=	“Igual a” e “diferente de ”	
		&	*E* para bits	

		^	*Ou exclusivo* para bits	
		|	*Ou* para bits	
		&&	*E* lógico	
		||	*Ou* lógico	
		c?t:f	Condição ternária	direita para esquerda
		= += -= *= /= %= <<= >>= &= ^= |=	Atribuição Atribuição por adição ou subtração Atribuição por multiplicação, divisão ou módulo (resto) Atribuição por deslocamento de bits Atribuição por operações lógicas	
		throw	Lançamento de exceção	---
		,	Vírgula	esquerda para direita
62	Mensagem:	**case label does not reduce to an integer constant case**		
	Origem:	Simulado		
	Tipo:	Error		

	Feedback:	O erro c*ase label does not reduce to an integer constant case* ocorre quando é utilizado as duas aspas na validação dos valores em comandos como switch().
		
		Veja o exemplo:
		
		`scanf("%d",&voto);`
		`switch (voto){`
		`  case "11": //ERROR`		

		c1++;
		break;
		....
		}
		
		Observe que o valor 11 do voto está sendo avaliado utilizando duas aspas, porém deve-se utilizar aspas simples tais como case '11':

63	Mensagem:	**format %f expects a matching double argument**
	Origem:	Simulado
	Tipo:	Error
	Feedback:	O alerta ***format %f expects a matching double argumen*t** ocorre quando o comando printf() está configurado para imprimir uma saída do tipo float, mas a variável está ausente. Veja o exemplo: printf("\nMedia: %.1f Conceito E - Reprovado por faltas"); Observe que no comando printf() consta a formatação da saída %.1f, no entanto, não existe a variável associada.
64	Mensagem:	**__builtin_memcpy writing**
	Origem:	Maratona
	Tipo:	Warning
	Feedback:	O alerta ***__builtin_memcpy writing ...*** ocorre quando o tamanho da variável destino é menor que o tamanho do conteúdo que irá receber. Lembre-se que no final de uma string existe o caracter '\0'. Por exemplo: char vogais[10]; strcpy(vogais, "aeiouAEIOU"); Observe que a variável vogal pode armazenar somente 10 bytes, no entanto, ela precisa armazenar 11 bytes em função do caracter final '\0' .
65	Mensagem:	**left-hand operand of comma expression has no effect**
	Origem:	Maratona
	Tipo:	Warning
	Feedback:	O alerta ***left-hand operand of comma expression has no effect*** ocorre quando é utilizado a vírgula (,) como casa decimal. Na linguagem C, os tipos de dados float e double utilizam o ponto (.) como separador das casas decimais. Exemplo: float salario_minimo = 998.00;

66	Mensagem:	**format %s expects argument of type char**
	Origem:	Maratona
	Tipo:	Warning

	Feedback:	O alerta ***format %s expects argument of type char *, but argument 2 has type char (*)*** ocorreu porque foi utilizado uma formatação %s em um comando scanf() junto com o operador & para armazenar dados em uma variável de tipo string. Para trabalhar com leitura de dados para variáveis strings, o operador & não deverá ser utilizado.
		
		Veja o exemplo:
		
		`char nome[20];`
		`scanf("%s", nome);`

67	Mensagem:	**expected identifier before (token**
	Origem:	Maratona
	Tipo:	Error

	Feedback:	O erro ***expected identifier before (token*** ocorre quando os parênteses necessários para o comando não estão completos.
		
		Veja o exemplo1:
		
		`if (peso >=91) && (idade >= 41) {...}`
		
		O comando *if()* necessita dos parênteses que envolverá as condições. Observe que os parênteses do exemplo1 isolam somente as condições, sendo necessário um ajuste no código semelhante ao exemplo2.
		
		Veja o exemplo2:
		`if ( (peso >=91) && (idade >= 41) ) {...}`

68	Mensagem:	**right-hand operand of comma expression has no effect**
	Origem:	Maratona
	Tipo:	Warning
	Feedback:	O alerta ***right-hand operand of comma expression has no effect*** ocorre quando é utilizado a vírgula (,) como casa decimal. Na linguagem C, os tipos de dados float e double utilizam o ponto (.) como separador das casas decimais. Exemplo: `valorkw=salario_minino*0,01;` Observe que foi utilizado o valor 0,01 na operação, o que na verdade deve ser 0.01

<table>
<tr><td rowspan="4">69</td><td>Mensagem:</td><td>lvalue required as left operand of assignment</td></tr>
<tr><td>Origem:</td><td>Maratona</td></tr>
<tr><td>Tipo:</td><td>Error</td></tr>
<tr><td>Feedback:</td><td>O erro lvalue required as left operand of assignment ocorre quando o operador utilizado não está de acordo com as regras da linguagem.

Veja o exemplo:

<code>if ((numero % 2) =! 0) { }</code>

Observe no exemplo que o operador utilizado para avaliar a diferença está trocado de =! troque para != que solucionará o problema. Os operadores relacionais são apresentados na tabela abaixo.

<table>
<tr><th>Símbolo</th><th>Operador</th><th>Exemplo</th><th>Significado</th></tr>
<tr><td>></td><td>Maior que</td><td>x > y</td><td>x é maior que y?</td></tr>
<tr><td>>=</td><td>Maior ou igual</td><td>x >= y</td><td>x é maior ou igual a y?</td></tr>
<tr><td><</td><td>Menor que</td><td>x < y</td><td>x é menor que y?</td></tr>
<tr><td><=</td><td>Menor ou igual</td><td>x <= y</td><td>x é menor ou igual a y?</td></tr>
<tr><td>==</td><td>Igualdade</td><td>x == y</td><td>x é igual a y?</td></tr>
<tr><td>!=</td><td>Diferente de</td><td>x != y</td><td>x é diferente de y?</td></tr>
</table></td></tr>
<tr><td rowspan="4">70</td><td>Mensagem:</td><td>comparison between pointer and integer</td></tr>
<tr><td>Origem:</td><td>Maratona</td></tr>
<tr><td>Tipo:</td><td>Warning</td></tr>
<tr><td>Feedback:</td><td>O alerta comparison between pointer and integer ocorre quando é utilizado os operadores relacionais para verificar conteúdo de uma variável char. Para variáveis que armazenam strings e caracter deve-se utilizar o comando strcmp() para verificar seu conteúdo.

Veja o exemplo:

<pre>char conceito;
if((conceito == 'A' || conceito == 'B' || conceito == 'C') && faltas < 4)
{...}</pre>
Observe que foi utilizado o operador de igualdade (==) para verificar o conteúdo da variável conceito.</td></tr>
</table>

71	Mensagem:	**zero-length gnu_printf format string**
	Origem:	Maratona
	Tipo:	Warning

	Feedback:	O alerta ***zero-length gnu_printf format string*** ocorre quando um comando de printf() é utilizado com valor vazio.
		
		Veja o exemplo:
		
		`printf("");`
		
		Observe que o comando printf() possui a seguinte sintaxe:
		`printf("Conversão/Formato do argumento",argumentos);`

72	Mensagem:	**expected } before**
	Origem:	Maratona
	Tipo:	Error

	Feedback:	O erro ***expected } before ...*** ocorre quando está faltando fechar o token {}.
		
		Veja o exemplo:
		
		`if (num3 < num4) {`
		`soma = soma + num3;`
		`else {`
		`soma = soma + num4;`
		`}`
		
		Observe que as chaves da primeira condição if (num3 < num4) { não foi finalizada antes da cláusula else {}

73	Mensagem:	**assignment to char from char * makes integer from pointer without a cast**
	Origem:	Maratona
	Tipo:	Warning
	Feedback:	O alerta ***assignment to char from char * makes integer from pointer without a cast*** ocorre quando tenta-se atribuir uma string a uma variável com o operador =. Para concatenar valores a uma variável de tipo string, utilize o comando strcat().
74	Mensagem:	**redeclared as different kind of symbol**
	Origem:	Maratona
	Tipo:	Error
	Feedback:	O erro ***redeclared as different kind of symbol*** ocorre quando uma variável está sendo definida em outra parte do programa com o mesmo nome.

		Veja o exemplo:
		
		`int soma_coluna(int coluna){`
		`  int matriz[5][5];`
		`  int somaCol = 0, linha, coluna;`
		`}`
		
		Observe que a variável coluna é passada por parâmetro para a função e dentro da função existe uma variável sendo declarada com o mesmo nome.

75	Mensagem:	**format %f expects argument of type double**
	Origem:	Maratona
	Tipo:	Warning
	Feedback:	O alerta ***format %f expects argument of type double*** ocorre quando a estrutura do printf() não está correta. Veja o exemplo: `printf("Aprovado %.2f\n",&media);` Observe que foi utilizado o operador & junto a variável a ser impressa. No entanto, esse operador é utilizado somente com o scanf() para tipos de dados inteiros e flutuantes.
76	Mensagem:	**incompatible type for argument**
	Origem:	Maratona
	Tipo:	Error

	Feedback:	O erro *incompatible type for argument* ocorre quando uma função espera um tipo de valor e é passado outro.
		
		Veja o exemplo:
		
		`//definição da função`
		`float maior(float *a, float *b, float *c, float *d, float *e){ }`
		
		`printf("Maior valor: %f", maior(a, b, c, d, e)); //chamada da função`
		
		Observe que a função maior está esperando receber endereços de memória para trabalhar com ponteiros e no entanto está recebendo os valores armazenados nas variáveis: a,b,c,d,e

77	Mensagem:	**division by zero**
	Origem:	Maratona
	Tipo:	Warning
	Feedback:	O alerta ***division by zero ocorre*** quando há uma divisão de um número não nulo por zero.
78	Mensagem:	**initializer-string for array of chars is too long char**

	Origem:	Maratona
	Tipo:	Warning
	Feedback:	O alerta ***initializer-string for array of chars is too long char*** ocorre quando o tamanho da variável char é menor que o valor que ela irá receber. Veja o exemplo: `char str2[15] = "aula musical de jazz";` A string "aula musical de jazz" contém 20 caracteres, no entanto, o tamanho da variável str2 é 15. Não esqueça que strings são finalizadas com o /0, que deve ser contabilizado no tamanho da variável que irá receber a string.
79	Mensagem:	**expected (before**
	Origem:	Maratona
	Tipo:	Error
	Feedback:	O erro ***expected '(' before*** ocorre quando um parêntese fechamento não possui o correspondente de abertura ou ausência dos parênteses para os comandos que necessitam sua utilização como por exemplo os comandos: if(), scanf(), printf().
80	Mensagem:	**passing argument 1 of printf from incompatible pointer type**
	Origem:	Maratona
	Tipo:	Warning
	Feedback:	O alerta ***passing argument 1 of printf from incompatible pointer type*** ocorre quando a sintaxe do comando printf() não está de acordo com o padrão ANSI. Relembrando, o comando printf() possui a seguinte estrutura: `printf("Conversão/Formato do argumento",argumentos);` Onde: <Conversão/Formato do argumento> como as mensagens que serão exibidas <argumentos> pode conter identificadores, expressões aritméticas ou lógicas e valores fixos
81	Mensagem:	**invalid use of void expression**
	Origem:	Maratona
	Tipo:	Error

	Feedback:	O erro ***invalid use of void expression*** ocorre quando se está utilizando uma função declarada como void em um comando ou espaço no programa que está esperando um valor.
		
		Veja o exemplo:
		
		`void classeIMC (float imcPessoa) {}`
		
		`printf ("\nVoce completara %d anos - seu IMC: %d - %d", anosCompletados(anoNascimento, 2019), calculoIMC(peso, altura), classeIMC(calculoIMC(peso, altura)));`
		
		Observe que o printf() está esperando um valor com o %d que seria resultante da função classeIMC(peso,altura), que foi definida como void, ou seja, não retorna valor.

82	Mensagem:	**this for clause does not guard**
	Origem:	Maratona
	Tipo:	Warning

	Feedback:	O alerta ***this for clause does not guard*** ocorre quando o comando for não está de acordo com a sintaxe do padrão ANSI.
		
		Lembrando, o comando for() possui a seguinte sintaxe:
		
		`for(valor_inicial; condição_final; valor_incremento)`
		`{`
		`  instruções;`
		`}`
		
		Não esqueça de abrir e fechar as chaves, pois elas fazem parte do comando.

83	Mensagem:	**conflicting types**
	Origem:	Maratona
	Tipo:	Error

	Feedback:	O erro ***conflicting types for ...*** ocorre quando existe duas declarações diferentes para a mesma variável ou assinatura de função de um tipo e definição da função de outro.
		
		Veja o exemplo1 - Variáveis:
		
		`int a,b,c;`
		`int *a,*b,*c;`
		Observe que int a e int *a se referem a mesma variável a.
		
		Veja o exemplo2 - funções:
		
		`char situacao(float imc);`
		`void situacao(float imc){}`

		Observe que a assinatura da função situacao() é do tipo char, no entanto, ao definição na programação foi implementada como void.
84	Mensagem:	**format not a string literal and no format arguments**
	Origem:	Maratona
	Tipo:	Warning

	Feedback:	O alerta ***format not a string literal and no format arguments*** ocorre quando não foi definida a formatação de entrada no comando scanf().
		
		Veja o exemplo:
		
		`char nome[50];`
		`scanf("%s", nome);`
		
		Observe que para strings a formatação é %s.

