Languages and Compilers Project Report

CMPN 403

Project Members

Ahmed Osama

Adel Adham

Mahmoud Zidan

Mohamed Hossam

1. Project Overview

1.1. Introduction

Designed and implemented a compiler for a simple programming C-like language using the Lex and Yacc compiler generating packages.

1.2. Language Rules

Variables and Constants

Variables and constants' types are integer (int), decimals (float) and characters (char) and boolean (bool).

All variable names must begin with a letter of the alphabet or an underscore(_), after the first initial letter, variable names can also contain letters and numbers. No spaces or special characters, however, are allowed.

i.e. int _x9=3; const float num=2.1; char r ='r'; bool isTrue=false;

Mathematical and Logical expressions

The mathematical operations that are valid on all numerical types are addition, subtraction, multiplication and division (+,-,*,/)

The logical expressions that are valid on all numerical types are AND, OR and NOT (&&,||,!).

Conditions

Conditions include both if-else statements and switch-case statements the exact syntax is shown in figures.

```
int x = 0;
int z = 1;
if (x == 0) {
  x = 1;
  if (x == 1) {
    int y = 3;
    x = 2;
  } else {
    int t = 2;
    x = 10;
  }
} else if ( z == 0 ) {
  z = 2;
}
```

```
int x = 3;
switch(x){
  case 5 : x = x + 2;
  case 8 : x = 1; break;
}
```

Loops

While, for and do-while loops are valid in our language. The exact syntax is shown in figures.

```
int x = 1;
while( x == 1 &&  x <= 1 || x == 20 ) {
  x = 67;
}
```

```
int x = 50;
do {
  x = x + 1;
} while ( x <= 100 )
```

```
for (int i=0;i<2;) {
    /* code */
    i=i+1;
}
```

2. Tools and Technologies

2.1. Lex (A Lexical Analyzer Generator)

Lex source is a table of regular expressions and corresponding program fragments. The table is translated to a program which reads an input stream, copying it to an output stream and partitioning the input into strings which match the given expressions.

2.2. YACC (Yet Another Compiler-Compiler)

Specified the structures of the input, together with code to be invoked as each such structure is recognized. Yacc turns such a specification into a subroutine that handles the input process.

3. Tokens

Token	Description
IF	If statement (if)
ELSE	Else statement (else)
FOR	For loop statement (for)
WHILE	While loop statement (while)
SWITCH	Switch statement (switch)
CASE	Case statement (case)
DEFAULT	Default for switch statement (default)
DO	Do for do-while loop statement (do)
BREAK	Break for case statement (break)
TYPE_INT	Variable type for integers (int)
TYPE_FLOAT	Variable type for floats (float)
TYPE_CAHR	Variable type for character (char)
TYPE_BOOL	Variable type for boolean (bool)
CONST	Constant statement (const)
PRINT	Print expressions
IDENTIFIER	The value of the variables' name
INT_VALUE	The value of the integer
FLOAT_VALUE	The value of the float
CHAR_VALUE	The value of the character
TRUE	true
FALSE	false

