Uma Proposta de Ferramenta para Simplificar a Depuração de Códigos em C, por Alunos Iniciantes

Marina S. Gomes[1], Érico M. H. do Amaral[1]

[1]Engenharia de Computação - Universidade Federal do Pampa (Unipampa)
Avenida Maria Anunciação Gomes de Godoy, 1650 – 96.413-172 – Bagé – RS – Brasil

{gomes.marina93, ericohoffamaral}@gmail.com

Abstract. *This paper aims to present the development of a support resource to the teaching and learning process of programming for beginner students, through the proposal of a system that turns the identifying and correction process of C language compilation errors simpler and more comprehensible, allowing that the student, at his first contact with the programming language, could be able to abstract the complexity involved on error correction during the programming process, stimulating, on this way, the knowledge building about the logic involved on the deployment of computational solutions for simple problems. This software tool is entitled CFacil analyzer.*

Resumo. *Este artigo tem por objetivo apresentar o desenvolvimento de um recurso de apoio ao processo de ensino e aprendizagem de programação, para alunos iniciantes, por meio da proposta de um sistema que torne a identificação e correção de erros de compilação em linguagem C mais simples e compreensível. Permitindo desta forma, que o estudante, em seus primeiros contatos com a linguagem de programação, consiga abstrair a complexidade envolvida na correção de erros durante o processo de programação, estimulando a construção de conhecimento sobre a lógica envolvida na implementação de soluções computacionais para problemas simples. Esta ferramenta de software intitulou-se Analisador CFacil.*

1. Introdução

As dificuldades no processo de ensino e aprendizagem da disciplina de Algoritmos e Programação, aliada a importância dos seus conteúdos para os cursos da área de Ciências Exatas e a diferenciada capacidade de aprendizagem de cada discente, constituem um cenário que dificulta ao docente atingir os diferentes estilos cognitivos presentes em sala de aula. Estes fatores justificam a proposta e o desenvolvimento de novos ambientes e metodologias para o ensino nesta área. Constatando estes pontos expostos, autores como: Likke *et. al* (2014), Yang *et. al* (2014) apontaram diferentes estratégias pedagógicas para esta disciplina.

Em estudos sobre o baixo desempenho dos alunos em conteúdos introdutórios a programação Jenkins (2002) e Pimentel&Nizam (2008) descreveram como causas para este problema: a dificuldade do estudante de pensar logicamente, ou seja, utilizar processos cognitivos de forma eficiente, visando construir de forma lógica uma solução; após, construir o algoritmo, a dificuldade enfrentada é a utilização de ambientes de programação e a transcrição para uma linguagem de programação, cuja sintaxe é vista como desafiadora para estudantes iniciantes e, conseguindo construir o seu pensamento e

DOI: 10.5753/cbie.wcbie.2016.1029

transcrevendo para uma linguagem computacional, o impedimento de progresso nesta fase é a realização de testes e depuração, ou seja, verificar se o algoritmo desenvolvido realiza o que é de seu propósito.

Mantendo o foco na última dificuldade relatada, realização de testes e depuração, More *et. al* (2011) diz que, a correção de erros é uma das tarefas mais importantes na programação, sendo uma das causas para o baixo desempenho nesta disciplina. Então, recorrendo a literatura, na teoria Contrucionista de Papert (1986), a qual estuda a interação aluno-computador, tendo como objetivo principal extrair o máximo de rendimento do estudante auxiliado por meio do computador, destaca-se, entre as ações propostas por esta teoria, que as estratégias pensadas possam ser visualizadas, manipuladas e corrigidas, a fim de serem aperfeiçoadas. Alinhando o Construcioninsmo ao assunto que se deseja estudar, a depuração de códigos, percebe-se que esta é uma tarefa muito importante no aprendizado do aluno, pois lhe permite a visualização do seu pensar, ou seja, da construção do seu raciocínio lógico e verificar as inconsistências presentes neste e corrigi-las.

