DESENVOLVIMENTO DE UMA LINGUAGEM DE PROGRAMAÇÃO INTERPRETADA PARA FINS ACADÊMICOS – DEXTER

Everton de Vargas Agilar, Giovani Rubert Librelotto

UNIFRA – Centro Universitário Franciscano
Rua das Andradas, 1614 – 97.010-032 – Santa Maria – RS – Brasil

everton_ti07@yahoo.com.br, giovani@unifra.br

Resumo. *Este artigo propõe o desenvolvimento de uma linguagem de programação interpretada para fins acadêmicos com sintaxe semelhante à Pascal além de conter alguns recursos de linguagens como C e Java, a fim de ser utilizado em disciplinas iniciais dos cursos de Sistemas de Informação e Ciência da Computação, bem como em outros cursos que possuem disciplinas de programação na sua grade curricular.*

Abstract. *This article proposes the development of a programming language interpreted for academic purposes with syntax similar to Pascal. This language contains some features of languages, like C and Java, to be used in initial subjects of the courses of Information Systems and Computer Science, as well as courses in other disciplines that have programming in their grade curriculum.*

1. Introdução

Compiladores e interpretadores permitem a construção de programas de computador utilizando uma linguagem de alto nível para codificação das instruções a serem executadas. No contexto de linguagens de programação, é um sistema que aceita como entrada um programa escrito em uma linguagem de programação (linguagem fonte) e produz como resultado um programa equivalente em outra linguagem (linguagem objeto). [Price and Toscani 2001]

Para que o projeto e implementação de uma linguagem de programação interpretada seja possível, é necessário estudar sobre a implementação de compiladores e interpretadores, suas diferenças de implementação, suas vantagens e desvantagens, além da arquitetura de cada um.

O artigo inicia com uma apresentação sobre compiladores e interpretadores destacando suas diferenças e alguns exemplos de linguagens de programação compilada e interpretada existentes. Após será apresentado o projeto da linguagem de programação Dexter e um comparativo com o compilador Pascal ZIM, uma linguagem de programação Pascal desenvolvido no Departamento de Ciências da Computação da Universidade de Brasília, utilizado como ferramenta de apoio ao ensino e aprendizagem da linguagem Pascal nesta instituição e por diversas outras instituições de ensino no Brasil.

2. Compiladores

Segundo Wirth (2005), um compilador é um programa que lê um programa escrito em linguagem de alto nível e produz um programa semanticamente equivalente, porém escrito em outra linguagem, o código objeto.

Antes dos compiladores, os programadores trabalhavam apenas com a linguagem de máquina, ou seja, com seqüências de 0s e 1s. Do ponto de vista dos computadores, a codificação de instruções como números binários é perfeitamente natural e muito eficiente. No entanto, nós seres humanos, temos grande dificuldade para entender e manipular esses números. As pessoas lêem e escrevem símbolos (palavras) com muito mais facilidade do que longas seqüências de dígitos [Patterson, Hennessy 2005]. A figura 1 demonstra o processo de compilação de um programa.

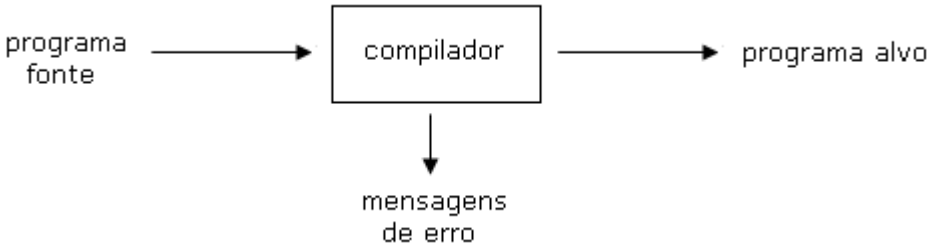

Figura 1. Processo de compilação

Exemplos de compiladores incluem os compiladores de C, C++, Pascal, Clipper, Fortran, etc.