85	Mensagem:	**Conflicting types for main**
	Origem:	Maratona
	Tipo:	Error
	Feedback:	O erro ***conflicting types for main ...*** ocorre quando existem duas funções main() no programa.
86	Mensagem:	**unknown type name string**
	Origem:	Maratona
	Tipo:	Error
	Feedback:	O erro ***unknown type name string*** ocorre quando uma variável é declarada como tipo string. Esse tipo não contém no padrão ANSI. Uma strings é um array de char e são declaradas como char variavel[quantidade]. Veja o exemplo: `char nome[10];`
87	Mensagem:	**format %c expects argument of type char**
	Origem:	Maratona
	Tipo:	Warning

	Feedback:	O warning ***format %c expects argument of type char*** ocorre quando o compilador espera um valor char e está recebendo outro tipo.
		
		Veja o exemplo:
		
		`char nome[10], genero;`
		`scanf("%s", nome);`
		`scanf("%c", genero);`

		Observe que foi utilizado o formatador %s para armazenar o valor digitado na variável nome pois esta variável foi definida como uma string, e %c para a variável genero visto que irá armazenar apenas 'M' ou 'F'.
88	Mensagem:	**incompatible types when initializing**
	Origem:	Maratona
	Tipo:	Error

	Feedback:	O erro ***incompatible types when initializing*** ocorre quando uma variável foi declarada para armazenar um tipo e, no entanto, tenta transformá-la outro.
		
		Veja o exemplo:
		
		`float maiorValor(float *a, float *b, float *c, float *d, float *e){`
		`  float maior=a;`
		`}`
		
		Observe que a variável a é um ponteiro definida como parâmetro da função. Neste caso a variável maior deveria receber o conteúdo deste endereço passado para a função, através de float maior = *a;

89	Mensagem:	**format %li expects argument of type long int, but argument 3 has type int**
	Origem:	Maratona
	Tipo:	Warning
	Feedback:	O alerta ***format %s expects a matching char*** o ocorre quando o comando prinft() espera um argumento correspondente a formatação informada. Veja o exemplo: `printf(" %s Peso normal");` Observe que o printf() está esperando um argumento de tipo string e não foi informado.

Fonte: Base de dados do Codein'play

APÊNDICE F

QUESTIONÁRIO SOBRE FERRAMENTAS UTILIZADAS

Caros(as) alunos(as),

O objetivo deste questionário é obter informações a respeito das ferramentas IDEs utilizadas em sala de aula. **Não precisa se identificar. Selecione apenas 1 opção para cada questão.**

Idade: _____ Data: _____ Gênero: [ ] M [ ] F

1. Qual IDE você utiliza para fazer os exercícios em aula?

[ ]DevC++ [ ] NetBeans [ ] Outra. Qual?_____

2. Você considera que o uso dessas ferramentas facilitam na resolução dos exercícios?

[ ]Sim [ ] Não [ ] Não sei responder

3. As mensagens emitidas pela ferramenta que você utiliza ajudam na localização dos erros?

[ ]Sim [ ] Não [ ] Não sei responder

4. A ferramenta ajuda a compreender e conhecer o significado o erro?

[ ]Sim [ ] Não [ ] Não sei responder

5. A ferramenta ajuda na digitação da sintaxe correta dos comandos?

[ ]Sim [ ] Não [ ] Não sei responder

6. A ferramenta ajuda na tradução das mensagens em inglês?

[ ]Sim [ ] Não [ ] Não sei responder

7. A ferramenta que você utiliza é simples de utilizar?

[ ]Sim [ ] Não [ ] Não sei responder

8. A ferramenta que você utiliza disponibiliza histórico das execuções?

[ ]Sim [ ] Não [ ] Não sei responder

APÊNDICE G

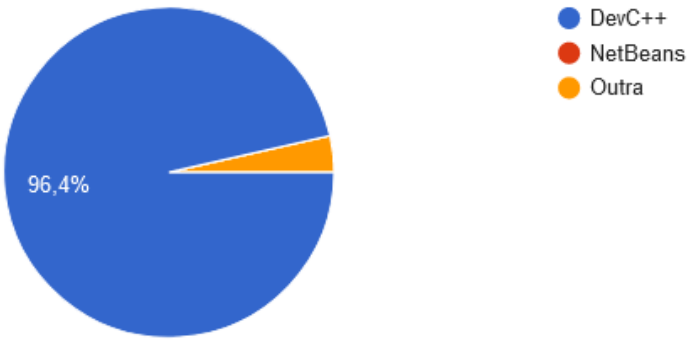
RESPOSTAS DO QUESTIONÁRIO SOBRE FERRAMENTAS UTILIZADAS

Este questionário foi aplicado em outubro de 2019 à turma de LPI (2019/02) e tinha como objetivo principal obter informações a respeito das ferramentas IDEs utilizadas em sala de aula. Participaram da pesquisa 28 alunos, os quais, de maneira voluntária, preencheram oito questões fechadas e uma questão aberta, esta era opcional. Os relatos dos alunos da questão aberta foram adicionados no decorrer do texto.

Q1-Qual IDE você utiliza para fazer os exercícios em aula?

96,4% dos alunos responderam que utilizam a IDE DevC++ para resolver os exercícios propostos na disciplina, e somente 3,6% utilizam outra ferramenta. Ao ser perguntado qual era o nome desta outra ferramenta, citaram o Code Blocks. A ferramenta NetBeans não foi pontuada.

Figura 47 - IDE utilizada para fazer os exercícios em aula

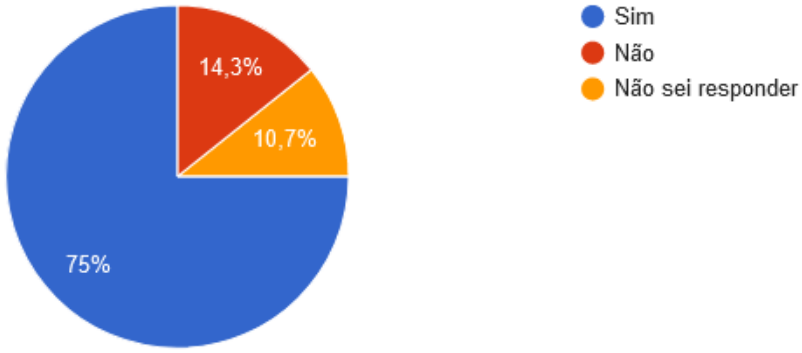


Fonte: Resultado questionário do Apêndice F

Q2-Você considera que o uso dessas ferramentas facilita na resolução dos exercícios?

75% relataram que a ferramenta DevC++ facilita na resolução dos exercícios; e 14,3% acham que não facilita. Segundo a narrativa de um aluno "*Houve dificuldade por ser a primeira vez. Gerou confusão*" no entendimento do retorno das mensagens do compilador. 10,7% não souberam responder.

Figura 48 - IDE facilita na resolução dos exercícios

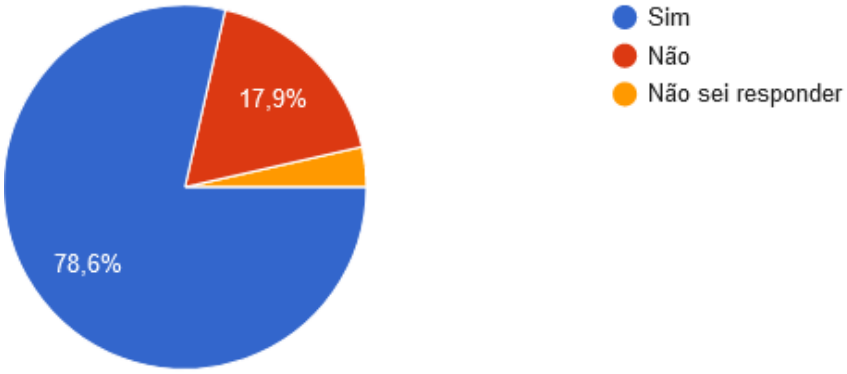


Fonte: Resultado questionário do Apêndice F

Q3-As mensagens emitidas pela ferramenta que você utiliza ajudam na localização dos erros?

78,6% das respostas mostram que as mensagens emitidas pelo GCC, por meio do DevC++, ajudam na localização dos erros no código; 17,9% são contrários, e 3,6% não souberam responder.

Figura 49 - Mensagens do compilador ajudam na localização dos erros

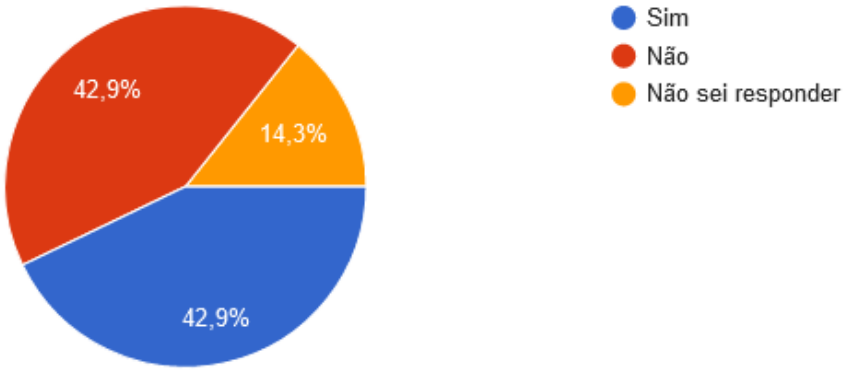


Fonte: Resultado questionário do Apêndice F

Q4-A ferramenta ajuda a compreender e conhecer o significado do erro?

Empatados com 42,9% cada, as opiniões dos estudantes se dividem sobre este quesito. Alguns alunos acham que sim, o DevC++ ajuda na compreensão e na significação sobre o erro, e outros são contrários. 14,3% não souberam responder.

Figura 50 - Ferramenta ajuda na compreensão e conhecimento sobre o erro

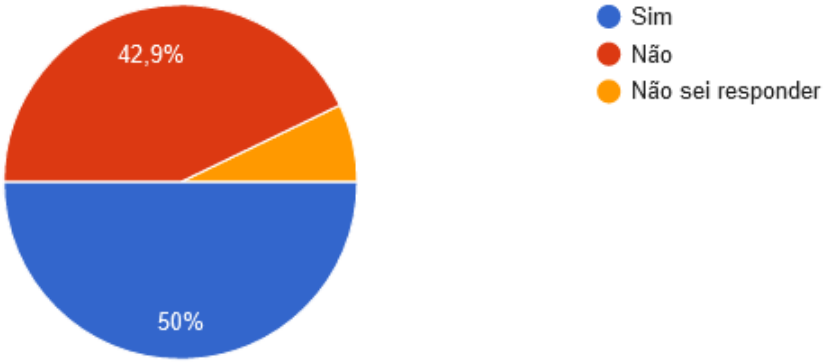


Fonte: Resultado questionário do Apêndice F

Q5-A ferramenta ajuda na digitação da sintaxe correta dos comandos?

50% dos alunos acham que a ferramenta auxilia na digitação correta dos comandos; há 42,9% que são contrários; e 7,1% dos alunos não souberam responder. Houve um relato nesta questão de que não saber inglês, dificultava na escrita correta da sintaxe dos comandos: "*Por eu não saber inglês*".

Figura 51 - Ferramenta auxilia na digitação da sintaxe correta dos comandos

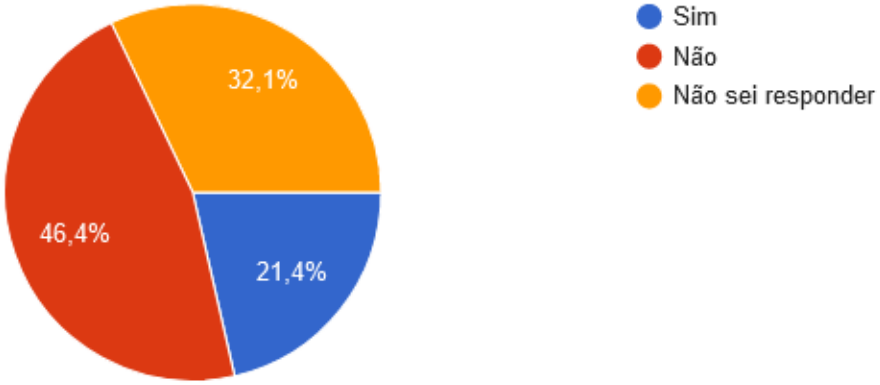


Fonte: Resultado questionário do Apêndice F

Q6-A ferramenta ajuda na tradução das mensagens em inglês?

46,4% dos estudantes afirmaram que o DevC++ não auxilia na tradução das mensagens emitidas pelo compilador; 21,4% acham que a ferramenta ajuda na tradução; e 32,1% não souberam responder. Observa-se, pelas falas, que os alunos não estão muito familiarizados com a ferramenta, quando comentam: "*Como não sei usar amplamente a ferramenta, mas deve ter alguma função que faça*" e "*Não percebi esse recurso*".

Figura 52 - Ferramenta ajuda na tradução das mensagens em inglês

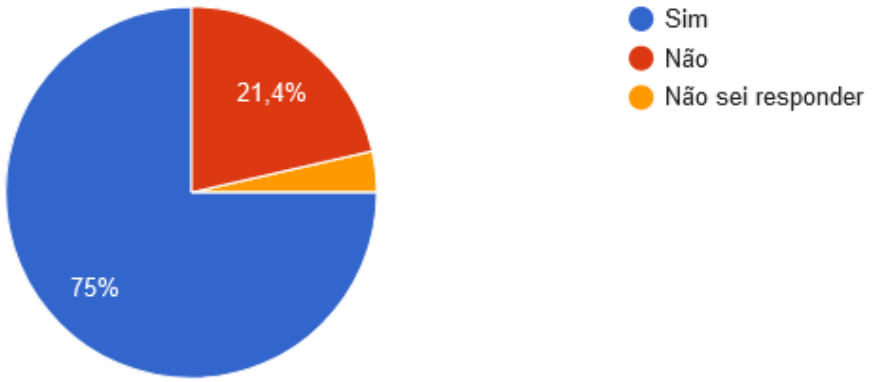


Fonte: Resultado questionário do Apêndice F

Q7-A ferramenta que você utiliza é simples de utilizar?

