Construção de Compiladores I [BCC328]

Atividade Prática

Análise Léxica da Linguagem *Torben*

Departamento de Computação
Universidade Federal de Ouro Preto
Prof. José Romildo Malaquias
14 de março de 2018

Resumo

Nesta atividade vamos implementar os analisadores léxico e sintático para uma pequena linguagem de programação.

Sumário

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), `/` (divisão), e `%` (resto da divisão inteira),
- 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	→	`litint`	
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*	**resto da divisão**
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 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 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 `true` (verdadeiro) e `false` (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
`\f`	avanço de formulário
`\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. Letras maiúsculas e minúsculas são distintas em um identificador.

3 Atividade: análise léxica

Tarefa 1: Implementação

Implemente um analisador léxico 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.

A estrutura do projeto está disponível, bastando completar as regras de análise léxica.

Por exemplo, a análise léxica do programa fonte seguinte:

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

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

int main() =
  printint(fatorial(7))
```

produz uma lista de símbolos léxicos similar a:

```
4:1-4:4 INT
4:5-4:13 ID(fatorial)
4:13-4:14 LPAREN
4:14-4:17 INT
4:18-4:19 ID(n)
4:19-4:20 RPAREN
4:21-4:22 EQ
5:3-5:5 IF
5:6-5:7 ID(n)
5:8-5:9 GT
5:10-5:11 LITINT(0)
5:12-5:16 THEN
6:5-6:6 LITINT(1)
7:3-7:7 ELSE
8:5-8:6 ID(n)
8:7-8:8 TIMES
8:9-8:17 ID(fatorial)
8:17-8:18 LPAREN
8:18-8:19 ID(n)
8:19-8:20 MINUS
8:20-8:21 LITINT(1)
8:21-8:22 RPAREN
10:1-10:4 INT
10:5-10:9 ID(main)
10:9-10:10 LPAREN
10:10-10:11 RPAREN
10:12-10:13 EQ
11:3-11:12 ID(print_int)
11:12-11:13 LPAREN
11:13-11:21 ID(fatorial)
11:21-11:22 LPAREN
11:22-11:23 LITINT(7)
11:23-11:24 RPAREN
11:24-11:25 RPAREN
12:1-12:1 EOF
```

Tarefa 2: Testes Automáticos

Ceritifique-se de que todos os testes implementados no projeto sucedam.

Tarefa 3: Testes

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

1. ```
   bool f(string s) =
     s = "pedro" || true
   ```
2. ```
   int main() =
     let a = 873 in
       let b = a ^ 3 in
         (a + b)/2
   ```
3. ```
   int main() =
     # program execution starts here
     let PesoPessoa = 45 in
       print_int(PesoPessoa + 2)
   ```
   ```

4.
```
int main() =
  print(-2342,
        56.7,
        "Letra \064.",
        "Escape \k inválida",
        "aspas\"internas\" e \\ barra",
        "ouro \n preto",
        "bom dia)
```

4 Aspectos semânticos