Um estudo sobre erros em Programação: reconhecendo as dificuldades de programadores iniciantes

Marina Gomes[1], Liliane Becker[1], Lucas Gestaro[1], Érico Amaral[1] , Liane Tarouco[2]

[1]Engenharia de Computação - Universidade Federal do Pampa (UNIPAMPA)
Bagé– RS – Brasil

[2]Programa de Pós-Graduação em Informática na Educação (PPGIE) – Universidade Federal do Rio Grande do Sul (UFRGS)
Porto Alegre – RS – Brasil

{gomes.marina93, lilianebb,lucasgestaro}@gmail.com,
ericoamaral@unipampa.edu.br, liane@penta.ufrgs.br

Abstract. *This paper introduces the first step of a project that aims to build a tool to assists the debugging of codes in C language by beginner students. The article describes a research conducted with beginners students in programming with the objective of evaluate and list the most common errors of these students during programming practices. From the information collected it was possible to perform a case-by-case analysis about the main errors committed. On the basis of observed results it is possible to point with objectivity topics that must be prioritized in the process of teaching and learning programming and in the tool in development.*

Resumo. *Este trabalho apresenta a primeira etapa de um projeto que visa construir uma ferramenta para auxiliar a depuração de códigos em linguagem C, por alunos iniciantes. O artigo descreve um estudo realizado com alunos iniciantes em programação, tendo por objetivo avaliar e elencar os erros mais comuns cometidos por estes estudantes durante as práticas de programação. A partir das informações coletadas foi possível realizar uma análise pontual sobre os principais erros cometidos. Com os resultados observados é possível apontar com objetividade temas que devem ser priorizados no processo de ensino- aprendizagem de programação e na ferramenta em desenvolvimento.*

1. Introdução

As dificuldades inerentes ao processo de ensino-aprendizagem de algoritmos e programação estão relacionadas a diversos fatores e com intuito de amenizar as dificuldades enfrentadas pelos acadêmicos, inúmeros grupos trabalham na realização de experimentos práticos e na construção de ferramentas e metodologias para facilitar o processo de ensino e aprendizagem dessa área, como: Yang *et al.* (2014) e Likke *et al.* (2014). Segundo Pimentel e Nizam (2008) e Jenkins (2002), algumas das causas para o baixo desempenho em disciplinas de algoritmos e programação são: falta de habilidades para resolução de problemas, linguagens de programação com sintaxes inadequadas para estudantes em início de curso, utilização de ambientes de programação, realização de testes e depuração de algoritmos.

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 instruções da linguagem de programação, assim como

DOI: 10.5753/cbie.wcbie.2015.1398

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

A teoria construcionista de Papert (1986), a qual possui como base o construtivismo cognitivo de Piaget, se propõe a explicar as relações aluno-computador para produzir o máximo de aprendizagem, construindo um conjunto de premissas a serem utilizadas quando aplicadas à ação de ensino que tem o computador como ferramenta. Na literatura construcionista pode-se destacar um princípio que corrobora com a aprendizagem de programação, a qual define que a possibilidade de visualizar, manipular e corrigir as estratégias pensadas permite aperfeiçoa-las. Desta maneira, a depuração de códigos torna-se uma tarefa importante na aprendizagem da lógica de programação, visto que possibilita ao aluno identificar inconsistências de seus conceitos sobre a construção algorítmica de programas.

Neste contexto, este estudo objetiva identificar os erros frequentemente cometidos por alunos iniciantes na prática de programação na linguagem C, a fim de organizar subsídios para propor uma solução que sirva de apoio ao processo de ensino-aprendizagem de algoritmos. Para alcançar os objetivos desta etapa da pesquisa foi proposta e implementada uma ferramenta para coleta de dados, sendo utilizada durante o primeiro semestre de 2015, por alunos dos cursos de engenharia e matemática, na disciplina de Algoritmos e Programação em uma Universidade Federal.

Este artigo encontra-se 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 alguns 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 considerações finais sobre o trabalho.