Reconhecido este contexto, o trabalho aqui descrito tem por objetivo apresentar o desenvolvimento de uma aplicação que ofereça ao discente uma forma simples e compreensível de verificar as mensagens de erros, durante a compilação de seus programas, por meio da tradução destas mensagens para o idioma nativo do aprendiz. Vislumbra-se, a partir desta solução, estimular a aprendizagem e evolução na disciplina de algoritmos e programação. A ferramenta desenvolvida trata-se de um analisador, o qual recebeu a denominação de Analisador CFacil. A linguagem utilizada para o ensino de algoritmos destacada neste estudo é a Linguagem C.

Este artigo está estruturado da seguinte maneira: na seção 2 é apresentada uma revisão na literatura sobre o tema, como a aprendizagem de programação e a depuração de códigos; a seção 3, apresenta a metodologia utilizada para realização da pesquisa e, na seção seguinte a implementação; na seção 5 são apresentados os resultados obtidos e discussões da pesquisa, e na seção 6 as conclusões alcançadas.

2. Referencial Teórico

Apresenta as bases teóricas utilizadas de acordo com o eixo que rege a pesquisa. A seção 2.1 aponta dificuldades identificadas por autores da área na aprendizagem de algoritmos e programação. Na seção 2.2 é apresentada a depuração de códigos como forma de construção de conhecimento pelo aluno.

2.1 O processo de Ensino/Aprendizagem de Programação

A disciplina de Algoritmos e Programação exige do aluno o desenvolvimento de competências para idealizar programas que resolvam problemas reais, competências estas que necessitam de uma base lógico-matemática desenvolvida, considerada por Friedrich *et al.* (2012) uma técnica de usar corretamente as leis do pensamento. Do professor se espera a interação, acompanhamento e avaliação do aluno, porém muitas das vezes isto se torna inviável devido aos diferentes estilos cognitivos e ritmos de aprendizagem presentes em sala de aula, tornando o ensino deficiente em algum aspecto. E, como consequência destas situações muitas dificuldades de aprendizagem não conseguem ser identificadas, levando a um alto índice de reprovações e desistências.

Para Menezes e Nobre (2002) são três problemas gerais que podem causar dificuldades no processo de ensino e aprendizagem da disciplina de Algoritmos e Programação: acompanhamento individualizado inviável devido ao grande número de

alunos por turma, não promoção de uma evolução gradual da aprendizagem, devido as formas de avaliação; diferenças na forma e ritmo de aprendizagem de cada aluno. Além destes, Kamiya & Brandão (2009) também apontam como obstáculos a configuração dos ambientes de programação e a sintaxe de seus comandos. Os esforços para identificação dos problemas inerentes ao ensino e aprendizagem desta disciplina tornam-se imprescindíveis e indispensáveis.

As abordagens mais comuns, utilizadas para enfrentar problemas relacionados ao ensino e aprendizagem de algoritmos e programação, estão calcadas no desenvolvimento de ambientes ou ferramentas computacionais que auxiliem o aluno na construção da lógica algorítmica e, ao professor na organização e ampliação da capacidade de atendimento aos problemas de aprendizagem.

2.2 Construindo conhecimento a partir da depuração de códigos

A atividade de programar requer do programador o domínio de uma linguagem de programação que possibilita a codificação de um programa, entendimento do problema apresentado e a capacidade de criar uma solução para este problema a partir da organização de seus pensamentos, descrevendo através de um algoritmo este raciocínio lógico e, transcrevendo para um programa de computador através de uma linguagem de programação. Desta maneira, a programação inibe a memorização de informação, mas sim incentiva as capacidades de formalizar o raciocínio lógico, refletir e planejar (MALTEMPI & VALENTE, 2000).

Papert (1986) diz que ao aprender a programar erros são cometidos e não se acerta na primeira tentativa, do seu ponto de vista cometer erros não é um absurdo, pelo contrário: cometer erros é uma oportunidade para construir conhecimento, aspectos defendidos também por Valente (1999) e Almeida (1991). Então, quando o aluno recebe o *feedback* do computador sobre a execução o seu programa e, o mesmo contém erros, este fato deve ser considerado uma oportunidade, visto que o estudante deverá identificar o que ocasionou o erro, utilizando para isso conhecimentos já adquiridos e, desta forma, construir novas estruturas mentais, até aperfeiçoar o seu programa.

