Faculty of engineering

Cairo university

Compiler Project - Phase 2

Team number 8

Team members:

Name	Sec.	Bn.
Sarah Mohamed	1	23
Salma Mohamed	1	27
Kareem Omar	2	6
Nourhan GamaL	2	31

Work load

Name	**Work**
Sarah Mohamed	● Variables and Constants declaration & their logic ● Mathematical and logical expressions & their logic ● Assignment statement & their logic ● Design suitable action rules to produce the output quadruples ● Semantic analysis: - Variable declaration conflicts. - Improper usage of variables regarding their type. - Variables used before being initialized and unused variables. - The addition of type conversion quadruples
Salma Mohamed	● Functions & its logic ● Semantic analysis : function arguments type and value and number ● Implement GUI
Kareem Omar	● While loops, repeat-until loops, for loops & their logic ● Block structure, nested scopes & their logic
Nourhan GamaL	● If-then-else statement, switch statement & their logic ● Symbol table ● Semantic analysis :Checking function return type

[1] Project Overview

The designed compiler using Lex and Yacc for C language. And therefore, we design a programming language like the C language.

Our programming language include

1. Variables and Constants declaration

Data types:

- Integer (int)
- Float (float)
- character(char)
- String (string)

Variable names are defined to be a single character from a-z.

Example:

Variable declaration:

```
int x = 1;
float y = 3.4;
char w =’f’;
string s =”compilers proj”;
```

Just like c language <datatype> <variable-name><=> <variable-value> ;

Constant declaration:

```
const int c =7;
const int n =7;
const float y = 9.5;
const char c = ‘c’;
```

Just like c language const <datatype> <variable-name> <=><variable-value> ;

2. Mathematical and Logical expressions

Supported operations:

“*”, “+”, “-“, “/”, ”=”, "&&","||", "!", "==", "!=", ">=", "<=", ">", "<"

```
  int x = 9 + 7;

  float y = 8.7 * 5.8;
```

3. Assignment statement:

```
   int x =8;
```

4. If-then-else statement:

```
1.  if(x==9){ x=0;}   else{x=1;}

2.  if(x==9 || y ==2){ x=0;}
    if(y==9 ){y=8;}  else{x=9;}

3.  if (x == 7 && y != 5) {
      int z = 1;
    } else if ( x == 4 ) {
      int z = 2;
    }
```

for loops:

```
    for ( i = 0; i < 10; i = i + 1 )
     { int x = 1;
      int y = x + 5; }
```

while loops:

```
     while(x==8) {c=0;}
```

repeat-until loops:

```
     do{x=x+1;}

     while(x<9)
```

switch-case statement:

```
     1.       switch(x) {
     case 3: x=9;
     case 6: x=7;}
     2.       switch(x) {case 3: x=9;
```

```
case 6: x=7;
default: x=2;}
```

Block structure (nested scopes)

```
if (x == 3) {
  x = 2;
  if (x == 8) {
    float num= 55.3;
    int f = 4;
    x = 1;
  } else {
    int y = 2;
    x = 10;
  }
} else if ( z > 5 ) {
  if ( z==2){
int q = a+4;
      } else{
int q = b+5;
}
}
```

Functions: function name consists of one character from a-z

```
int f ()
{
               int a ;
               return 3;
           }
float f ( int a , int b)
{
        float s = a + b;
        return s;
        }
float f ( float s )
{
        return s;
 }
```

```
void f ()
{
int a = b * c;
}

 void f (int b , float c)
{
int a = b * c;
}
```

Function call:

```
int s = f ( a,b);
f (1,2);
```

2. Tools and Technologies

2.1 Lex (A Lexical Analyzer Generator) (used flex instead)

During the first phase the compiler reads the input and converts strings in the source to tokens. With regular expressions we can specify patterns to lex so it can generate code that will allow it to scan and match strings in the input. Each pattern specified in the input to lex has an associated action. Typically an action returns a token that represents the matched string for subsequent use by the parser. Initially we will simply print the matched string rather than return a token value.

2.2 YACC (Yet Another Compiler-Compiler)   (used bison instead)

YACC generates a syntax analyzer (parser) for a given grammar.

3. Tokens

Token	**Description**
TYPE_INT	Variable type for integers (int)
TYPE_FLT	Variable type for floats (float)
TYPE_CHR	Variable type for character (char)
TYPE_STR	Variable type for string (string)
TYPE_CONST	Constant statement (const)
ID	The value of the variables' name
NUM	Integer value assigned to a variable or constant
FLOATING_NUM	Decimal value assigned to a variable or constant
CHAR_VALUE	Character value assigned to a variable or constant
IF	If statement (if)
ELSE	Else statement (else)
ELSEIF	Else if statement (else if)
FOR	For loop statement (for)
WHILE	While loop statement (while)
SWITCH	Switch statement (switch)
CASE	Case statement (case)
DO	Do for do-while loop statement (do)
BREAK	Break statement (break)
Default	If no case (default)

