CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA

FUNDAÇÃO DE ENSINO "EURÍPIDES SOARES DA ROCHA"

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

IMPLEMENTAÇÃO DE UM INTERPRETADOR DE ALGORITMOS NA LINGUAGEM C QUE IRÁ AUXILIAR NO PROCESSO DE APRENDIZADO DE PROGRAMAÇÃO

CLAUDINEI BRITO JUNIOR

ORIENTADOR(A): PROF. ME. MAURICIO DUARTE

Marília - SP
Dezembro/2017

CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA

FUNDAÇÃO DE ENSINO "EURÍPIDES SOARES DA ROCHA"

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

IMPLEMENTAÇÃO DE UM INTERPRETADOR DE ALGORITMOS NA LINGUAGEM C QUE IRÁ AUXILIAR NO PROCESSO DE APRENDIZADO DE PROGRAMAÇÃO

CLAUDINEI BRITO JUNIOR

Monografia apresentada ao Centro Universitário Eurípides de Marília como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciência da Computação.
Orientador(a): Prof. Me. Mauricio Duarte

Marília - SP

Dezembro /2017

Portanto, quer comais quer bebais, ou façais outra qualquer coisa, fazei tudo para glória de Deus.

1 Coríntios 10.31

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

Capítulo 1

COMPILADORES E INTERPRETADORES

As linguagens de programação visam construir instruções e comandos para o computador executar determinado processamento. Um problema é que os computadores operam na base binária, ou seja, só compreendem os dígitos 0 e 1, enquanto os seres humanos têm facilidade em conversar apenas na própria linguagem humana. As linguagens de programação servem para encurtar a distância entre a linguagem de máquina e a linguagem natural humana.

Visto que as linguagens de programação servem para aproximar a linguagem do homem da linguagem da máquina, ainda é sabido que a própria linguagem de programação não consegue fazer com que o homem entenda a linguagem de máquina ou que a máquina entenda a linguagem do homem. Para isso, existem os compiladores, que são programas que tem como objetivo ler um texto numa linguagem de programação – linguagem natural – e então traduzir este texto para a linguagem de máquina.

Segundo Aho, Sethi e Ullman (1986, p.1):

> Posto de forma simples, um compilador é um programa que lê um programa escrito numa linguagem – a linguagem fonte – e o traduz num programa equivalente numa outra linguagem – a linguagem alvo. Como importante parte deste processo de tradução, o compilador relata a seu usuário a presença de erros no programa fonte.

Existem diversas classificações de compiladores, como compiladores de uma passagem, de passagens múltiplas, de carregar e executar, depuradores ou otimizadores. Além da classificação, o método de abordagem na tradução da linguagem origem para a linguagem destino, também é importante, neste caso, resumidos em compilação e interpretação.

O compilador faz um mapeamento do programa escrito em linguagem de alto nível para um programa correspondente na linguagem de máquina. A execução do programa escrito em alto nível, basicamente é dividida entre compilação – que é a transformação do programa fonte em programa objeto – e a própria execução – que é o processamento/execução do programa objeto, gerando assim resultados.

Diferentemente do compilador, o interpretador apenas executa o programa, sem compila-lo, ou seja, na interpretação não existe um mapeamento do programa fonte em um programa objeto. O interpretador lê a primeira instrução do programa fonte, faz todas as análises no código – sintática, léxica e semântica -, depois disso, converte esta instrução para linguagem de máquina e ordena que o computador execute esta instrução. Após isso, o interpretador lê a próxima instrução do programa, faz a mesma análise e executa, assim sucessivamente até a última instrução do programa. Na interpretação, a cada nova execução – leitura, análise e execução da instrução – a instrução anterior é perdida, ou seja, fica apenas uma instrução na memória em cada instante. Na próxima execução do programa, novamente todo o processo se repete, pois, há uma nova tradução e nova análise para execução, assim, comando a comando.

1.1 História dos compiladores

Em 1952, Grace Hopper fez o primeiro compilador para a linguagem de programação A-0, ainda naquela época, este compilador se parecia mais com um carregador ou linkador do que propriamente dito um compilador conforme a noção moderna de compiladores.

Logo em seguida, no ano de 1958, foi finalizado o primeiro compilador de ALGOL 58, desenvolvido especificamente para o computador Z22. O ALGOL 58 foi

desenvolvido por uma equipe de 3 alemães e um suíco, Friedrich L. Bauer, Hermann Bottenbruch, Klaus Samelson e Heinz Rutishauser.

