Faculty of Engineering
Credit Hours System

Cairo University

Languages and Compilers Project Report

CMPN 403

Project Members

Abdallah Khaled Sobehy

Mostafa Ashraf Fateen

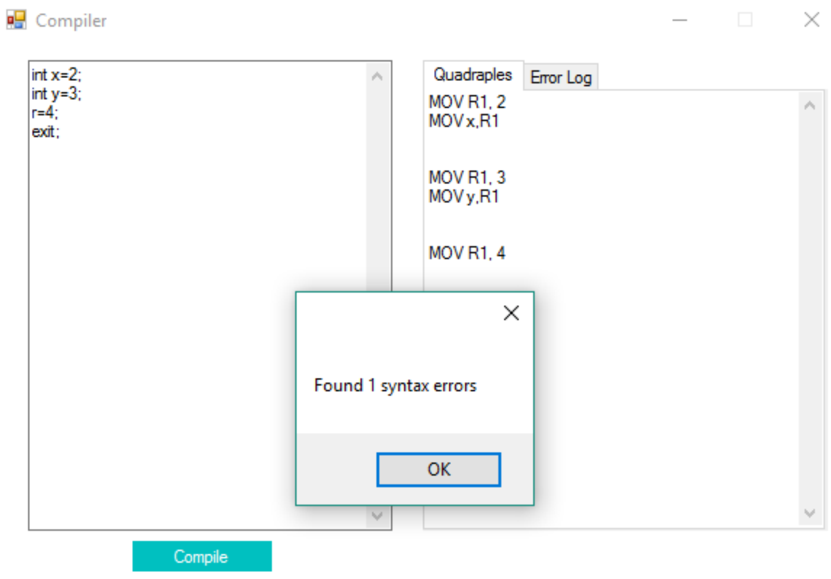
Ramy Adel Alfred

Yousra Samir Mohamed

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 and a simple GUI using C#.



```
int x=2;
int y=3;
r=4;
exit;
```

```
MOV R1, 2
MOV x,R1

MOV R1, 3
MOV y,R1

MOV R1, 4
```

Found 1 syntax errors

Figure 1 GUI of the compiler

1.2. Language Rules

Variables and Constants

Variables and constants' types are integer (int), decimals (float) and characters (char). Their names are defined to be single character from a-z (lower case).

i.e. int x=3; const float y=2.1; char r ='r';

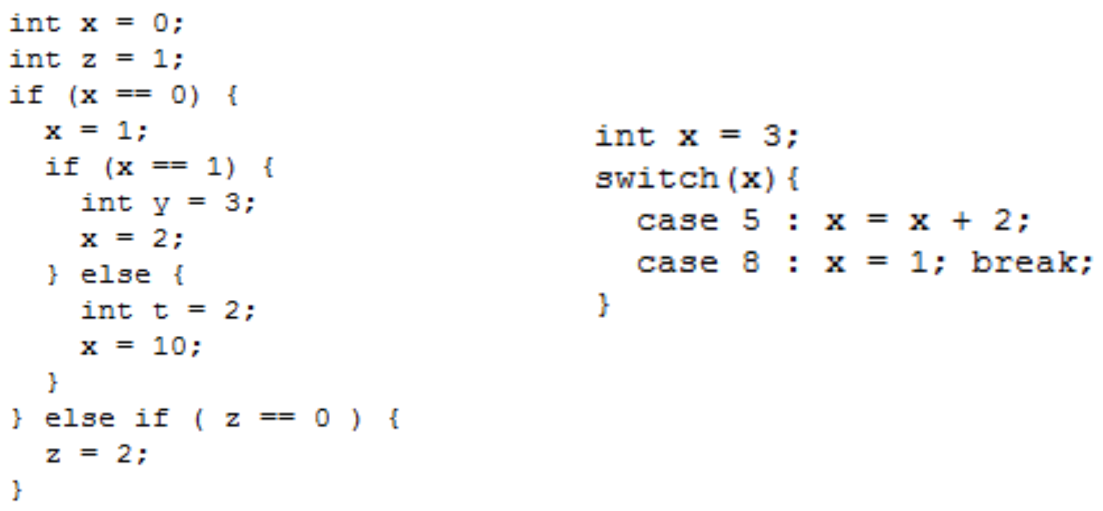
Mathematical and Logical expressions

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, NOT and XOR (&,|,~,^).

