Programação Orientada a Objetos [BCC221]

Atividade Prática

Implementando uma linguagem de programação super simples

Departamento de Computação
Universidade Federal de Ouro Preto
Prof. José Romildo Malaquias
29 de junho de 2015

Resumo

Esta atividade propõe a implementação de um interpretador e um compilador para uma linguagem de programação bastante simples, utilizando os mecanismos de herança, polimorfismo e ligação dinâmica no paradigma orientado a objetos.

A implementação pode ser feita usando a linguagem orientada a objetos de sua preferência.

O trabalho poderá ser desenvolvido em duplas.

A implementação deverá ser submetido pelo moodle até o dia 03/07/2015 (sexta-feira).

Sumário

1 A linguagem L0

Considere uma linguagem de programação **L0** com as seguintes construções, aqui expressas informalmente usando uma sintaxe concreta:

- O único **tipo** de dados é **inteiro**.
- Uma **variável** (isto é, uma variável inteira) tem um nome, que é um identificador formado por uma sequência de letras, dígitos decimais e sublinhados, começando com uma letra.
- Uma **constante** é um número inteiro negativo, nulo ou positivo.
- Uma **expressão** pode ser:
  - uma **variável**, cujo valor é obtido consultando a memória; caso a variável não esteja na memória, o seu valor é zero.
  - uma **constante**, cujo valor é a própria constante.

  - uma **operação binária**, formada por duas expressões conectadas por um operador, com a semântica esperada para as expressões aritméticas, relacionais e lógicas; o seu valor é obtido aplicando-se a operação indicada aos valores dos dois operandos.

- Os **operadores** (todos binários) são:
  - aritméticos: `+`, `-`, `*`, `/`, `%`
  - relacionais: `=`, `<>`, `<`, `<=`, `>`, `>=`
  - lógicos: `&&`, `||`

- Um **comando** pode ser:
  - o **comando nulo**, da forma `skip`, que não tem nenhum efeito
  - o **comando de leitura**, da forma `read x`, que especifica uma variável e cuja execução consiste em extrais a próxima palavra da entrada padrão, convertendo-a para um número inteiro, e armazenando-o na variável especificada
  - o **comando de impressão**, da forma `print e`, cuja execução consiste em avaliar a expressão `e` e inserir um texto que represesnta o seu valor na saída padrão
  - o **comando de atribuição**, da forma `x := e`, onde `x` é uma variável e `e` é uma expressão, cuja execução consiste em avaliar a expressão `e` e armazenar o seu valor na variável `x`
  - o **comando condicional**, da forma `if e then c1 else c2` onde `e` é uma expressão, e `c1` e `c2` são comandos; sua execução consiste em avaliar a expressão `e` e, se o seu valor for diferente de zero (considerado verdadeiro), executar o comando `c1`, ou, se o seu valor for igual a zero (considerado falso), executar o comando `c2`
  - o **comando de repetição**, da forma `from c1 until e loop c2` onde `c1` e `c2` são comandos e `e` é uma expressão; sua execução consiste em executar o comando `c1` e repetir a execução do comando `c2` enquanto o valor da expressão `e` for diferente de zero (considerado verdadeiro).
  - o **comando composto**, formado por uma sequência de comandos separados por `;`, precedida de `begin` e seguida de `end`; sua execução consiste na execução sequencial dos comandos que a constituem.

- Um **programa** é um comando.

Exemplo: programa em **L0** que lê dois números e exibe a soma dos mesmos:

```
begin
  read num1;
  read num2;
  print num1 + num2
end
```

Exercise 1

Usando esta sintaxe concreta, escreva um programa em **L0** para ler dois inteiros (não negativos), e calcular e exibir o seu máximo divisor comum aplicando o algoritmo de Euclides, sem usar multiplicação.
Uma implementação deste algoritmo em C++ é apresentado a seguir:

```
#include <iostream>
using namespace std;
int main()
{
   int a, b;
   cin >> a >> b;
   while (b != 0)
   {
      int resto = a % b;
      a = b;
      b = resto;
   }
   cout << a;
   return 0;
}
```

2 Sintaxe abstrata

A sintaxe abstrata é uma estrutura de dados hierárquica usada para representar um programa internamente no interpretador ou no compilador da linguagem.

Exercise 2

Defina um conjunto de classes para representar a sintaxe abstrata de um programa em **L0**. Use herança para criar uma hierarquia entre as classes. Implemente as seguintes classes com os construtores abaixo especificados (usando a notação da linguagem Java).

```
public enum Op { ADD, SUB, MUL, DIV, MOD,
                 EQ, NE, LT, LE, GT, GE,
                 AND, OR }
public abstract class Expressao
public class ExpNum extends Expressao
  ExpNum(int num)
public class ExpVar extends Expressao
  ExpVar(String identificador)
public class ExpOp extends Expressao
  ExpOp(Op op, Expressao a, Expressao b)

public abstract class Comando
public class ComSkip extends Comando
  ComSkip()
public class ComRead extends Comando
  ComRead(String x)
public class ComPrint extends Comando
  ComPrint(Expressao e)
public class ComAtrib extends Comando
  ComAtrib(String x, Expressao e)
public class ComIf extends Comando
  ComIf(Expressao e, Comando c1, Comando c2)
public class ComLoop extends Comando
  ComLoop(Comando c1, Expressao e, Comando c2)
public class ComComposto extends Comando
  // use Comando[] para representar a sequência de subcomandos
  ComComposto(Comando ... comandos)
  ComComposto(List<Comando> l)
```

A árvore de sintaxe abstrata do programa que lê dois números e exibe a soma dos mesmos pode ser construída em Java usando as classes apresentadas da seguinte maneira:

```
new ComComposto(
    new ComRead("num1"),
    new ComRead("num2"),
    new ComPrint(
      new ExpOp(
        Op.ADD,
        new ExpVar("num1"),
        new ExpVar("num2") )))
```

Exercise 3

Escreva um programa que produza uma árvore sintática abstrata representando o programa que calcula o máximo divisor comum que você escreveu anteriormente.

3 Interpetação

Exercise 4

Escreva um interpretador para a linguagem **L0**, ou seja, um programa que executa qualquer programa em **L0** dado como uma instância da classe `Comando`. Utilize um mapeamento para representar a memória.
A classe `Expressao` deverá ter um método para avaliar uma expressão. A memória deve ser passada como argumento para o método.
A classe `Comando` deverá ter um método para interpretar um comando. A memória deve ser passada como argumento para o método.
Na execução de um programa é necessário simular uma memória. Para tanto pode-se utilizar um mapeamento (estutura de dados) da biblioteca padrão. As chaves do mapeamento são os nomes das variáveis, e os valores associados às chaves são os valores das variáveis. Durante a execução de um programa assuma que o valor de uma variável que não conste do mapeamento seja zero.
Para executar um programa (que é um comando) acrescente um método abstrato na classe Comando. Este método deve receber a "memória"como argumento, e quando chamado, executa o programa. Implemente este método em cada subclasse de Comando de acordo com as regras específicas de execução para a forma de comando correspondente.
Eventualmente será necessário avaliar expressões quando o programa estiver sendo executado. Para tanto acrescente um método abstato na classe Expressao que recebe a "memória"como argumento, e resulta no valor calculado para a expressão. Implemente este método em cada subclasse de Expressao de acordo com as regras específicas de avaliação para a forma de expressão corresondente.
Utilize o interpretador de **L0** para executar os programas em **L0** que calcula a soma e que calcula o máximo divisor comum de dois números.