O início do desenvolvimento de interpretadores, foi marcado pelas linguagens SpeedCoding e Lisp.

Em 1953, surgiu a primeira linguagem de programação de alto nível, a SpeedCoding, implementada por John Backus, esta linguagem foi criada para um computador IBM, no caso o IBM 701. O sistema SpeedCoding era um intepretador que fornecia pseudo-instruções para funções matemáticas. Este interpretador, ocupava em média 30% da memória disponível do IBM 701.

Em 1960 surgiu outra linguagem de programação de alto nível interpretada. Naquele ano, era implementado o Lisp por Steve Russell num IBM 704.

Nos dias atuais, existem diversas linguagens de programação de alto-nível que são interpretadas, entre elas, Python, JavaScript, CINT, Visual Basic Script, entre outras.

1.2 Diferença entre um interpretador e um compilador

Interpretadores e compiladores são *softwares* muito parecidos, porém com implementações e execuções distintas.

Os compiladores traduzem o código-fonte de uma linguagem para outra. Como dado de entrada, o compilador recebe o código-fonte numa linguagem origem e então produz um programa correspondente na linguagem destino. A linguagem origem normalmente é uma linguagem de alto nível[1], e então traduz o programa para linguagem de máquina[2]. O compilador lê o programa-fonte como um todo, o analisa e gera o código binário, e então a cada nova execução do programa, o computador não compilará novamente o código-fonte, uma vez que ele já está em código binário pois assim foi gerado pelo compilador.

[1] Linguagem de programação de alto nível: É uma linguagem de programação comumente usada e compreendida pelo ser humano, como por exemplo Java, C, C++, Python.

[2] Linguagem de máquina: A linguagem de máquina é o código-fonte escrito usando as instruções de máquina, chamado de código binário, representadas em bits.

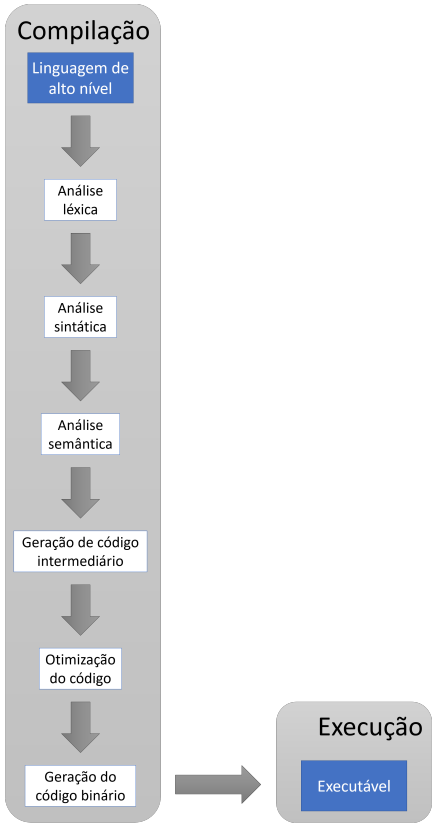

Figura 1 - Processo de execução através do compilador | Fonte: Próprio Autor

O interpretador comumente não traduz o programa fonte desenvolvido em linguagem de alto nível, mas simplesmente o interpreta, analisa e executa linha a linha. A cada nova execução do programa, o computador faz todo o processo de interpretação novamente, uma vez que este mesmo programa não existe como código binário pois foi apenas interpretado.

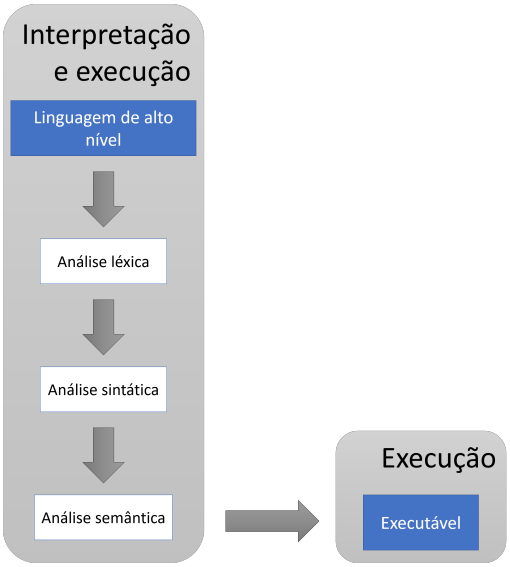

Figura 2 - Processo de execução através do interpretador | Fonte: Próprio Autor

1.3 Fases da interpretação

Assim como a compilação, a interpretação do código-fonte passará por algumas fases antes da efetiva execução do programa desenvolvido. A cada linha do código fonte, o interpretador irá submeter esta linha a três tipos de análises e então caso tenha passado com sucesso por estas fases, passa esta instrução para o computador executar.

