Construção de Compiladores

Luís Felipe
Vitor Lopes

Descrição

Nesta atividade vamos descrever uma pequena linguagem de programação, sendo uma variação da micro linguagem *straight-line*.

Neste roteiro vamos considerar uma linguagem, que chamaremos de Linguagem X, e baseada nessa linguagem vamos incluir:

- comentários de linha;
- comentários de bloco;
- literais booleanos, inteiros e string;
- os tipos booleano (bool), inteiro (int) e string;
- operadores aritméticos: - (simétrico e subtração), * (multiplicação), / (divisão), e % (resto da divisão inteira);
- operadores relacionais: == (igual), != (diferente), > (maior que), >= (maior ou igual a), < (menor que), e <= (menor ou igual a);
- operadores lógicos: && (e lógico), e || (ou lógico);

Os operadores relacionais '==', '!=', '>', '<', '>=' e '<=' tem precedência menor do que os demais operadores, e não são associativos.

Os operadores aritméticos '*' e '/' tem a maior precedência, enquanto que '+' e '-' tem precedência intermediária

A linguagem *straight-line* é descrita por Appel no final do capítulo 1.

Estrutura Léxica

Comentários:

Linha: começam com //.

Bloco: começam com /* e terminam com */.

Literais:

Inteiros: são formados pela sequência de um ou mais dígitos numéricos.

String: são formados por uma sequência de caracteres delimitada por aspas duplas.

Booleanos: apenas podem ser **true** ou **false**.

Identificadores: são sequências de letras maiúsculas ou minúsculas, dígitos decimais e sublinhados (underline) obrigatoriamente iniciando-se com uma letra minúscula.

Operadores: são os símbolos que significam uma expressão matemática relacional, aritmética ou lógica.

Gramática

Fun → TypeId (TypeIds) = Exp	declaração de função
TypeId → bool id	tipo booleano
TypeId → int id	tipo inteiro
TypeId → string id	tipo string
Exp → id	identificador (variável)
Exp → id := Exp	atribuição
Exp → litbool	literal booleano
Exp → litint	literal inteiro
Exp → litstring	literal string
Exp → Exp + Exp	operação aritmética soma
Exp → + Exp	operação aritmética positivo
Exp → Exp – Exp	operação aritmética subtração
Exp → – Exp	operação aritmética negação
Exp → Exp * Exp	operação aritmética multiplicação
Exp → Exp / Exp	operação aritmética divisão
Exp → Exp % Exp	operação aritmética resto da divisão

Gramática

Exp → Exp == Exp	**operação relacional igual**
Exp → Exp != Exp	**operação relacional diferente**
Exp → Exp > Exp	**operação relacional maior**
Exp → Exp >= Exp	**operação relacional maior ou igual**
Exp → Exp < Exp	**operação relacional menor**
Exp → Exp <= Exp	**operação relacional menor ou igual**
Exp → Exp && Exp	**operação lógica “e”**
Exp → Exp || Exp	**operação lógica “ou”**
Exp → Exps	**expressão sequência**
Exp → id (Exps)	**chamada de função**
Exp → if Exp then Exp else Exp	**expressão condicional**
Exp → while Exp do Exp	**expressão de repetição**
Exp → (Exps)	**expressão sequência**

Cada Exp é uma expressão que pode ser avaliada para produzir um comando ou valor, para no final ter algum efeito.

Exemplo

```
/*
    Exemplo de comentário
*/


int fibonacci (int n) =
    if  n < 2  than   n
    else  fibonacci(n-1) + fibonacci(n-2)
```

Bibliografia

[1] Andrew, W. Appel, and P. Jens. "Modern compiler implementation in Java." (2002).