Construção de Compiladores I [BCC328]

Atividade prática

Análise Léxica

Departarmento de Computação
Universidade Federal de Ouro Preto
César Olimpio Antunes Lima

Sumário

1. A linguagem

Este capítulo apresenta uma apresentação breve da linguagem proposta para implementação na matéria de Compiladores I (BCC328). A linguagem proposta irá contemplar os seguintes itens:

- os tipos inteiros, booleanos e *string*;
- literais inteiros;
- literais booleanos;
- literais *string*;
- definição de variáveis;
- definição de funções;
- operadores aritméticos: + (soma), - (simétrico e subtração), * (multiplicação), / (divisão) e % (resto da divisão inteira);
- operadores relacionais: == (igualdade), != (diferença), > (maior que), >= (maior ou igual a), < (menor que) e <= (menor ou igual a);
- operadores lógicos: and (e lógico) e or (ou lógico);
- atribuição;
- expressão de decisão;
- expressão de repetição;
- escopo de código;
- comentários de linha;
- comentários de bloco.

A linguagem será fortemente tipada e irá usar caracteres de formatação, como a quebra-de-linha e o caracter de identação (tab) para definir fim de expressões e contexto de código.

2. Gramática

Apresentamos a seguir uma gramática livre de contexto (com comentários) para a linguagem proposta, que define a sintaxe de todas as construções da linguagem.

Scope	→	*ExpFun ScopeRest*	definição de escopo
ScopeRest	→		
ScopeRest	→	\n *Scope ScopeRest*	
ExpFun	→	*Exp*	
ExpFun	→	*Fun*	
ExpFun	→	*Indents ExpFun*	indentação
Indents	→		
Indents	→	\t *Indents*	
Fun	→	*TypeId* (*TypeIds*): *Scope*	declaração de função
Exp	→	return *Exp*	retorno de função
TypeId	→	bool id	tipo booleano
TypeId	→	int id	tipo inteiro
TypeId	→	string id	tipo string
TypeIds	→		
TypeIds	→	*TypeId TypeIdsRest*	
TypeIdsRest	→		
TypeIdsRest	→	, *TypeId TypeIdsRest*	
Exp	→	litbool	literais
Exp	→	litint	
Exp	→	litstring	
Exp	→	*TypeId*	declaração de variável
Exp	→	*TypeId* = *Exp*	
Exp	→	id = *Exp*	atribuição
Exp	→	*Exp* + *Exp*	operações aritméticas
Exp	→	*Exp* - *Exp*	
Exp	→	*Exp* * *Exp*	
Exp	→	*Exp* / *Exp*	
Exp	→	-*Exp*	
Exp	→	*Exp* % *Exp*	resto da divisão
Exp	→	*Exp* == *Exp*	operações relacionais
Exp	→	*Exp* != *Exp*	
Exp	→	*Exp* > *Exp*	
Exp	→	*Exp* >= *Exp*	
Exp	→	*Exp* < *Exp*	

Exp	→	*Exp* <= *Exp*	
Exp	→	*Exp* and *Exp*	operações lógicas
Exp	→	*Exp* or *Exp*	
Exp	→	id (*ExpParams*)	chamada de função
Exp	→	if *Exp*: *Scope*	expressão condicional
Exp	→	if *Exp*: *Scope* else: *Scope*	
Exp	→	while *Exp*: *Scope*	expressões de repetição
Exp	→	for *Exp*; *Exp*; *Exp*: *Scope*	
ExpParams	→		parâmetros de função
ExpParams	→	*Exp ParamsRest*	
ParamsRest	→		
ParamsRest	→	, *Exp ParamsRest*	

3. Aspectos léxicos

Comentários de linha começam com o caracter # e se estendem até o final da linha. Comentários de bloco são delimitados pelas sequências de caracteres /* e */ e podem ser aninhados.

Ocorrências de caracteres de espaço e comentários entre os símbolos léxicos são ignorados, servindo apenas para separar os símbolos léxicos. Os caracteres de quebra-de-linha são usados para diferenciar expressões. Caracterres de tabulação horizontal são usados para identificar o escopo ao qual a expressão pertence.

