Simplificando a depuração de códigos na Linguagem C - Uma solução para 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 presentes no processo de ensino e aprendizagem da disciplica de Algoritmos e Programação são alvos de inúmeros estudos por parte de pesquisadores, como: Souza *et al.* (2013), Yang *et al.* (2014) e Likke *et al.* (2014), que com intuito de amenizar estas dificuldades tem proposto e apresentado ferramentas e metodologias para essa área.

Para os autores Pimentel&Nizam (2008) e Jenkins (2002), estas dificuldades podem ser causadas pela dificuldade na resolução de problemas, ou seja, não conseguem interpretar e formalizar seu pensamento lógico para solucionar o problema; além disso, utilizar uma linguagem de programação e ambientes de programação, também agravam esta situação, já que muitas das vezes se mostram desafiadores para alunos iniciantes.

Para More *et al.* (2011) a correção dos erros é uma das tarefas mais importantes na programação, chamada de depuração de códigos, sendo uma das causas de alunos obterem baixo desempenho nesta disciplina. Os estudantes enfrentam dificuldades em representar suas abstrações em comandos para linguagem de programação, assim como

entender e interpretar as mensagens de erros exibidas pelo compilador e por consequência corrigir os erros de sintaxe.

Utilizando a teoria construcionista de Papert (1986), que estuda a utilização do computador pelo aluno afim de produzir o máximo de aprendizagem, e, focando em um dos seus princípios para que isto ocorra, o princípio de que o aluno possa manipurar as suas estratégias pensadas, ou seja, o aluno constrói o seu pensamento sobre algo ou problema e o implementa em um computador e ao final pode visualizar e pensar sobre o que havia planejado e se este está de acordo com o que desejava, podemos utilizar isto para a aprendizagem de programação, mais especificamente, na tarefa de depuração de códigos.

A tarefa de depuração de códigos é uma das mais importantes na aprendizagem da lógica e programação, pelo fato de possibilitar que o aluno identifique inconsistências de seus conceitos sobre a construção algorítmica de programas. Com base neste contexto, este trabalho tem como objetivo o desenvolvimeno de um recurso de apoio ao processo de ensino e aprendizagem, visando aprimorar a capacidade dos alunos em resolverem problemas de forma algorítmica, por meio de um ambiente de pré-compilação que ofereça ao aluno iniciante na linguagem de programação C um diálogo mais simples e compreensível, reportando mensagens de erros no idioma português, estas provenientes de análises realizadas no programa fonte. E, desta forma, estimular a construção de conhecimentos sobre lógica computacional em estudantes iniciantes. A ferramenta desenvolvida trata-se de um analisador, comumente chamado de Analisador CFacil.

Afim de apresentar a construção desta pesquisa, este artigo está estruturado da seguinte forma: na seção 2 é apresentada uma revisão na literatura sobre o tema, como a aprendizagem de programação e a depuração de códigos, além de trabalhos correlatos; 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 são apresentadas as conclusões alcançadas.

2. Referencial Teórico

Esta seção tem por objetivo apresentar as bases teóricas utilizadas para construção deste trabalho, sendo abordados assuntos selecionados de acordo com o eixo que rege a pesquisa. A seção 2.1 aponta algumas das 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 do conhecimento pelo aluno, e na seção 2.3 são apresentados alguns trabalhos correlatos.

2.1. Aprendizagem de Algortimos e Programação

Os motivos que tornam a aprendizagem de algoritmos e programação por alunos iniciantes são muitos, dentre eles destacam-se, segundo Menezes e Nobre (2002): dificuldade, por parte do professor, de dispor de um atendimento diferenciado de acordo com cada dificuldade pelo grande número de alunos por turma, diferentes estilos e tempos de aprendizagem, formas de avaliação utilizadas e, para Kamiya & Brandão (2009) ainda há a dificuldade na utilização de ambientes de programação e a sintaxe de uma linguagem de programação.