2. Referencial Teórico

Nesta Seção apresentam-se as bases teóricas utilizadas para realização deste trabalho. A Seção 2.1 aponta algumas das dificuldades identificadas por autores da área na aprendizagem de algoritmos e programação. Na Seção 2.2 é mostrada a depuração de códigos como forma de construção do conhecimento pelo aluno, e na Seção 2.3 são expostos alguns trabalhos correlatos.

2.1. Aprendizagem de Algoritmos e Programação

Muitos são os motivos que tornam a aprendizagem dos alunos na disciplina de Algoritmos e Programação difícil. Menezes e Nobre (2002) apontam três problemas gerais: inviabilidade da realização de um acompanhamento individualizado devido ao grande número de alunos por turma; apenas avaliações escritas ou trabalhos individuais, não promovendo uma evolução gradual da aprendizagem; heterogeneidade da turma, disparidade de conhecimento e ritmo de aprendizagem. Além destes, Kamiya e Brandão (2009) apontam como obstáculos a configuração dos ambientes de programação e a sintaxe de seus comandos.

O ensino de linguagens de programação tem como objetivo que os alunos desenvolvam um conjunto de competências necessárias para conceber programas capazes de resolver problemas reais. Contudo, observa-se que existe uma grande dificuldade em compreender e aplicar certos conceitos abstratos de programação por uma parte significativa dos alunos. Ainda, destaca-se a complexidade na construção do raciocínio lógico, o que é considerado por Friedrich *et al.* (2012) uma técnica de usar corretamente as leis do pensamento.

As causas de insucesso para a disciplina de Algoritmos e Programação são acarretadas pela dificuldade na organização de raciocínio, elaboração de estratégias de resolução de problemas, lógica, atenção e concentração. Deste modo, pode-se afirmar que as habilidades de raciocínio lógico, tais como tentar, observar, avaliar e deduzir não estão sendo devidamente desenvolvidas pelos alunos nas aulas de algoritmos, interferindo diretamente em sua capacidade de construir soluções algorítmicas para problemas [Barcelos *et al.* 2009].

Por ser uma tarefa difícil para muitos alunos, diversos grupos de pesquisa têm desenvolvido softwares educativos com intuito de auxiliar estudantes e professores neste processo. Estas ferramentas de auxilio são baseadas em teorias de aprendizagem e, uma destas teorias é o Construcionismo de Papert (1986).

2.2. Depuração de códigos como construtor de conhecimento

A atividade de programação exige o domínio de uma linguagem de programação por parte do programador, o que irá possibilitar a codificação do programa e seu processamento pelo computador. Ela também requer o conhecimento da situação-problema abordada e alguma criatividade, uma vez que uma solução pode ser expressa de diferentes maneiras. A construção de programas inibe, portanto, a reprodução e memorização de informações e requer a formalização de raciocínio lógico, reflexão, dedicação à atividade e pesquisa em relação ao problema a ser modelado e também quanto à linguagem de programação adotada [Maltempi e Valente 2000]. Para Da Silva *et al.* (2014) são muitos os desafios para o aprendizado de programação, devido à complexidade desta tarefa, além da necessidade de uma competência para solucionar problemas algoritmicamente.

Papert (1986) afirma que, quando se aprende a programar, dificilmente se acerta na primeira tentativa. Isso é verídico, pois por se tratarem de alunos iniciantes em programação, é possível e muito provável que os alunos cometam erros ao utilizar uma linguagem de programação, visto que além de pensarem na lógica de um problema é necessário também utilizar a sintaxe correta da linguagem adotada.

Do ponto de vista construcionista, cometer um erro não significa algo condenável, muito pelo contrário: o erro é tido como oportunidade ideal para a construção do conhecimento. Almeida (1999) destaca que o erro passa a ser um revisor de ideias e não mais um objeto de frustração. Valente (1999) também diz que o processo de achar e corrigir um erro constitui uma oportunidade única para o aprendiz aprender sobre um determinado conceito envolvido na solução do problema ou sobre estratégias de resolução de problemas.

