Credit Hours System
CMPN403

Cairo University
Faculty of Engineering

Compilers Project

Name	ID
Rana Mostafa	1142309
Fady Nasser	1142158
Taghreed Hassan Ahmed	1142163
Chahira Hamza	11417005

Project Overview:

A simple compiler using Lex and Yacc programming languages with simple GUI where you can declare variables, constant and use Mathematical and logical expressions, Assignment statement, If-then-else statement, while loops, repeat-until loops, for loops, switch statement and also you can write Block structure (nested scopes where variables may be declared at the beginning of blocks).

Tools and Technologies used:

- Flex
- Bison
- Dev-Cpp (GCC)

List of tokens and a description of each.

Token	Description
{	OBRACE
{	EBRACE
(	ORBRACKET
)	ERBRACKET
;	SEMICOLON
:	COLON
,	COMMA
++	INCREMENT
--	DECREMENT
+=	PLUS EQUAL
-=	MINUS EQUAL
*=	MULTIPLY EQUAL
/=	DIVIDE EQUAL
>	GREATER THAN
<	LESS THAN
>=	GREATER THAN OR EQUAL
<=	LESS THAN OR EQUAL
==	EQUILTY CONDITION
!=	NOT EQUAL
+	PLUS
-	MINUS

*	MULTIPLY
/	DIVIDE
^	POWER
=	ASSIGN
%	Get REMAINDER
&&	AND
||	OR
!	NOT
while	WHILE LOOP
for	for LOOP
if	If condition
else	else
print	PRINT
bool	BOOLEAN
int	INTEGER
float	FLOAT
double"	DOUBLE
long	LONG
char	CHAR;
string	STRING
const	CONSTANT
do	DO
break	BREAK
switch	SWITCH
case	CASE
false	FALSE
true	TRUE
default	DEFAULT
return	RETURN

