Construção de Compiladores I [BCC328]

Atividade Prática

Análise Léxica da Linguagem *Torben*

Departamento de Computação
Universidade Federal de Ouro Preto
Prof. José Romildo Malaquias
27 de outubro de 2015

Resumo

Nesta atividade vamos implementar um analisar léxico para uma pequena linguagem de programação.

Sumário

1 A linguagem *Torben* **1**

2 Aspectos léxicos **3**

3 Atividade: análise léxica **3**

1 A linguagem *Torben*

Torben Ægidius Mogensen apresenta no capítulo 4 de seu livro *Introduction to Compiler Design* uma pequena linguagem de programação para fins didáticos.

Nas atividades propostas neste roteiro vamos considerar uma linguagem, que chamaremos de *Torben*, baseada nessa linguagem, porém estendida para incluir:

- comentários de linha,
- comentários de bloco,
- literais booleanos,
- o tipo `string`,
- literais strings,
- operadores aritméticos: `-` (simétrico e subtração), `*` (multiplicação), e `/` (divisão),
- operadores relacionais: `<>` (diferente), `>` (maior que), `>=` (maiour ou igual a), `<` (menor que), e `<=` (menor ou igual a),
- operadores lógicos: `&&` (e lógico), e `||` (ou lógico).
- atribuição,
- expressão de repetição, e
- expressão sequência.

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

Program	→	*Funs*	programa
Funs	→	*Fun*	
Funs	→	*Fun* *Funs*	
Fun	→	*TypeId* `(` *TypeIds* `)` `=` *Exp*	declaração de função
TypeId	→	`bool` `id`	tipo booleano
TypeId	→	`int` `id`	tipo inteiro
TypeId	→	`string` `id`	tipo string
TypeIds	→		lista de parâmetros
TypeIds	→	*TypeId* *TypeIdsRest*	
TypeIdsRest	→		
TypeIdsRest	→	`,` *TypeId* *TypeIdsRest*	
Exp	→	`litbool`	literais
Exp	→	`litnum`	
Exp	→	`litstring`	
Exp	→	`id`	variável
Exp	→	`id` `:=` *Exp*	atribuição
Exp	→	*Exp* `+` *Exp*	operações aritméticas
Exp	→	*Exp* `-` *Exp*	
Exp	→	*Exp* `*` *Exp*	
Exp	→	*Exp* `/` *Exp*	
Exp	→	*Exp* `%` *Exp*	
Exp	→	`-` *Exp*	
Exp	→	*Exp* `=` *Exp*	operações relacionais
Exp	→	*Exp* `<>` *Exp*	
Exp	→	*Exp* `>` *Exp*	
Exp	→	*Exp* `>=` *Exp*	
Exp	→	*Exp* `<` *Exp*	
Exp	→	*Exp* `<=` *Exp*	
Exp	→	*Exp* `&&` *Exp*	operações lógicas
Exp	→	*Exp* `||` *Exp*	
Exp	→	`id` `(` *Exps* `)`	chamada de função
Exp	→	`if` *Exp* `then` *Exp* `else` *Exp*	expressão condicional
Exp	→	`while` *Exp* `do` *Exp*	expressão de repetição
Exp	→	`let` `id` = *Exp* `in` *Exp*	expressão de declaração
Exp	→	`(` *Exps* `)`	expressão sequência
Exps	→		
Exps	→	*Exp* *ExpsRest*	
ExpsRest	→		
ExpsRest	→	`,` *Exp* *ExpsRest*	

A precedência relativa e a associatividade dos operadores é indicada pela tabela a seguir, em ordem decrescente de precedência.

operadores	**associatividade**
`-` (unário)	
`*`, `/`, `%`	esquerda
`+`, `-` (binário)	esquerda
`=`, `<>`, `>`, `>=`, `<`, `<=`	
`&&`	esquerda
`||`	esquerda
`:=`	direita
`then`, `else`, `do`, `in`	direita

Observe que um programa em *Torben* é uma sequência de declarações de funções.

Um programa deve definir uma função sem argumentos chamada `main` que resulta em um inteiro. A execução do programa inicia-se pela chamada desta função `main`.

2 Aspectos léxicos

Comentários de linha em *Torben* começam com os caracteres `//` 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 brancos** (espaço, tabulação horizontal e nova linha) e comentários entre os símbolos léxicos são ignoradas, servindo apenas para separar símbolos léxicos.

Os **literais inteiros** são formados por uma sequência de um ou mais dígitos decimais.

Os **literais booleanos** são `#t` (verdadeiro) e `#f` (falso).