3. Interpretadores

Interpretadores são programas processadores que lêem um programa escrito em linguagem de alto nível (código fonte), e produz o "efeito de execução" do algoritmo original, ou seja, não convertem o programa fonte em um programa objeto, em vez disso, executam os algoritmos diretamente. [Price and Toscani 2001]

Técnicas interpretativas permitem grande flexibilidade na implementação de linguagens de programação para diferentes equipamentos de computação e tornam algumas construções complexas dos compiladores muito mais fáceis de serem implementadas com interpretadores. [Price and Toscani 2001]

Os interpretadores são freqüentemente usados para executar linguagens de comandos, dado que cada operador numa tal linguagem é tipicamente uma invocação de uma rotina complexa.

Exemplos de interpretadores incluem os interpretadores de BASIC, SQL, Java, bash, PL/SQL e PHP. Existem também interpretadores para linguagens tipicamente compiladas, como é o caso de C e C++. São utilizados principalmente para depuração em ambientes integrados de desenvolvimento (IDE – *Integrated Development Enviroment*). A figura 2 ilustra o processo de interpretação.

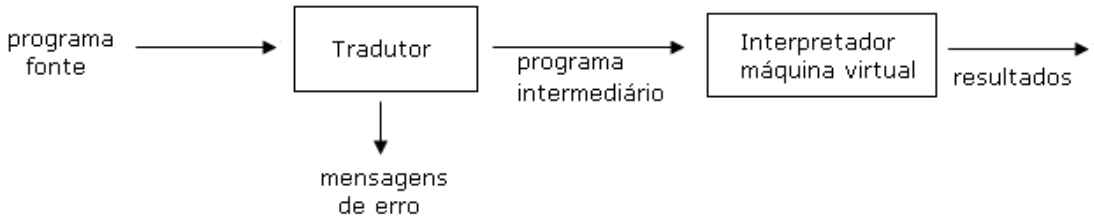

Figura 2. Processo de interpretação

4. Fases de um Compilador e um Interpretador

Compiladores e interpretadores operam em fases, cada uma das quais transforma o programa fonte de uma representação para outra. Na prática, algumas fases podem ser agrupadas e a representação intermediária entre as mesmas não precisa ser explicitamente construída. [Aho ***et al.*** 2005]

A figura 3 demonstra as fases de um compilador e de um interpretador.

