Construção de Compiladores I [BCC328]

Atividades Práticas

Compilação da linguagem *Panda*

Departamento de Computação
Universidade Federal de Ouro Preto
Prof. José Romildo Malaquias
4 de novembro de 2015

Resumo

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

Sumário

1 A linguagem *Panda*

Andrew Appel apresenta em seu livro *Modern Compiler Implementation in Java* (primeira edição) uma pequena linguagem de programação para fins didáticos chamada Tiger.

Vamos considerar uma linguagem, que chamaremos de *Panda*, baseada em Tiger, porém com algumas diferenças sintáticas e semânticas

2 Sintaxe

Apresentamos a seguir uma gramática livre de contexto para *Panda*, que define a sintaxe de todas as construções da linguagem.

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

Program	→	*Decs*	
Decs	→	*Dec*	
Decs	→	*Dec* *Decs*	
Dec	→	`type` `id` `=` *Type*	declaração de tipo
Dec	→	`var` `id` `:` `id` `=` *Exp*	declaração de variável
Dec	→	`var` `id` `=` *Exp*	
Dec	→	`function` `id` `(` *Params* `)` `:` `id` `=` *Exp*	declaração de função
Dec	→	`function` `id` `(` *Params* `)` `=` *Exp*	
Type	→	`id`	tipo nomeado
Type	→	`[` `id` `]`	tipo array
Type	→	`{` *Params* `}`	tipo registro
Param	→	`id` `:` `id`	
Params	→		
Params	→	*Param* *ParamsRest*	
ParamsRest	→		
ParamsRest	→	`,` *Param* *ParamsRest*	
Var	→	`id`	variável simples
Var	→	*Var* `[` *Exp* `]`	variável indexada
Var	→	*Var* `.` `id`	campo de registro
Exp	→	`nil`	nil
Exp	→	`litint`	literais
Exp	→	`litreal`	
Exp	→	`litbool`	
Exp	→	`litchar`	
Exp	→	`litstring`	
Exp	→	`@` `id` `{` fields `}`	expressão registro
Exp	→	`@` `id` `[` exps `]`	expressão array
Exp	→	*Var*	variável
Exp	→	`-` *Exp*	operações aritméticas
Exp	→	*Exp* `+` *Exp*	
Exp	→	*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	→	*Var* `:=` *Exp*	atribuição
Exp	→	`id` `(` *Exps* `)`	chamada de função
Exp	→	`if` *Exp* `then` *Exp* `else` *Exp*	expressões condicionais
Exp	→	`if` *Exp* `then` *Exp*	
Exp	→	`while` *Exp* `do` *Exp*	expressão de repetição
Exp	→	`break`	
Exp	→	`let` *Decs* `in` *Exp*	expressão de declaração
Exp	→	`(` *ExpSeq* `)`	expressão sequência
Exps	→		
Exps	→	*Exp* *ExpsRest*	
ExpsRest	→		
ExpsRest	→	`,` *Exp* *ExpsRest*	
ExpSeq	→		
ExpSeq	→	*Exp* *ExpSeqRest*	
ExpSeqRest	→		
ExpSeqRest	→	`;` *Exp* *ExpSeqRest*	

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

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

3 Aspectos léxicos

Ocorrências de **caracteres brancos** (espaço, tabulação horizontal, nova linha, e retorno de carro) e comentários entre os símbolos léxicos são ignorados.

Comentários de linha em *Panda* começam com o caractere `#` e se estendem até o final da linha. **Comentários de bloco** são delimitados pelas sequências de caracteres `{#` e `#}` e podem ser aninhados.

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

Os **literais reais** são formados por uma sequência de um ou mais dígitos decimais seguida do símbolo `.`, seguido de uma sequência de um ou mais dígitos decimais. Uma e somente uma das duas sequências de dígitos é opcional.

Os **literais booleanos** são `true` (verdadeiro) e `false` (falso).

Os **literais caracteres** são formados por um caracter gráfico ou uma sequência de escape, delimitado pelo símbolo `'`. As únicas sequências de escape válidas são as indicadas na tabela a seguir.

sequência de escape	**descrição**
`\\`	`\`
`\'`	`'`
`\t`	tabuação horizontal
`\n`	nova linha
`\r`	retorno de carro
`\b`	retrocesso
`\^c`	caracter de controle *c*, sendo *c* uma letra maiúscula, `@`, `[`, `\`, `]`, `^`, `_` ou `?`
`\`*`ddd`*	caracter de código *ddd*, sendo *ddd* uma sequências de 3 dígitos decimais

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 indicadas na tabela a seguir.

sequência de escape	**descrição**
`\\`	`\`
`\"`	`"`
`\t`	tabuação horizontal
`\n`	nova linha
`\r`	retorno de carro
`\b`	retrocesso
`\^c`	caracter de controle *c*, sendo *c* uma letra maiúscula, `@`, `[`, `\`, `]`, `^`, `_` ou `?`
`\`*`ddd`*	caracter de código *ddd*, sendo *ddd* uma sequências de 3 dígitos decimais

Os primeiros 32 caracteres na tabela ASCII são códigos de controle não imprimíveis e são usados para controlar periféricos, tais como impressoras. A tabela 1 lista todos os caracteres de controle ASCII.

Decimal	Hexadecimal	Abbreviation	Caret notation	Escape code	Name
0	00	NUL	`^@`	`\0`	Null character
1	01	SOH	`^A`		Start of Header
2	02	STX	`^B`		Start of Text
3	03	ETX	`^C`		End of Text
4	04	EOT	`^D`		End of Transmission
5	05	ENQ	`^E`		Enquiry
6	06	ACK	`^F`		Acknowledgment
7	07	BEL	`^G`	`\a`	Bell
8	08	BS	`^H`	`\b`	Backspace
9	09	HT	`^I`	`\t`	Horizontal Tab
10	0A	LF	`^J`	`\n`	Line feed
11	0B	VT	`^K`	`\v`	Vertical Tab
12	0C	FF	`^L`	`\f`	Form feed
13	0D	CR	`^M`	`\r`	Carriage return
14	0E	SO	`^N`		Shift Out
15	0F	SI	`^O`		Shift In
16	10	DLE	`^P`		Data Link Escape
17	11	DC1	`^Q`		Device Control 1 (oft. XON)
18	12	DC2	`^R`		Device Control 2
19	13	DC3	`^S`		Device Control 3 (oft. XOFF)
20	14	DC4	`^T`		Device Control 4
21	15	NAK	`^U`		Negative Acknowledgment
22	16	SYN	`^V`		Synchronous idle
23	17	ETB	`^W`		End of Transmission Block
24	18	CAN	`^X`		Cancel
25	19	EM	`^Y`		End of Medium
26	1A	SUB	`^Z`		Substitute
27	1B	ESC	`^[`	`\e`	Escape
28	1C	FS	`^\`		File Separator
29	1D	GS	`^]`		Group Separator
30	1E	RS	`^^`		Record Separator
31	1F	US	`^_`		Unit Separator
127	7F	DEL	`^?`		Delete

Tabela 1: Caracteres de controle da tabela ASCII.

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.

4 Atividade: análise léxica

Tarefa 1: Implementação

Implemente um analisador léxico usando um gerador de analisador léxico para a linguagem *Panda*.

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 a lista de símbolos léxicos:

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

Tarefa 2: Implementação

Elaborar um conjunto de testes e usá-los para testar o seu analisador léxico. O conjunto de testes deve contemplar todos os aspectos léxicos relevantes da linguagem *Panda*.