Portanto, nota-se que, após o aluno receber o *feedback* do computador sobre a execução de seu programa, ele passa a tentar identificar a origem de seu erro e saná-lo, ou ainda, empenhar-se na construção de melhorias em seu programa. Esse processo de depuração é, para o aluno, um momento de pensar sobre o pensar [Valente 1999]. A atividade de depuração na programação é muito importante do ponto de vista da aprendizagem e é exatamente por isso que deve ser estimulada. Assim, o ciclo da descrição-execução-reflexão-depuração descrito por Valente (1993), configura-se um exercício árido, que demanda esforço, dedicação, concentração e motivação por parte do aluno. Nesse ponto, torna-se imprescindível a atuação de um profissional da educação, que favoreça a aprendizagem do aluno.

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, realizou uma pesquisa junto aos alunos com intuito de identificar as maiores dificuldades no aprendizado de programação.

Como resultados, obteve que a um grande percentual de desistência na disciplina de Algoritmos e Programação e que a maior parte dos alunos submetidos à pesquisa tiveram seu primeiro contato com a programação na disciplina. Foram identificadas também as dificuldades no aprendizado, em que os alunos apontaram que a maior dificuldade é a de encontrar erros no próprio programa (erros de compilação), seguida de desenvolver um programa para executar uma tarefa. Além disso, foi questionado aos alunos que tipos de materiais os ajudariam no processo de ensino-aprendizagem da disciplina.

3. Metodologia

A pesquisa realizada neste trabalho foi classificada quanto a sua natureza como aplicada e, exploratória quanto aos objetivos. A metodologia adotada para o desenvolvimento deste trabalho consistiu inicialmente no levantamento de um referencial bibliográfico. Na etapa seguinte, denominada passo 2, foi desenvolvido um instrumento de coleta de dados para esta pesquisa, em que foi aplicado a uma turma de algoritmos e programação. Este instrumento foi utilizado durante aproximadamente 3 meses. Em seguida, no passo 3, os dados foram analisados para identificar os erros mais comuns cometidos pelos alunos na prática de programação em linguagem C.

Na etapa 4, foi realizada a submissão de um questionário aos alunos da turma de algoritmos e programação e feita a análise dos dados conseguidos com este questionário. E, no passo 5 foi feita a comparação entre os resultados conseguidos com a aplicação do instrumento de coleta de dados e o questionário submetido. A Figura 1 apresenta a descrição da metodologia.

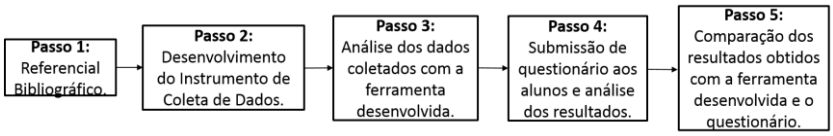


Figura 1. Descrição da Metodologia

4. Implementação

Nesta seção, serão apresentados o desenvolvimento e a dinâmica de aplicação do instrumento de coleta de dados e, será também mostrado o questionário submetido aos alunos.

4.1. Instrumento de coleta de dados

Com a finalidade de coletar informações sobre os 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.

Para a implementação da aplicação, denominada CFacil, utilizou-se a tecnologia de Shell Script, no qual o script desenvolvido é apresentado na Figura 2.

```
#!/bin/bash

#######################################
# Script que testa se arquivo existe e #
# joga a saída do gcc para um arquivo  #
#######################################
echo "Informe o arquivo de saída e o arquivo.c a ser compilado..."
read saída 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" "$arquivo" #saída padrão
        gcc -o "$saida" "$arquivo" 2>> relatorio_erros #saída para arquivo

fi
```

Figura 2. Trecho de código CFacil.