Conditions

Conditions include both if-else statements and switch-case statements the exact syntax is shown in figure (2)



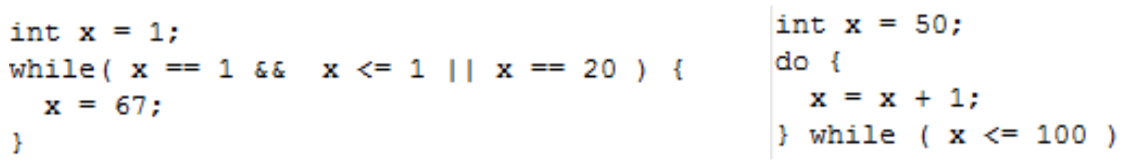
```
int x = 0;
int z = 1;
if (x == 0) {
  x = 1;
  if (x == 1) {
    int y = 3;
    x = 2;
  } else {
    int t = 2;
    x = 10;
  }
} else if ( z == 0 ) {
  z = 2;
}
```

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

Figure 2 The exact syntax of if-else and switch-case statements

Loops

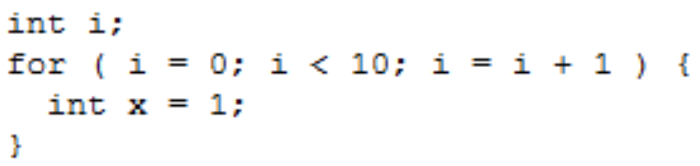
While, for and do-while loops are valid in our language. The exact syntax is shown in figure (3) and figure (4).



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

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

Figure 3 The exact syntax of the while and do-while loops



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

Figure 4 The exact syntax of the for loops

2. Tools and Technologies

2.1. Lex (A Lexical Analyzer Generator)

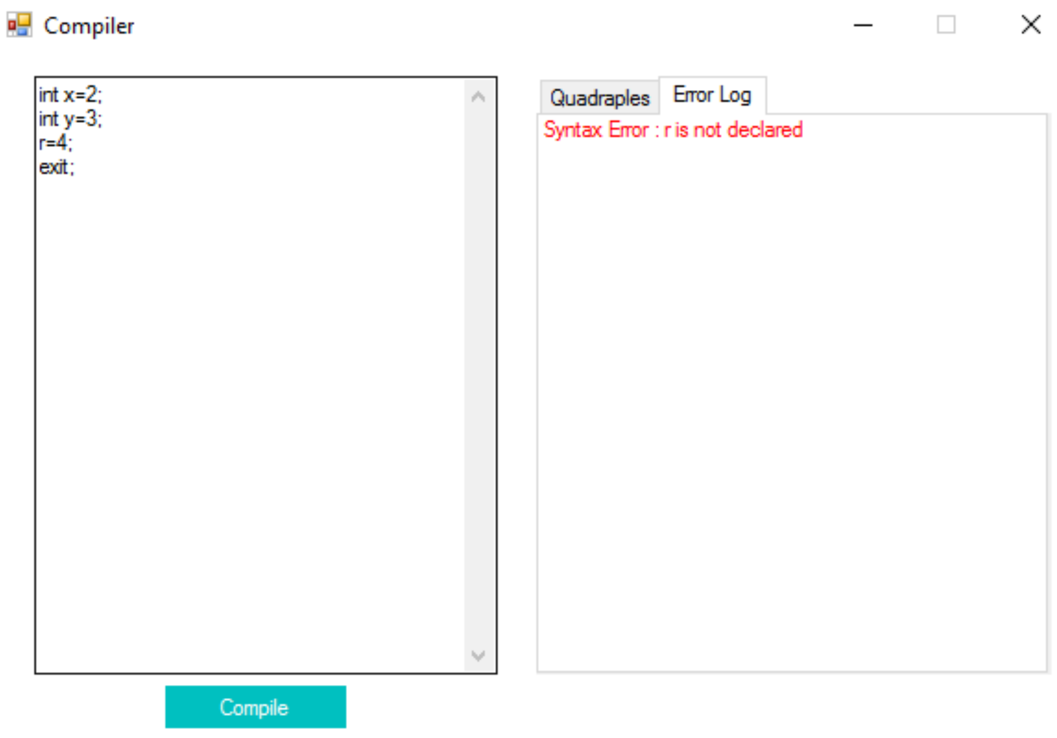
Lex source is a table of regular expressions and corresponding program fragments. The table is translated to a program which reads an input stream, copying it to an output stream and partitioning the input into strings which match the given expressions.

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.