Se observa ainda, no meio acadêmico, uma grande dificuldade de os alunos compreenderem e utilizarem os conceitos de algoritmos e programação expostos em sala de aula, ou seja, através destes conceitos contruirem seu pensamento lógico, que para Friedrich *et al.* (2012) é a técnica de usar corretamente as leis do pensamento e conseguirem elaborar uma solução para um problema de forma algorítmica.

Todos estes problemas expostos levam a um baixo desempenho na disciplina de Algoritmos e Programação, e por esta causa, grupos de pesquisa têm se empenhado em desenvolver softwares que possam auxiliar alunos e professores neste processo.

2.2. Depuração de códigos como construtor do conhecimento

A construção de programas requer a compreensão de um problema, a utilização de uma linguagem de uma linguagem de programação e a lógica que resolva este problema, a utilização de conceitos já aprendidos, que de forma organizada, cheguem a uma solução. Para isto, não há a necessidade de memorização de como resolver tal problema, mas sim a utilização do raciocínio lógico que deve ser utilizado para modelar o problema e a utilização de uma linguagem de programação (MALTEMPI & VALENTE, 2000).

Segundo Papert (1986) a aprendizagem de programação se constrói com erros, pois dificilmente se acerta na primeira tentativa. Ainda mais, quando se trata de alunos iniciantes nesta tarefa, por terem que se preocupar com a utilização de uma linguagem de programação e sua sintaxe correta, e pensar logicamente em uma solução. Ou seja, segundo Papert, o erro é uma oportunidade de construir conhecimento. Para Almeida (1991) e Valente (1999) o erro funciona como um revisor de ideias, pois permite que o aluno possa aprender sobre determinado conceito aplicado em sua solução e pensar sobre as estratégias que utilizou para resolver seu problema.

Então, na atividade de depuração, quando o aluno recebe o *feedback* sobre a execução do seu programa, ele é capaz de identificar quais são as inconsistências presentes em seu programa e tentar corrigi-las. E, desta forma, ele passa a utilizar seus conhecimentos prévios para melhoramento de sua solução proposta, e é por isso que a tarefa de depuração na aprendizagem de programação deve ser estimulada.

2.3. Trabalhos Correlatos

Vários grupos têm trabalhado na realização de experimentos para identificar as maiores dificuldades dos alunos na aprendizagem de programação. Cabe destacar o trabalho de Cechinel *et al.* (2008), em que como parte de seu trabalho realizou uma pesquisa junto aos alunos com intuito de identificar as maiores dificuldades no aprendizado de programação. Como resultados observou um alto percentual de evasões na disciplina de Algoritmos e Programação e, que a maior parte dos alunos submetidos a pesquisa tiveram seu primeiro contato com a programação nesta disciplina. Algumas dificuldades reportadas pelos estudantes estão relacionadas a identificação dos erros nos seus programas (erros de compilação) e, do próprio processo de desenvolvimento utilizando a linguagem de programação.

3. Metodologia

A patir das observações feitas sobre a utilização de uma depuração de códigos mais atraente ao aluno, ou seja, de forma que ele possa compreender as mensagens de erros geradas pelo compilador, este trabalho vislumbra a construção de uma ferramenta para o apoio ao processo de ensino e aprendizagem de algoritmos, que sirva para professores e alunos, como forma de facilitar o ensino e a aprendizagem prática de programação.

Desta forma, a pesquisa realizada neste trabalho foi classificada, quanto a sua natureza, como aplicada e exploratória quanto aos objetivos. O método adotado para o desenvolvimento deste trabalho consistiu inicialmente no levantamento de um referencial bibliográfico. O passo seguinte para esta pesquisa foi o desenvolvimento de um instrumento de coleta de dados, que foi aplicado a uma turma de algoritmos e programação de uma universidade. Este instrumento foi utilizado durante 3 meses. Em

seguida os dados foram analisados para identificar os erros mais comuns cometidos pelos alunos na prática de programação em linguagem C, que serviu para nortear o desenvolvimento da ferramenta.