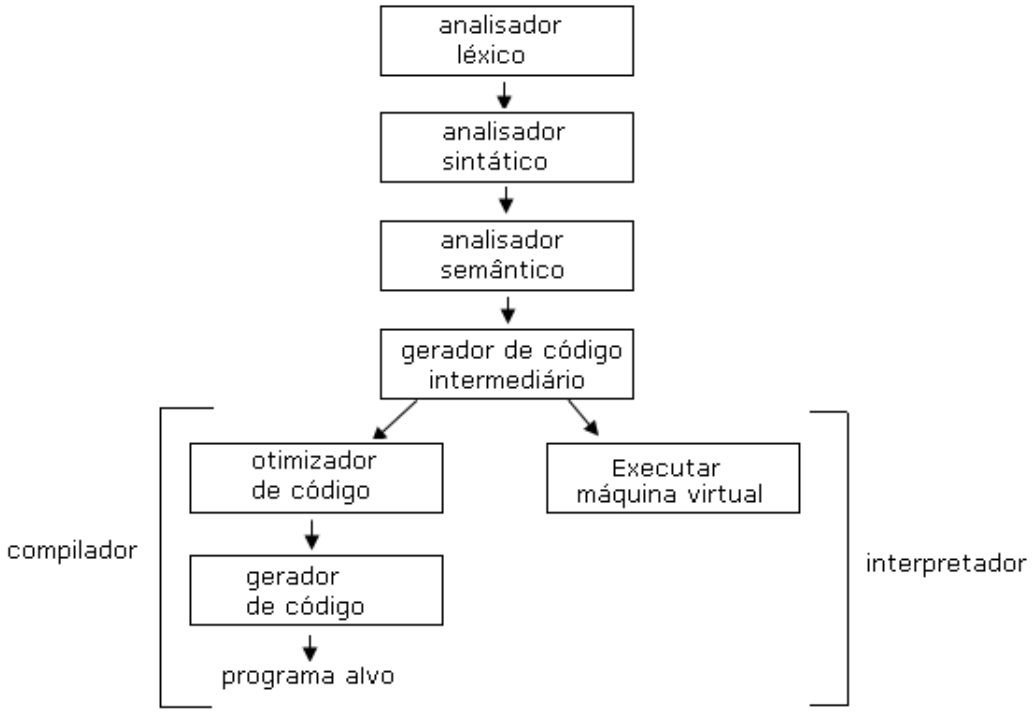

Figura 3. Fases de um compilador e um interpretador

5. Compiladores x Interpretadores

Compiladores e interpretadores possuem semelhanças em relação a sua implementação principalmente na fase de análise (análise léxica, sintática e semântica). Já na fase de síntese (geração de código intermediário, otimização e geração de código objeto), as diferenças são mais visíveis. Enquanto a fase de síntese do compilador se propõe a gerar código objeto, um interpretador executa as operações imediatamente à medida que lê o programa fonte ou o código intermediário. [Aho ***et al.*** 2005]

A implementação de compiladores é bem mais complexa que a dos interpretadores, por conta da geração de código objeto, mas o desempenho da execução de um programa compilado é muito superior a um interpretado, pois um programa interpretado precisa fazer uma tradução das operações. [Wirth 2005]

Um programa interpretado ao ser executado necessita do interpretador para traduzir as operações, à medida que o programa é executado, enquanto que um programa compilado não necessita do compilador, podendo ser executado diretamente.

Ao projetar uma linguagem de programação é necessário analisar a abordagem mais apropriada: compilar ou interpretar. Pode-se também ter os dois tipos de abordagens para uma mesma linguagem de programação. Isso depende muito do uso que será dado à linguagem de programação.

6. Interpretador DEXTER

Para o desenvolvimento da linguagem de programação interpretada DEXTER será utilizado Pascal, como linguagem de programação. O projeto poderá ser compilado em qualquer plataforma que forneça um compilador para Pascal, como Windows (Delphi, Turbo Pascal, Free Pascal) ou Linux (Free Pascal, Kylix).

A maioria dos compiladores fornece SDKs (*Software Development Kit*) para o desenvolvimento de software formado por bibliotecas de rotinas e classes já prontas. Pretende-se evitar ao máximo o uso de tais bibliotecas, a não ser quando necessário, preferindo-se programar todos os recursos necessários facilitando assim o aprendizado e a compilação do projeto em outros compiladores Pascal que podem não ter as mesmas bibliotecas.

O interpretador desenvolvido executará programas escritos na linguagem DEXTER, uma linguagem de programação estruturada semelhante à Pascal bem simplificada. Como o objetivo do projeto é acadêmico, à parte referente ao tratamento de erros deverá ser dada mais atenção, pois uma linguagem de caráter acadêmico deveria ajudar o aluno na localização dos erros de sintaxe. Em linguagens de programação profissionais, as mensagens de erro freqüentemente exigem experiência.

A seguir, será dado uma descrição geral dos recursos que serão implementados no interpretador DEXTER:

- Chamada de funções e procedimento com recursividade
- Estrutura em blocos: begin end;
- Estrutura de Controle:
  - if-then-else
  - case of
  - switch
- Estrutura de Laço:
  - for (;;)
  - for do
  - repeat while
  - repeat until
  - while do
- Operadores Aritméticos: + - / *

- Operadores Relacionais: > < >= <= <> == !< !> !>= !<=
- Tipos de dados: Integer, Boolean, string, Real
- Tratamento de arquivo orientado a objetos como em Java
- Métodos de console: Read, ReadLn, Write, WriteLn
- Declaração de variáveis
- Declaração de constantes
- Declaração de variáveis locais

Alguns recursos têm duas versões, uma versão estilo Pascal e outra versão estilo C, como é o caso das estruturas de controle "*case of*" do Pascal e "*switch*" do C. A razão para isto é permitir codificar instruções com sintaxe tanto do Pascal quanto do C para facilitar o aprendizado destas duas linguagens. A figura 4 apresenta um exemplo de programa em DEXTER.

```
program programa02;

procedure PrintMenu();
begin
  WriteLn('Programa para demonstrar chamada de procedimento');
  WriteLn('------------------------------------------');
  WriteLn();
  WriteLn('               *** Controle de Estoque ***');
  WriteLn('               --------------------------');
  WriteLn('               1 - Cadastro de Produtos');
  WriteLn('               2 - Cadastro de Grupos');
  WriteLn('               3 - Cadastro de Clientes');
  WriteLn('               4 - Cadastro de Fornecedores');
  WriteLn('               5 - Movimentos');
  WriteLn('               6 - Relatorios');
  WriteLn('               7 - Configuracao');
  WriteLn('               8 - Sair');
end;

begin
  PrintMenu;
end.
```