ID	The identifier like variable name
NUM	Any integer value like 1,2,3
FLOATING_NUM	Any float value like 3,7 , 8.9
CHAR_VALUE	Any character between two quotes like ‘d’
STRING_VALUE	Any sentence between two quotes “compiler project”
RETURN	Return in function (return)
TYPE_VOID	Type of void function (void)
GT	Greater than comparison operator (<)
LT	Larger than comparison operator (>)
INC	Increment (++)
DEC	Decrement (--)
AND	And operation (&&)
OR	Oring operation (||)
Not	Not operation (!)
EQ	Equal (=)
NOTEQ	Not equal (!=)
GTE	Greater or equal (>=)
LTE	Less or equal (<=)
Show_symbol_table	Prints the current variables in the symbol table
exit_command	Exit the program (exit)

4. Language Production Rules

```
statement       : variable_declaration_statement ';'
statement       :assign_statement ';'
statement       : constant_declaration_statement ';'
statement       : conditional_statement
statement       : math_expr ';'
statement       : increment_decrement ';'
statement       : exit_command ';'
statement       : statement variable_declaration_statement ';'
statement       : statement assign_statement ';'
statement       : statement constant_declaration_statement ';'
statement       : statement conditional_statement
statement       : statement math_expr ';'
statement       : open_brackets statement close_brackets statement
statement       : statement open_brackets statement close_brackets
statement       : function_statement
statement       : statement function_statement
statement       :  functionCall
statement       : statement functionCall
statement       : statement exit_command ';'
statement       : print_symble ';'
statement       : statement print_symble ';'
statement       :  //empty line
function_statement :TYPE_INT ID func_open_brace '(' argList ')' '{'  statement  returnFunc ';'func_close_brace
                |TYPE_FLT ID func_open_brace'('argList ')' '{' statement returnFunc ';' func_close_brace
                |TYPE_CHR ID func_open_brace'(' argList')' '{' statement returnFunc ';' func_close_brace
                |TYPE_STR ID func_open_brace'(' argList')' '{' statement returnFunc ';' func_close_brace
                |TYPE_VOID ID   func_open_brace '(' argList')' '{'  statement  func_close_brace
func_open_brace :
func_close_brace : '}'
functionCall: ID '(' argListCall ')' ';'
returnFunc:   RETURN
             | RETURN NUM
              | RETURN ID
              | RETURN FLOATING_NUM
              | RETURN CHAR_VALUE
```

```
                 | RETURN STRING_VALUE
argList:    TYPE_INT  ID count
             |TYPE_FLT  ID count
             |TYPE_CHR  ID count
             |TYPE_STR  ID count
             |
             ;
data:     NUM
        | ID
        |FLOATING_NUM
        | CHAR_VALUE
        ;
dataType:  TYPE_INT
            |TYPE_FLT
            |TYPE_CHR
            |TYPE_VOID
            |TYPE_STR
            ;
argListCall: data  countt
            |
            ;
countt: ',' data countt
        | //empty
count: ',' TYPE_INT ID count
             |',' TYPE_FLT ID count
            |',' TYPE_CHR ID count
            |',' TYPE_STR ID count
            |',' TYPE_VOID ID count
            | //empty
            ;
conditional_statement :if_statement {;}
                              |while_loop {;}
                              |for_loop {;}
                              |do_while {;}
                              |switch_statement{;}
                              ;
switch_statement: SWITCH '(' math_expr ')' switch_body;
Switch_body:    open_brackets cases close_brackets
                    |open_brackets cases default close_brackets
cases: CASE
```

```
math_expr  ':' statement case_break
                    |cases cases
case_break:
                  | BREAK ';'
default: DEFAULT ':' statement
            | DEFAULT ':' BREAK ';'
do_while: DO '{' statement '}' {close_brace();} WHILE '('condition')' ';'
for_loop:FOR '(' assign_statement for_first_semi_colon condition for_second_semi_colon
assign_statement ')'for_open_brac statement for_close_brac
for_first_semi_colon: ';'
for_second_semi_colon : ';'
for_open_brac : '{'
for_close_brac : '}'
while_loop :WHILE '(' condition ')' while_open_brace statement while_close_brace
while_open_brace : '{'
while_close_brace : '}'
if_statement : IF '(' condition ')'if_open_brace statement if_close_brace
                      | IF '(' condition ')'if_open_brace statement if_close_brace Else_last statement
if_close_brace
                      | IF '(' condition ')'if_open_brace statement if_close_brace ELSE if_statement
Else_last : ELSE '{'
if_open_brace : '{'
if_close_brace : '}'
condition :'(' condition ')' {;}
                  | condition OR comparing_condition
                  | condition AND comparing_condition
                  | NOT condition
                  | comparing_condition
                  ;
comparing_condition :math_expr EQ math_expr
                                | math_expr NOTEQ math_expr
                                | math_expr GTE math_expr
                                | math_expr GT math_expr
                                | math_expr LTE math_expr
                                | math_expr LT math_expr
                                ;
math_expr     :'('math_expr')'
                    |math_expr '+' mult_div_expr
                    | math_expr '-' mult_div_expr
                    | '~' math_expr
                    | math_expr '|' mult_div_expr
                    | math_expr '&' mult_div_expr
                    | math_expr '^' mult_div_expr
                    |mult_div_expr
                    ;
mult_div_expr:mult_div_expr '*' math_element
                      |mult_div_expr '/' math_element
                      |math_element
                    ;
math_element:        NUM
                            | FLOATING_NUM
                            | ID
                            | '('math_expr')'
                          ;
assign_statement: math_expr_assignment
                          |ID '=' math_element_NUM
                           | ID '=' math_element_FLT
                           | ID '=' math_element_CHR
                          | ID '=' math_element_STR
                           | ID '=' math_element_ID
                           | ID '=' ID '(' argListCall ')'
                           ;
math_expr_assignment  : ID '='  '('math_expr')'
                                  |ID '='  math_expr '+' mult_div_expr
                                  |ID '='  math_expr '-' mult_div_expr
                                  | ID '='  '~' math_expr
                                  | ID '='  math_expr '|' mult_div_expr
                                  |ID '='  math_expr '&' mult_div_expr
                                  |ID '='  math_expr '^' mult_div_expr
                                  |ID '='  mult_div_expr '*' math_element
                                   |ID '=' mult_div_expr '/' math_element
                                ;
math_element_NUM:        NUM
math_element_FLT: FLOATING_NUM
math_element_CHR: CHAR_VALUE
math_element_STR: STRING_VALUE
math_element_ID : ID
increment_decrement: ID DEC
                               | ID INC
```

```
                         ;
math_expression_init     :'('math_expr')'
                         |math_expr '+' mult_div_expr
                         | math_expr '-' mult_div_expr
                         | '~' math_expr
                         | math_expr '|' mult_div_expr
                         | math_expr '&' mult_div_expr
                         | math_expr '^' mult_div_expr
                         |mult_div_expr '*' math_element
                         |mult_div_expr '/' math_element                          ;
variable_declaration_statement:
          TYPE_INT ID
          |TYPE_FLT ID
          |TYPE_CHR ID
          |TYPE_STR ID
          |TYPE_INT ID '=' math_expression_init
          |TYPE_FLT ID '=' math_expression_init
          |TYPE_INT ID '=' math_element_ID
          |TYPE_FLT ID '=' math_element_ID
          |TYPE_INT ID '=' math_element_NUM
          |TYPE_FLT ID '=' math_element_FLT
          |TYPE_CHR ID '=' ID
          |TYPE_STR ID '=' ID
          |TYPE_CHR ID '=' CHAR_VALUE
          |TYPE_CHR ID '=' FLOATING_NUM
          |TYPE_STR ID '=' STRING_VALUE
          |TYPE_STR ID '=' FLOATING_NUM
          |TYPE_INT ID '=' ID   '(' argListCall ')'
          |TYPE_FLT ID '=' ID '('argListCall ')'
          |TYPE_CHR ID '=' ID '(' argListCall')'
          |TYPE_STR ID '=' ID '(' argListCall')'
          ;
constant_declaration_statement: TYPE_CONST TYPE_INT ID '=' math_expr
                                | TYPE_CONST TYPE_FLT ID '=' math_expr
                                | TYPE_CONST TYPE_CHR ID '=' CHAR_VALUE
                                 | TYPE_CONST TYPE_STR ID '=' STRING_VALUE
open_brackets: '{' { open_brace(); } ;
close_brackets: '}' { close_brace(); };
```