Estas fases são executadas a cada nova linha do código.

```c
1   int main(){
2
3       float nota1, nota2, media;
4
5       nota1 = 8;
6       nota2 = 7;
7       media = (nota1 + nota2) / 2;
8
9       if (media >= 7){
10          printf("Aluno aprovado");
11      }else{
12          printf("Aluno reprovado");
13      }
14
15      printf("Fim!");
16
17  }
```

Figura 3 - Exemplo de código escrito na linguagem C | Fonte: Próprio Autor

Por exemplo, ao passar por um interpretador o código acima descrito, o interpretador irá interpretar, analisar e executar cada linha individualmente.

- Na linha 3, irá alocar espaço na memória para 3 variáveis do tipo ponto flutuante (*float*);
- Nas linhas 5 e 6 irá atribuir as notas às variáveis anteriormente declaradas;
- Na linha 7 irá fazer a soma e depois a divisão por 2 do valor destas variáveis;
- Na linha 9 irá verificar se a média é maior ou igual à 7, caso seja, executa a linha 10, tal qual exibe um texto descrevendo que o aluno foi aprovado;

- A linha 11 será executada caso média não seja maior ou igual a 7, e então irá executar a linha 10, que exibirá um texto descrevendo que o aluno foi reprovado;
- Na linha 17, exibirá um texto notificando sobre o fim do programa;

Caso houvesse um erro na linha 13, até lá o programa seria executado normalmente, ou seja, o texto de aprovado ou reprovado, seria exibido normalmente e a única linha que não seria executada, seria a linha 15.

1.4 Análise léxica

A análise léxica é o primeiro componente a ser desenvolvido, tanto num interpretador como num compilador. Aho, Sethi e Ullman (1986, p. 38) descreve: “Sua tarefa principal é a de ler os caracteres de entrada e produzir uma sequência de *tokens* [3]que o *parser* utiliza para a análise sintática.”.

A análise léxica é responsável pela conexão entre o código fonte e as demais partes do código. O objetivo do analisador léxico é ler os caracteres de entrada e então fracionar o programa fonte de entrada em trechos principais compostos e com identificação própria, chamados de *tokens*.

Ao observar o texto fonte, o analisador léxico faz a identificação de qualquer sequência de caracteres e identifica os símbolos terminais da gramática e da linguagem. Também é função do analisador léxico, a eliminação de caracteres desnecessários (comentários, espaços em branco, etc), e a função mais relevante é fazer a classificação dos *tokens* conforme tabela abaixo.

Tabela 1 - Exemplo de classificação de *tokens*, analisando o código entre as linhas 3 e 7 da Figura 3

Linha	Token	Símbolo
3	float	Palavra reservada
3	nota1	Identificador

[3] T*oken* é uma sequência de caracteres que representa um símbolo terminal.

Linha	Token	Símbolo
3	nota2	Identificador
3	media	Identificador
5	nota1	Identificador
5	=	Símbolo especial
5	8	Constante inteira
6	nota2	Identificador
6	=	Símbolo especial
6	7	Constante inteira
7	media	Identificador
7	=	Símbolo especial
7	(	Símbolo especial
7	nota1	Identificador
7	+	Símbolo especial
7	nota2	Identificador
7	)	Símbolo especial
7	/	Símbolo especial
7	2	Constante inteira

Pode ser definido como um token, qualquer sequência de caracteres que formem uma palavra-chave, operadores, identificadores, constantes, literais, cadeias e símbolos de pontuação (parênteses, vírgulas e ponto e vírgula).

Em resumo, o analisador léxico será como uma interface entre o programa fonte e o analisador sintático, transformando o programa fonte numa sequência de símbolos terminais da gramática, tais quais serão consumidos pelo analisador sintático.

