

Universidade Federal do Amazonas - UFAM
Instituto de Computação – IComp, Curso de Ciência da Computação
Compiladores
Projeto I: Calculadora Avançada (versão 1.0)

Objetivo

Neste projeto você deve escrever um interpretador de uma calculadora com suporte a funções do usuário. Embora simples, este projeto ilustra a construção do núcleo de uma linguagem dinâmica, uma vez que para suportar funções do usuário será necessário usar uma representação intermediária e um avaliador consciente de contextos de variáveis.

Descrição

A calculadora a ser implementada possui notação infixada, com suporte aos seguintes comandos, constantes e expressões:

Comandos, constantes e expressões	Definição
def ident = expressao	Define <i>ident</i> como expressão. Ex: def x = 10
def funcao(parametros) = expressao	Define uma função do usuário. Ex: def max(a, b) = if a>b then a else b
if ExpCond then ThenExp else ElseExp	Expressão if retorna ThenExp se ExpCond é avaliada como verdadeira. Caso contrário, retorna ElseExp. Ex: if n = 0 then 1 else if n = 1 then 2 else max(a,b)
+ - * / %	Soma, subtração, multiplicação, divisão e resto
> < = >= <= !=	Operadores relacionais >, <, =, >=, <= e != (diferente)
pi, e	Constantes π e de Euler (e)
sqrt, exp e ln	Raiz quadrada, exponencial e logaritmo natural

A seguir temos uma especificação formal, em EBNF, da linguagem a ser suportada:

Tokens:

```
number = digit {digit} [ '.' digit {digit} ].  
ident = letter {digit | letter }.
```

Regras de Produção:

```
AdvCalc = { Def | Exp } EOF.  
  
Def = "def" ident [ "(" ParamNames ")" ] "=" Exp.  
ParamNames = ident { "," ident }.  
  
Exp = T { ( "+" T | "-" T ) }.  
T = U { ( "*" U | "/" U | "%" U ) }.  
U = ( "-" F | F ).  
F = number | VarOrFunc | IFExp | "(" Exp ")" .  
  
VarOrFunc = ident [ "(" Params ")" ].  
Params = Exp { "," Exp }.  
  
IFExp = "if" ExpL "then" Exp "else" Exp.  
ExpL = Exp OpRel Exp.  
OpRel = ">" | ">=" | "<" | "<=" | "=" | "!=".
```

Note, pela especificação que funções primitivas (*sqrt*, *log*, etc) serão tratadas como funções do usuário. Assim, elas não aparecem na gramática dada. A seguir temos exemplos de interações com a calculadora:

Calculadora Avancada (digite 's' para sair)

Exemplos de expressões: ");

- 2+5-3*4

- 3*(1+4)

- def abs(n) = if n>=0 then n else -n

- def fat(n) = if n=0 then 1 else n*fat(n-1)

- fat(5) + abs(-10)

? def abs(x) = if x>0 then x else -x

function [x] [if, [>, x, 0], x, [*, -1, x]]

? def fat(n) = if n=0 then 1 else n*fat(n-1)

function [n] [if, [=, n, 0], 1, [*, n, [fat, [-, n, 1]]]]

? fat(5)*abs(-4)

480

? def x=x+1

Exception: x not found!

? def x=0

0

? def x=x+1

1

? x

1

? def max(a,b) = if a>b then a else b

function [a, b] [if, [>, a, b], a, b]

? def q(n)=n*n

function [n] [*, n, n]

? def cubo(n) = n*q(n)

function [n] [*, n, [q, n]]

? cubo(3)

27

? pi

3.1415927

? e

2.7182817

? exp(5)

148.41316

? ln(10)

2.3025851

? sqrt(8)

2.828427

? def pow(n, m) = if m = 0 then 1 else n*pow(n, m-1)

function [n, m] [if, [=, m, 0], 1, [*, n, [pow, n, [-, m, 1]]]]

? def area(raio) = raio * pow(pi, 2)

function [raio] [*, raio, [pow, pi, 2]]

? area(10)

98.69605

? def log(n, b) = ln(n)/ln(b)

function [n, b] [/ , [ln, n], [ln, b]]

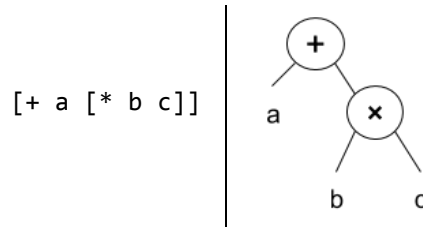
? log(8, 2)

3

? s

Como se pode notar, a calculadora suporta expressões infixadas com ordem de precedência

tradicional, definição de variáveis e definição de funções. Como este interpretador permite definições de funções, a estratégia de implementação mais adequada é uma tradução da notação infixada para uma árvore semântica abstrata (AST, do *inglês abstract syntactic tree*), seguida da interpretação da AST por um avaliador. Por exemplo, a expressão $a + b * c$ deveria ser traduzida para a seguinte AST (denotada por uma lista e uma árvore):



O avaliador, com base nas definições existentes no ambiente global, pode então avaliar a expressão. No caso dado, a raiz $+$ indica que uma soma deve ser realizada com o valor dos filhos a e $[* b c]$. *Recursivamente*, é necessário avaliar $[* b c]$ que se trata de uma multiplicação entre os elementos b e c , cujos valores devem ter sido previamente definidos no contexto global. De posse do valor de $[* b c]$, é possível somá-lo ao valor de a , resolvendo a expressão completa.