Por isso, a atividade de depuração é tão importante durante o processo de aprendizagem, pois leva o aluno a pensar sobre o *feedback* que recebera e a buscar soluções para isto, demandando do aluno concentração, dedicação e o motivando a melhorar a sua solução.

3. Metodologia

Pensando na forma de como tornar a depuração de códigos, utilizada como estratégia de ensino na disciplina de algoritmos e programação, mais atraente ao aluno, onde ele compreenda as mensagens de erro geradas mais facilmente e que estas sejam simples, este trabalho tem como objetivo a construção de uma ferramenta de software para apoio ao processo de ensino e aprendizagem de programação.

A pesquisa realizada neste trabalho é classificada como aplicada quanto a sua natureza e exploratória quanto aos objetivos. O planejamento/desenvolvimento da pesquisa seguiu um conjunto de etapas previamente definidas, sendo a primeira etapa voltada para o levantamento bibliográfico sobre o tema de estudo, a etapa seguinte contemplou o estudo e implementação de um instrumento para coleta de dados (Projeto Piloto), o qual teve por objetivo elencar quais os principais erros cometidos pelos alunos durante a prática de programação em linguagem C. Na etapa 03 estes dados foram

analisados através de um minerador de textos (Sobek[1]), fornecendo resultados que serviram de base para o desenvolvimento do Analisador CFacil (etapa 04). Os testes realizados com o CFacil (etapa 05) ocorreram com a participação dos alunos de uma turma de algoritmos e programação, durante 5 aulas de 4 períodos cada, por meio da qual foram coletados os dados necessários a avaliação das contribuições deste estudo.

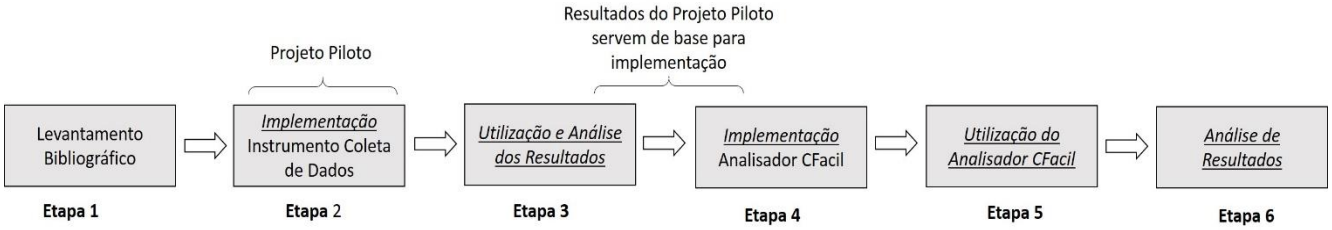


Figura 01. Metodologia

Para a análise dos resultados (etapa 06), adotou-se novamente o Sobek. E, com o intuito de buscar mais informações sobre a efetividade da ferramenta foi disponibilizado, aos participantes dos experimentos, um instrumento de pesquisa (questionário) sobre as atividades de programação e compilação com o CFacil. A Figura 01 apresenta de forma esquematizada a metodologia utilizada, demonstrado todas as etapas descritas.

4. Implementação

Nesta seção serão apresentados detalhes da implementação da ferramenta proposta. A seção 4.1, Projeto Piloto, trata do desenvolvimento de um instrumento de coleta de dados, a fim de elencar dados para nortear o desenvolvimento da ferramenta de *software* CFacil. E, a seção 4.2 apresenta o desenvolvimento do Analisador CFacil e estratégias utilizadas no seu desenvolvimento.

4.1. Projeto Piloto

Afim de buscar mais informações sobre os erros cometidos por alunos iniciantes em programação na linguagem C, foi realizado o desenvolvimento de um instrumento de monitoramento e coleta das mensagens de erros geradas durante o processo de compilação utilizando o compilador GCC.

Foi utilizada para implementação a tecnologia de *Shell Script*, onde há a solicitação ao aluno de um nome para o arquivo de saída, o qual será gerado, e o nome do arquivo que se deseja compilar. Após, o arquivo enviado é compilado duas vezes, uma para ser apresentado no terminal e outra para ser armazenado no arquivo de *logs*, que contém os erros do código construído pelo estudante.