Os **literais string** são formados por uma sequência de caracteres gráficos delimitada por aspas (`"`). Na sequência de caracteres o caracter `\` é especial e inicia uma sequência de escape. As únicas sequências de escape válidas são indicados na tabela a seguir.

sequência de escape	**descrição**
`\\`	`\`
`\"`	`"`
`\t`	tabuação horizontal
`\n`	nova linha
`\r`	retorno de carro
`\b`	backspace
`\`*ddd*	caracter de código *ddd*, sendo *d* qualquer dígito decimal

Identificadores são sequências de letras maiúsculas ou minúsculas, dígitos decimais e sublinhados (`_`), começando com uma letra minúscula. Letras maiúsculas e minúsculas são distintas em um identificador.

3 Atividade: análise léxica

Tarefa 1: Implementação

Implemente um analisador léxico *ad hoc* para a linguagem *Torben*.

A sua aplicação deverá aceitar o nome do arquivo a ser analisado como argumento na linha de comando, e exibir a sequência de *tokens* obtidas pela análise léxica do arquivo.

Para cada `token` o seu analisador léxico deverá informar:

- o tipo do `token`,
- o valor semântico do `token`, quando relevante, e
- a posição (número da linha e coluna) em que o `token` aparece no programa fonte.

Todos os possíveis erros léxicos devem ser reportados corretamente pelo seu analisador léxico. Ao reportar um erro, deve-se exibir a posição (linha e coluna) em que o erro foi detectado, e uma mensagem de diagnóstico.

Por exemplo, a análise léxica do programa fonte seguinte:

```
{# programa para cálculo do fatorial
   de um número inteiro #}

function fatorial(n: int): int =
  if n > 0 then
    1                           // caso base
  else
    n * fatorial(n-1)           // caso recursivo

var main: void :=
  print_int(fatorial(7))
```

produz uma lista de símbolos léxicos similar a:

```
fat.torben:  4.  0-  4.  8   FUNCTION
fat.torben:  4.  9-  4. 17   ID(fatorial)
fat.torben:  4. 17-  4. 18   LPAREN
fat.torben:  4. 18-  4. 19   ID(n)
fat.torben:  4. 19-  4. 20   COLON
fat.torben:  4. 21-  4. 24   ID(int)
fat.torben:  4. 24-  4. 25   RPAREN
fat.torben:  4. 25-  4. 26   COLON
fat.torben:  4. 27-  4. 30   ID(int)
fat.torben:  4. 31-  4. 32   EQ
fat.torben:  5.  2-  5.  4   IF
fat.torben:  5.  5-  5.  6   ID(n)
fat.torben:  5.  7-  5.  8   GT
fat.torben:  5.  9-  5. 10   LITINT(0)
fat.torben:  5. 11-  5. 15   THEN
fat.torben:  6.  4-  6.  5   LITINT(1)
fat.torben:  7.  2-  7.  6   ELSE
fat.torben:  8.  4-  8.  5   ID(n)
fat.torben:  8.  6-  8.  7   TIMES
fat.torben:  8.  8-  8. 16   ID(fatorial)
fat.torben:  8. 16-  8. 17   LPAREN
fat.torben:  8. 17-  8. 18   ID(n)
fat.torben:  8. 18-  8. 19   MINUS
fat.torben:  8. 19-  8. 20   LITINT(1)
fat.torben:  8. 20-  8. 21   RPAREN
fat.torben: 10.  0- 10.  3   VAR
fat.torben: 10.  4- 10.  8   ID(main)
fat.torben: 10.  8- 10.  9   COLON
fat.torben: 10. 10- 10. 14   ID(void)
fat.torben: 10. 15- 10. 17   ASSIGN
fat.torben: 11.  2- 11. 11   ID(print_int)
fat.torben: 11. 11- 11. 12   LPAREN
fat.torben: 11. 12- 11. 20   ID(fatorial)
fat.torben: 11. 20- 11. 21   LPAREN
fat.torben: 11. 21- 11. 22   LITINT(7)
fat.torben: 11. 22- 11. 23   RPAREN
fat.torben: 11. 23- 11. 24   RPAREN
fat.torben: 12.  0- 12.  0   EOF
```

Tarefa 2: Testes

Teste o seu analisador léxico de *Torben* com os seguintes programas fontes:

1. ```
   int fatorial(int n) =
     if n > 0 then
       1
     else
       n * fatorial(n-1)

   int main() =
     print_int(fatorial(7))
   ```
2. ```
   bool f(string s) =
     s = "pedro" || #t
   ```

3.
```
int main() =
  let a = 873 in
    let b = a ^ 3 in
      (a + b)/2
```

4.
```
int main() =
  // program execution starts here
  let PesoPessoa = 45 in
    print_int(PesoPessoa + 2)
```

5.
```
int main() =
  print(-2342,
        56.7,
        "Letra \064.",
        "Escape \k inválida",
        "aspas\"internas\" e \\ barra",
        "ouro \n preto",
        "bom dia)
```