Figura 4. Exemplo de um programa de controle de estoque

7. Arquitetura do Interpretador DEXTER

Interpretadores de um modo geral compõem-se de funções padronizadas que compreendem a análise do programa fonte e a posterior síntese (execução). [Price and Toscani 2001]

O interpretador DEXTER é dividido em fases. Cada fase é responsável por uma função bem definida: análise léxica, análise sintática e síntese. Essas funções são mapeadas em classes Pascal.

Uma classe, segundo Lischner (2000), é um super-registro composto de campos, funções e procedimentos (chamados métodos) e propriedades. Classes são recursos presentes em linguagens orientadas a objetos (como Pascal, C++, Java) que permitem abstrair a complexidade de um projeto, dividindo-se em partes lógicas. A divisão do interpretador em classes permite melhor compreensão e clareza das fases do interpretador e como estas fases estão interligadas.

Duas classes principais e sete classes utilitárias formam o núcleo do projeto do interpretador. A tabela 1 seguir lista as classes do interpretador.

Tabela 1. Classes do interpretador DEXTER

Nome da Classe	Descrição Funcional
TInterpretador	Principal classe do interpretador, responsável pela análise sintática, semântica e síntese (interpretação do programa)
TAnalisadorLexico	Segunda classe principal, responsável pela análise léxica do programa fonte.
TToken	Classe utilitária que representa um *token* reconhecido pelo analisador léxico.
TVariavel	Classe utilitária que representa um identificador. Um identificador pode ser do tipo *Integer, string, Boolean* ou *Real*.
TFunction	Classe utilitária que representa uma função ou procedimento da linguagem DEXTER.
TPilha	Classe utilitária que representa uma estrutura de dados do tipo Pilha utilizada para armazenamento de variáveis locais de funções e procedimentos, etc.
TTabelaVariaveis	Classe utilitária que representa uma estrutura de dados do tipo lista para armazenamento de variáveis do programa.
TTabelaFunctions	Classe utilitária que representa uma estrutura de dados do tipo lista para armazenamento de funções e procedimentos do programa.
TKeyword	Enumeração dos códigos numéricos das palavras chaves.

7.1 Funcionamento do Interpretador DEXTER

Nesta seção do artigo, será descrito o funcionamento do interpretador DEXTER, detalhando as classes do interpretador e como estas classes estão formadas para implementação das fases típicas de um interpretador.

TInterpretador é a primeira e principal classe do interpretador. O interpretador instancia um objeto chamado “Interpretador” desta classe para iniciar a interpretação do programa passado como parâmetro através do prompt de comando do sistema operacional. Este objeto por sua vez, cria e inicializa vários outros objetos necessários para fazer a interpretação do programa: *Scan* (analisador léxico), *Vars* (tabela de variáveis), *Functions* (tabela de funções), *Pilha*, *Result* (variável *Result* global do Pascal).

Antes que um programa possa ser interpretado, é necessária a obtenção das unidades léxicas (*tokens*) [Price and Toscani 2001]. O analisador léxico, representado pelo objeto *Scan*, é invocado pelo objeto *Interpretador*, para obter os *tokens* do programa. Internamente, ao ser criado, *Scan* percorre o código fonte completo, separando os *tokens*, descartando linhas em branco e comentários e armazenando-os em

uma lista duplamente encadeada na forma de objetos *TTokens*. Cada *token* mantém seu tipo e valor. O tipo de um *token* é determinado por uma enumeração *TKeyword*.

O interpretador então executa o programa, realizando a análise sintática e semântica enquanto obtém os *tokens* na forma de objetos *TToken* do analisador léxico *Scan*. O interpretador processa diretamente os *tokens*. Segundo Price and Toscani (2001), uma técnica mais eficiente seria gerar uma linguagem intermediária, que seria então interpretada. A geração de um código intermediário poderá ser feito no futuro.