Os literais inteiros são formados por uma sequência de um ou mais digitos decimais. Os literais booleanos são true (verdadeiro) ou false (falso). Os literais string são formados por uma sequência de caracteres gráficos delimitada por aspas duplas (“”). Na sequência de caracteres, o caracter \ é especial e indica uma sequência de escape. As únicas sequências de escape válidas são indicadas na tabela a seguir.

4. Documentação

4.1. Primitivas

O tipo int

Armazena um valor numérico inteiro. Uma variável do tipo int é automaticamente inicializada com o valor 0.

```
int a = 3
print(a) # 3
```

O tipo bool

Armazena um valor booleano, sendo true (verdadeiro) ou false (falso). Uma variável do tipo bool é automaticamente inicializada com o valor false.

```
bool a = true
print(a) # true
```

O tipo string

Armazena uma cadeia de caracteres. Uma variável do tipo string é automaticamente inicializada com o valor “” (string vazia).

```
string a = “hello world”
print(a) # hello world
```

O tipo void

Uma função obrigatóriamente deve retornar um valor, a não ser que ela seja do tipo void. Nesse caso, não é necessário que a função tenha um retorno. Essa função obrigatoriamente irá retornar um valor do tipo void.

```
void _print (string text):
   print(text)
_print(“hello world”)
```

4.2. Estruturas de decisão

Estrutura if

Executa um trecho de código caso uma expressão dada dê verdadeiro.

```
string number = 1
```

```
if number == 1:
    print(“success”) # success
```

Estrutura if-else

Executa um trecho de código caso uma expressão dada dê verdadeiro, e executa um trecho de código caso contrário.

```
string number = 1
if number == 2:
    print(“success”)
else:
    print(“failure”) # failure
```

4.3. Estruturas de repetição

Estrutura while

Executa um trecho de código em repetição enquanto a expressão dada der verdadeiro.

```
int number = 5
while (number > 0):
    print(number)
    number = number - 1
# 5 4 3 2 1
```

Estrutura for

Executa um trecho de código em repetição enquanto a expressão dada der verdadeiro. Além disso, executa uma expressão no inicio da estrutura, e uma expressão a cada execução da instrução de repetição.

```
for int i = 5; i > 0; i = i - 1:
    print(i)
# 5 4 3 2 1
```

4.4. Funções

Funções podem ser declaradas em qualquer escopo. Uma função existe dentro do escopo em que ela foi declarada e em seus escopos filhos. Uma função deve possuir um tipo e devem, obrigatóriamente, retornar um valor daquele tipo, a não ser que ela seja do tipo void. Quando o método return for chamado, a execução do código da função é encerrado, e o valor da expressão do return é retornado.

```
int sum (int a, int b):
    int c = a + b
    return c
    print(c) # não alcançado
int res = sum (2, 3)
print(res) # 5
```

4.5. Escopos de código

Escopos de código são regiões de código definidas por sua identação. Variáveis e funções declaradas dentro de um escopo só podem ser acessadas pelo escopo que elas pertencem e por escopos internos a ela.

O aumento no nivel de identação de código (representado pelo caracter especial \t) representa o início de um novo escopo, e o recuo da identação de código representa o fim daquele escopo. As funções, tais como as estruturas if, if-else, while e for exigem a definição de um escopo interno após a declaração dos mesmos.

5. Exemplos

Fibonacci

```
int fib(int n):
   if n == 1 or n == 2:
      return 1
   return fib(n-1) + fib(n-2)
int fib5 = fib(5) # 5
```

Fatorial

```
int fat(int n):
   if n == 1:
      return 1
   return n*fat(n-1)
int fat5 = fat(5) # 120
```

Par-ímpar

```
bool par(int n):
    return n%2 == 0
bool impar(int n):
    return par(n-1)
par(2) # true
par(3) # false
impar(4) # false
impar(5) # true
```