1.4.1 Tratamento de erros

Os erros são difíceis de serem identificador durante a análise léxica, isso acontece, pois, o analisador léxico possui uma visão muito pontual do código fonte. Pela Figura 4, é perceptível que há um erro na declaração de variáveis.

```
3        flota nota1, nota2, media;
```

Figura 4 - Exemplo incorreto de declaração de variáveis Fonte: Próprio Autor

Porém, o analisador léxico irá analisar este código, e identificará o token *flota* (que deveria ser *float*) como um identificador, e então a atribuição de um erro nesta linha ficará por conta das próximas fases de análise do código fonte.

1.5 Análise sintática

A principal função da análise sintática é verificar se as construções usadas no programa estão gramaticalmente corretas. Para avaliar se a construção do programa está válida, são definidas estruturas sintáticas por meio de uma gramática livre de contexto. Recebendo uma gramática livre de contexto GLC e uma sentença (programa fonte) S, o analisador sintático avalia se a sentença S pertence à linguagem construída por GLC.

O analisador sintático recebe uma cadeia de *tokens* vindoura da análise léxica e então analisa se esta cadeia pode ser representada pela gramática da linguagem fonte. Também é papel do analisador sintático reportar os erros de sintaxe de forma compreensível. A saída do analisador sintático é uma representação da árvore gramatical para a sequência de *tokens* produzida pelo analisador léxico na etapa anterior.

A análise sintática analisa a organização dos tokens no código. Os modelos de análise sintática comumente utilizam gramáticas livres de contexto, tal qual, devem retratar uma gramática formal. Estas gramáticas livres de contexto, podem ser definidas através de algoritmos que fazem a derivação de todas as possíveis construções da linguagem. Estas derivações determinam se uma sequência de tokens obedece à sintaxe da linguagem de programação.

A árvore gramatical representada a partir da análise sintática pode ser construída de forma declarada, através de estrutura de dados, ou então ficar inerente nas camadas que aplicam as especificações de produção desta gramática durante a verificação.

1.5.1 Métodos de análise e subclasses gramaticais

Os métodos mais usados e eficientes para a construção de um analisador sintático, são classificados como *top-down* ou *bottom-up*. Os analisadores *top-down* constroem árvores sintáticas começando do topo do código (raiz) e vão até o fundo (folhas), lendo a entrada da esquerda para a direita. Os analisadores *bottom-up*, fazendo o inverso do top-down, começam analisando das folhas e vão até a raiz, mas, mantém a mesma característica de leitura da entrada, da esquerda para a direita.

Para gramáticas construídas sob subclasses gramaticais como a LL e LR (que tem expressões o suficiente para representar a maioria das construções sintáticas das linguagens de programação), tem como saída do analisador sintático, uma árvore de derivação gramatical segundo a sequência de tokens recebida do analisador léxico.

1.5.2 Tratamento de erros

Conforme mencionado por Aho, Sethi e Ullman (1986, p. 73), os erros num programa-fonte podem ser:

> "Léxicos, tais como errar a grafia de um identificador, palavra-chave ou operador;
> Sintáticos, tais como uma expressão aritmética com parênteses não balanceados;
> Semânticos, tais como um operador aplicado a um operando incompatível;
> Lógicos, tais como uma chamada infinitamente recursiva.".

No tocante aos erros sintáticos, o analisador sintático deve relatar a presença de erros de forma exata e precisa, além de se recuperar de forma eficaz e robusta ao encontrar um trecho do programa com erro, para que possa se detectar os erros posteriores a estes detectados. O analisador não deve apenas detectar a presença de um erro, mas também deve procurar tratá-lo e assim procurar adivinhar o que o programador tinha em mente no momento de ter escrito um erro. Porém,

normalmente os erros não ocorrem quando são detectados, estes poderiam existir muito antes da sua detecção pelo analisador sintático. Os métodos de análise sintática LL e LR, são eficientes no momento de detectar erros no momento de sua criação, impedindo com que este erro se propague.

