Linguagem Tiny - Implementação por meio de orientação por objetos

Felipe Buzatti Nascimento , Leonardo Vilela Teixeira

[1]Departamento de Ciência da Computação
Universidade Federal de Minas Gerais (UFMG)

buzatti@dcc.ufmg.br , vilela@dcc.ufmg.br

Resumo. *Este trabalho produz um interpretador para a linguagem Tiny. Para construir o interpretador foi utilizada uma estrutura de classes seguindo o padrão de projetos interpretador definido em [Gamma et al. 1995].*

1. Estrutura de classes

A estrutura básica de classes consiste em três superclasses virtuais puras, a classe Comando, Expressao e ExpressaoBool, que declaram as três funções utilizadas no processo: Interpreta, Calcula e Avalia.

1.1. classe Comando:

A classe comando define a função Interpreta e é herdada pelas classes ComandoAtribuicao, ComandoFor, ComandoIf, ComandoRead, ComandoWhile, ComandoWrite.

```
public: virtual  Comando() ;
virtual void Interpreta( Contexto& C ) = 0;
```

Algoritmo 1: class Comando

1.2. classe Expressao:

Classe relacionada a expressões aritméticas, define a função Calcula. É herdada pelas classes ExpressaoAritmetica e Fator

```
public: virtual  Expressao() ;
virtual double Calcula(Contexto& C) = 0;
```

Algoritmo 2: class Expressao

1.3. classe ExpressaoBool:

Classe relacionada a expressões booleanas, define a função Avalia. É herdada pelas classes ExpressaoBooleana e ExpressaoRelacional.

```
public: virtual  ExpressaoBool() ;
virtual bool Avalia (Contexto& C) = 0;
```

Algoritmo 3: class ExpressaoBool

```
private: std::vector<double> variaveis;
public: Contexto();
 Contexto();
double obtemVariavel(char nomeVar);
void defineVariavel(char nomeVar, double valor);
```

Algoritmo 4: class Contexto

1.4. classe Contexto:

Classe responsável pela estrutura que armazena e fornece acesso às variáveis.

1.5. classe ListaComandos:

Classe responsável pela estrutura de lista de comandos, que armazena os comandos do programa.

```
private: std::vector<Comando *> lista_de_comandos;
public: ListaComandos(Comando * C);
 ListaComandos();
void AdicionaComando(Comando * C);
void Interpreta(Contexto& C);
```

Algoritmo 5: class ListaComandos

2. Algoritmos

As três funções principais são Interpreta, Avalia e Calcula. Essas são definidas em cada classe de uma forma diferente, dependendo da classe. A aplicação dessas funções se dá através de atribuição polimórfica, com essas funções sendo chamadas para objetos declarados como Comando, ExpressaoBooleana ou Expressao. Como trata-se de funções virtuais, é chamada a função da classe à qual pertence o objeto apontado.

2.1. virtual void Interpreta(Contexto& C)

Função típica de comandos. Executa o comando relativo à classe. Por exemplo, na classe ComandoWhile, essa função testa o condicional, e executa a lista de comandos recebida pela classe enquanto a condição for verdadeira. Já na classe ComandoAtribuicao a função recebe o nome da variável e o valor a ser atribuído e através das funções definidas pela classe Contexto modifica o valor da variável.

2.2. virtual double Calcula(Contexto& C)

Função relativa a expressões aritméticas. Faz acesso a variáveis, soma, subtrai, multiplica, ou faz raiz quadrada, dependendo da operação definida pelo objeto. Retorna o resultado do cálculo como um double.

2.3. virtual bool Avalia (Contexto& C)

Função típica de classes que definem expressões booleanas e relacionais. Realiza comparações, AND's, OR's, etc. O resultado dessas operações é um booleano.

3. IMPLEMENTAÇÃO

Cada classe está implementada em uma dupla de *<nome da classe>.h* e *<nome da classe>.cpp*. Há ainda um arquivo *parser.yy* usado pelo analisador sintático bison, responsável por produzir a partir desse os arquivos *parser.h* e *parser.cpp*. Há ainda os arquivos *scanner.ll* e *scanner.h* relativos ao análisador léxico Flex, que gera a partir desses o arquivo *scanner.cpp*.

Encapsulando o parser gerado pela dupla Bison/Flex, há a classe Driver. Essa classe é responsável por instanciar os analisadores léxico e sintático, conectá-los, alimentá-los com a entrada ("Código Fonte"), armazenar o resultado do processo de parsing e tratar erros de parsing encontrados no processo. A classe fornece os métodos: parse_stream, parse_string e parse_file, que iniciam o processo de parsing através de um stream, uma string ou um arquivo, respectivamente.

4. Compilação

O programa deve ser compilado pelo compilador GCC através de um makefile na pasta raiz.

5. Execução

O programa pode receber por parâmetro o nome do arquivo-fonte cujo código deve ser interpretado. Se não for informado um arquivo, o interpretador irá interpretar o código passado pela entrada padrão.

5.1. Formato da entrada

A entrada para o programa é um código fonte na sintaxe da linguagem tiny. A gramática da linguagem se encontra num arquivo junto ao código fonte do interpretado. Um exemplo de programa válido seria:

```
read(a);
read(b);

while( a > 0 ) do
    c := c + b;
    a := a - 1;
endw;

writeVar(c);
writeln;

endp
```

Algoritmo 6: Programa válido em tiny

5.2. Formato da saída

A saída do interpretador tiny será a saída gerada pelo programa fonte interpretado, ou mensagens de erro e de requisição de entrada, em caso de erro ou interpretação de um programa fornecido pela entrada padrão, respectivamente.

Referências

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Publishing Company.