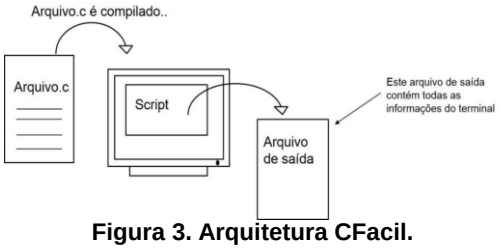


Figura 3. Arquitetura CFacil.

A arquitetura de funcionamento do software é demonstrada na Figura 3, em que 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 enviado para o arquivo que contém os erros. Na Figura 4 é apresentada a interface de execução do CFacil durante o processo de compilação de um determinado programa, enquanto na Figura 5 tem-se o arquivo de saída, gerado pela aplicação.

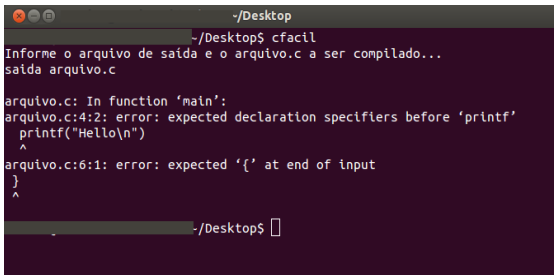


Figura 4. Execução do CFacil.

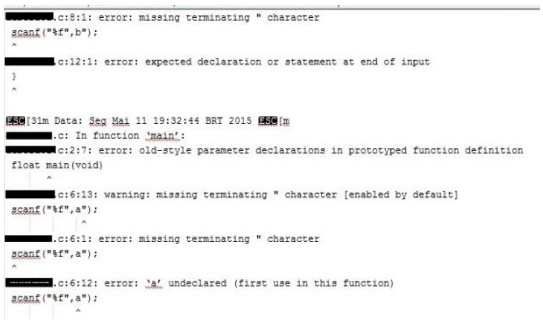


Figura 5. Arquivo de saída CFacil.

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 um laboratório de programação. As atividades acadêmicas da disciplina ocorreram em encontros semanais, sendo 4 horas/aulas, em um período total de aproximadamente 3 meses. Todas as atividades de programação seguiram o plano de ensino, sendo que a utilização do CFacil ocorreu com supervisão docente, os estudantes realizaram as tarefas de programação propostas, e ocorria a armazenagem dos logs de compilação. Todos os logs gerados durante o período do experimento foram agrupados em um único arquivo (logs.txt), com um tamanho total de 556KB, por meio do qual foram executadas todas as análises necessárias para a obtenção dos resultados desta etapa do projeto.

4.2. Questionário submetido aos alunos

A fim de validar os resultados observados com o CFacil e compor um elemento para comparação destes, foi submetido aos alunos participantes do experimento um instrumento de pesquisa na forma de um questionário composto por um conjunto de questões abertas e fechadas, que tinham a finalidade de ratificar a identificação dos principais erros de programação reconhecidos pelo compilador GCC, além de reunir as opiniões destes discentes sobre as atividades de depuração durante as tarefas de programação. O questionário, em pauta, seguiu uma estrutura em tópicos.

O primeiro tópico objetivou observar os problemas de aprendizado, identificando desta forma, as principais dificuldades enfrentadas pelos alunos no processo de ensino-aprendizagem, com o foco na complexidade dos conteúdos, prática da programação e a utilização de uma linguagem procedural; O tópico 2 (conhecimentos do acadêmico e materiais de apoio ao processo de ensino-aprendizagem) teve por objetivo questionar a percepção do estudante sobre qual tipo de material ou ferramental poderia ser adotado como recurso de apoio no ensino de programação.

5. Resultados e discussões

Nesta seção apresentam-se os resultados conseguidos com a aplicação do instrumento de coleta de dados e do questionário submetido aos alunos.

5.1 Resultados Instrumento de coleta de dados (CFacil)

Nesta fase do estudo, é possível afirmar a efetividade da ferramenta CFacil. Por meio dos logs gerados por este software, foi possível observar os principais tipos de erros cometidos pelos acadêmicos durante a programação com a linguagem C e, também definir o quantitativo de ocorrências dos mesmos. Durante os experimentos foram constatadas 1.297 compilações com o instrumento de coleta, sendo que deste total 71% das compilações não foram bem sucedidas, devido a erros identificados nos códigos desenvolvidos pelos discentes.