2.3. C# (Windows Form Application)

Used C# to develop a GUI for the compiler shown in figure (5).



```
int x=2;
int y=3;
r=4;
exit;
```

```
Syntax Error : r is not declared
```

Figure 5 GUI of the compiler

3. Tokens

Token	**Description**
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)
TYPE_INT	Variable type for integers (int)
TYPE_FLT	Variable type for floats (float)
TYPE_CHR	Variable type for character (char)
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
exit_command	Exits the program (exit)
AND	Logical AND used in comparison (&&)
OR	Logical OR used in comparison (||)
NOT	Logical NOT used in comparison (!)
EQ	Equal to comparison operator (==)
NOTEQ	Not equal to comparison operator (!=)
GTE	Greater than or equal comparison operator (<=)
LTE	Larger than or equal comparison operator (>=)
GT	Greater than comparison operator (<)
LT	Larger than comparison operator (>)
INC	Increment (++)
DEC	Decrement (--)
Show_symbol_table	Prints the current variables in the symbol table

4. Language Production Rules

- Statement: variable_declaration_statement
- Statement: assign_statement
- Statement: constant_declaration_statement
- Statement: math_expr
- Statement: exit_command
- Statement: show_symbol_table
- Statement: statement variable_declaration_statement
- Statement: statement assign_statement
- Statement: statement constant_declaration_statement
- Statement: statement math_expr
- Statement: statement exit_command
- Statement: statement show_symbol_table
- Statement: open_brace statement close_brace statement {;}
- Statement: statement open_brace statement close_brace {;}
- conditional_statement: if_statement
- conditional_statement: while_loop {;}
- conditional_statement: for_loop {;}
- conditional_statement: do_while {;}
- conditional_statement: switch_statement{;}
- switch_statement: SWITCH '(' math_expr ')'
- switch_body: open_brace cases close_brace
- switch_body: open_brace cases default close_brace
- cases: CASE
- cases: cases cases
- case_break: BREAK
- default: DEFAULT ':' statement
- do_while: DO '{' statement '}' WHILE '('condition')'
- for_loop: FOR '(' assign_statement for_sep1 condition for_sep2 assign_statement ')'for_ob statement for_cb
- while_loop: WHILE '(' condition ')' while_open_brace statement while_closed_brace
- while_open_brace: '{'
- while_closed_brace: '}'
- if_statement: IF '(' condition ')'if_open_brace statement if_closed_brace
- if_statement: IF '(' condition ')'if_open_brace statement if_closed_brace ELSE_FINAL statement if_closed_brace
- if_statement: IF '(' condition ')'if_open_brace statement if_closed_brace ELSE if_statement
- ELSE_FINAL: ELSE '{'
- if_open_brace: '{'
- if_closed_brace: '}'
- condition: '(' condition ')'

- condition: condition OR high_p_condition
- condition: condition AND high_p_condition
- condition: NOT condition
- condition: high_p_condition
- high_p_condition: math_expr EQ math_expr
- high_p_condition: math_expr NOTEQ math_expr
- high_p_condition: math_expr GTE math_expr
- high_p_condition: math_expr GT math_expr
- high_p_condition: math_expr LTE math_expr
- high_p_condition: math_expr LT math_expr
- math_expr: '('math_expr')'
- math_expr: math_expr '+' high_priority_expr
- math_expr: math_expr '-' high_priority_expr
- math_expr: '~' math_expr
- math_expr: math_expr '&' high_priority_expr
- math_expr: math_expr '|' high_priority_expr
- math_expr: math_expr '^' high_priority_expr
- math_expr: high_priority_expr
- high_priority_expr: high_priority_expr '*' math_element
- high_priority_expr: high_priority_expr '/' math_element
- high_priority_expr: math_element
- math_element: NUM
- math_element: FLOATING_NUM
- math_element: ID
- math_element: '('math_expr')'
- assign_statement: ID '=' math_expr
- variable_declaration_statement: TYPE_INT ID
- variable_declaration_statement: TYPE_FLT ID
- variable_declaration_statement: TYPE_CHR ID
- variable_declaration_statement: TYPE_FLT ID
- variable_declaration_statement: TYPE_INT ID '=' math_expr
- variable_declaration_statement: TYPE_FLT ID '=' math_expr
- variable_declaration_statement: TYPE_CHR ID '=' CHAR_VALUE
- open_brace: '{'
- close_brace: '}'
- constant_declaration_statement: TYPE_CONST TYPE_INT ID '=' math_expr
- constant_declaration_statement: TYPE_CONST TYPE_FLT ID '=' math_expr
- constant_declaration_statement: TYPE_CONST TYPE_CHR ID '=' CHAR_VALUE

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