As coletas das informações, com a aplicação, aconteceram em uma turma da disciplina de algoritmos e programação, com um universo amostral de 22 alunos por aula. As atividades de programação realizadas seguiram o plano de ensino da disciplina e a utilização da ferramenta ocorreu com supervisão docente.

[1] Minerador de textos Sobek desenvolvido pela UFRGS é capaz de identificar conceitos ou termos mais relevantes em um texto a partir da análise de frequências de termos no material textual analisado (KLEMANN, 2011)

A análise realizada nos dados armazenados permitiu identificar a grande dificuldade que os alunos iniciantes na programação de computadores têm em reconhecer seus erros de programação. E, a partir dos resultados alcançados com este estudo, pôde-se nortear a implementação do Analisador CFacil, ou seja, quais seriam os comandos e estruturas da linguagem C que deveriam estar presentes na ferramenta.

4.2. Desenvolvimento do Analisador CFacil

O Analisador CFacil se propõe a analisar um arquivo de código fonte da linguagem C e determinar se certa sentença encontrada neste arquivo está de acordo com a linguagem utilizada, ou seja, está sintaticamente correta. Para que o CFacil atenda sua finalidade, foram utilizados conceitos e técnicas de compiladores como: as fases de análise léxica, sintática e semântica, as quais foram adotadas para a verificação do arquivo fonte.

A análise léxica realiza a leitura de um código fonte e o transforma em uma sequência de símbolos léxicos, que são chamados *tokens.* São *tokens* da linguagem C os comandos *for*, *if*; os identificadores de tipo, entre outros. A análise sintática forma sentenças de uma linguagem a partir dos *tokens* recebidos, provenientes da análise léxica. E, a análise semântica verifica tipos e a unicidade da declaração de variáveis (AHO *et al.* 2008).

Por se tratar de uma aplicação utilizada por alunos iniciantes na linguagem de programação C, o Analisador CFacil reconhece um subconjunto de comandos da linguagem, porém não deixa de contemplar recursos que são fundamentais para demonstrar aspectos importantes da linguagem, como: variáveis locais e globais dos tipos *int*, *char* e *float*; constantes; estruturas de seleção; estruturas de repetição; funções da biblioteca padrão da linguagem C (*printf*, *scanf*, entre outras); funções criadas pelos usuários; operadores. Além disso, tomou-se o cuidado de trazer a implementação da ferramenta os recursos que foram mais recorrentes em erros, segundo a análise realizada com o projeto piloto desenvolvido.

Algumas restrições foram impostas afim de facilitar o desenvolvimento da aplicação, que são: as estruturas de seleção e repetição implementadas, deveriam obrigatoriamente iniciar e terminar seus blocos de códigos entre chaves, caso contrário seriam acusados erros durante a análise do código fonte.

Assim como é padrão a todos os programas na linguagem C, no Analisador CFacil também deve-se começar pela chamada a função *main()* e terminar com um *token* '}'. Todas as funções que venham a ser declaradas no programa fonte devem ser chamadas a partir da função *main()*. Portanto, para executar um programa C, deve-se começar no início da função *main()* e parar quando ela terminar.

Como, comandos na linguagem C podem não ser palavras-chave, eles são então expressões e, por isso, se desenvolveu um analisador de expressões. O analisador de expressões atua juntamente com o analisador léxico e é um dos subsistemas mais importantes e necessários para o CFacil. O analisador de expressões desenvolvido para o CFacil é um analisador recursivo descendente, ou seja, são funções recursivas que processam uma expressão.

A análise léxica é realizada por meio da leitura sequencial do código fonte. Mas, antes de realizar a análise léxica do código fonte, o analisador realiza uma varredura para encontrar e guardar todas as posições de funções presentes no programa. Após, o analisador começa a interpretar cada *token* retornado pela análise léxica, ou seja, realiza a análise sintática.

Para facilitar esta tarefa, utilizou-se a construção de autômatos para facilitar a validação de estruturas da linguagem, pois segundo Aho *et al.* (2008), a verificação de linguagens (estruturas gramaticais) é usualmente realizada através de autômatos. A adoção desta estrutura tornou mais fácil à implementação do analisador, pois era possível saber de antemão quais os *tokens* que tanto o analisador de expressões, quanto o analisador sintático e semântico deveriam receber em determinado instante.