Um tratamento de erros (diferentemente da detecção de erros) deve apontar o erro ocorrido, e também exibir o local onde o erro aconteceu para que então o programador possa ver o registro de erro, entender o equívoco e corrigir. O tratamento de erros do compilador de uma forma geral (não apenas do analisador sintático) é um tanto quanto complexo, pois, o compilador não deve encerrar a atividade ao encontrar um erro, mas sim, encontrá-lo e prosseguir com a análise para então exibir os outros erros e facilitar para a correção por parte do programador. Existem diversos métodos e estratégias genéricas que são aplicadas à análise sintática para tentar prosseguir com a análise após encontrar um erro. No geral, estas estratégicas comumente se resumem a encontrar um erro e pular para um estado do programa, onde existe a possibilidade de o resto do programa não estar totalmente incorreto e fazer sentido continuar a análise naquele ponto. Mas ainda assim, esta estratégia pode ser falha, pois ao procurar ignorar algum erro para procurar outros erros, o analisador sintático pode gerar supostos erros na cadeia de análise, sendo que o programador não gerou efetivamente estes erros. Por exemplo, na figura abaixo, a ausência do ponto e vírgula na declaração da variável *media* na 4ª linha fará com que o analisador sintático registre este erro, mas para prosseguir, despreza esta linha, de forma a ignorar esta declaração e assim, irá gerar outros erros em próximas fases da análise, como por exemplo, a não declaração da variável *media*, erros estes até então inexistentes.

```
1   int main(){
2       float nota1 = 8;
3       float nota2 = 9;
4       float media
5
6       media = (nota1 + nota2) / 2;
7
8       if (media > 7)
9           printf("Aprovado");
10      else
11          printf("Reprovado")
12
13  }
```

Figura 5 - Exemplo de código em linguagem C com problema sintático | Fonte: próprio Autor

É inevitável a geração de erros conforme relatado acima, mas uma solução que os compiladores têm para que o compilador não exiba mensagens de erros inexistentes ao programador, é inibir mensagens de erros vindouras de erros descobertos já anteriormente, ou seja, caso exista algum erro relacionado à uma declaração anteriormente problemática (no exemplo, relacionado à variável *media*), o compilador não irá exibir este novo erro, tal qual, pode ter sido gerado pelo próprio compilador.

Para analisar o programa e identificar se o mesmo está correto ou não, a análise sintática deve-se utilizar da gramática da linguagem de programação como parâmetro.

Capítulo 2

SOFTWARES DE APOIO AO APRENDIZADO

Neste capítulo são abordados os conceitos de softwares educacionais, apresentando a sua definição, classificações e aplicação.

2.1 Definição de software educacional

O software educacional é uma categoria de software que faz mais do que apenas servir de ferramenta para o aprendizado do aluno. Mais do que apenas computadores disponíveis na escola para gozo do aluno como ferramenta didática, para que alunos de quaisquer níveis educacionais consigam fazer da informática seu aliado no aprendizado, é necessário que existam softwares que estejam preparados para fornecer informação e transformá-las em conhecimento.

Segundo Gomes (2003, p. 02):

> O software educacional é um software desenvolvido com interfaces e artefatos educativos para servir de mediador entre atividades educativas nas diversas áreas do conhecimento.

Uma das maiores ou talvez a maior dificuldade no desenvolvimento de softwares educacionais está diretamente relacionada ao fato que no processo de compreensão, desenvolvimento e aplicação, há uma diferença impactante entre as representações que designers, programadores e professores têm acerca dos

processos de ensino e aprendizagem. Para solucionar este problema, é necessário que haja uma grande interação entre programadores e professores, mas além da interação, é necessário que um conheça um pouco mais do universo do outro, visto que existe uma grande dificuldade em compartilhar conceitos de diferentes áreas do conhecimento, e aplicar estes conceitos num software educacional.

Valente (1999, p. 71) reitera que o software educacional por si só não é capaz de fazer com que o aluno aprenda

> O computador pode ser um importante recurso para promover a passagem da informação ao usuário ou facilitar o processo de construção de conhecimento. No entanto, por intermédio da análise dos softwares, é possível entender que o aprender (memorização ou construção de conhecimento) não deve estar restrito ao software, mas à interação do aluno-software.

2.2 Classificações dos softwares educacionais

Alguns softwares educacionais apresentam suas específicas características que maximizam a compreensão, enquanto outros softwares necessitam de um maior envolvimento do professor, que desenvolverá situações que completam a compreensão.

2.2.1 Tutoriais

Os softwares tutoriais são softwares onde a informação é organizada de acordo com uma sequência pedagógica e didática, e então, esta sequência é apresentada ao estudante, para que este possa cumprir a sequência e obter determinado resultado que irá fazer com que a sua compreensão seja efetivada.