Simultaneamente, com a aplicação do instrumento de coleta de dados, especificou-se uma solução para o problema de pesquisa, utilizando a *Unified Modeling Language* (UML) como recurso para modelagem da aplicação proposta. Especificado o modelo foi iniciado o desenvolvimento da ferramenta, construída sob a plataforma de programação C. A etapa de testes do software ocorreu com a participação dos alunos de uma turma de algoritmos e programação, durante 5 aulas de 4 períodos cada, através da qual foram coletados dados necessários a avaliação das contribuições deste estudo. A análise dos resultados foi realizada com o software minerador de texto Sobek. E, para finalizar a validação da ferramenta foi disponibilizado aos estudantes um instrumento de pesquisa (questionário) sobre os experimentos realizados.

4. Implementação

Neste capítulo serão apresentados detalhes da implementação da ferramenta proposta.

4.1. Instrumento de coleta de dados

Com a finalidade de coletar informações sobre erros cometidos por alunos iniciantes em programação na linguagem C, foi realizado a análise e desenvolvimento de uma ferramenta de monitoramento das saídas geradas durante o processo de compilação utilizando o compilador GCC (Gomes *et al.* 2015).

Para a implementação desta aplicação utilizou-se a tecnologia de *Shell Script*, onde o *script* desenvolvido é apresentado na Figura 01.

```
#!/bin/bash

########################################
# Script que testa se arquivo existe e #
# joga a saida do gcc para um arquivo  #
########################################
echo "Informe o arquivo de saida e o arquivo.c a ser compilado..."
read saida arquivo

if [ -e "$arquivo" ]
then
        #echo "O arquivo '$arquivo' existe"
        echo
        data=$(date)
        echo -e "\033[31m Data: $data \033[m" >> relatorio_erros
        gcc -o "$saida" "$arquivo" #saida padrão
        gcc -o "$saida" "$arquivo" 2>> relatorio_erros #saida para arquivo

fi
```

Figura 01. Trecho de código CFacil

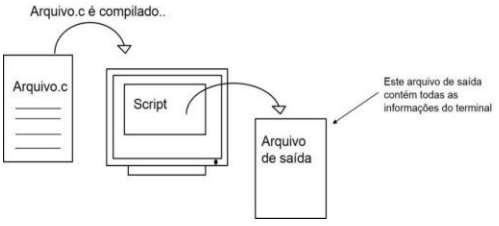

Figura 02. Arquitetura CFacil

A arquitetura de funcionamento do software é demonstrada na Figura 02, onde ocorre a solicitação ao usuário de um nome para o arquivo de saída e o nome do arquivo que se deseja compilar. Após isto, o arquivo enviado pelo usuário é compilado duas vezes, uma para ser apresentado no terminal e outra para ser armazenada no arquivo de logs, que contém os erros no código construído pelo estudante.

A coleta de dados, com a aplicação, ocorreu em uma turma da disciplina de algoritmos e programação, com um universo amostral de 22 alunos por aula, todas realizadas em laboratório de programação. As atividades acadêmicas da disciplina ocorreram em encontros semanais, 4 horas/aulas, em um período total de 3 meses. Todas as atividades de programação seguiram o plano de ensino, sendo que a utilização da ferramenta ocorreu com supervisão docente, onde os estudantes realizavam as tarefas de programação propostas, com o armazenamento dos logs de compilação.

A partir da análise dos dados armazenados, pode-se perceber a grande dificuldade dos alunos em reconhecer as mensagens de erros geradas pelo compilador. Esta etapa serviu para estipular quais seriam os pontos/estruturas da linguagem C que deveriam estar presentes na ferramenta, explicitando desta forma os requisitos para a construção do Analisador, o qual recebeu a denominação de CFacil.

4.2 Desenvolvimento do Analisador CFacil

O CFacil tem como objetivo realizar a análise de um arquivo de logs e, a partir desta, inferir se uma determinada sentença da linguagem está correta. Como já especificado, a aplicação faz a análise de um arquivo de código fonte da linguagem C e determina se certa sentença desta linguagem está correta. Para isso, são utilizados conceitos e técnicas de compiladores, ou seja, são realizadas as fases de análise léxica, sintática e semântica no arquivo.