O tratamento dos erros de sintaxe no Analisador CFacil é tratado por uma função especifica, a qual exibe a mensagem de erro e a linha na qual o erro foi encontrado, além de exibir em qual função este erro ocorreu, se na principal ou em uma função declarada pelo usuário. Como os erros, neste caso de um analisador recursivo, podem ser encontrados em rotinas recursivas, a maneira mais simples encontrada para lidar com esta situação foi de desviar a análise para um lugar seguro, ou seja, fazer o analisador "pular" para a próxima linha de código.

A execução do Analisador CFacil é executada por meio de um *script* e tem o seguinte comportamento: no momento da compilação é solicitado ao usuário um nome para o arquivo de saída (executável) e um nome para o arquivo que se deseja compilar (código fonte), semelhante a sintaxe utilizada para a compilação utilizando o GCC. Primeiramente é feita a execução do arquivo que faz a análise do código fonte, em seguida é avaliado se não foi encontrado nenhum erro nesta análise; se forem encontrados erros é gerado um arquivo com o relatório de erros com o Analisador CFacil, denominado "relatorio_erros_cfacil" para posterior análise pelo professor, e apresentado na tela os erros ao usuário.

Se o Analisador CFacil não encontrar erros é feita a compilação com o GCC. Esta compilação com o GCC é necessária, pois a ferramenta CFacil não consegue prever todos os erros que o usuário possa cometer. Desta forma, se forem encontrados erros é gerado um arquivo de erros com o compilador GCC denominado 'relatorio_erros_gcc'. Se não forem encontrados erros o *script* cria o arquivo executável e o usuário pode "rodar" o programa da forma convencional.

5. Resultados e Discussões

Nesta seção serão apresentados os resultados e discussões sobre o trabalho. Na seção 5.1 serão mostrados exemplos de código que a ferramenta pode reconhecer e algumas discussões sobre este assunto. E, na seção 5.2 serão mostrados os resultados de um questionário de opinião submetido aos alunos, requisitando o parecer sobre a tarefa de compilação e depuração de códigos em linguagem C com o compilador GCC e o Analisador CFacil.

5.1. Exemplos de programas-fonte reconhecidos

Um primeiro exemplo, conforme a Figura 04, representa a saída do Analisador CFacil ao reconhecer que não há uma biblioteca padrão declarada, necessária para o funcionamento das funções *printf()* e *scanf()*. O erro apresenta a falta da declaração da biblioteca padrão da linguagem C e a linha em que ocorreu, que pode ser uma linha acima ou abaixo de onde de fato o erro aconteceu.

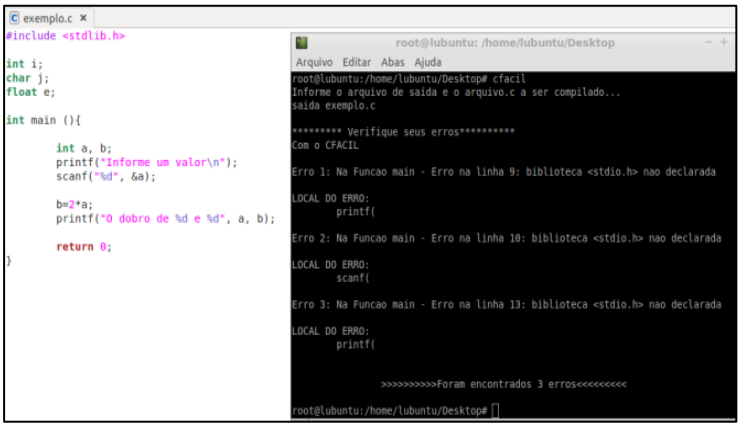


Figura 04. Funcionamento do Analisador CFacil – *printf()* e *scanf()*

O mesmo programa anterior, na Figura 05, apresenta a saída do Analisador quando há a falta do ponto-e-vírgula ao final de comandos da linguagem, retorno com tipo diferente da função e, novamente a não declaração de bibliotecas padrão.