Numa das subclassificações de softwares tutoriais, o software tem o domínio do cenário de ensino e do que pode ser apresentado ao aluno, que pode mudar de tópicos, apertando a tecla ENTER, ou então, o software altera a ordem de acordo com as respostas dadas pelo aluno. Uma outra subclassificação, é onde o aluno tem o domínio e pode escolher o que deseja ver.

Independentemente da subclassificação, a informação que está disponível ao aluno foi programada e organizada previamente por alguém. Ele está limitado a essa informação e então o computador assume a função de uma máquina de ensinar. A interação entre ele e o computador baseia-se na leitura da tela ou na escuta da informação fornecida pelo computador, no avanço pelos tópicos, apertando a tecla ENTER, na escolha de informação, usando o mouse e/ou resposta de perguntas que são digitadas no teclado.

2.2.2 Programação

Quando o aluno desenvolve algo para ser executado no computador, o computador pode ser visto como uma ferramenta que solucionará problemas. O programa desenvolvido pelo aluno, ainda que simples e básico, utiliza conceitos, estratégias e uma forma de agir para resolução dos problemas. Desta forma, a produção de um programa no computador obriga que o aluno processe informação, transforme esta informação em conhecimento aplicável e assim, solucione um problema.

Programar um software para o computador, usando uma linguagem de programação (ainda que de interface visual, "arrasta e solta"), permite identificar diversas ações, que acontecem em termos do ciclo de solução de problemas: descrição, execução, reflexão, depuração e descrição.

2.2.2.1 Descrição

A descrição da resolução do problema em termos da linguagem de programação, significa lançar mão de toda a cadeia de conhecimento para reproduzir e explicitar a sequência algorítmica da resolução do problema.

2.2.2.2 Execução

A execução por parte do computador da descrição algorítmica anteriormente desenvolvida é a descrição de como o problema é resolvido em termos de linguagem de programação, ou seja, como o computador irá executar e solucionar o problema. Visto que o computador é apenas uma máquina, esta execução fornece

um feedback exato e imediato, visto que o resultado desta etapa é fruto do que foi solicitado ao computador executar.

2.2.2.3 Reflexão

A reflexão sobre o que foi produzido pelo computador produzirá diversos níveis de raciocínio, níveis este, que causarão alterações na estrutura mental e comportamental do aluno. A reflexão sobre o resultado do programa deve causar uma das seguintes ações por parte do aluno: Ou ele não modifica o processo visto que suas ideias iniciais sobre a solução do problema são iguais ou se assemelham aos resultados pelo computador, entendendo que o problema está resolvido; ou o aluno depura o processo, quando o resultado não é igual ou semelhante ao resultado da intenção original.

2.2.2.4 Depuração

A refinação dos conhecimentos através da busca de novas informações e da reflexão faz com que o aluno busque informação sobre: conceitos de um específico conhecimento (ele não sabe a fórmula de bhaskara por exemplo), alguma prática da linguagem de programação, computação ou estratégias (ele não sabe como fazer potenciação na linguagem de programação). Após buscar e concluir a procura por esta nova informação, esta informação é incorporada pela estrutura mental (de informação se transforma em conhecimento) e então é utilizada pelo aluno no programa para modificar a descrição que estava definida anteriormente e então se repete o ciclo: descrição, execução, reflexão, depuração e descrição.

2.2.3 Processadores de texto

Na utilização de softwares como processadores de texto, as ações desenvolvidas pelo aluno também são cabíveis de análise em termos do ciclo descrição, execução, reflexão, depuração e descrição. Ao redigir um texto, a comunicação entre o computador acontece através do idioma natural e pelos comados do processador de texto para formatar o texto (alinhamento de texto, sublinhas trechos, etc.). Os processadores de texto, comumente são intuitivos e

simples de usar, de modo a facilitar a expressão escrita de nossos pensamentos. A utilização de processadores de texto vai bem até a parte da execução, visto que o processador de texto só pode executar o aspecto de formato do texto (colocar em itálico palavras de outros idiomas, iniciar com letra maiúscula uma frase, etc.) ou alguns estilos da escrita (corrigir erros de concordância numeral, erros de gramática, etc.), porém, não pode executar o conteúdo e apresentar um feedback em termos de semântica daquilo que foi redigido.

2.2.4 Multimídia e internet

Antes de entender o funcionamento da multimídia e internet como software educacional, é necessário compreender a diferença entre o uso de uma multimídia já pronta (estática) e o uso de sistemas de composição para o aluno desenvolver sua própria multimídia.