75% dos estudantes acham que a ferramenta DevC++ é simples de ser utilizada, mas, para que ela se torne fácil, o estudante precisa ter "*[...] familiaridade de uso*", segundo relato; 21,14% são contrários; e 3,6% não souberam responder.

Figura 53 - Ferramenta é simples de utilizar

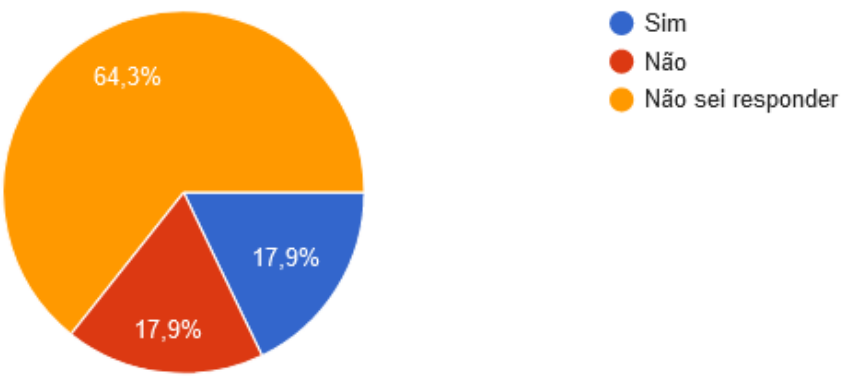


Fonte: Resultado questionário do Apêndice F

Q8-A ferramenta que você utiliza disponibiliza histórico das execuções?

Esta pergunta visa identificar se o estudante consegue visualizar os históricos de suas execuções dentro da IDE adotada. 64,3% dos estudantes não souberam responder e relataram que "*Não percebi esse recurso*"; 17,9% que acham que não é disponibilizado histórico das execuções, e o mesmo percentual para os outros, que acham que sim.

Figura 54 - Disponibilização de histórico das execuções



Fonte: Resultado questionário do Apêndice F

APÊNDICE H

QUESTÕES UTILIZADAS NO SIMULADO, DESAFIO E MARATONA

SIMULADO

Q#1 - Título: Leitura de quantidade indeterminada de números inteiros (3.0 pt)

Faça um algoritmo que:

a) Leia uma quantidade indeterminada de números inteiros (parar quando o usuário digitar '0' Zero). Uso correto de estrutura de repetição **1.0 ponto**.

b) Apresente a soma dos valores ímpares, a quantidade de valores pares. **1.0 ponto**

c) Apresente o Maior valor e o Menor Valor. O zero (finalizador) NÃO deve ser considerado. **1.0 ponto**.

Q#2 - Título: Calcule o quadrado ou a quantidade de divisores de 10 números inteiros (3.0 pt)

Faça um algoritmo que leia 10 valores inteiros, com estrutura de repetição [**1 ponto**]. Se o valor for ímpar, apresentar a quantidade de divisores [**1 ponto**], se o valor for par, apresentar o quadrado [**1 ponto**]. Uso obrigatório de estrutura de repetição.

Q#3 - Título: Eleição para Miss RS (3.0 pt)

Em uma eleição para Miss RS existem quatro candidatas. Os votos são informados através de códigos. Os dados utilizados para a contagem dos votos obedecem à seguinte codificação:

- 11 = Candidata 1
- 22 = Candidata 2
- 33 = Candidata 3
- 44 = Candidata 4
- 77 = voto em branco
- Qualquer número diferente de 0 = Voto Nulo

Elabore um algoritmo que leia os códigos da candidata que receberá o voto, até a eleição ser finalizada com o voto 0. Escreva no final:

- Total de votos para cada candidata
- Percentual de VOTOS VÁLIDOS* de cada candidata
- Total de votos nulos

- Total de votos em branco

* Votos Válidos = Total de votos subtraídos os votos NULOS.

DESAFIO CODEIN'PLAY

Q#1 - Título: Retângulos

Autor do Problema: Eduardo do Araújo – finalista da Maratona de Programação SBC 2017
Adaptado por Karen Cristina Braga

Como trabalho final da disciplina de Linguagem de Programação I, o professor resolveu passar um trabalho que envolvesse toda a turma. Ele começou a escrever várias perguntas estranhas no quadro e assim que terminou disse que o trabalho final era fazer uma biblioteca de algoritmos que respondessem todas aquelas perguntas. A verdadeira lição não era as questões em si, mas o trabalho em equipe e como a expressão "dividir para conquistar" deve fazer parte da vida de um profissional de computação. Apesar de claramente exigir um pouco de raciocínio, os alunos logo começaram a discutir sobre a utilidade de uma biblioteca para aquelas perguntas estranhas. Ignorando os comentários, o professor divide as questões entre os alunos e avisa que a biblioteca tem que estar completa para que todos possam ser aprovados.

José acabou pegando uma bem fácil: "**Se traçarmos X linhas horizontais e Y linhas verticais, quantos retângulos formaremos?**" Antes mesmo de ir para casa ele pegou uma folha de papel e começou a fazer alguns desenhos. Através deles, visualizar a quantidade de retângulos ficou mais fácil e logo conseguiu generalizar o caso para que pudesse desenvolver o algoritmo. Como não podia falhar com seus amigos, ele tinha que ter certeza que seu algoritmo está funcionando. Então, preparou uma bateria de teste e pediu que sua equipe resolvesse o mesmo problema para que pudesse comparar seus resultados.

Ajude José e sua turma a passar na disciplina!

OBSERVAÇÃO: Considere que as linhas nunca são coincidentes e que retângulos formados pela união de 2 ou mais retângulos não devem ser contados.

Entrada:

Quantidade de linhas verticais e horizontais

Saída:

Para cada entrada, imprimir a quantidade de retângulos formados pelas linhas.

Caso de Teste do José:

Entrada	**Saída**
2 1	A quantidade de retangulo(s) para 2 linhas verticais e 1 linhas horizontais: 0
2 2	A quantidade de retangulo(s) para 2 linhas verticais e 2 linhas horizontais: 1
5 3	A quantidade de retangulo(s) para 5 linhas verticais e 3 linhas horizontais: 8

Desenhos do José:

Q#2 - Título: Contando as Letras

Autor do Problema: Eduardo do Araújo – finalista da Maratona de Programação SBC 2017

Adaptado por Karen Cristina Braga

Um pesquisador linguístico está escrevendo um artigo sobre como as diferentes línguas baseadas no alfabeto romano tem sido utilizada desde sua origem. Em uma das seções do artigo o pesquisador queria estabelecer as diferenças entre essas línguas. O primeiro índice a ser avaliado era a diferença da intensidade da utilização de vogais e consoantes entre elas.

O pesquisador então coletou vários textos escritos em diferentes línguas baseadas nesse alfabeto. Selecionou aleatoriamente vários trechos desses textos e para fazer a contagem das vogais e consoantes contatou a Diretoria de Ensino do Instituto Federal do Rio Grande do Sul (IFRS) *Campus* Porto Alegre em busca de alguns alunos que pudessem ajudar a desenvolver um programa capaz de realizar essa contagem. Rapidamente, a Diretoria indicou a sua equipe

para ajudar o pesquisador.

Entrada:
Cada entrada deverá possuir no máximo 80 caracteres [a-z] e espaços em branco. Caracteres diferente dos permitidos não deverão ser contabilizados. As letras não devem ser acentuadas.

ATENÇÃO: Lembre-se que a Linguagem de Programação C é *case sensitive,* ou seja, a letra A é diferente da letra a.

Saída:
Para cada entrada, imprima a quantidade de vogais e consoantes junto com a palavra digitada.

OBSERVAÇÃO: Considerar as vogais como: a,e,i,o,u; demais letras do alfabeto como consoantes.

Exemplo:

Entrada	**Saída**
2	O texto 2 contem 0 vogais e 0 consoantes
barbada	O texto barbada contem 3 vogais e 4 consoantes
Algoritmo	O texto Algoritmo contem 4 vogais e 5 consoantes

IMPORTANTE
Utilize o comando **fgets()** para ler o texto do teclado.
Sintaxe: fgets (str, tamanho, stdin);
Exemplo: fgets(texto, 80, stdin);

MOTIVO: A função gets(), da biblioteca padrão do C (*stdio*), pode gerar um problema de vazamento de memória, ou até mesmo, injeção de código malicioso no programa, pois essa função não limita o número de caracteres a serem lidos da entrada padrão (*stdin*).
FONTE: http://rberaldo.com.br/c-por-que-usar-fgets-em-vez-de-gets/

MARATONA DE PROGRAMAÇÃO

Q#1 - Título: Peso Médio

Desenvolver um algoritmo que leia o nome (até 10 caracteres) e peso de três pessoas. Calcular o peso médio dessas pessoas e imprimí-lo no final com o nome das pessoas participantes da pesquisa. (Não utilizar estrutura de repetição)

Q#2 - Título: Concatenando Textos

Leia duas strings. Se as strings forem iguais escreva **strings iguais**. Caso contrário, concatene as duas strings e imprima a string resultante.

Q#3 - Título: Calculando a Corrida de Charrete

Uma locadora de charretes cobra R$ 10,00 de taxa para cada 3 horas de uso destas e R$5,00 para cada 1 hora abaixo destas 3 horas. Fazer um algoritmo que dado a quantidade de horas que a charrete foi usada calcular e escrever quanto o cliente terá que pagar.

Exemplo:

Quantidade em Horas	**R$ a pagar**
4h (3h+1h)	R$ 15,00
10h (3h+3h+3h+1h)	R$ 35,00

Q#4 - Título: Somando os Menores Valores de Três

Dados quatro números distintos, desenvolver um algoritmo que determine e imprima a soma dos três menores.

Q#5 - Título: Quilowatts

Faça um algoritmo que leia o total de kw gasto por uma residência, calcule e imprima o valor em reais de 1 kw, o valor em reais a ser pago pela residência e o novo valor a ser pago por essa residência com um desconto de 10%. Dado que 100 quilowatts custa 1/7 do salário mínimo. A quantidade de kw gasto pela residência será?

Salário Mínimo = R$ 998.00

Q#6 - Título: Função com Três Valores Inteiros

Crie um algoritmo que leia 3 valores inteiros e em uma função que receba estes 3 valores, retorne um valor inteiro com o cálculo do quadrado do 1º valor + a soma dos outros dois valores.

Q#7 - Título: Vetor com 8 Elementos e Índice do Vetor dos Números > 30

Criar um algoritmo que leia 8 valores inteiros e armazene-os em um vetor. Imprimir o vetor e informar o valor e seu índice na posição do vetor que contêm números maiores que 30.

Q#8 - Título: Campeonato e suas categorias de atletas

Faça um programa que leia o peso e a idade de 05 atletas. Informe para cada atleta qual a categoria ele pertence e gere um relatório no final. Conforme abaixo:

18 anos até 40 anos	**41 anos e acima**
Adulto	Veterano

Peso < 70kg	**Entre 80kg e 90kg**	**Peso >=91kg**
Leve	Médio	Pesado

Exemplo:

	Entrada	**Mensagem**
Atleta 01	Peso: 55 Idade: 19	Categoria Adulto Leve
Atleta 02	Peso: 80 Idade: 43	Categoria Adulto Medio
Atleta 03	Peso: 99 Idade: 32	Categoria Adulto Pesado
Atleta 04	Peso: 56 Idade: 60	Categoria Veterano Leve
Atleta 05	Peso: 85 Idade: 51	Categoria Veterano Medio

Resultado Final:

- 1 Atletas Categoria Adulto Leve
- 0 Altetas Categoria Adulto Medio
- 1 Atletas Categoria Adulto Pesado
- 1 Atletas Categoria Veterano Leve
- 2 Altetas Categoria Veterano Medio
- 0 Atletas Categoria Veterano Pesado

Items avaliados:

- Leitura correta dos dados (printf e scanf)
- Atribuição correta na idade/peso
- Uso correto de estrutura de repetição
- Impressão do Resultado Final correto

OBSERVAÇÃO: Fique atento a formatação de saída do TESTE BÁSICO da questão.
Não acentue as palavras no printf()

ATENÇÃO: Siga a seguinte ordem para a leitura dos dados: PESO, IDADE

Q#9 - Título: Votação para cassação de Senador

Em uma votação para cassação de senador, cada votante deve escolher entre as opções (Favorável/Contrário/Abstenção).

Os votos são informados através de códigos.

Os dados utilizados para a contagem dos votos obedecem à seguinte codificação:

- 111 para Favorável;
- 222 para Contrário;
- 333 para Abstenção;

Elabore um algoritmo que realize a leitura dos votos até que seja inseridos o **10 votos**.