A análise léxica faz a leitura do programa fonte, caractere a caractere, e o traduz em uma sequência de símbolos léxicos, chamados de tokens. São exemplos de tokens da linguagem C: comando `for`, função `printf()`, identificador `int`, entre outros. A partir destes tokens a análise sintática constitui uma sentença da linguagem. E, a análise semântica realiza a verificação de tipos e a unicidade da declaração de variáveis. (AHO, 2008).

O analisador CFacil reconhece um subconjunto restrito de comandos da linguagem C, contemplanto aspetos importantes da linguagem, como recursos fundamentais e necessários para demonstrar aspectos da linguagem e recursos identificados como recorrentes em erros, segundo a análise prévia realizada na primeira estapa desta pesquisa. Com base nestas características foram implementados as seguintes funcionalidades no CFacil:

- Variáveis locais e globais
- Variáveis inteiras, caracteres, *float*
- Constantes inteiras, caracteres, *float*
- Os comandos *if-else*, *if*, *switch*
- Os laços *do-while*, *while* e *for*
- Número limitado de funções da biblioteca padrão
- Retorno de funções (*return*)
- Operadores: +, -, *, /, %, <, >, <=, >=, ==, !=, - unário e + unário
- Comentários
- Funções parametrizadas com variáveis locais

Com o intuito de facilitar a agregação destes recursos ao Analisador foi necessário impor a restrição de que os alvos de *if-else*, *if*, *while*, *for*, *switch* e *do-while* tivessem seus blocos de códigos encerrados entre chaves, ou seja, esses comandos não poderiam ser encontrados isolados no código, sob pena de serem acusados erros durante a análise feita pelo Analisador CFacil.

Por padrão todos os programas em C devem começar com uma chamada a função *main()* e terminar com o último *token* ‘}’ ou um comando *return* dentro de *main()*. Quaisquer outras funções que estejam contidas no programa precisam ser chamadas direta ou indiretamente a partir da função principal *main()*. Portanto, para executar um programa C, deve-se começar no início da função *main()* e parar quando *main()* terminar.

Qualquer comando na linguagem C, que não seja um comando de palavra-chave, é por definição uma expressão. Desta forma, fez-se necessário desenvolver um analisador de expressões. O analisador de expressões que está juntamente com o analisador léxico é um dos subsistemas mais importantes e necessários para o CFacil. Muitos compiladores comerciais usam um analisador baseado em tabelas, que é comumente gerado por um programa gerador de analisadores. Embora os analisadores baseados em tabelas sejam mais rápidos que os criados com outros métodos, eles são complexos de criar manualmente. Para a construção do Analisador CFacil utilizou-se a

técnica de análise recursiva descendente, que implementa as regras de produções feitas. Um analisador recursivo descendente é essencialmente uma coleção de funções recursivas que processam uma expressão.

A função *get_token()*, contida no CFacil, realiza a análise léxica do código, por meio de uma leitura sequencial, retornando sempre o próximo símbolo lógico do código-fonte e, é fundamental para todos os analisadores e compiladores. Porem, antes de o anlisador começar de fato a executar/analisar um programa, ele executa algumas ações administrativas, como: encontrar e guardar todas as posições de funções que estejam presentes no código, além de encontrar variáveis locais e globais. Após isto, ele efetivamente analisa o código e utiliza para isso uma função de interpretação.

Esta função é que decide qual ação tomar em função do token de entrada. Para reconhecer cada uma das estruturas de palavras-chave, ou seja, comandos como *if*, *for*, *while* entre outros, fez-se uso da construção de autômatos para facilitar a validação destas estruturas. Segundo Aho *et al.* (2008), a verificação de linguagens (estruturas gramaticais) é usualmente realizada através de autômatos que as reconhecem. Com base neste estudo teórico sobre conceitos de compiladores, foram descritos os autômatos finitos para as estruturas gramaticais e, após, foram implementadas as estruturas no Analisador CFacil. 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.