A utilização da multimídia se assemelha e muito aos tutoriais, porém, se diferenciando pelo fato da multimídia ter outras facilidades como a intercalação de textos, imagens, vídeos, sons, etc., facilitando ainda mais a compreensão da ideia.

Tanto no tutorial como na multimídia e internet, o aluno não está descrevendo seus pensamentos e ações, mas sim, decidindo entre várias opções disponibilizadas pelo software. Depois de escolhida determinada opção, o computador exibe a informação disponível e o aluno pode refletir e então tomar ações embasado sob esta informação adquirida e quiçá transformá-la em conhecimento, aplicando-a. A navegação do aluno na internet ou multimídia acontece pelas ações de avanças e voltar sobre os tópicos de informação.

Com uma gama gigantesca de informações, é possível que o aluno navegue superficialmente por um número grande de tópicos, mas também é possível que navegue em poucos, mas aprofunde-se nas informações discorridas neste, mas ainda como nos tutoriais, o aluno continuará restrito às informações que o software disponibiliza.

A navegação do aluno, pode mantê-lo alocado nos tópicos durante um longo período de tempo, porém, não pode existir uma certeza sobre a transformação das informações visualizadas em conhecimento aplicado. Desta forma, conclui-se que o uso de softwares multimídia e da própria internet auxiliam o aluno a percorrer e conhecer mais informações, porém, não garante que este aluno tenha compreendido

tudo e transformado aquelas informações em conhecimento, e então, após ter passado um período, aquele amontanhado de informações visualizadas, não servirão mais para nada visto que não foram aplicadas, fazendo com que sejam esquecidas.

2.2.5 Jogos

Jogos educacionais como software também são analisados na forma do ciclo descrição, execução, reflexão, depuração e descrição. Têm características semelhantes às dos tutorais e softwares de simulação, assemelhando-se mais de um ou do outro de acordo com o nível de interatividade e exposição de ideias, do aluno para o computador. Comumente, jogos procuram desafiar e fomentar no aluno o desejo por vitória, cercando-o em uma competição contra a inteligência artificial do computador ou contra os colegas. A maneira mais simples de se fazer isso é apresentar um contexto e alternativas para que o aluno possa identificar e defender uma ideia dentre as apresentas, tomando-a como correta e as outras alternativas como incorretas. Uma outra possibilidade de jogabilidade é o que se assemelha às simulações fechadas, onde as regras do jogo são definidas previamente, fazendo com que o aluno deva jogar o jogo e então, criar hipóteses, estratégias e aplicar conhecimentos prévios para elaborar novos conhecimentos e solucionar um problema.

Uma limitação dos jogos educacionais é que estes têm a função de cercar o aluno em uma competição e então esta competição favorecer o aluno em termos de vitórias, mas desfavorecer o aluno em termos de informação e conhecimento. Os jogos não podem ser descartados como estratégia de educação por software, pois podem criar situações e condições para o aluno aplicar os conceitos e estratégias previamente adquiridos, porém, o aluno pode aplicar os conceitos e estratégicas de forma positiva ou negativa, sem entender se aquela aplicação resultará em resultados bons ou ruins para a sua vitória no jogo e para a construção de seu conhecimento.

REFERÊNCIAS

AHO, Alfred V.; SETHI, Ravi; ULLMAN, Jeffrey D. **Compiladores - Princípios, técnicos e ferramentas**. Rio de Janeiro: LTC - Livros Técnicos e Científicos Editora S.A., 1986. 351 p.

VALENTE, José Armando. **O Computador na sociedade do conhecimento**. Campinas: UNICAMP: Nied, 1999. 115 p.

PRICE, Ana Maria de Alencar. **Implementação de linguagens de programação: Compiladores**. Porto Alegre: UFRGS: Sagra Luzzatto, 2001. 195 p.

FURLANETO, Mayara Ribeiro. **DESENVOLVIMENTO DE UM SOFTWARE EDUCACIONAL DO JOGO TANGRAM, USANDO OPENGL**. 2010. 58 p. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação)- UNIVEM, Marília, 2010.

GOMES, Alex Sandro; WANDERLEY, Eduardo Garcia. **Elicitando requisitos em projetos de Software Educativo**. 2003. 12 p. Artigo (Workshop em Informática na Educação - WIE)- WIE, Recife: UFPE, 2003.