A vantagem de usar ASTs em lugar de uma interpretação direta é a possibilidade de armazenar uma AST para interpretação posterior. Por exemplo, dada a expressão:

```
def max(a, b) = if a>b then a else b
```

A AST $[a, b] [if, [>, a, b], a, b]$ pode ser armazenada, associada ao identificador *max*. Assim, por exemplo, ao ser invocado *max*(3, 5) em uma expressão posterior, a AST pode ser recuperada e avaliada em um contexto de referência onde a e b são variáveis locais iniciadas com os valores 3 e 5. Logo, um cuidado especial deve ser seguido com os ambientes de referência, como se vê no exemplo:

```
? def a = 100
  100
? def b = 200
  200
? def max(a,b) = if a>b then a else b
  function [a, b] [if, [>, a, b], a, b]
? max(1000, 500)
  1000
? a
  100
? b
  200
```

Note que mesmo que a seja menor que b no contexto global ($a = 100 < b = 200$), quando invocada com valores 1000 e 500, $a > b$, já que no contexto local, $a = 1000 > b = 500$. A execução de *max*, contudo, não afeta os valores globais de a e b que permanecem 100 e 200, após a sua execução. Em suma, dada uma AST, ela deve ser avaliada em seu contexto *local*.

Avaliação

O seu interpretador será avaliado pela gramática de atributos entregue, os códigos de apoio em Java e, fundamentalmente, pela execução de expressões de teste.

Observações

- (a) Trabalho feito por, no máximo, dois alunos.
- (b) Plágio não será tolerado, com anulação dos projetos envolvidos.
- (c) Em anexo, encontram-se a definição EBNF da linguagem dada, além de códigos mínimos para interface do usuário, para o interpretador e o AST.

ANEXOS

advcalc.atg

```
import java.lang.Exception;

COMPILER AdvCalc

Interpreter aci;

public void setInterpreter(Interpreter aci)
{
    this.aci = aci;
}

CHARACTERS
    digit = "0123456789".
    letter = 'A'..'Z' + 'a'..'z'.

TOKENS
    number = digit {digit} [ '.' digit {digit} ].
    ident = letter {digit | letter }.

IGNORE '\n' + '\r' + '\t'

PRODUCTIONS

AdvCalc          (. ASTExp exp; .)
= { CExp<out exp> (. try {
    ASTExp res = aci.eval(exp);
    System.out.println(" " + res);
} catch (Exception e) {
    System.err.println(" Error: "
        + e.getMessage());
    }
    .)
    } EOF
.

CExp<out ASTExp v>    (. v = null; .)
= Def | Exp.

Def = "def" ident ( "(" ParamNames ")" | ) "=" Exp.
ParamNames = ident { "," ident }.

Exp = T { ( "+" T | "-" T ) }.
T = U { ( "*" U | "/" U | "%" U ) }.
U = ("-" F | F ).
F = number | VarOrFunc | IFExp | "(" Exp ")" .

VarOrFunc = ident ( "(" Params ")" | ).
Params = Exp { "," Exp }.

IFExp = "if" ExpL "then" Exp "else" Exp.
ExpL = Exp OpRel Exp.
OpRel = ">" | ">=" | "<" | "<=" | "=" | "!=".

END AdvCalc.
```

Interpreter.java

```
import java.lang.Exception;

class ASTExp
{
}

class Interpreter
{
    public ASTExp eval(ASTExp exp) throws Exception
    {
        return null;
    }
}
```

Interpret.java

```
import java.io.*;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Interpret
{
    public static void main(String argv[])
    {
        Interpreter aci = new Interpreter();

        String str;
        System.out.println("Calculadora Avancada (digite 's' para sair)");
        System.out.println("Exemplos de expressoes: ");
        System.out.println("- 2+5-3*4");
        System.out.println("- 3*(1+4)");
        System.out.println("- def abs(n) = if n>=0 then n else -n");
        System.out.println("- def fat(n) = if n=0 then 1 else n*fat(n-1)");
        System.out.println("- fat(5) + abs(-10)\n");
        while (true) {
            try {
                System.out.print("? ");
                BufferedReader bufferRead = new
                    BufferedReader(new InputStreamReader(System.in));
                str = bufferRead.readLine();
                if (str.equalsIgnoreCase("s"))
                    System.exit(0);
                InputStream stream = new
                    ByteArrayInputStream(str.getBytes("UTF-8"));
                Scanner s = new Scanner(stream);
                Parser p = new Parser(s);
                p.setInterpreter(aci);
                p.Parse();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```