Quando o CFacil encontra um erro de sintaxe, ele chama a função *sntx_err()* com um valor de enumeração que corresponde ao tipo do erro encontrado. Além de apresentar a mensagem do erro a função *sntx_err()* também exibe o número da linha na qual foi encontrado o erro, que pode ser a linha seguinte à que de fato contém o erro e, também exibe em qual função este erro ocorreu. Como erros de sintaxe podem ser encontrados em rotinas profundamente aninhadas ou recursivas, a maneira mais simples de lidar com um erro é desviar para um lugar seguro, ou seja, ao retornar da função *sntx_err()* o programa “pula” para a próxima linha, que conterá um novo comando.

A aplicação 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, após é avaliado se não foi encontrado nenhum erro nesta análise; se forem encontrados erros é gerado um arquivo com o relatório de erros com a ferramenta CFacil, denominado “relatorio_erros_cfacil” para posterior analise pelo professor, e apresentado na tela os erros ao usuário.

Se não forem encontrados erros com a ferramenta CFacil é feita a compilação com o compilador 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 nesta compilação 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 pela ferramenta

O primeiro exemplo, na Figura 03, mostra a saída que o analisador apresenta ao reconhecer que não há uma biblioteca declarada para uma função interna da linguagem C, neste caso para as funções *printf()* e *scanf()*. O programa apresenta o erro por falta de biblioteca e também a linha que ocorreu o erro, que pode ser uma linha acima ou abaixo de onde de fato o erro aconteceu. O analisador CFacil também reconhece funções da biblioteca *string.h*.

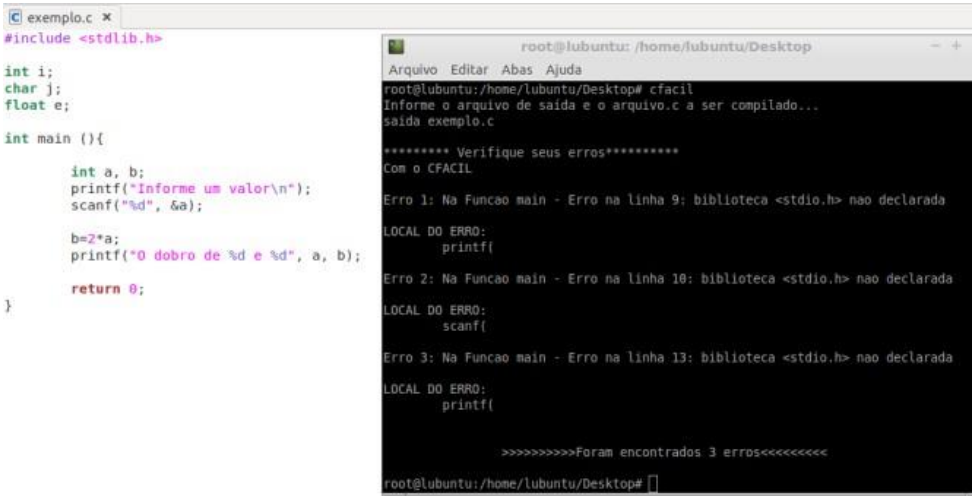

Figura 03. Funcionamento do analisador CFacil – printf() e scanf()

O segundo exemplo, na Figura 04, ainda com o mesmo programa anterior, apresenta detalhes para quando há falta do *token* ponto-e-vírgula ao final de comandos da linguagem, e também o retorno incompatível com o tipo da função, além dos erros pela falta da declaração da biblioteca de entrada e saída.