Com o objetivo de destacar os erros com maior incidência durante o experimento, adotou-se o Sobek[1], um software de mineração de dados, utilizado para identificar conceitos/termos mais relevantes em um determinado texto, a partir da análise de frequência destes no material textual [Klemann 2011]. A partir da aplicação deste no arquivo logs.txt, foi possível gerar um diagrama com a frequência dos principais termos e suas relações, conforme a Figura 6, na qual nesta análise é possível perceber que os termos com maior incidência são *error* e *warning*, além do termo *function* que sempre está presente nas compilações, pois indica em qual função encontra-se o erro/*warning*. Destaca-se também o termo *error expected*, onde ocorrem os erros nas funções *printf*, *scanf* e a falta de *tokens*.

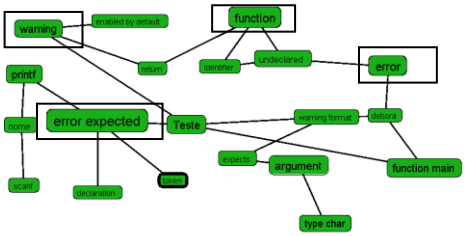


Figura 6. Diagrama de análise das saídas do CFacil com Sobek.

Gráfico 1. Resultados observados para o conceito *error*.

Ao analisar o vínculo dos erros identificados pelo Sobek, destaca-se que a partir das 922 compilações com erros ocorridas, a qual representa 71% das execuções do CFacil, foi possível elencar que aconteceram 1249 alertas (*warning*) e 2236 erros (*error*), ou seja, cada compilação errada continha, em média, 1 alerta e 2 erros.

O Gráfico 1 apresenta a incidência para o conceito *error*, no qual é denotado como erro mais comum a falta de *tokens* como parênteses ou chaves, o qual corresponde

[1] Disponível em: http://sobek.ufrgs.br/index.htm

a 49%. Em segundo, tem-se a falta de declaração das variáveis, a qual equivale a 20%, outros erros compõem um percentual de 19 pontos, a falta de terminações por aspas computa 8% e, por fim os erros na declaração de variáveis representaram apenas 4% dos erros.

Os resultados observados para o conceito *warning* são mostrados no Gráfico 2, que embora sejam apenas avisos, a chamada *warning* pode resultar no mal funcionamento do programa desenvolvido. No gráfico para o conceito *warning*, destaca-se o aviso com maior incidência a leitura de *strings* com um percentual de 32%. Em segundo, compreendendo 28% temos o erro nas funções de leitura do teclado (*fgetc* e *fgets*), após com 15% o uso de retorno em funções *void*, com 8% o erro de tipos, ou seja, declarar *int* e utilizá-lo como *float*. Em seguida com 7% destaca-se a falta de argumentos (*%d*,*%f*,*%c*) nas funções *scanf* e *printf* e, o erro na leitura de caracteres. E, por último com 3% o aviso por declaração incorreta de bibliotecas.

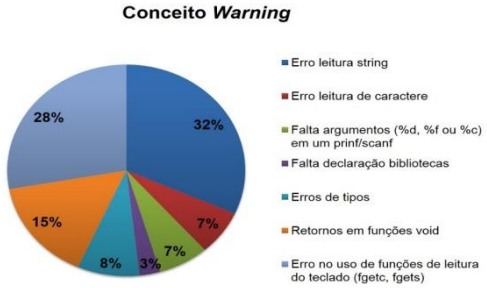
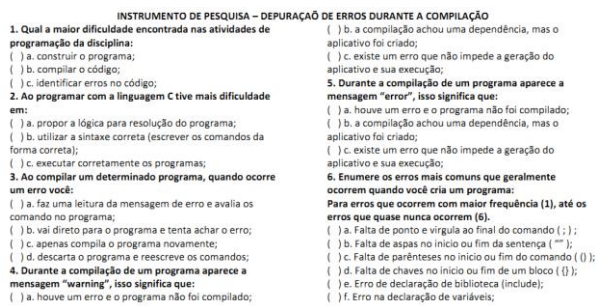


Gráfico 2. Resultados observados para o conceito *warning*.