Calcule e escreva:

I. Total de votos para cada opção;

Imprima o resultado de acordo com a regra:

- se o total de votos favoráveis é maior que os Contrários: Imprimir Resultado "Aprovado";
- senão se o total de votos contrários é maior que os favoráveis: imprimir: "Rejeitado";
- senão se o total de votos favoráveis é igual aos contrários: imprimir: "empate";

Exemplo de resultado:

- 6 votos Favoraveis
- 4 votos Contrarios
- 0 Abstencoes

Resultado: Aprovado

OBSERVAÇÃO: Fique atento a formatação de saída do TESTE BÁSICO da questão.
Não acentue as palavras no printf()

Q#10 - Título: Cálculo da média ponderada de 4 alunos e suas faltas

Faça um programa que leia a quantidade de faltas, 3 notas de 4 alunos e calcule a média final deste aluno. (Não use vetor!)

- A primeira nota (n1) é referente à entrega dos exercícios (nota de 0 a 10), peso 1;
- A segunda nota (p1) é referente à primeira prova (nota de 0 a 10) peso 2.5;
- A terceira nota (p2) e referente à segunda prova (nota de 0 a 10) peso 5;

A média ponderada é calculada pela fórmula: media=(n1+3*p1+5*p2)/9

Converter a média para conceitos de acordo com a tabela:

Se faltas forem **maiores que 5**	então reprovado por faltas (conceito E)
0~6.9	Conceito D - Reprovado
7.0~8.0	Conceito C
8,1~9.0	Conceito B
9.1~10	Conceito A

Itens avaliados:

- Apresentação de notas e conceitos. Leitura Correta das notas (Uso de comando de entrada/saída)
- Cálculo correto da média;
- Uso correto de Estrutura de Repetição para ler os 3 alunos;
- Apresentar os conceitos corretamente (Uso IF/ELSE)

OBSERVAÇÃO: Fique atento a formatação de saída do TESTE BÁSICO da questão.
Não acentue as palavras no printf()

ATENÇÃO: Siga a seguinte ordem para a leitura dos dados: FALTA, NOTA1 (N1), NOTA2 (P1), NOTA3(P2)

Q#11 - Título: Ordenado 3 números com ponteiros

Faça um programa que leia **3 números** e, usando ponteiros, coloque-os em ordem crescente e descrente, imprimindo-os nas duas ordens.

DICA: Identifique o maior, menor e meio entre os trê números para imprimir a ordem.

Q#12 - Título: Trabalhando com Matrizes

Faça um programa que realize algumas operações sobre uma **matriz 5 X 5.**

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

O algoritmo que inicializa a matriz está implementado e sobre esta matriz efetue as seguintes operações:

a) Calcular e apresentar Soma dos elementos divisíveis por 3 da diagonal SECUNDÁRIA. Obrigatório usar repetição.
b) Calcular e apresentar Soma de todos os elementos ímpares (usar repetição)
c) Buscar na matriz qual o maior valor, apresentar o maior valor;

d) Definir uma função que recebe uma matriz como parâmetro e o número da coluna; a função deve calcular a soma da coluna indicada e retornar o valor;
e) Usando laço e a função definida no item "D", apresentar a soma para cada coluna;
f) Definir função que calcula a soma de toda a matriz; a função deverá fazer uso da função definida no item "D".

Q#13 - Título: Calculando IMC de 5 pessoas

Escreva um programa que leia o nome, ano de nascimento, peso (kg) e altura (metros) para **5 pessoas**.

a) Para cada pessoa, o programa deve informar a idade que a pessoa completará no **ano corrente(2019)**; informar o imc e a classificação do imc; USAR laço de repetição;
b) Ao final, Indicar o peso total do grupo de pessoas;
c) Ao final, Indicar a altura média das pessoas;
d) Defina uma função que recebe como parâmetro o ano de nascimento e o ano corrente. O programa deve retornar quantos anos a pessoa irá completar neste ano;
e) Defina uma função que recebe como parâmetro o peso (kg) e a altura (metros) e retorna o imc desta pessoa;
f) Defina uma função que recebe como parâmetro o imc e imprima na tela a classificação do imc, de acordo com a tabela abaixo:

Resultado	**Situação**
Abaixo de 17	Muito abaixo do peso
Entre 17 e 18,49	Abaixo do peso
Entre 18,5 e 24,99	Peso normal
Entre 25 e 29,99	Acima do peso
Entre 30 e 34,99	Obesidade I
Entre 35 e 39,99	Obesidade II (severa)
Acima de 40	Obesidade III (mórbida)

ATENÇÃO: Siga a seguinte ordem para a leitura dos dados: ANO, PESO, ALTURA

Q#14 - Título: Conhecendo a classe eleitoral

Ler a idade de uma pessoa e informar a sua classe eleitoral.

a. Nao leitor (abaixo de 16 anos)

b. Eleitor obrigatorio (entre 18 e 65 anos)

c. Eleitor facultativo (entre 16 e 18 e maior de 65 anos)

ATENÇÃO: Não acentue as palavras no printf() do resultado final.

Q#15 - Título: Trabalhando com passagem de parâmetros por referência e valor

Escreva um programa que **leia 5 valores float a,b,c,d, e**:

a) Escreva uma função que recebe como parâmetro os **ponteiros** para float A,B,C, D, E e retorne o maior valor.

b) Escreva uma função que recebe como parâmetro os valores A,B,C, D, E e retorne o menor valor.

Observe que a primeira função você utilizará passagem de **parâmetro por referência**, e na segunda a passagem de **parâmetro será por valor**.

APÊNDICE I

REQUISITOS FUNCIONAIS, MODELO ER E SCRIPT DE BANCO DE DADOS

REQUISITOS FUNCIONAIS

Cada requisito identificado possui um código associado e é apresentado no formato RF #X, onde X é o código do requisito.

- **RF #01:** A ferramenta deverá trabalhar com perfil de acesso que inicialmente conterá: Administrador, Professor, e Aluno. Cada perfil indicará as permissões do usuário dentro da ferramenta;
- **RF #02:** Permitir o gerenciamento de Turmas e Disciplinas, possibilitando adicionar e remover alunos;
- **RF #03:** Permitir o gerenciamento de Eventos, associando a uma Turma e Disciplina, informando data de início e término para ocorrer. Possibilitar vincular um conjunto de questões. Proporcionar a consulta das entregas realizadas pelos alunos, informando junto à questão os status: Não Iniciado, Não Entregue, Resolvido com/sem erros, bem como acesso ao histórico das execuções da questão produzidos pelos estudantes;
- **RF #04:** Permitir o gerenciamento de Questões, podendo cadastrar novas questões, editá-las, e reutilizá-las em vários Eventos. Faz parte do cadastro da questão:
    - Nível de dificuldade: fácil, médio ou difícil;
    - Dica;
    - Enunciado;
    - Esqueleto inicial código;
    - Solução modelo definido pelo professor;
    - Tópicos associados à questão: Variáveis e Tipos de Dados, Estruturas de Controle, Estruturas de Decisão, Matrizes e Vetores, Manipulação de Strings, Funções e Ponteiros;
    - Recurso de exceção para forçar a saída do laço.
- **RF #05:** Permitir o gerenciamento dos casos de teste associando-os a uma questão;
- **RF #06:** O sistema deverá trabalhar com correção automática de Exercícios. As correções automáticas devem ser realizadas imediatamente após a execução das

questões para correção, para que o *feedback* seja imediato.

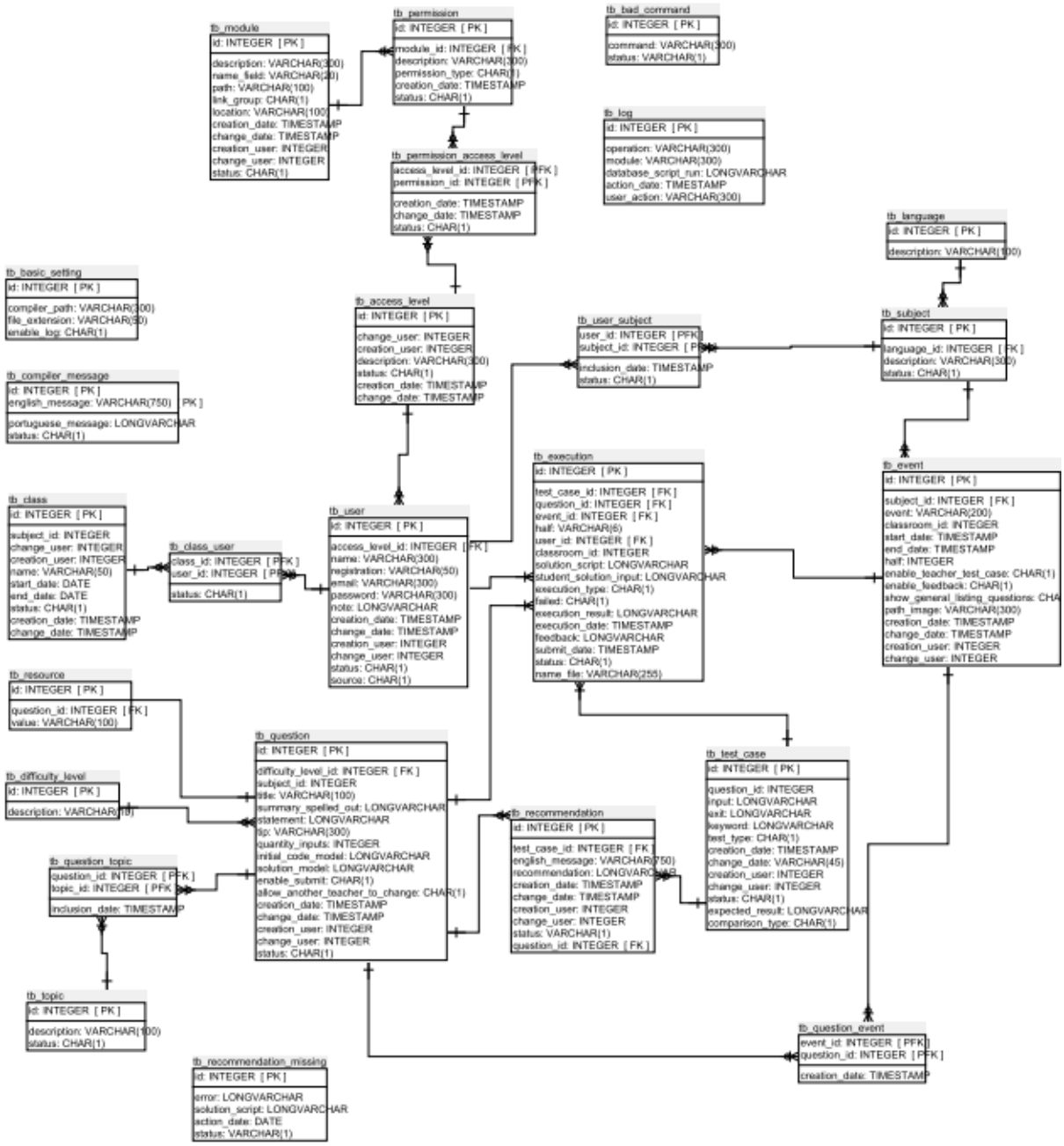
- **RF #07:** Disponibilizar ao Professor, os seguintes Relatórios:
    - Lista de submissões/não submissões em eventos dos alunos matriculados na turma;
    - Quantidade de erros ocorridos por tipo de erro;
    - Quantidade de acertos por tentativas por questão e por aluno separados por até 3 tentativas, 4 tentativas e mais do que 4 tentativas;
    - Quantidade de execuções, quantidade de acertos e erros por aluno e por questão.
- **RF #08:** Disponibilizar ao Aluno, o seguinte Relatório:
    - Desempenho geral por tópico e por questão
- **RF #09:** Permitir que o aluno responda os exercícios propostos pelo Professor, executando sua solução ilimitadas vezes, até o estudante Finalizar a questão. Disponibilizar recurso de parar a execução do código em caso de necessidade (*Loop* infinito);
- **RF #10:** Ao abrir a questão, o sistema deverá consultar se o estudante não iniciou a resolução. Se encontrar, deverá carregar o último desenvolvimento na área de programas; caso contrário, apresentar o esqueleto básico cadastrado pelo professor para o exercício;
- **RF #11:** Permitir o gerenciamento de alunos, possibilitando cadastrar, editá-lo e inativá-lo no ambiente. Possibilitar o cadastro em lote e inscrição pelo próprio aluno. Uma vez selecionada a opção de inscrição, viabilizar a opção do Professor aceitá-lo ou rejeitá-lo na disciplina;
- **RF #12:** Permitir o gerenciamento de comandos maliciosos a serem verificados no momento da execução do código pelo aluno;
- **RF #13:** Permitir ao Administrador configurar o ambiente, informando o caminho no qual está instalado o compilador e a opção de habilitar o log da ferramenta;
- **RF #14:** Permitir o gerenciamento das mensagens do compilador e suas recomendações (*feedback*) que serão utilizadas no momento que o ambiente identificar uma erro;

- **RF #15:** Permitir o Aluno acompanhar o histórico das execuções da questão, possibilitando fazer o download das versões dos códigos-fontes e/ou compartilhar com uma colega. Todos os códigos gerados pela ferramenta deverão possuir assinatura com as informações: nome do aluno, data/hora da execução, título da questão e evento na qual pertence (se houver);
- **RF #16:** Possibilitar a visualização dos caminhos percorridos pelo compilador no código executado (Teste de cobertura);
- **RF #17:** Permitir o Aluno testar o seu código com entradas informadas pelo professor ou com suas próprias;
- **RF #18:** Permitir o registro de LOGs na ferramenta de todas as ações realizadas pelo usuário;
- **RF #19:** Disponibilizar um espaço com os alertas da ferramenta. Os alertas apresentam os erros registrados que não possuem uma *feedback* associado;
- **RF #20:** Possibilitar o Aluno filtrar as questões por:
    - Nível de Dificuldade: fácil, médio ou difícil;
    - Tópico: Variáveis e Tipos de Dados, Estruturas de Controle, Estruturas de Decisão, Matrizes e Vetores, Manipulação de Strings, Funções e Ponteiros;
    - Disciplina;
    - Evento.