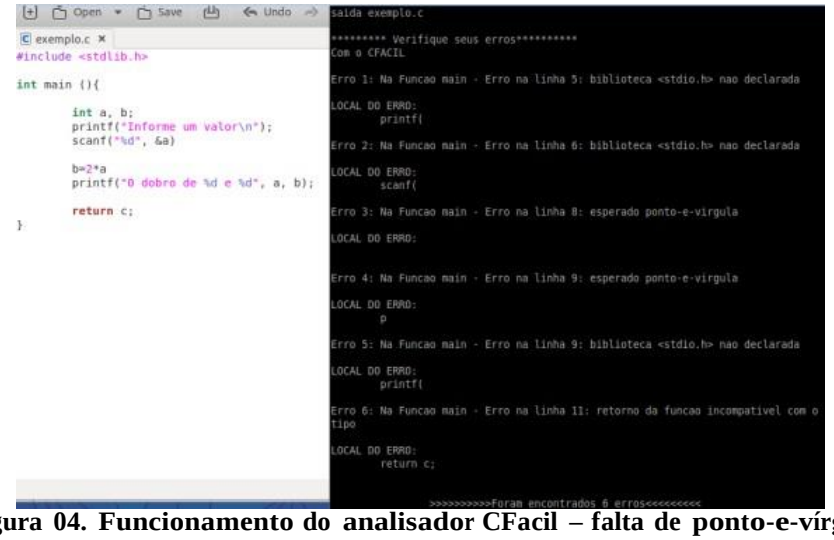

Figura 04. Funcionamento do analisador CFacil – falta de ponto-e-vírgula

E por fim, o último exemplo, verificar se um número é par ou ímpar, mostra o uso da estrutura *if-else* na Figura 05. Neste exemplo são reconhecidos erros de tipos de

variáveis incompatíveis em uma expressão, como mostrado na linha 11; a falta de ponto-e-vírgula no final de comandos e as chaves desbalanceadas na estrutura *if-else*. 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 05.

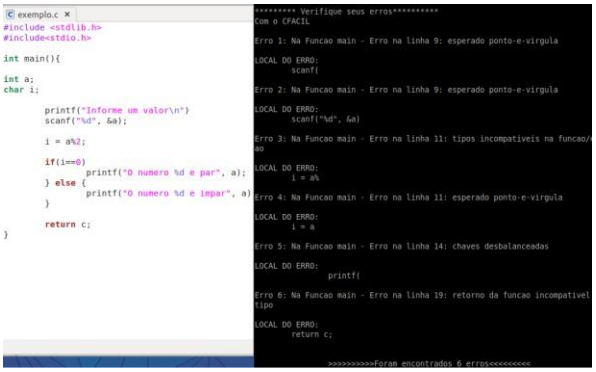

Figura 05. Funcionamento do analisador CFacil – falta de ponto-e-vírgula e chaves

Figura 06. 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 06 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 comados, 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 fazer uma complementação das 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, aqui 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).

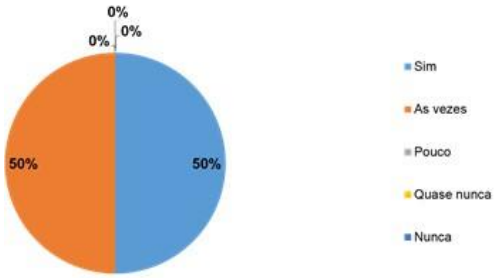

Grafico 01. Você consegue entender/localizar facilmente os erros ao compilar seu programa com o compilador GCC/CodeBlocks/DevC?

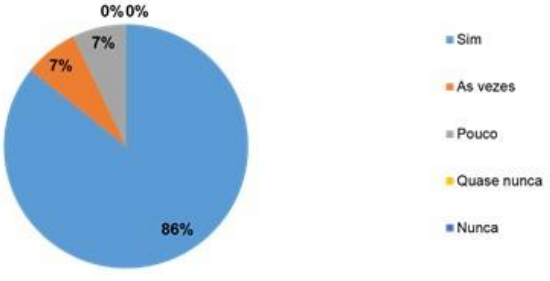

Grafico 02. Você acredita que utilizando o CFacil (que apresenta os erros de programação de forma simplificada e em Português) ajudaria no entendimento da disciplina de programação?