8. Trabalhos Relacionados

Muitos trabalhos têm sido realizados na área de compiladores e interpretadores. Alguns destes trabalhos foram desenvolvidos por estudantes enquanto cursavam cursos de graduação ou pós-graduação em Instituições de Ensino. Um dos trabalhos que merecem atenção é o compilador Pascal ZIM.

O compilador Pascal ZIM, desenvolvido no Departamento de Ciências da Computação da Universidade de Brasília, é fruto de muitos anos de pesquisa e trabalho na área de tradutores e linguagens de programação. Este compilador é atualmente utilizado em muitas Instituições de Ensino no Brasil como ferramenta de apoio ao ensino e pesquisa no estudo das linguagens de programação, em especial Pascal.

Composto por um compilador Pascal simples, um editor de texto e um *help* (arquivo de ajuda) permitem aos estudantes ou interessados em aprender a programar na linguagem Pascal utilizar um ambiente de fácil aprendizado.

O compilador Pascal ZIM implementa um subconjunto da linguagem Pascal e contém as estruturas de dados, funções e comandos mais utilizados por iniciantes no estudo dessa linguagem. O arquivo de ajuda que acompanha o produto especifica as instruções suportadas.

De forma semelhante ao Pascal ZIM, o projeto do interpretador DEXTER começou com um projeto na disciplina de Projeto Interdisciplinar I em um Curso de Sistemas de Informação, visando propiciar o aprendizado e permitir por em prática os conhecimentos que serão adquiridos na área de compiladores e interpretadores.

9. Conclusão

Este trabalho apresentou a proposta de uma linguagem de programação interpretada para fins acadêmicos, fornecendo uma visão geral sobre compiladores e interpretadores, suas diferenças de implementação e arquitetura, com exemplos de linguagens de programação interpretada e compilada.

O artigo também abordou sobre os trabalhos na área e destacou o compilador Pascal ZIM, desenvolvido na Universidade de Brasília. Assim como este compilador, o interpretador DEXTER também busca implementar um subconjunto da linguagem Pascal além de conter alguns recursos da linguagem C e Java a fim de ser utilizado em disciplinas iniciais dos cursos de Sistemas de Informação e Ciência da Computação, bem como em outros cursos que possuem disciplinas de programação na sua grade curricular.

Referências

Aho, A.V., Sethi, R., Ullmann, J. D. (2005) Compiladores: Princípios, Técnicas e Ferramentas, Rio de Janeiro, Editora LTC

Crenshaw, J. W, Ph.D (1995) Apostila sobre criação de compiladores disponível em http://br.geocities.com/feliposz/compiladores/crenshaw
Tradução e adaptação: Soranz, F.

Curado, L. R. A, Página do Compilador Pascal ZIM obtido em http://br.geocities.com/pascalzim/

Lischner, Ray (2000), Delphi – O Guia Essencial, Rio de Janeiro, RS, Editora Campus, pp. 34-76

Mesezes, P. B (2000), Linguagens Formais e Autômatos, Instituto de Informática da UFRGS, 3°ed Porto Alegre, Editora Sagra Luzzato

Patterson, D. A, Hennessy, J. L (2005) Organização e Projeto de Computadores, 3° ed, Editora Campus

Price, A. M. A., Toscani, S. S., (2001) Implementação de Linguagens de Programação: Compiladores, Instituto de Informática da UFRGS, 2°ed Porto Alegre, Editora Sagra Luzzato.

Instituto de Matemática e Estatística da Universidade de São Paulo – USP, Projeto de algoritmos com Pilhas disponível em http://www.ime.usp.br/~pf/algoritmos/aulas/pilha.html

Department of Computer Engineering and Industrial Automation (DCA) at the School of Electrical and Computer Engineering (FEEC), State University of Campinas (Unicamp), Site sobre compiladores e estrutura de dados disponível em http://www.dca.fee.unicamp.br/cursos/EA876/apostila/HTML/node37.html, acesso em 25 de abril de 2008.

Wirth, N. (2005), Compiler Construction, livro em pdf disponível em http://www.oberon.ethz.ch/WirthPubl/CBEAll.pdf