Os resultados observados a partir dos gráficos construídos para os termos *error* e *warning* permitem inferir que a grande maioria dos erros cometidos pelos alunos são erros de sintaxe da linguagem C. Estes erros são: por falta de fechamento de parênteses e/ou chaves em estruturas como *if*, *else*, *printf*, *scanf*; erro por não terminação de comando com ponto e vírgula; erro na leitura de *strings*, ou seja, ler 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, uma porcentagem considerável de erros por não declaração ou redeclaração de variáveis.

5.2 Resultados do instrumento de pesquisa (questionário)

O questionário foi aplicado à mesma turma que utilizou o CFacil, sendo esta constituída por alunos dos cursos de engenharia e matemática. Uma amostra das questões que compuseram o instrumento é apresentada na Figura 7.


INSTRUMENTO DE PESQUISA – DEPURAÇÃO DE ERROS DURANTE A COMPILAÇÃO

1. Qual a maior dificuldade encontrada nas atividades de programação da disciplina:
( ) a. construir o programa;
( ) b. compilar o código;
( ) c. identificar erros no código;

2. Ao programar com a linguagem C tive mais dificuldade em:
( ) a. propor a lógica para resolução do programa;
( ) b. utilizar a sintaxe correta (escrever os comandos da forma correta);
( ) c. executar corretamente os programas;

3. Ao compilar um determinado programa, quando ocorre um erro você:
( ) a. faz uma leitura da mensagem de erro e avalia os comando no programa;
( ) b. vai direto para o programa e tenta achar o erro;
( ) c. apenas compila o programa novamente;
( ) d. descarta o programa e reescreve os comandos;

4. Durante a compilação de um programa aparece a mensagem "warning", isso significa que:
( ) a. houve um erro e o programa não foi compilado;
( ) b. a compilação achou uma dependência, mas o aplicativo foi criado;
( ) c. existe um erro que não impede a geração do aplicativo e sua execução;

5. Durante a compilação de um programa aparece a mensagem "error", isso significa que:
( ) a. houve um erro e o programa não foi compilado;
( ) b. a compilação achou uma dependência, mas o aplicativo foi criado;
( ) c. existe um erro que não impede a geração do aplicativo e sua execução;

6. Enumere os erros mais comuns que geralmente ocorrem quando você cria um programa:
Para erros que ocorrem com maior frequência (1), até os erros que quase nunca ocorrem (6).
( ) a. Falta de ponto e vírgula ao final do comando (;) ;
( ) b. Falta de aspas no início ou fim da sentença ("");
( ) c. Falta de parênteses no início ou fim do comando (());
( ) d. Falta de chaves no início ou fim de um bloco ({});
( ) e. Erro de declaração de biblioteca (include);
( ) f. Erro na declaração de variáveis;

Figura 7. Instrumento de pesquisa (questionário).

Gráfico 3. Dificuldades encontradas na atividade de programação.

O Gráfico 3 mostra que a principal dificuldade encontrada pelos dicentes na disciplina é identificar erros no próprio código, com um percentual de 50%. Isto pode

ser validado pela quantidade de compilações com erros (71%) que foram encontradas pela ferramenta CFacil. Estas compilações errôneas eram muitas vezes a recompilação de um mesmo código, pois o aluno não conseguiu encontrar o erro que cometeu no código e, além disto, frequentemente esta recompilação continha mais erros do que da primeira vez. Em seguida a maior dificuldade é construir o programa (36%), ou seja, utilização de uma linguagem de programação e, após com 14%, compilar o código.