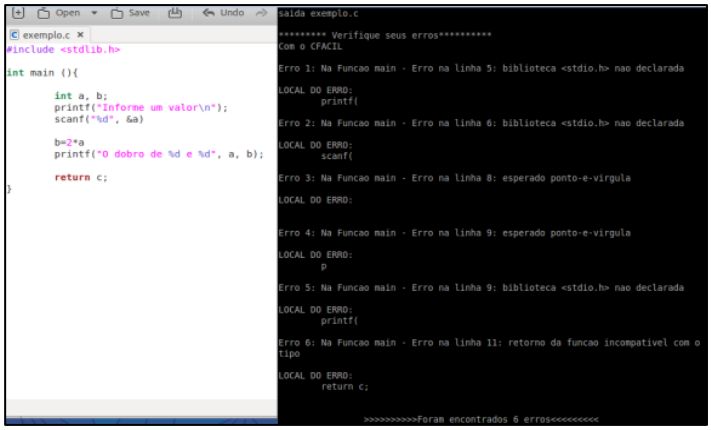


Figura 05. Funcionamento do Analisador CFacil – falta de ponto-e-vírgula

O último exemplo, na Figura 06, apresenta a utilização de uma estrutura de seleção da linguagem C. Mostra a saída do Analisador para quando há desbalanceamento das chaves de abertura e fechamento desta estrutura, além da utilização de variáveis com tipo incompatível ao seu. Assim como no compilador GCC, o Analisador continua a verificação do programa e pode encontrar erros que não existam, em virtude de algum erro que tenha sido cometido anteriormente, conforme o Erro 2 mostrado na Figura 06.

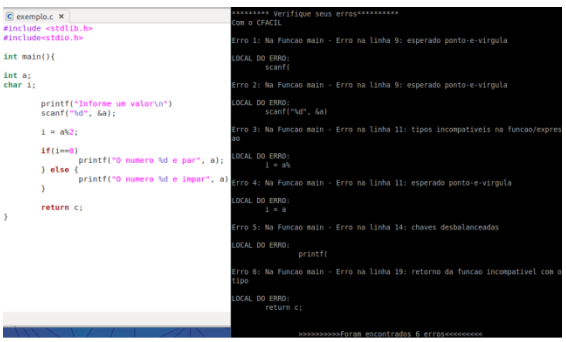


Figura 06. Funcionamento do analisador CFacil – falta de ponto-e-vírgula e chaves

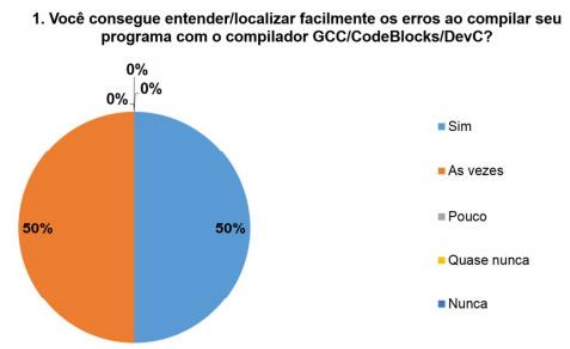
Figura 07. Grafo de análise do arquivo 'relatorio_erros_cfacil'

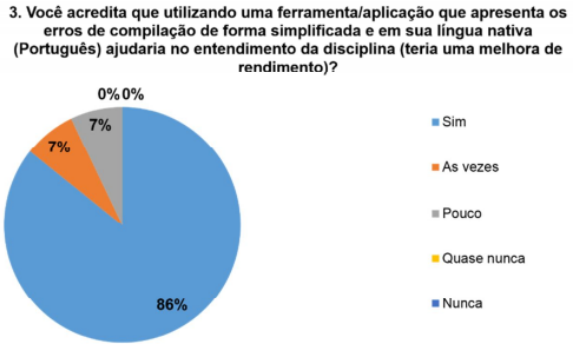
A fim de identificar a incidência dos erros tratados pelo CFácil durante os experimentos, após a captura e registro dos dados, foram realizadas análises textuais dos arquivos de *log* utilizando o software Sobek, com o intuito de elencar quais os termos mais recorrentes verificados e, desta forma reconhecer a amplitude do Analisador proposto. A Figura 07 apresenta um grafo gerado pelo Sobek sobre a utilização do Analisador CFacil (fonte: 'relatorio_erros_cfacil'), através do qual é possível perceber que os erros se concentraram na espera de *tokens* como abre e fecha chaves, abre e fecha parênteses e ponto e vírgula no final de comandos, além da não declaração de variáveis ou nomes de variáveis, sendo todos estes *tokens* reconhecidos pelo CFacil, demonstrando assim a sua efetividade.