Ou buscar todas as questões ativas na ferramenta paginada. Na listagem das questões informar, junto ao título da questão:

- Nível de Dificuldade: fácil, médio ou difícil; Tópico associado; Disciplina associada; Linguagem de Programação necessária para responder; Status da resolução: Resolvido ou Não Resolvido.

MODELO RELACIONAL



SCRIPT DE BANCO DE DADOS

codeinplay.sql

```
-- phpMyAdmin SQL Dump
-- version 4.8.4
-- https://www.phpmyadmin.net/
--
-- Host: 127.0.0.1:3307
```

```
-- Generation Time: 19-Jan-2020 às 13:25
-- Versão do servidor: 10.3.12-MariaDB
-- versão do PHP: 7.2.14

SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
SET AUTOCOMMIT = 0;
START TRANSACTION;
SET time_zone = "+00:00";


/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8mb4 */;

--
-- Database: `codeinplay`
--

DELIMITER $$
--
-- Procedures
--
DROP PROCEDURE IF EXISTS `sp_enroll_user`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `sp_enroll_user` (IN `in_name` VARCHAR(300), IN `in_email` VARCHAR(300), IN `in_registration` INT(11), IN `in_subject_id` INT(11), IN `in_note` TEXT, IN `in_password` VARCHAR(300)) BEGIN

  declare new_user_id int;

    insert tb_user(id, access_level_id, name, registration, email, password, note, creation_date, status) values(default, 1, in_name, in_registration, in_email, in_password, in_note, now(), 2);

            set new_user_id = last_insert_id();

  insert tb_user_subject(user_id, subject_id, inclusion_date, status) values(new_user_id, in_subject_id, now(), 2);

END$$

DELIMITER ;

-- --------------------------------------------------------

--
-- Estrutura da tabela `tb_access_level`
--

DROP TABLE IF EXISTS `tb_access_level`;
CREATE TABLE IF NOT EXISTS `tb_access_level` (
 `id` int(11) NOT NULL AUTO_INCREMENT,
 `description` varchar(300) DEFAULT NULL,
 `status` char(1) DEFAULT NULL,
 `creation_date` timestamp NULL DEFAULT NULL,
 `change_date` timestamp NULL DEFAULT NULL,
 `creation_user` int(11) DEFAULT NULL,
 `change_user` int(11) DEFAULT NULL,
 PRIMARY KEY (`id`),
 KEY `fk_access_level_user_creation` (`creation_user`),
 KEY `fk_access_level_user_change` (`change_user`)
) ENGINE=InnoDB AUTO_INCREMENT=21 DEFAULT CHARSET=latin1;

--
-- Extraindo dados da tabela `tb_access_level`
```

```
--

INSERT INTO `tb_access_level` (`id`, `description`, `status`, `creation_date`, `change_date`, `creation_user`, `change_user`)
VALUES
(1, 'Aluno', '1', '2016-04-29 03:00:00', '2019-09-08 21:59:22', NULL, NULL),
(2, 'Professor', '1', '2019-01-10 04:54:19', '2019-09-13 22:00:43', NULL, NULL),
(3, 'Administrador', '1', '2016-04-29 03:00:00', '2019-09-13 22:00:39', NULL, NULL);

-- --------------------------------------------------------

--
-- Estrutura da tabela `tb_bad_command`
--

DROP TABLE IF EXISTS `tb_bad_command`;
CREATE TABLE IF NOT EXISTS `tb_bad_command` (
 `id` int(11) NOT NULL AUTO_INCREMENT,
 `command` varchar(300) NOT NULL,
 `status` varchar(1) NOT NULL,
 PRIMARY KEY (`id`)
) ENGINE=MyISAM AUTO_INCREMENT=6 DEFAULT CHARSET=latin1;

-- --------------------------------------------------------

--
-- Estrutura da tabela `tb_basic_setting`
--

DROP TABLE IF EXISTS `tb_basic_setting`;
CREATE TABLE IF NOT EXISTS `tb_basic_setting` (
 `id` int(11) NOT NULL AUTO_INCREMENT,
 `compiler_path` varchar(300) DEFAULT NULL,
 `file_extension` varchar(50) DEFAULT NULL,
 `enable_log` char(1) DEFAULT NULL,
 PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=latin1;

--
-- Extraindo dados da tabela `tb_basic_setting`
--

INSERT INTO `tb_basic_setting` (`id`, `compiler_path`, `file_extension`, `enable_log`) VALUES
(1, 'gcc', '.exe', '1');

-- --------------------------------------------------------

--
-- Estrutura da tabela `tb_class`
--

DROP TABLE IF EXISTS `tb_class`;
CREATE TABLE IF NOT EXISTS `tb_class` (
 `id` int(11) NOT NULL AUTO_INCREMENT,
 `subject_id` int(255) DEFAULT NULL,
 `change_user` int(255) DEFAULT NULL,
 `creation_user` int(255) DEFAULT NULL,
 `name` varchar(50) DEFAULT NULL,
 `start_date` date DEFAULT NULL,
 `end_date` date DEFAULT NULL,
 `status` char(1) DEFAULT NULL,
 `creation_date` timestamp NULL DEFAULT NULL,
 `change_date` timestamp NULL DEFAULT NULL,
 PRIMARY KEY (`id`),
 KEY `fk_class_creation_user_user` (`creation_user`),
```

```
 KEY `fk_class_change_user_user` (`change_user`),
 KEY `fk_class_subject` (`subject_id`)
) ENGINE=MyISAM AUTO_INCREMENT=6 DEFAULT CHARSET=latin1;

-- --------------------------------------------------------

--
-- Estrutura da tabela `tb_class_user`
--

DROP TABLE IF EXISTS `tb_class_user`;
CREATE TABLE IF NOT EXISTS `tb_class_user` (
 `class_id` int(11) DEFAULT 0,
 `user_id` int(11) DEFAULT NULL,
 `status` char(1) DEFAULT NULL,
 UNIQUE KEY `fk_class_user` (`class_id`,`user_id`),
 KEY `fk_class_user_user` (`user_id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

-- --------------------------------------------------------

--
-- Estrutura da tabela `tb_compiler_message`
--

DROP TABLE IF EXISTS `tb_compiler_message`;
CREATE TABLE IF NOT EXISTS `tb_compiler_message` (
 `id` int(11) NOT NULL AUTO_INCREMENT,
 `english_message` varchar(750) NOT NULL,
 `portuguese_message` mediumtext DEFAULT NULL,
 `status` char(1) DEFAULT NULL,
 PRIMARY KEY (`id`,`english_message`)
) ENGINE=InnoDB AUTO_INCREMENT=18 DEFAULT CHARSET=latin1;

-- --------------------------------------------------------

--
-- Estrutura da tabela `tb_difficulty_level`
--

DROP TABLE IF EXISTS `tb_difficulty_level`;
CREATE TABLE IF NOT EXISTS `tb_difficulty_level` (
 `id` int(11) NOT NULL AUTO_INCREMENT,
 `description` varchar(10) DEFAULT NULL,
 PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=latin1;

--
-- Extraindo dados da tabela `tb_difficulty_level`
--

INSERT INTO `tb_difficulty_level` (`id`, `description`) VALUES
(1, 'Fácil'),
(2, 'Médio'),
(3, 'Difícil');

-- --------------------------------------------------------

--
-- Estrutura da tabela `tb_event`
--

DROP TABLE IF EXISTS `tb_event`;
CREATE TABLE IF NOT EXISTS `tb_event` (
```

```
 `id` int(11) NOT NULL AUTO_INCREMENT,
 `event` varchar(200) DEFAULT NULL,
 `subject_id` int(11) DEFAULT NULL,
 `classroom_id` int(11) NOT NULL,
 `start_date` timestamp NULL DEFAULT NULL,
 `end_date` timestamp NULL DEFAULT NULL,
 `half` int(11) DEFAULT NULL,
 `enable_teacher_test_case` char(1) DEFAULT NULL,
 `enable_feedback` char(1) DEFAULT NULL,
 `show_general_listing_questions` char(1) DEFAULT NULL,
 `path_image` varchar(300) DEFAULT NULL,
 `creation_date` timestamp NULL DEFAULT NULL,
 `change_date` timestamp NULL DEFAULT NULL,
 `creation_user` int(11) DEFAULT NULL,
 `change_user` int(11) DEFAULT NULL,
 PRIMARY KEY (`id`),
 KEY `fk_event_subject` (`subject_id`),
 KEY `fk_event_user_creation` (`creation_user`),
 KEY `fk_event_user_change` (`change_user`)
) ENGINE=InnoDB AUTO_INCREMENT=16 DEFAULT CHARSET=latin1;

-- --------------------------------------------------------

--
-- Estrutura da tabela `tb_execution`
--

DROP TABLE IF EXISTS `tb_execution`;
CREATE TABLE IF NOT EXISTS `tb_execution` (
 `id` int(11) NOT NULL AUTO_INCREMENT,
 `event_id` int(255) DEFAULT NULL,
 `question_id` int(11) DEFAULT NULL,
 `test_case_id` int(11) DEFAULT NULL,
 `half` varchar(6) DEFAULT NULL,
 `user_id` int(11) DEFAULT NULL,
 `classroom_id` int(11) DEFAULT NULL,
 `solution_script` mediumtext DEFAULT NULL,
 `student_solution_input` text DEFAULT NULL,
 `execution_type` char(1) DEFAULT NULL,
 `failed` char(1) DEFAULT NULL,
 `execution_result` mediumtext DEFAULT NULL,
 `execution_date` timestamp NULL DEFAULT NULL,
 `feedback` text DEFAULT NULL,
 `submit_date` timestamp NULL DEFAULT NULL,
 `status` char(1) DEFAULT NULL,
 `name_file` varchar(255) DEFAULT NULL,
 PRIMARY KEY (`id`),
 KEY `fk_execution_question` (`question_id`),
 KEY `fk_execution_test_case` (`test_case_id`),
 KEY `fk_execution_event` (`event_id`)
) ENGINE=InnoDB AUTO_INCREMENT=3530 DEFAULT CHARSET=latin1;

-- --------------------------------------------------------

--
-- Estrutura da tabela `tb_language`
--

DROP TABLE IF EXISTS `tb_language`;
CREATE TABLE IF NOT EXISTS `tb_language` (
 `id` int(11) NOT NULL AUTO_INCREMENT,
 `description` varchar(100) DEFAULT NULL,
 PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=45 DEFAULT CHARSET=latin1;
```

```
--
-- Extraindo dados da tabela `tb_language`
--

INSERT INTO `tb_language` (`id`, `description`) VALUES
(5, 'C(GCC)');

-- --------------------------------------------------------

--
-- Estrutura da tabela `tb_log`
--

DROP TABLE IF EXISTS `tb_log`;
CREATE TABLE IF NOT EXISTS `tb_log` (
 `id` int(11) NOT NULL AUTO_INCREMENT,
 `operation` varchar(300) DEFAULT NULL,
 `module` varchar(300) DEFAULT NULL,
 `database_script_run` mediumtext DEFAULT NULL,
 `action_date` timestamp NULL DEFAULT NULL,
 `user_action` varchar(300) DEFAULT NULL,
 PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=1986 DEFAULT CHARSET=latin1;

-- --------------------------------------------------------

--
-- Estrutura da tabela `tb_module`
--

DROP TABLE IF EXISTS `tb_module`;
CREATE TABLE IF NOT EXISTS `tb_module` (
 `id` int(11) NOT NULL AUTO_INCREMENT,
 `description` varchar(300) DEFAULT NULL,
 `name_field` varchar(20) DEFAULT NULL,
 `path` varchar(100) DEFAULT NULL,
 `link_group` char(1) DEFAULT NULL,
 `location` varchar(100) DEFAULT NULL,
 `creation_date` timestamp NULL DEFAULT NULL,
 `change_date` timestamp NULL DEFAULT NULL,
 `creation_user` int(11) DEFAULT NULL,
 `change_user` int(11) DEFAULT NULL,
 `status` char(1) DEFAULT NULL,
 PRIMARY KEY (`id`),
 KEY `fk_module_user_creation` (`creation_user`),
 KEY `fk_module_user_change` (`change_user`)
) ENGINE=InnoDB AUTO_INCREMENT=17 DEFAULT CHARSET=latin1;

--
-- Extraindo dados da tabela `tb_module`
--

INSERT INTO `tb_module` (`id`, `description`, `name_field`, `path`, `link_group`, `location`, `creation_date`, `change_date`, `creation_user`, `change_user`, `status`) VALUES
(1, 'Configurações Básicas', 'basic_settings', 'BasicSetting/', '1', NULL, NULL, NULL, NULL, NULL, '1'),
(2, 'Questões', 'questions', 'didatic/question/', '2', NULL, NULL, NULL, NULL, NULL, '1'),
(3, 'Eventos', 'events', 'didatic/event/', '2', NULL, NULL, NULL, NULL, NULL, '1'),
(4, 'Mensagem do Compilador', 'message_compiler', 'didatic/CompilerMessage/', '2', NULL, NULL, NULL, NULL, NULL, '1'),