As dificuldades na interpretação das mensagens de erros, apresentadas no Gráfico 4, podem ser atribuídas ao fato do idioma utilizado pelo compilador ser o inglês, o que gera um nível de distanciamento e dificuldade de compreensão, além do fato da organização da mensagem gerada pelo compilador. Por este gráfico é possível perceber um equilíbrio quanto às dificuldades no entendimento da mensagem de erro gerada pelos compiladores, ou seja, todas estas questões interferem ao aluno compreender as mensagens geradas na compilação. Fazendo uma análise dos gráficos 3 e 4, é possível perceber o porquê de os alunos terem dificuldades em identificar os erros no próprio código, pois eles têm grande dificuldade em entender as mensagens de erros geradas.

Gráfico 4. Dificuldades na interpretação das mensagens de erro.

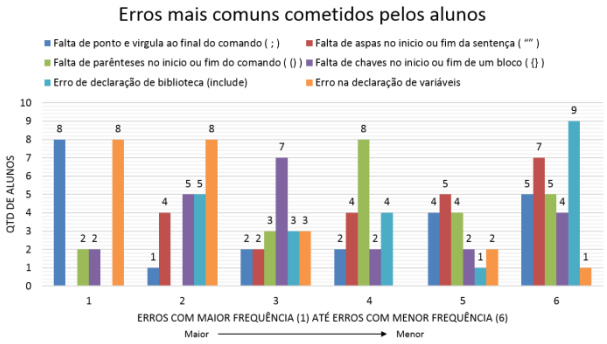


Gráfico 5. Erros mais comuns cometidos pelos alunos.

O Gráfico 5 apresenta, segundo os alunos, quais são os erros que ocorrem quando eles criam um programa. Foi utilizado 1 para os erros que ocorrem com maior frequência, até 6 para os erros que ocorrem com menor frequência. É possível observar pelo gráfico que os erros que ocorrem com maior frequência são a falta de ponto e vírgula ao final de um comando e erros na declaração de variáveis (não declaração de variáveis, não utilizar as regras de nome para variáveis).

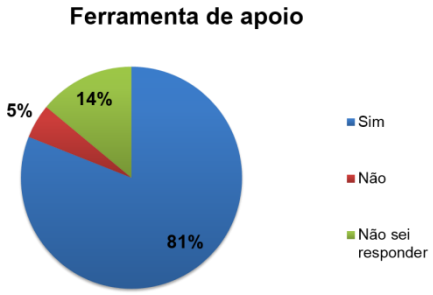


Gráfico 6. Resposta ao questionamento da utilização de uma ferramenta de apoio.

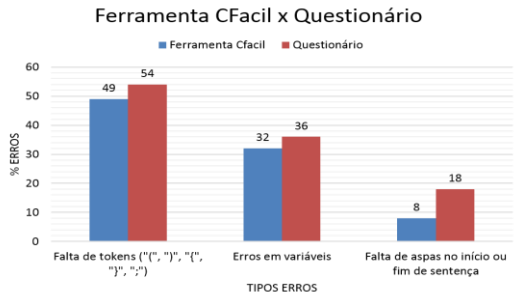


Gráfico 7. Comparação Ferramenta CFacil e Questionário

No Gráfico 6 é demonstrada a resposta ao questionamento feito aos estudantes sobre a possibilidade de construir programas de forma mais eficiente se as mensagens

de erros apresentadas pelo compilador do C fossem mais amigáveis (em português com exemplos da utilização correta do comando).

O Gráfico 7 apresenta algumas comparações entre a ferramenta CFacil e o questionário submetido aos alunos, o questionário elencava da maior dificuldade para a menor. É possível perceber então, que os maiores percentuais de erros encontrados, tanto pelo CFacil (49%), quanto pelo questionário (54%) são erros por falta de tokens, como: "{","}", "(", ")", ";". Em seguida, são encontrados grandes percentuais de erros cometidos em variáveis (não declaração, erros na utilização de tipos, nomes de variáveis em desacordo com regras da linguagem C), 32% para o CFacil e 36% para o questionário. Após também se percebe uma quantidade de erros consideráveis pela falta de aspas no início ou fim de sentenças, 8% para o CFacil e 18% para o questionário.