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) acreditam que o CFacil é uma ferramenta efetiva para o entendimento de erros, enquanto apenas 50% (Gráfico 01) conseguem 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 analisar de forma ampla o desfecho dos testes com o Analisador CFacil é possível inferir que a aplicação desenvolvida é válida, apesar de poder ser melhorada, e que foi aceita pelo público alvo como uma solução que pode auxiliá-los na aprendizagem da disciplina de algoritmos e programação. O CFacil pode também auxiliar o professor em suas aulas práticas, pois apresenta-se como um recurso ativo, através do qual o docente poderá identificar as principais dificuldades dos estudantes e, se dedicar a esclarece-las de forma pontual. Sendo uma ferramenta de *software*, o analisador CFacil deve sempre estar em constante readequação as necessidades de seus usuários, que neste caso é o "polimento" da linguagem utilizada nas mensagens de erros, apesar de já ser bem clara em sua colocação.

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 proposto e implementado uma ferramenta para servir como solução de apoio aos alunos, na correção de seus erros de programação.

Como etapa inicial desta pesquisa desenvolveu-se um Projeto Piloto com o intuito de determinar qual o percentual de erros cometidos por alunos iniciantes (Gomes *et al.* 2015) 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, 1999). 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. Compiladores Princípios, Técnicas e Ferramentas. São Paulo: Pearson Addison-Wesley, 2008.

ALMEIDA, M. E. B.; A Informática Educativa na Usina Ciência da UFAL, Maceió-AL, Anais do II Seninfe, NIES/UFAL, 1991.

CECHINEL, C., COGO, G., BETEMPS, C., TAVARES, R.; Desenvolvimento de Objetos de Aprendizagem para o Apoio à Disciplina de Algoritmos e Programação. Simpósio Brasileiro de Informática na Educação (SBIE), 2008.

DE SOUZA, M. B., MOREIRA, J. L. G., LOBO, F. L., & DOS SANTOS ALENCAR, M. A. Uma Abordagem Metodológica voltada para o Ensino-Aprendizagem de Algoritmos. RENOTE, v. 11, n. 1, 2013.

FRIEDRICH, R. V., dos SANTOS, D. S., dos SANTOS KELLER, R., PUNTEL, M. D., & BIASOLI, D.; 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, 2012.

GOMES, M., BECKER, L., GESTARO, L., AMARAL, E., & TAROUCO, L. M. R. Um estudo sobre erros em programação-Reconhecendo as dificuldades de programadores iniciantes. In: Anais dos Workshops do Congresso Brasileiro de Informática na Educação. 2015.

JENKINS, T. On the difficulty of learning to program. In Proceedings of 3rd Annual LTSN_ICS Conference Loughborough University, 2002.

KAMIYA, R. R., & BRANDÃO, L. O.; 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, 2009.

LIKKE, M., COTO, M., MORA, S., VANDEL, N., JANTZEN, C. Motivating programming students by Problem Based Learning and LEGO robots. In: Global Engineering Education Conference (EDUCON, IEEE), 2014.

MENEZES, C. S., NOBRE, I. A. M. Um ambiente cooperativo para apoio a cursos de introdução a programação. Congresso da Sociedade Brasileira de Computação, 2002.

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

MORE, A., KUMAR, J., RENUMOL, V. G.; Web Based Programming Assistance Tool for Novices. IEEE International Conference on Technology for Education, 2011.

PAPERT, S.; Constructionism: a new opportunity for elementary science education. Cambridge, Epistemology and Learning Group, Massachusetts Institute of Technology, 1986.

PIMENTEL, E., NIZAM, O.; Ensino de Algoritmos baseado na Aprendizagem Significativa utilizando o Ambiente de Avaliação NetEdu. WEI/SBC Anais do XXXVIII Congresso da Sociedade Brasileira de Computação, 2008.

VALENTE, J.A. Computadores e Conhecimento: repensando a Educação. Campinas: UNICAMP, 1993.

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

YANG, T., YANG, S., HWANG, G.; Development of an Interactive Test System for Students Improving Learning Outcomes in a Computer Programming Course. Advanced Learning Technologies (ICALT), 2014.