5.2 Questionário de avaliação

Para complementar as análises realizadas nas práticas de programação com o Analisador CFacil e com o compilador GCC foi disponibilizado aos alunos um instrumento de pesquisa na forma de um questionário, com 5 questões fechadas, o qual teve como objetivo reconhecer a impressão dos alunos sobre a utilização da aplicação desenvolvida.

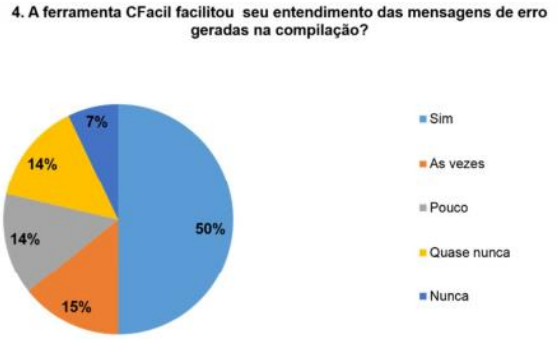
Dentre os resultados obtidos, são demonstrados o entendimento do estudante sobre o processo de compilação com o compilador GCC e as mensagens geradas por ele (Gráfico 01), assim como a utilização de uma aplicação como o CFacil e as mensagens geradas por ela (Gráfico 02).





A observação dos resultados apresentou-se de forma positiva ao se comparar as respostas dos Gráficos 01 e 02, visto que 86% dos alunos (Gráfico 02) apontaram que o CFacil é uma ferramenta efetiva para o entendimento de erros, enquanto apenas 50% (Gráfico 01) conseguiram entender os erros apresentados diretamente pelos compiladores. Corroborando com esta conclusão, quando questionados sobre a disponibilização do CFacil para utilização, 100% dos alunos afirmaram que utilizariam a ferramenta em algum momento durante as aulas de programação.

Além disso, ao avaliar de forma ampla o desfecho dos testes com o Analisador é possível inferir que a aplicação desenvolvida é válida, apesar de poder ser melhorada, conforme o resultado demonstrado no Gráfico 03. Contudo o Gráfico 04 aponta que a ferramenta foi aceita pelo público alvo como uma solução que pode auxiliá-los na aprendizagem da disciplina de algoritmos.



Como um software de auxílio ao processo de ensino e aprendizagem de algoritmos e programação, o Analisador CFacil pode, não somente auxiliar aos alunos, mas também aos professores em suas aulas práticas, pois se comporta como uma forma de verificação de pontos falhos do entendimento de programação, auxilia o docente a identificar as principais dificuldades dos estudantes e, de dedicar a esclarecer dúvidas pontuais sobre o conteúdo abordado naquele momento. Por fim, para que o Analisador CFacil atenda às necessidades de seus usuários de forma plena, o desenvolvimento da ferramenta de *software* tenta estar sempre em processo de readequação as necessidades de seus usuários, seja para correção de *bugs* ou inserção de novos comandos e estruturas da linguagem C.

6. Considerações finais

Neste trabalho foi abordado um problema observado por diversos autores da área de ensino e aprendizagem de algoritmos e programação, o qual é caracterizado pela dificuldade dos alunos em compreender e corrigir erros encontrados na compilação de seus algoritmos implementados em disciplinas introdutórias de programação. Como resposta a este problema foi proposta e implementada uma ferramenta para servir como solução de apoio aos alunos, na correção de seus erros de programação. Na etapa inicial desta pesquisa foi desenvolvido um Projeto Piloto com o intuito de determinar qual o percentual de erros cometidos por alunos iniciantes e, desta maneira verificar, de forma experimental, a validade da proposta.