(5, 'Recomendações(Feedback)', 'recommendations', 'didatic/recommendation/', '2', NULL, NULL, NULL, NULL, NULL, '1'),
(6, 'Usuário', 'user', 'administration/user/', '3', NULL, NULL, NULL, NULL, NULL, '1'),
(7, 'Cadastro em Lote', 'batch_register', 'administration/BatchRegister/', '3', NULL, NULL, NULL, NULL, NULL, '1'),
```

```
(8, 'Nível de Acesso', 'access_level', 'administration/AccessLevel/', '3', NULL, NULL, NULL, NULL, NULL, '1'),
(9, 'Inscrições', 'subscriptions', 'administration/subscription/', '3', NULL, NULL, NULL, NULL, NULL, '1'),
(10, 'Histórico de Execução', 'execution_history', 'ExecutionHistory/', '4', NULL, NULL, NULL, NULL, NULL, '1'),
(11, 'Visualizar Log', 'view_log', 'ViewLog/', '4', NULL, NULL, NULL, NULL, NULL, '1'),
(12, 'Comandos Proibidos', 'bad_command', 'BadCommand/', '1', NULL, NULL, NULL, NULL, NULL, '1'),
(13, 'Turmas', 'classroom', 'organization/classRoom/', '3', NULL, NULL, NULL, NULL, NULL, '1'),
(14, 'Relatório de Erros', 'report_error', 'report/reportError/', '6', NULL, NULL, NULL, NULL, NULL, '1'),
(15, 'Relatório de Tentativas Acertos', 'report_attempt', 'report/reportAttempt/', '6', NULL, NULL, NULL, NULL, NULL, '1'),
(16, 'Relatório de Execuções', 'report_execution', 'report/reportExecution/', '6', NULL, NULL, NULL, NULL, NULL, '1');

-- --------------------------------------------------------

--
-- Estrutura da tabela `tb_permission`
--

DROP TABLE IF EXISTS `tb_permission`;
CREATE TABLE IF NOT EXISTS `tb_permission` (
 `id` int(11) NOT NULL AUTO_INCREMENT,
 `module_id` int(11) DEFAULT NULL,
 `description` varchar(300) DEFAULT NULL,
 `permission_type` char(1) DEFAULT NULL,
 `creation_date` timestamp NULL DEFAULT NULL,
 `status` char(1) DEFAULT NULL,
 PRIMARY KEY (`id`),
 KEY `fk_permission_module` (`module_id`)
) ENGINE=InnoDB AUTO_INCREMENT=28 DEFAULT CHARSET=latin1;

--
-- Extraindo dados da tabela `tb_permission`
--

INSERT INTO `tb_permission` (`id`, `module_id`, `description`, `permission_type`, `creation_date`, `status`) VALUES
(1, 1, 'visualizar', '1', '2019-01-11 02:00:00', '1'),
(2, 1, 'incluir / editar / excluir', '2', '2019-01-11 02:00:00', '1'),
(3, 2, 'visualizar', '1', '2019-01-11 02:00:00', '1'),
(4, 2, 'incluir / editar / excluir', '2', '2019-01-11 02:00:00', '1'),
(5, 3, 'visualizar', '1', '2019-01-11 02:00:00', '1'),
(6, 3, 'incluir / editar / excluir', '2', '2019-01-11 02:00:00', '1'),
(7, 4, 'visualizar', '1', '2019-01-11 02:00:00', '1'),
(8, 4, 'incluir / editar / excluir', '2', '2019-01-11 02:00:00', '1'),
(9, 5, 'visualizar', '1', '2019-01-11 02:00:00', '1'),
(10, 5, 'incluir / editar / excluir', '2', '2019-01-11 02:00:00', '1'),
(11, 6, 'visualizar', '1', '2019-01-11 02:00:00', '1'),
(12, 6, 'incluir / editar / excluir', '2', '2019-01-11 02:00:00', '1'),
(13, 7, 'visualizar', '1', '2019-01-11 02:00:00', '1'),
(14, 7, 'incluir / editar / excluir', '2', '2019-01-11 02:00:00', '1'),
(15, 8, 'visualizar', '1', '2019-01-11 02:00:00', '1'),
(16, 8, 'incluir / editar / excluir', '2', '2019-01-11 02:00:00', '1'),
(17, 9, 'visualizar', '1', '2019-01-11 02:00:00', '1'),
(18, 9, 'incluir / editar / excluir', '2', '2019-01-11 02:00:00', '1'),
(19, 10, 'visualizar', '1', '2019-01-11 02:00:00', '1'),
(20, 11, 'visualizar', '1', '2019-01-11 02:00:00', '1'),
(21, 12, 'visualizar', '1', '2019-01-11 02:00:00', '1'),
(22, 12, 'incluir / editar / excluir', '2', '2019-01-11 02:00:00', '1'),
(23, 13, 'visualizar', '1', '2019-01-11 02:00:00', '1'),
(24, 13, 'incluir / editar / excluir', '2', '2019-01-11 02:00:00', '1'),
(25, 14, 'visualizar', '1', '2019-01-11 02:00:00', '1'),
(26, 15, 'visualizar', '1', '2019-01-11 02:00:00', '1'),
(27, 16, 'visualizar', '1', '2019-01-11 02:00:00', '1');

-- --------------------------------------------------------

--
```

```
-- Estrutura da tabela `tb_permission_access_level`
--

DROP TABLE IF EXISTS `tb_permission_access_level`;
CREATE TABLE IF NOT EXISTS `tb_permission_access_level` (
 `access_level_id` int(11) NOT NULL,
 `permission_id` int(11) NOT NULL,
 `creation_date` timestamp NULL DEFAULT NULL,
 `change_date` timestamp NULL DEFAULT NULL,
 `status` char(1) DEFAULT NULL,
 PRIMARY KEY (`access_level_id`,`permission_id`),
 KEY `fk_permission_access_level_permission` (`permission_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Extraindo dados da tabela `tb_permission_access_level`
--

INSERT INTO `tb_permission_access_level` (`access_level_id`, `permission_id`, `creation_date`, `change_date`, `status`)
VALUES
(2, 1, '2019-03-13 00:00:59', '2019-09-13 22:00:43', '1'),
(2, 4, '2019-03-13 01:19:37', '2019-09-13 22:00:43', '1'),
(2, 5, '2019-03-13 00:14:08', '2019-09-13 22:00:43', '1'),
(2, 7, '2019-03-13 00:14:08', '2019-09-13 22:00:43', '1'),
(2, 9, '2019-03-13 00:22:10', '2019-09-13 22:00:43', '1'),
(2, 11, '2019-03-13 00:01:49', '2019-09-13 22:00:43', '1'),
(2, 13, '2019-03-13 00:22:10', '2019-09-13 22:00:43', '1'),
(2, 15, '2019-03-13 00:52:47', '2019-09-13 22:00:43', '1'),
(2, 17, '2019-03-13 00:54:20', '2019-09-13 22:00:43', '1'),
(2, 19, '2019-03-13 00:55:20', '2019-09-13 22:00:43', '1'),
(2, 20, '2019-03-13 00:55:20', '2019-09-13 22:00:43', '1'),
(2, 22, '2019-09-09 19:51:19', '2019-09-13 22:00:43', '1'),
(2, 24, '2019-09-09 19:51:19', '2019-09-13 22:00:43', '1'),
(2, 25, '2019-09-13 04:04:46', '2019-09-13 22:00:43', '1'),
(2, 26, '2019-09-13 04:01:03', '2019-09-13 22:00:43', '1'),
(2, 27, '2019-09-13 22:00:43', NULL, '1'),
(3, 2, '2019-01-13 19:40:46', '2019-09-13 22:00:39', '1'),
(3, 4, '2019-01-13 19:40:46', '2019-09-13 22:00:39', '1'),
(3, 6, '2019-02-16 21:20:14', '2019-09-13 22:00:39', '1'),
(3, 8, '2019-01-13 19:40:46', '2019-09-13 22:00:39', '1'),
(3, 10, '2019-01-13 19:40:46', '2019-09-13 22:00:39', '1'),
(3, 12, '2019-01-13 19:40:46', '2019-09-13 22:00:39', '1'),
(3, 14, '2019-01-13 19:40:46', '2019-09-13 22:00:39', '1'),
(3, 16, NULL, '2019-09-13 22:00:39', '1'),
(3, 18, '2019-01-13 19:40:46', '2019-09-13 22:00:39', '1'),
(3, 19, '2019-01-17 19:50:40', '2019-09-13 22:00:39', '1'),
(3, 20, '2019-01-13 19:40:46', '2019-09-13 22:00:39', '1'),
(3, 22, '2019-09-04 20:53:38', '2019-09-13 22:00:39', '1'),
(3, 24, '2019-09-09 19:51:14', '2019-09-13 22:00:39', '1'),
(3, 25, '2019-09-13 04:02:49', '2019-09-13 22:00:39', '1'),
(3, 26, '2019-09-13 04:04:36', '2019-09-13 22:00:39', '1'),
(3, 27, '2019-09-13 22:00:39', NULL, '1');

-- --------------------------------------------------------

--
-- Estrutura da tabela `tb_question`
--

DROP TABLE IF EXISTS `tb_question`;
CREATE TABLE IF NOT EXISTS `tb_question` (
 `id` int(11) NOT NULL AUTO_INCREMENT,
 `subject_id` int(11) DEFAULT NULL,
 `difficulty_level_id` int(11) DEFAULT NULL,
```

```
 `title` varchar(100) DEFAULT NULL,
 `summary_spelled_out` mediumtext DEFAULT NULL,
 `statement` mediumtext DEFAULT NULL,
 `tip` varchar(300) DEFAULT NULL,
 `quantity_inputs` int(11) DEFAULT NULL,
 `initial_code_model` mediumtext DEFAULT NULL,
 `solution_model` mediumtext DEFAULT NULL,
 `enable_submit` char(1) DEFAULT NULL,
 `allow_another_teacher_to_change` char(1) DEFAULT NULL,
 `creation_date` timestamp NULL DEFAULT NULL,
 `change_date` timestamp NULL DEFAULT NULL,
 `creation_user` int(11) DEFAULT NULL,
 `change_user` int(11) DEFAULT NULL,
 `status` char(1) DEFAULT NULL,
 PRIMARY KEY (`id`),
 KEY `fk_question_subject` (`subject_id`),
 KEY `fk_question_difficulty_level` (`difficulty_level_id`),
 KEY `fk_question_user_creation` (`creation_user`),
 KEY `fk_question_user_change` (`change_user`)
) ENGINE=InnoDB AUTO_INCREMENT=80 DEFAULT CHARSET=latin1;

-- --------------------------------------------------------

--
-- Estrutura da tabela `tb_question_event`
--

DROP TABLE IF EXISTS `tb_question_event`;
CREATE TABLE IF NOT EXISTS `tb_question_event` (
 `event_id` int(11) NOT NULL,
 `question_id` int(11) NOT NULL,
 `creation_date` timestamp NULL DEFAULT NULL,
 PRIMARY KEY (`event_id`,`question_id`),
 KEY `fk_question_event_question` (`question_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-- --------------------------------------------------------

--
-- Estrutura da tabela `tb_question_topic`
--

DROP TABLE IF EXISTS `tb_question_topic`;
CREATE TABLE IF NOT EXISTS `tb_question_topic` (
 `question_id` int(11) NOT NULL,
 `topic_id` int(11) NOT NULL AUTO_INCREMENT,
 `inclusion_date` timestamp NULL DEFAULT NULL,
 PRIMARY KEY (`question_id`,`topic_id`),
 KEY `fk_question_topic_topic` (`topic_id`)
) ENGINE=InnoDB AUTO_INCREMENT=8 DEFAULT CHARSET=latin1;

-- --------------------------------------------------------

--
-- Estrutura da tabela `tb_recommendation`
--

DROP TABLE IF EXISTS `tb_recommendation`;
CREATE TABLE IF NOT EXISTS `tb_recommendation` (
 `id` int(11) NOT NULL AUTO_INCREMENT,
 `test_case_id` int(11) DEFAULT NULL,
 `english_message` varchar(750) DEFAULT NULL,
 `recommendation` mediumtext DEFAULT NULL,
 `creation_date` timestamp NULL DEFAULT NULL,
```

```
 `change_date` timestamp NULL DEFAULT NULL,
 `creation_user` int(11) DEFAULT NULL,
 `change_user` int(11) DEFAULT NULL,
 `status` varchar(1) DEFAULT NULL,
 `question_id` int(11) DEFAULT NULL,
 PRIMARY KEY (`id`),
 KEY `fk_recommendation_test_case` (`test_case_id`),
 KEY `fk_recommendation_user_change` (`change_user`),
 KEY `fk_recommendation_user_creation` (`creation_user`),
 KEY `fk_recommendation_question` (`question_id`)
) ENGINE=InnoDB AUTO_INCREMENT=131 DEFAULT CHARSET=latin1;

-- --------------------------------------------------------

--
-- Estrutura da tabela `tb_recommendation_missing`
--

DROP TABLE IF EXISTS `tb_recommendation_missing`;
CREATE TABLE IF NOT EXISTS `tb_recommendation_missing` (
 `id` int(11) NOT NULL AUTO_INCREMENT,
 `error` text NOT NULL,
 `solution_script` text NOT NULL,
 `action_date` date NOT NULL,
 `status` varchar(1) NOT NULL DEFAULT '1',
 PRIMARY KEY (`id`)
) ENGINE=MyISAM AUTO_INCREMENT=50 DEFAULT CHARSET=latin1;

-- --------------------------------------------------------

--
-- Estrutura da tabela `tb_resource`
--

DROP TABLE IF EXISTS `tb_resource`;
CREATE TABLE IF NOT EXISTS `tb_resource` (
 `id` int(11) NOT NULL AUTO_INCREMENT,
 `question_id` int(11) DEFAULT NULL,
 `value` varchar(100) DEFAULT NULL,
 PRIMARY KEY (`id`),
 KEY `fk_resource_question` (`question_id`)
) ENGINE=InnoDB AUTO_INCREMENT=51 DEFAULT CHARSET=latin1;

