Relatório
Construção de Compiladores

Gerador de analisador semântico – Bison

Victor Hugo Cândido de Oliveira
RA 121041451

Objetivo

Desenvolvimento de um gerador de analisador semântico utilizando a ferramenta de análise sintátia/semântica GNU Bison em conjunto com a ferramenta de análise léxica Flex.

Linguagem

Algumas modificações foram necessárias para a implementação do terceiro projeto, entre elas estão:

-declarações múltiplas não são mais válidas, apenas uma declaração por comando.

-atribuições só podem ser feitas nos formatos:

```
<ID> = <expressão numérica ou valor absoluto>
<ID>++
<ID>--
<ID> = <ID >
```

ou seja, expressões no formato:

```
<ID> = <ID > <operador> <qualquer coisa>
```

(envolvendo elemento(s) não numérico) não serão aceitas.

-comandos de declaração e atribuição devem ser feitos em linhas separadas:

```
<tipo de dado> <ID>;
<ID> = <expressão, valor ou ID> ;
```

-modificações na estrutura para eliminar conflitos.

A linguagem L aceita pelo analisador desenvolvido será especificada pelas regras a seguir retiradas do arquivo parser.output gerado pelo Bison.

Os tokens reconhecidos são:

Delimitadores: '(' ')' ';' '{' '}' ','

Operadores: '*' '+' '-' '/'

Atribuição: '='

MAIN – token da função principal

IF – token “if”

ELSE – token “else”

DO – token “do”

WHILE – token “while”

FOR – token “for”

BREAK – token “break”

PRINT – token “print”

RETURN – token “return”

INT_TYPE – token “int”

FLOAT_TYPE – token “float”

STRING – strings utilizadas no comando “print”

INTEGER – número inteiro

REAL – número real

ID – token para identificadores e nome de funções

MATH_INC – incremento de 1 “++”

MATH_DEC – decremento de 1 “--”

LOG_EQL – operador lógico binário EQUIVALE A “==”

LOG_LT – operador lógico binário MENOR QUE “<”

LOG_GT – operador lógico binário MAIOR QUE “>”

LOG_AND – operador lógico binário E “&&”

LOG_OR – operador lógico binário OU “||”

LOG_NOT – operador lógico unário NÃO “!”

Código gerado pelo Bison (parser.output)

```
$accept: program $end
program: MAIN '(' ')' '{' commands '}' program_end
       | error
program_end: %empty
           | ID '(' ')' '{' commands '}' program_end
           | error
commands: command_list
command_list: '{' command_list '}'
            | command command_list
            | %empty
command: cmd ';'
       | stmt
       | error
stmt: if_stmt
    | for_stmt
    | while_stmt
    | dowhile_stmt
cmd: PRINT STRING
   | BREAK
   | RETURN
   | attrib
   | declaration
if_stmt: IF '(' l_expr ')' '{' commands '}' else_stmt
else_stmt: %empty
         | ELSE '{' commands '}'
for_stmt: FOR '(' ffirst ';' l_expr ';' fthird ')' '{' commands '}'
```

```
 ffirst: %empty
      | attrib_list
fthird: %empty
      | cmd ',' fthird
      | cmd
while_stmt: WHILE '(' l_expr ')' '{' commands '}'
dowhile_stmt: DO '{' commands '}' WHILE '(' l_expr ')' ';'
declaration: INT_TYPE ID
           | FLOAT_TYPE ID
attrib_list: attrib ',' attrib_list
           | attrib
attrib: i_attrib
      | r_attrib
      | ID MATH_INC
      | ID MATH_DEC
      | ID '=' ID
      | error
i_attrib: ID '=' i_expr
r_attrib: ID '=' r_expr
l_expr: l_expr LOG_EQL l_factor
      | l_expr LOG_AND l_factor
      | l_expr LOG_OR l_factor
      | l_expr LOG_GT l_factor
      | l_expr LOG_LT l_factor
      | LOG_NOT l_expr
      | l_factor
l_factor: '(' l_expr ')'
        | INTEGER
        | REAL
        | ID
i_expr: i_expr '+' i_term
      | i_expr '-' i_term
      | i_term
r_expr: r_expr '+' r_term
      | r_expr '-' r_term
      | r_term
i_term: i_term '*' i_factor
      | i_term '/' i_factor
      | i_factor
r_term: r_term '*' r_factor
      | r_term '/' r_factor
```

```
      | r_factor
i_factor: '(' i_expr ')'
        | INTEGER
r_factor: '(' r_expr ')'
        | REAL
```

Conteúdo

Seguem os seguintes arquivos:

- `scanner.l` – código do gerador de analisador léxico (Flex)
- `parser.y` – código do gerador de analisador sintático (Bison)
- `symboltable.h`/`symboltable.c` – tabela de símbolos
- `compila.sh` – código em shell script para compilar o scanner.l, o parser.y e gerar o executável exec com os códigos `lex.yy.c`, `parser.tab.c` e `symboltable.c`
- `input_scanner` – exemplo de código a ser analisado pela linguagem

Para compilar e executar

Execute o script compila.sh com o comando

```
./compila.sh
```

Execute o executável exec gerado pelo script e insira o código pelo próprio stdin

```
./exec
```

ou execute o código passando um arquivo como parâmetro (ex: `input_scanner`)

```
./exec input_scanner
```

Saída do analisador

O analisador acusará erros de sintaxe como falta de parênteses, chaves, ponto e vírgula, declarações incompletas, etc.

O analisador apontará erros sintáticos como:

-atribuição de tipos diferentes

-utilização de uma variável não declarada

-redeclaração de uma varíavel com tipo diferente ou não

E irá inserir a tabela de símbolos formada (com valores atuais, lexema e tipos) no arquivo "`symboltable`".

Caso existam erros o analisador os apontará aproximando o cursor da ocorrẽncia, caso contrário a análise será concluída com sucesso.