Construção de Compiladores I
Atividade: Mini-Linguagem

Bernardo Marotta e Melina Lopes

4 de Abril de 2018

Resumo

A proposta dessa atividade é especificar uma mini linguagem de programação simples que aborde alguns dos conceitos apresentados em sala de aula.

1 Sumário

2 A linguagem

A linguagem **Lhama** (Language of Heuristic Algorithms and Multithread Architecture) é uma mini-linguagem construída na disciplina de Construção de Compiladores I. É uma linguagem imperativa simples e de tipagem estática. Na linguagem, existem tipos primitivos com inteiro e ponto flutuante e tipos compostos, como arrays e estruturas. Existem também operadores aritméticos, relacionais, lógicos, espresses etc.

3 Aspectos léxicos

Comentários de linha: Começam com o caracter # e comentam a linha inteira
Comentários de bloco: Abre com ++[e fecha com]++
Caracteres brancos: Não interferem no interpretador léxico (servem de separador)
Literais inteiros: Sequência de dígitos decimais
Literais booleanos: true e false
Literais string: Delimitado por aspas duplas , dentro da string, pode ser usado qualquer caracter especial, usando $ para começar e terminar a sequência a ser escapada.

Identificadores: Sequência de letras e/ou dígitos. Pode conter alguns caracteres especiais como _, -, &, etc.

4 Aspectos sintáticos

A fim de mostrar os aspectos sintáticos, será apresentada uma GLC que define a sintaxe das construções da linguagem.

Declaração de funções:
Program → Functions
Functions → Function
Functions → Function Functions
Function → Type (Types): Exp end
Type → bool id
Type → int id
Type → string id
Type → float id
Types → Type TypeRest
TypeRest → , Type TypeRest
TypeRest → λ

Tipos literais:
Exp → id
Exp → id = Exp
Exp → Lit
Exp → Type[Lit]
Lit → lint
Lit → lstring
Lit → lbool
Lit → lfloat
Lit → λ
Lits → Lit LitRest
LitRest → , Lit LitRest
LitRest → λ

Operações aritméticas:
Exp → Exp + Exp
Exp → Exp - Exp
Exp → Exp * Exp
Exp → Exp % Exp
Exp → Exp / Exp

Operações relacionais:
Exp → Exp = Exp

$Exp \rightarrow Exp > Exp$
$Exp \rightarrow Exp < Exp$
$Exp \rightarrow Exp <= Exp$
$Exp \rightarrow Exp >= Exp$

Operações lógicas:
$Exp \rightarrow Exp$ and Exp
$Exp \rightarrow Exp$ or Exp

Chamada:
$Exp \rightarrow$ id (Exp)

Expressões:
$Exp \rightarrow$ if Exp : Exp else Exp end
$Exp \rightarrow$ while Exp : end
$Exp \rightarrow$ for Exp in Exp : end
$Exp \rightarrow$ for Exp , Exp : end
$Exp \rightarrow Exps$
$Exps \rightarrow Exp\ ExpRest$
$Exps \rightarrow$, Exp ExpRest
$Exps \rightarrow$ + Exp ExpRest
$Exps \rightarrow \lambda$

Precedência:
*,/,%
+, -
=, >, <, >=, <=
and
or
else, then, :

5 Aspectos semânticos

Atribuição:
A atribuição é feita através do operador =.
Ex: a = 5

Operadores:
O uso de operadores é feito de forma simples, com as expressões dos dois lados do operador.
Ex: a < 5, a > 5, a > 5 and b < 10

Declaração:
A declaraç ao de variáveis ou funções segue o padrão:

Ex: a = 5,
a[] = [1,2,3]
a[1] = [0]
bool function(int)

Condicionais:
Sendo c1 uma sequência de comandos e c2 outra:
if (BOOL) : c1 end
else c2 end

Repetição:
while (BOOL):
c1
end

for (ITEM) in (LISTA):
c2
end

for (i) , (i2):
c3
end

6 Biblioteca padrão

A id das funções da biblioteca padrão começam com lh.
Exemplo: lh.getLine(Exp) Guarda a entrada do teclado na variável id (entre parênteses).
Outros exemplos:

lh.print(Exp)	Imprime na tela o conteúdo
lh.pow(Exp, Exp)	Calcula potência
lh.fibonacci(Exp)	Calcula fibonacci
lh.factorial(Exp)	Calcula fatorial

7 Exemplos

Função Fibonacci em Lhama Language:

```
function fibonacci(int n):
    a = 0
```

```
    b = 1
    for i = 1, n:
      a = b
      b = a + b
    end
    return a
end

function insertionSort(int array[], int n):
    int i, key, j;
    for 1, n:
      key = arr[i]
      j = i-1
      while (j >= 0 and arr[j] > key)
        arr[j+1] = arr[j]
        j = j-1
      end
      arr[j+1] = key
    end
end

int main():
    int a
    lh.getLine(a)
    fibonacci(a)
    lh.print("The fibonacci of a is", a)
end
```