-- --------------------------------------------------------

--
-- Estrutura da tabela `tb_subject`
--

DROP TABLE IF EXISTS `tb_subject`;
CREATE TABLE IF NOT EXISTS `tb_subject` (
 `id` int(11) NOT NULL AUTO_INCREMENT,
 `language_id` int(11) NOT NULL,
 `description` varchar(300) DEFAULT NULL,
 `status` char(1) DEFAULT NULL,
 PRIMARY KEY (`id`),
 KEY `fk_subject_language` (`language_id`)
) ENGINE=InnoDB AUTO_INCREMENT=9 DEFAULT CHARSET=latin1;

--
-- Extraindo dados da tabela `tb_subject`
--

INSERT INTO `tb_subject` (`id`, `language_id`, `description`, `status`) VALUES
```

```
(7, 5, 'LPI', '1');

-- --------------------------------------------------------

--
-- Estrutura da tabela `tb_test_case`
--

DROP TABLE IF EXISTS `tb_test_case`;
CREATE TABLE IF NOT EXISTS `tb_test_case` (
 `id` int(11) NOT NULL AUTO_INCREMENT,
 `question_id` int(11) DEFAULT NULL,
 `input` text DEFAULT NULL,
 `exit` text DEFAULT NULL,
 `keyword` text DEFAULT NULL,
 `test_type` char(1) DEFAULT NULL,
 `creation_date` timestamp NULL DEFAULT NULL,
 `change_date` varchar(45) DEFAULT NULL,
 `creation_user` int(11) DEFAULT NULL,
 `change_user` int(11) DEFAULT NULL,
 `status` char(1) DEFAULT NULL,
 `expected_result` text DEFAULT NULL,
 `comparison_type` char(1) DEFAULT NULL,
 PRIMARY KEY (`id`),
 KEY `fk_test_case_question` (`question_id`),
 KEY `fk_test_case_user_creation` (`creation_user`),
 KEY `fK_test_case_user_change` (`change_user`)
) ENGINE=InnoDB AUTO_INCREMENT=268 DEFAULT CHARSET=latin1;

-- --------------------------------------------------------

--
-- Estrutura da tabela `tb_topic`
--

DROP TABLE IF EXISTS `tb_topic`;
CREATE TABLE IF NOT EXISTS `tb_topic` (
 `id` int(11) NOT NULL AUTO_INCREMENT,
 `description` varchar(100) DEFAULT NULL,
 `status` char(1) DEFAULT NULL,
 PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=8 DEFAULT CHARSET=latin1;

--
-- Extraindo dados da tabela `tb_topic`
--

INSERT INTO `tb_topic` (`id`, `description`, `status`) VALUES
(1, 'Variáveis e Tipos de Dados', '1'),
(2, 'Estruturas de Controle', '1'),
(3, 'Estruturas de Decisão', '1'),
(4, 'Matrizes e Vetores', '1'),
(5, 'Funções', '1'),
(6, 'Manipulação de strings', '1'),
(7, 'Ponteiros', '1');

-- --------------------------------------------------------

--
-- Estrutura da tabela `tb_user`
--

DROP TABLE IF EXISTS `tb_user`;
CREATE TABLE IF NOT EXISTS `tb_user` (
```

```
 `id` int(11) NOT NULL AUTO_INCREMENT,
 `access_level_id` int(11) NOT NULL,
 `name` varchar(300) DEFAULT NULL,
 `registration` varchar(50) DEFAULT NULL,
 `email` varchar(300) DEFAULT NULL,
 `password` varchar(300) DEFAULT NULL,
 `note` text DEFAULT NULL,
 `creation_date` timestamp NULL DEFAULT NULL,
 `change_date` timestamp NULL DEFAULT NULL,
 `creation_user` int(11) DEFAULT NULL,
 `change_user` int(11) DEFAULT NULL,
 `status` char(1) DEFAULT NULL,
 `source` char(1) DEFAULT NULL,
 PRIMARY KEY (`id`),
 KEY `fk_user_access_level` (`access_level_id`),
 KEY `fk_creation_user` (`creation_user`),
 KEY `fk_change_user` (`change_user`)
) ENGINE=InnoDB AUTO_INCREMENT=247 DEFAULT CHARSET=latin1;

--
-- Extraindo dados da tabela `tb_user`
--

INSERT INTO `tb_user` (`id`, `access_level_id`, `name`, `registration`, `email`, `password`, `note`, `creation_date`, `change_date`, `creation_user`, `change_user`, `status`, `source`) VALUES
(34, 3, 'Administrador', '123', 'administrador@administrador.com', 'if13Dpsyoylmc', 'Administrador', '2019-01-14 04:44:30', NULL, 1, NULL, '1', '1'),
-- --------------------------------------------------------

--
-- Estrutura da tabela `tb_user_subject`
--

DROP TABLE IF EXISTS `tb_user_subject`;
CREATE TABLE IF NOT EXISTS `tb_user_subject` (
 `user_id` int(11) NOT NULL,
 `subject_id` int(11) NOT NULL,
 `inclusion_date` timestamp NULL DEFAULT NULL,
 `status` char(1) DEFAULT NULL,
 PRIMARY KEY (`user_id`,`subject_id`),
 KEY `fk_user_subject_subject` (`subject_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Constraints for dumped tables
--

--
-- Limitadores para a tabela `tb_access_level`
--
ALTER TABLE `tb_access_level`
 ADD CONSTRAINT `fk_access_level_user_change` FOREIGN KEY (`change_user`) REFERENCES `tb_user` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION,
 ADD CONSTRAINT `fk_access_level_user_creation` FOREIGN KEY (`creation_user`) REFERENCES `tb_user` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION;

--
-- Limitadores para a tabela `tb_event`
--
ALTER TABLE `tb_event`
 ADD CONSTRAINT `fk_event_subject` FOREIGN KEY (`subject_id`) REFERENCES `tb_subject` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION,
 ADD CONSTRAINT `fk_event_user_change` FOREIGN KEY (`change_user`) REFERENCES `tb_user` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION,
```

```
 ADD CONSTRAINT `fk_event_user_creation` FOREIGN KEY (`creation_user`) REFERENCES `tb_user` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION;

--
-- Limitadores para a tabela `tb_execution`
--
ALTER TABLE `tb_execution`
 ADD CONSTRAINT `fk_execution_event` FOREIGN KEY (`event_id`) REFERENCES `tb_event` (`id`),
 ADD CONSTRAINT `fk_execution_question` FOREIGN KEY (`question_id`) REFERENCES `tb_question` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION,
 ADD CONSTRAINT `fk_execution_test_case` FOREIGN KEY (`test_case_id`) REFERENCES `tb_test_case` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION;

--
-- Limitadores para a tabela `tb_module`
--
ALTER TABLE `tb_module`
 ADD CONSTRAINT `fk_module_user_change` FOREIGN KEY (`change_user`) REFERENCES `tb_user` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION,
 ADD CONSTRAINT `fk_module_user_creation` FOREIGN KEY (`creation_user`) REFERENCES `tb_user` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION;

--
-- Limitadores para a tabela `tb_permission`
--
ALTER TABLE `tb_permission`
 ADD CONSTRAINT `fk_permission_module` FOREIGN KEY (`module_id`) REFERENCES `tb_module` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION;

--
-- Limitadores para a tabela `tb_permission_access_level`
--
ALTER TABLE `tb_permission_access_level`
 ADD CONSTRAINT `fk_permission_access_level_access_level` FOREIGN KEY (`access_level_id`) REFERENCES `tb_access_level` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION,
 ADD CONSTRAINT `fk_permission_access_level_permission` FOREIGN KEY (`permission_id`) REFERENCES `tb_permission` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION;

--
-- Limitadores para a tabela `tb_question`
--
ALTER TABLE `tb_question`
 ADD CONSTRAINT `fk_question_difficulty_level` FOREIGN KEY (`difficulty_level_id`) REFERENCES `tb_difficulty_level` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION,
 ADD CONSTRAINT `fk_question_subject` FOREIGN KEY (`subject_id`) REFERENCES `tb_subject` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION,
 ADD CONSTRAINT `fk_question_user_change` FOREIGN KEY (`change_user`) REFERENCES `tb_user` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION,
 ADD CONSTRAINT `fk_question_user_creation` FOREIGN KEY (`creation_user`) REFERENCES `tb_user` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION;

--
-- Limitadores para a tabela `tb_question_event`
--
ALTER TABLE `tb_question_event`
 ADD CONSTRAINT `fk_question_event_event` FOREIGN KEY (`event_id`) REFERENCES `tb_event` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION,
 ADD CONSTRAINT `fk_question_event_question` FOREIGN KEY (`question_id`) REFERENCES `tb_question` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION;

--
-- Limitadores para a tabela `tb_question_topic`
--
ALTER TABLE `tb_question_topic`
```

```
 ADD CONSTRAINT `fk_question_topic_question` FOREIGN KEY (`question_id`) REFERENCES `tb_question` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION,
 ADD CONSTRAINT `fk_question_topic_topic` FOREIGN KEY (`topic_id`) REFERENCES `tb_topic` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION;

--
-- Limitadores para a tabela `tb_recommendation`
--
ALTER TABLE `tb_recommendation`
 ADD CONSTRAINT `fk_recommendation_question` FOREIGN KEY (`question_id`) REFERENCES `tb_question` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION,
 ADD CONSTRAINT `fk_recommendation_test_case` FOREIGN KEY (`test_case_id`) REFERENCES `tb_test_case` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION,
 ADD CONSTRAINT `fk_recommendation_user_change` FOREIGN KEY (`change_user`) REFERENCES `tb_user` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION,
  ADD CONSTRAINT `fk_recommendation_user_creation` FOREIGN KEY (`creation_user`) REFERENCES `tb_user` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION;

--
-- Limitadores para a tabela `tb_resource`
--
ALTER TABLE `tb_resource`
  ADD CONSTRAINT `fk_resource_question` FOREIGN KEY (`question_id`) REFERENCES `tb_question` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION;

--
-- Limitadores para a tabela `tb_subject`
--
ALTER TABLE `tb_subject`
  ADD CONSTRAINT `fk_subject_language` FOREIGN KEY (`language_id`) REFERENCES `tb_language` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION;

--
-- Limitadores para a tabela `tb_test_case`
--
ALTER TABLE `tb_test_case`
  ADD CONSTRAINT `fK_test_case_user_change` FOREIGN KEY (`change_user`) REFERENCES `tb_user` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION,
  ADD CONSTRAINT `fk_test_case_question` FOREIGN KEY (`question_id`) REFERENCES `tb_question` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION,
 ADD CONSTRAINT `fk_test_case_user_creation` FOREIGN KEY (`creation_user`) REFERENCES `tb_user` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION;

--
-- Limitadores para a tabela `tb_user`
--
ALTER TABLE `tb_user`
 ADD CONSTRAINT `fk_change_user` FOREIGN KEY (`change_user`) REFERENCES `tb_user` (`id`),
 ADD CONSTRAINT `fk_creation_user` FOREIGN KEY (`creation_user`) REFERENCES `tb_user` (`id`);

--
-- Limitadores para a tabela `tb_user_subject`
--
ALTER TABLE `tb_user_subject`
  ADD CONSTRAINT `fk_user_subject_subject` FOREIGN KEY (`subject_id`) REFERENCES `tb_subject` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION,
 ADD CONSTRAINT `fk_user_subject_user` FOREIGN KEY (`user_id`) REFERENCES `tb_user` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION;
COMMIT;

/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
```