O desenvolvimento da aplicação foi realizado e testado com alunos de uma turma da disciplina de algoritmos e programação. Estes testes basearam-se em conteúdos que estavam sendo abordados naquele momento da disciplina, ou seja, declaração de variáveis, estruturas de repetição, estruturas de controle, vetores e outros. Com a análise dos resultados obtidos foi possível verificar que houve uma melhora no quesito de recompilações dos programas, pois os alunos conseguiram de fato enxergar seus erros e corrigi-los. Além disso, observou-se uma redução significativa em alguns tipos de erros cometidos, pois, com o entendimento de suas falhas e, de forma proativa nas compilações seguintes, os estudantes conseguiram corrigir seus códigos.

Em resumo, a ferramenta CFacil foi considerada válida, visto que as dificuldades apresentadas pelos alunos ao aprender a programar são notórias e a aplicação conseguiu amenizar esta situação. Segundo Papert (1986), dificilmente se conseguirá uma compilação sem erros na primeira tentativa, pois se tratam de alunos iniciantes em programação. Mas, o fato é que cometer erros não significa algo condenável, pelo contrário, o erro pode ser tido como uma oportunidade para a construção de conhecimento, podendo ser considerado como um revisor de ideias e não um objeto para frustração (ALMEIDA, 1991). Assim sendo, o CFacil pode ser reconhecido como um

revisor de ideias, apresentando os erros cometidos pelos alunos de forma mais simples e auxiliando na construção do conhecimento sobre a disciplina de algoritmos e programação.

Referências

Aho, V. A.; Lam, S. M.; Sethi, R.; Ullman, D. J. (2008) "Compiladores Princípios, Técnicas e Ferramentas." Pearson Addison-Wesley.

Almeida, M. E. B.; (1991) "A Informática Educativa na Usina Ciência da UFAL", In: Anais do II Seninfe, NIES/UFAL, Maceio-AL.

Friedrich, R. V., dos Santos, D. S., dos Santos Keller, R., Puntel, M. D., & Biasoli, D. (2012) "Proposta Metodológica para a Inserção ao Ensino de Lógica de Programação com Logo e Lego Mindstorms." In: Anais do Simpósio Brasileiro de Informática na Educação, v. 23, n. 1.

Jenkins, T. (2002) "On the difficulty of learning to program." In: Proceedings of 3rd Annual LTSN_ICS Conference Loughborough University.

Kamiya, R. R., Brandão, L. O.; (2009) "iVProg-um sistema para introdução à Programação através de um modelo Visual na Internet." In: Anais do XX Simpósio Brasileiro de Informática na Educação. Florianópolis, SC.

Klemann, M., Reategui, E., Rapkiewicz, C. (2011) "Análise de Ferramentas de Mineração de Textos para Apoio à Produção Textual." In: XXII Simpósio Brasileiro de Informática na Computação (SBIE).

Likke, M., Coto, M., Mora, S., Vandel, N., Jantzen, C. (2014) "Motivating programming students by Problem Based Learning and LEGO robots." In: Global Engineering Education Conference (EDUCON, IEEE).

Menezes, C. S., Nobre, I. A. M. (2002)"Um ambiente cooperativo para apoio a cursos de introdução a programação." In: Congresso da Sociedade Brasileira de Computação.

Maltempi, M.V., Valente, J.A.; (2000) "Melhorando e Diversificando a Aprendizagem via Programação de Computadores." In: International Conference on Engeneering and Computer Education – ICECE.

More, A., Kumar, J., Renumol, V. G.; (2011) "Web Based Programming Assistance Tool for Novices." In: IEEE International Conference on Technology for Education.

Papert, S. (1986) "Constructionism: a new opportunity for elementary science education." Cambridge, Epistemology and Learning Group, Massachusetts Institute of Technology.

Pimentel, E., Nizam, O. (2008) "Ensino de Algoritmos baseado na Aprendizagem Significativa utilizando o Ambiente de Avaliação NetEdu." In: WEI/SBC Anais do XXXVIII Congresso da Sociedade Brasileira de Computação.

Valente, J. A.; (1999) "Análise dos diferentes tipos de softwares usados na Educação" In: Valente, J.A. (org). O computador na sociedade do conhecimento. Ed. Campinas: UNICAMP/NIED.

Yang, T., Yang, S., Hwang, G.; (2014) "Development of an Interactive Test System for Students Improving Learning Outcomes." In: Computer Programming Course. Advanced Learning Technologies (ICALT).