Cabe destacar que um dos erros menos cometidos pelos alunos, tanto no questionário, quanto na ferramenta CFacil é erros na declaração de bibliotecas (*include*), pode ser visto pelos gráficos 1 e 5.

6. Considerações Finais

A partir dos dados analisados, tanto do instrumento de coleta de dados, quanto do questionário submetido aos alunos, é possível perceber a dificuldade que os alunos iniciantes em programação têm em reconhecer as mensagens de erros geradas pelo compilador e corrigi-las. Essa observação pode ser validada pelo número de compilações recolhidas com o instrumento de coleta de dados e o grande percentual de compilações com erros que ocorreram. Em uma observação detalhada sobre estas compilações percebe-se que, em média, os alunos compilam seus programas mais de 5 vezes e mesmo assim não conseguem encontrar os erros cometidos.

Os erros cometidos muitas vezes são simples, como é o caso de esquecer o ponto e vírgula no final de comandos ou erros na declaração de variáveis, conforme apontado pelo questionário e também pelo instrumento de coleta de dados. Porém, conforme o questionamento feito aos alunos de o porquê eles não compreenderem as mensagens de erros, a resposta é dada pelo Gráfico 4, pois as mensagens são no idioma inglês e a estrutura em que são apresentadas não ajuda a alunos iniciantes a compreenderem.

Com a revisão bibliográfica feita, sabe-se que é de extrema importância a etapa de depuração na aprendizagem de programação, pois a partir dela o aluno pode construir conhecimentos sobre a disciplina, porém da forma em que é abordada não é possível. Então, foi perguntado aos alunos se eles tivessem ao seu dispor uma ferramenta, que oferecesse um *feedback* de mensagens mais amigável, eles conseguiriam construir programas de forma mais eficiente e a resposta foi sim.

Este trabalho foi a primeira etapa do projeto de uma ferramenta que auxilie na depuração de códigos em linguagem C, que forneça um *feedback* de compilação mais amigável aos alunos e que desta forma seja possível a construção de conhecimento, por parte dos alunos, de forma mais significativa. Esta etapa serviu para a validação da construção desta ferramenta.

Referências

Almeida, M. E. B., (1999). A Informática Educativa na Usina Ciência da UFAL. Anais do II Seninfe, NIES/UFAL, Maceió.

Barcelos, R. J. S., Tarouco, L., Berch, M. (2009). O uso de mobile learning no ensino de algoritmos, Renote, v.7, n.3.

Cechinel, C., Cogo, G., Betemps, C., Tavares, R. (2008). Desenvolvimento de Objetos de Aprendizagem para o Apoio à Disciplina de Algoritmos e Programação. In Anais

Simpósio Brasileiro de Informática na Educação: II Workshop de Ambientes de apoio à Aprendizagem de Algoritmos e Programação. Fortaleza: Brasil, CDROM.

Da Silva, T. S. C., de AR Tedesco, P. C., & de Melo, J. C. (2014). A importância da motivação dos estudantes e o uso de técnicas de engajamento para apoiar a escolha de jogos no ensino de programação. In Anais do Simpósio Brasileiro de Informática na Educação (Vol. 25, No. 1, pp. 11-15).

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 (Vol. 23, No. 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. Anais do XX Simpósio Brasileiro de Informática na Educação. Florianópolis, SC.

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).

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 2000

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.

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

Papert, S. (1986). Construcionism: a new opportunity for elementar 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 Anais do XXXVIII Congresso da Sociedade Brasileira da Computação: Workshop sobre educação em computação.

Valente, J. (1993) Computadores e Conhecimentos: repensando a Educação. Campinas: UNICAMP.

Valente, J. (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 na Interactive Test System for Students Improving Learning Outcomes in a Computer Programming Course. In: Advanced Learning Technologies (ICALT).