4. Language Production Rules

```
line                    : /*epsilon*/
                        | line scope
                        | line stmt
                        ;

stmt                    : declaration semicolon
                        | const_declaration semicolon
                        | assignment semicolon
                        | PRINT expressions semicolon
                        | if_stmt
                        | while_stmt
                        | do_while_stmt
                        | for_stmt
                        | switch_stmt
                        ;

semicolon               : SEMICOLON
                        | /*epsilon*/
                        ;

stmt_list               : stmt
                        | stmt_list stmt
                        ;

const_declaration       : CONST type IDENTIFIER ASSIGN_OP expressions
                        ;

declaration             : type IDENTIFIER ASSIGN_OP expressions
                        | type IDENTIFIER
                        ;

assignment              : IDENTIFIER ASSIGN_OP expressions
                        ;

type                    : TYPE_INT
                        | TYPE_FLOAT
                        | TYPE_BOOL
                        | TYPE_CHAR
                        ;

sign                    : /*epsilon*/
                        | ADD_OP
                        | SUB_OP
                        ;
```

```
46  expressions             : expression
47                          ;
48
49  expression              : sign term
50                          | sign LEFT_BRACE expression RIGHT_BRACE
51                          | expression ADD_OP expression
52                          | expression SUB_OP expression
53                          | expression MUL_OP expression
54                          | expression DIV_OP expression
55                          | expression MOD_OP expression
56                          | expression LE_OP expression
57                          | expression GE_OP expression
58                          | expression EQ_OP expression
59                          | expression NE_OP expression
60                          | expression L_OP expression
61                          | expression G_OP expression
62                          | expression AND_OP expression
63                          | expression OR_OP expression
64                          | NOT_OP term
65                          | NOT_OP LEFT_BRACE expression RIGHT_BRACE
66                          ;
67
68  logic_expression        : expression
69                          ;
70
71  term                    : INT_VALUE
72                          | FLOAT_VALUE
73                          | CHAR_VALUE
74                          | TRUE
75                          | FALSE
76                          | IDENTIFIER
77                          ;
78
79  if_header               : IF LEFT_BRACE logic_expression RIGHT_BRACE
80  if_else_header          : if_header scope ELSE
81                          ;
82
83  if_stmt                 : if_header scope
84                          | if_else_header scope
85                          ;
```

```
86
87  while                   : WHILE
88                          ;
89
90  while_header            : while LEFT_BRACE logic_expression RIGHT_BRACE
91                          ;
92
93  while_stmt              : while_header scope
94                          ;
95
96  do                      : DO
97                          ;
98
99  do_while_stmt           : do scope WHILE LEFT_BRACE logic_expression RIGHT_BRACE semicolon
100                         ;
101
102 for                     : FOR LEFT_BRACE declaration SEMICOLON
103                         | FOR LEFT_BRACE assignment  SEMICOLON
104                         ;
105
106 for_header              : for logic_expression SEMICOLON
107                         ;
108
109 for_stmt                : for_header RIGHT_BRACE scope
110                         ;
111
112 switch_header           : SWITCH LEFT_BRACE expression RIGHT_BRACE LEFT_CURLYBRACKET
113                         ;
114
115 switch_stmt             : switch_header case_block RIGHT_CURLYBRACKET
116                         ;
117
118 case_block              : case_stmt
119                         | case_block case_stmt
120                         ;
121
122 case_header             : CASE term COLON
123                         ;
124
125 break                   : BREAK
126                         ;
```

```
127
128  case_stmt                : case_header stmt_list break semicolon
129                           | case_header stmt_list
130                           | case_header
131                           | DEFAULT COLON stmt_list
132                           ;
133
134  scope                    : scope_start line scope_end
135                           ;
136
137  scope_start              : LEFT_CURLYBRACKET
138                           ;
139
140  scope_end                : RIGHT_CURLYBRACKET
141                           ;
```

5. Quadruples

Quadruple	Description
ADD R1,R2,R3	R3 = R1+R2
SUB R1,R2,R3	R3 = R1-R2
DIV R1,R2,R3	R3 = R1/R2
MUL R1,R2,R3	R3 = R1*R2
MOD R1,R2,R3	R3 = R1%R2
LE R1,R2,R3	R3 = R1<=R2
GE R1,R2,R3	R3 = R1>=R2
EQ R1,R2,R3	R3 = R1==R2
NE R1,R2,R3	R3 = R1!=R2
LS R1,R2,R3	R3 = R1<R2
GR R1,R2,R3	R3 = R1>R2
AND R1,R2,R3	R3 = R1&&R2
OR R1,R2,R3	R3 = R1||R2
NOT R1,-,R3	R3 = !R1
MINUS R1,-,R3	R3 = -R1
MOV R1,-,R3	R3 = R1
JMP R1,R2,LABEL	Go to Label if R1==R2
JMPN R1,R2,LABEL	Go to Label if R1!=R2