5. Quadruples

Quadruple	**Description**
JMP labelX	Unconditional jump to label X
JT RF,labelX	Jump to label X if RF is true
JF RF,labelX	Jump to label X if RF is false
NOT RX	~RX
MOV RX, RY	RX=RY
ADD R1,R2,R3	R1 = R2+R3
SUB R1,R2,R3	R1=R2-R3
OR R1,R2,R3	R1=R2|R3
AND R1,R2,R3	R1=R2&R3
XOR R1,R2,R3	R1=R2 xor R3
MUL R1,R2,R3	R1=R2*R3
DIV R1,R2,R3	R1=R2/R3
CMPE R1,R2,R3	R1 true if R2 == R3 and vice versa
CMPNE R1,R2,R3	R1 true if R2 != R3 and vice versa
CMPGE R1,R2,R3	R1 true if R2 >= R3 and vice versa
CMPG R1,R2,R3	R1 true if R2 > R3 and vice versa
CMPLE R1,R2,R3	R1 true if R2 <= R3 and vice versa
CMPL R1,R2,R3	R1 true if R2 < R3 and vice versa
PUSH_<TYPE><VALUE>	Push <value> to the top of stack
POP_<TYPE><dst>	Pop S1 and save it to <dst>
PROC <ID>	Define procedure (function)
CALL <ID>	Calls a procedure (function)
RET	Return from a procedure (function)