ANEXO A

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO RIO GRANDE DO SUL – IFRS
PRÓ-REITORIA DE PESQUISA, PÓS-GRADUAÇÃO E INOVAÇÃO – PROPPI
COMITÊ DE ÉTICA EM PESQUISA – CEP

TERMO DE CONSENTIMENTO LIVRE E ESCLARECIDO

Prezado (a) Senhor (a):

Você está sendo respeitosamente convidado(a) a participar do projeto de pesquisa intitulado: **Codein'play: Um ambiente de mediação do erro a partir da avaliação de exercícios de programação de computadores**, cujo objetivo é verificar como um ambiente de *feedback*/recomendação pode contribuir na mediação do erro durante o aprendizado, provendo informações detalhadas sobre o erro para que ele sirva como parte integrante do processo de formação de conceitos em programação de computadores aos estudantes. Este projeto está vinculado ao Mestrado Profissional em Informática na Educação - MPIE.

A pesquisa será feita no/a Instituto Federal do Rio Grande do Sul – IFRS, através de Observação das aulas de Linguagem de Programação I, Preenchimento de Questionário pelos alunos, utilização da ferramenta proposta para este trabalho em exercícios em sala de aula e entrevista com professor da disciplina, que poderá ser gravada e/ou filmada, após minha autorização. Para a coleta de dados será utilizado/a Observação, Aplicação de questionário, captura da resolução dos exercícios pela ferramenta Codein'play e Entrevista.

==========

Fui alertado (a) que este estudo apresenta risco mínimo, isto é, pode ocorrer algum tipo de desconforto emocional ou constrangimento ao participante, com a possibilidade de não conseguir realizar satisfatoriamente as atividades propostas em sala sabendo que os erros serão gravados e depois analisados. Além disso, os participantes, também, poderão optar por não querer participar das atividades com o ambiente computacional Codein'play, sem

prejuízo da avaliação, permitindo o uso de outras ferramentas, como o NetBeans por exemplo. Caso isso ocorra, serei encaminhado para o pesquisador que está realizando a pesquisa ou professor da disciplina a fim de receber o acompanhamento necessário. Além disso, diante de qualquer tipo de questionamento ou dúvida poderei realizar o contato imediato com um dos pesquisadores responsáveis pelo estudo que fornecerá os esclarecimentos necessários.

Foi destacado que minha participação no estudo é de extrema importância, uma vez que espera-se:

- Ambiente online com conjunto de exercícios para a linguagem de programação C (compilador GCC), com opção de salvar os códigos para, em outro momento, dar continuidade ou consultar;
- Execução de bateria de testes unitários fornecidos pelo professor, para conferência do código pelo aluno;
- Identificação dos erros sintáticos, semânticos, lexicais e lógicos com *feedback* dos possíveis problemas para entendimento e correção do erro, e recomendação de materiais de apoio para reforço de conceitos sobre os conteúdos relacionados ao exercício;
- Mensagens de erro em Português;
- Apresentação visual do resultado do teste de cobertura com as entradas de dados informados pelo aluno.

Estou ciente e me foram assegurados os seguintes direitos:

- da liberdade de retirar o meu consentimento, a qualquer momento, e deixar de participar do estudo, sem que isso me traga prejuízo de qualquer ordem;

- da segurança de que não serei identificado (a) e que será mantido caráter confidencial das informações relacionadas à minha privacidade;

- de que serão mantidos todos os preceitos ético-legais durante e após o término da pesquisa, de acordo com a Resolução 466/2012 do Conselho Nacional de Saúde;

- do compromisso de ter acesso às informações em todas as etapas do estudo, bem como aos resultados, ainda que isso possa afetar meu interesse em continuar participando da pesquisa;

- de que não haverá nenhum tipo de despesa ou ônus financeiro, bem como não haverá nenhuma recompensa financeira relacionada à minha participação;

- de que não está previsto nenhum tipo de procedimento invasivo, coleta de material biológico, ou experimento com seres humanos;

- de não responder qualquer pergunta que julgar constrangedora ou inadequada.

Eu _____, portador do documento de identidade _____, aceito participar da pesquisa intitulada: **Codein'play: Um ambiente de mediação do erro a partir da avaliação de exercícios de programação de computadores**. Fui informado (a) dos objetivos do presente estudo de maneira clara e detalhada, bem como sobre a metodologia que será adotada, sobre os riscos e benefícios envolvidos. Recebi uma cópia deste termo de consentimento e me foi dada a oportunidade de ler e esclarecer as minhas dúvidas.

Porto Alegre, ___ de _____de _____.

_____ _____

Assinatura do (a) participante

_

Assinatura do (a) pesquisador(a)

Em caso de dúvidas com respeito aos aspectos éticos deste estudo, poderei consultar:

CEP/IFRS
E-mail: cepesquisa@ifrs.edu.br
Endereço: Rua General Osório, 348, Centro, Bento Gonçalves, RS, CEP: 95.700-000
Telefone: (54) 3449-3340

Pesquisador(a) principal: KAREN CRISTINA BRAGA
Telefone para contato: 51 98121-6381
E-mail para contato: karencbraga@gmail.com

ANEXO B

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO RIO GRANDE DO SUL – IFRS
PRÓ-REITORIA DE PESQUISA, PÓS-GRADUAÇÃO E INOVAÇÃO – PROPPI
COMITÊ DE ÉTICA EM PESQUISA – CEP

AUTORIZAÇÃO INSTITUCIONAL

Eu, **Marcelo Rauh Schmitt**, responsável pela instituição **Instituto Federal do Rio Grande do Sul – IFRS**, autorizo a realização da pesquisa intitulada **Codein'play: Um ambiente de mediação do erro a partir da avaliação de exercícios de programação de computadores**, a ser conduzido pelos pesquisadores abaixo relacionados. Fui informado pelo responsável do estudo sobre objetivos, metodologia, riscos e benefícios aos participantes da pesquisa, bem como das atividades que serão realizadas na instituição a qual represento.

Foi assegurado pelo pesquisador responsável que os dados coletados serão mantidos em absoluto sigilo de acordo com a Resolução do Conselho Nacional de Saúde nº 466/2012, que trata da Pesquisa envolvendo seres humanos e que serão utilizados tão somente para a realização deste estudo.

Esta instituição está ciente de suas co-responsabilidades como instituição co-participante do presente projeto de pesquisa e de seu compromisso no resguardo da segurança e bem-estar dos participantes de pesquisa, dispondo de infra-estrutura necessária para a garantia de tal segurança e bem-estar.

Serão disponibilizados, ao pesquisador:

- Acesso ao laboratório de informática com a turma da disciplina de Linguagem de Programação I para observação, acompanhamento e avaliação da ferramenta Codein'play;
- Hospedagem da ferramenta no servidor do IFRS com suporte a linguagem PHP, MySQL.

Porto Alegre, 07 de NOVEMBRO de 2018.

_____

Assinatura e carimbo do responsável institucional
Cargo que ocupa na instituição

Em caso de dúvidas com respeito aos aspectos éticos deste estudo, consultar:

CEP/IFRS
E-mail: cepesquisa@ifrs.edu.br
Endereço: Rua General Osório, 348, Centro, Bento Gonçalves, RS, CEP: 95.700-000
Telefone: (54) 3449-3340

Pesquisador(a) principal: KAREN CRISTINA BRAGA
Telefone para contato: 51 98121-6381
E-mail para contato: karencbraga@gmail.com

Demais pesquisadores:
Nome: FABIO YOSHIMITSU OKUYAMA
Telefone para contato: 51 98119-7992
E-mail para contato: fabio.okuyama@poa.ifrs.edu.br

Nome: MÁRCIA AMARAL CORRÊA DE MORAES
Telefone para contato: 51 99967-9526
E-mail para contato: marcia.moraes@poa.ifrs.edu.br

ANEXO C

PARECER CONSUBSTANCIADO DO CEP

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO RIO

Plataforma Brasil

PARECER CONSUBSTANCIADO DO CEP

DADOS DO PROJETO DE PESQUISA

Título da Pesquisa: ifCODE: Um ambiente de mediação do erro a partir da avaliação de exercícios de programação de computadores

Pesquisador: KAREN CRISTINA BRAGA

Área Temática:

Versão: 1

CAAE: 02644218.7.0000.8024

Instituição Proponente: INSTITUTO FEDERAL DE EDUCACAO, CIENCIA E TECNOLOGIA DO RIO

Patrocinador Principal: Financiamento Próprio

DADOS DO PARECER

Número do Parecer: 3.066.476

Apresentação do Projeto:

O ensino de algoritmos é requisito fundamental para os cursos de computação no ensino superior e técnico. Porém, é preocupante a quantidade de desistências de estudantes do curso de computação em função da dificuldade em compreender e aplicar conceitos abstratos de programação na resolução dos problemas propostos em sala de aula. Pesquisadores buscam identificar as dificuldades de aprendizagem dos alunos encontradas nas disciplinas iniciais de programação e mapear os erros mais frequentes. Na perspectiva tradicional de ensino, o erro é considerado uma barreira na formação do conhecimento e este deveria ser eliminado do processo de ensino-aprendizagem. No entanto, abordagens pedagógicas atuais salientam a importância do erro na educação, porém, pouco, ainda, se faz a partir deste na prática. Sabe-se da necessidade em detectar as dificuldades na aprendizagem da programação e, consequentemente, os erros nos códigos gerados durante a jornada educacional para que se consiga mediar o problema a tempo. Uma vez detectados, é essencial garantir o acesso aos registros de erros pelos professores e alunos, a fim de identificá-los e problematizá-los para reflexão, e desta reflexão, os novos conhecimentos serem assimilados e acomodados. Em via de regra, um erro tende a ocorrer em certo contexto, em um momento específico do desenvolvimento cognitivo do aluno e não é isolado de outros erros. Em face disso, a proposta deste projeto é o desenvolvimento de um ambiente para o registro dos erros ocorridos durante a resolução dos exercícios estabelecidos para o ensino da programação de computadores, que auxilie na mediação do erro através de feedbacks

Endereço: Rua General Osório, 348
Bairro: CENTRO **CEP:** 95.700-086
UF: RS **Município:** BENTO GONCALVES
Telefone: (54)3449-3340 **E-mail:** cepesquisa@ifrs.edu.br

automáticos definidos pelo professor.

- Projeto de pesquisa do Mestrado Profissional em Informática na Educação do IFRS – Campus Porto Alegre.

Objetivo da Pesquisa:

- Os objetivos estão descritos.

Avaliação dos Riscos e Benefícios:

- Os riscos e os benefícios da pesquisa estão descritos, bem como as medidas de proteção de risco.

Comentários e Considerações sobre a Pesquisa:

- Os questionários e o roteiro estão adequados aos objetivos propostos da pesquisa.

Considerações sobre os Termos de apresentação obrigatória:

- TCLE ok.

Recomendações:

- Os questionários e o roteiro estão adequados aos objetivos propostos da pesquisa.

Conclusões ou Pendências e Lista de Inadequações:

Não foram observados óbices éticos.

O projeto está aprovado e, após a finalização da última etapa, conforme cronograma cadastrado na Plataforma Brasil, o pesquisador possui o prazo de 60 dias para envio do relatório final via Plataforma.

Considerações Finais a critério do CEP:

Este parecer foi elaborado baseado nos documentos abaixo relacionados:

Tipo Documento	Arquivo	Postagem	Autor	Situação
Informações Básicas do Projeto	PB_INFORMAÇÕES_BÁSICAS_DO_PROJETO_1250978.pdf	08/11/2018 21:03:12		Aceito
Projeto Detalhado / Brochura Investigador	projeto_detalhado.pdf	08/11/2018 21:02:25	KAREN CRISTINA BRAGA	Aceito
Folha de Rosto	folha_rosto.pdf	08/11/2018 20:57:22	KAREN CRISTINA BRAGA	Aceito
Declaração de Instituição e	autorizacao_institucional.pdf	08/11/2018 20:57:02	KAREN CRISTINA BRAGA	Aceito

Infraestrutura	autorizacao_institucional.pdf	08/11/2018 20:57:02	KAREN CRISTINA BRAGA	Aceito
TCLE / Termos de Assentimento / Justificativa de Ausência	TCLE.pdf	06/11/2018 23:01:28	KAREN CRISTINA BRAGA	Aceito
Outros	questionario_2.pdf	04/11/2018 22:13:43	KAREN CRISTINA BRAGA	Aceito
Outros	questionario_1.pdf	04/11/2018 22:13:28	KAREN CRISTINA BRAGA	Aceito
Outros	Roteiro_entrevista.pdf	04/11/2018 22:13:04	KAREN CRISTINA BRAGA	Aceito

Situação do Parecer:
Aprovado

Necessita Apreciação da CONEP:
Não

BENTO GONCALVES, 07 de Dezembro de 2018

Assinado por:
MARCELO MALLET SIQUEIRA CAMPOS
(Coordenador(a))

ANEXO D

PLEA: PLanejamento, Execução e Avaliação

O PLEA, disponível no AVA Moodle, foi criado pelo mestrando Márcio Carvalho aluno do curso de Mestrado Profissional em Informática na Educação (MPIE) - IFRS e aplicado no Desafio Codein'play como uma das tarefas a serem entregues pelas equipes. O PLEA é composto de 9 questões, a seguir:

Q#1 - O que o exercício está pedindo que seu programa faça?

Q#2 - Quais são as informações de entrada para o seu programa?
O que deverá ser digitado pelo usuário?
Quais são os tipos de dados necessários para resolver a questão?
Quais são as informações e qual a saída do seu programa? (tela, um arquivo, impressora, ...)

Q#3 - O programa precisa fazer algum tipo de cálculo? (contar, somar, subtrair, totalizar, ...)

Q#4 - Quais são os comandos necessários para que meu programa faça o que é pedido no enunciado? (printf, scanf, ges, puts, getch, for, if, else, swicth, while, ...)

Q#5 - Qual é o passo a passo para que seu programa faça o que o enunciado pede? Como ele começa? Como ele termina? E o que tem no meio disso tudo?

Q#6 - O ambiente está adequado para que você possa criar seu programa de forma agradável? Alguma perturbação?

Q#7 - Estou conseguindo seguir o planejamento para realizar a atividade?

Q#8 - Quais foram as dificuldades que você teve e o que você percebeu que precisa estudar?

Q#9 - Como foi seu planejamento para realização da atividade?
Como foi a execução para criação do seu programa?
O que é possível melhorar para próxima oportunidade?
Fique confortável para escrever algo para nós :P