List of the language production rules:

```
Type:    INT   {$$=0;}
         | FLOAT{$$=1;}
         | CHAR {$$=2;}
         | STRING {$$=3;}
         | BOOL {$$=4;}
         ;

Constant : CONST INT         {$$=5;}
           | CONST FLOAT     {$$=6;}
           | CONST CHAR      {$$=7;}
           | CONST STRING    {$$=8;}
           | CONST BOOL      {$$=9;}
           ;

no_declaration:   FLOATNUMBER                  { char c[] = {}; ftoa($1, c, 6); $$ = con(c, 1); }
         | INTEGERNUMBER                       { char c[] = {}; itoa($1, c, 10); $$ = con(c, 0); }
         | IDENTIFIER                          { $$ = getId($1,brace); }
         | no_declaration PLUS   no_declaration { $$ = opr(PLUS, 2, $1, $3); }
         | no_declaration MINUS no_declaration  {$$= opr(MINUS,2,$1,$3);}
         | no_declaration MUL no_declaration    {$$= opr(MUL, 2 ,$1,$3);}
         | no_declaration  DIV   no_declaration {$$= opr(DIV, 2 ,$1,$3);}
         | no_declaration  REM   no_declaration {$$= opr(REM, 2 ,$1,$3);}
         | no_declaration  POWER no_declaration {$$= opr(POWER, 2 ,$1,$3);}
         | MINUS no_declaration %prec UMINUS    {$$ = opr(UMINUS, 1, $2); }
         | IDENTIFIER INCREMENT                 {$$=opr(INCREMENT,1,$1);}
         | IDENTIFIER DECREMENT                 {$$=opr(DECREMENT,1,$1);}
         | ORBRACKET no_declaration ERBRACKET   {$$=$2;}    ;

increments: IDENTIFIER  INCREMENT                   {$$=opr(INCREMENT,1,$1);}
          | IDENTIFIER DECREMENT                    {$$=opr(DECREMENT,1,$1);}
          | IDENTIFIER PEQUAL no_declaration         {$$= opr(PEQUAL, 2 ,$1,$3);}
          | IDENTIFIER MEQUAL no_declaration         {$$= opr(MEQUAL, 2 ,$1,$3);}
          | IDENTIFIER MULEQUAL no_declaration       {$$= opr(MULEQUAL, 2 ,$1,$3);}
          | IDENTIFIER DIVEQUAL no_declaration       {$$= opr(DIVEQUAL, 2 ,$1,$3);}
          ;
```

```
forExpression : increments                          {$$=$1;}
              | IDENTIFIER ASSIGN no_declaration     {$$ = opr(ASSIGN, 2, getId($1,brace), $3);};

booleanExpression: expression AND expression        { $$ = opr(AND, 2, $1, $3); }
            | expression OR expression              { $$ = opr(OR , 2, $1, $3); }
            | NOT expression                        { $$ = opr(NOT, 1, $2); }
            | DataTypes GREATER DataTypes           { $$ = opr(GREATER, 2, $1, $3); }
            | DataTypes LESS DataTypes              { $$ = opr(LESS, 2, $1, $3); }
            | DataTypes GE DataTypes                { $$ = opr(GE, 2, $1, $3); }
            | DataTypes LE DataTypes                { $$ = opr(LE, 2, $1, $3); }
            | DataTypes NE DataTypes                { $$ = opr(NE, 2, $1, $3); }
            | DataTypes EQ DataTypes                { $$ = opr(EQ, 2, $1, $3); }
            | ORBRACKET booleanExpression ERBRACKET {  $$ = $2; }
            ;
```

```
stmt:   Type IDENTIFIER SEMICOLON   %prec IFX                          {$$=id(indexCount,$1,brace,Accepted,$2);printf("Declaration\n");indexCount++;}

      | IDENTIFIER ASSIGN expression SEMICOLON                         {$$ = opr(ASSIGN,2, getId($1,brace), $3); printf("Assignment\n");}

      | Type IDENTIFIER ASSIGN expression SEMICOLON                   {$$ = opr(ASSIGN,2, id(indexCount,$1,brace,Accepted,$2), $4); indexCount++; printf

      | Constant IDENTIFIER ASSIGN expression SEMICOLON              { $$ = opr(ASSIGN,2, id(indexCount,$1,brace,Constant,$2), $4); indexCount++;printf(

      | increments SEMICOLON                                         {$$=$1; printf("Increments\n");}

      | WHILE ORBRACKET expression ERBRACKET stmt                    {$$ = opr(WHILE,2, $3, $5); printf("While loop\n");}

      | DO braceScope WHILE ORBRACKET expression ERBRACKET SEMICOLON  {$$ = opr(DO,2, $2, $5);printf("Do while\n");}

      | FOR ORBRACKET INT IDENTIFIER ASSIGN INTEGERNUMBER SEMICOLON
        expression SEMICOLON
        forExpression ERBRACKET
        braceScope                                          {char c[] = {}; itoa($6, c, 10);$$ = opr(FOR,4, opr(ASSIGN, 2, getId($4,brace), con(c, 0


      | IF ORBRACKET expression ERBRACKET braceScope %prec IFX            {$$ = opr(IF, 2, $3, $5);printf("If statement\n");}

      | IF ORBRACKET expression ERBRACKET braceScope   ELSE braceScope    {$$ = opr(IF, 3, $3, $5, $7); printf("If-Elsestatement\n");}

      | SWITCH ORBRACKET IDENTIFIER ERBRACKET switchScope                 {$$ = opr(SWITCH, 2, $3, $5);printf("Switch case\n");}

      | PRINT expression  SEMICOLON                                       {$$ = opr(PRINT, 1, $2); printf("Print\n");}

      | braceScope                                                       {$$=$1; printf("New braces scope\n");}
```

```
DataTypes:no_declaration        {  $$ = $1; }
        | CHARACTER             { $$ = con($1, 2); }
        | FALSE                 { $$ = con("false", 4); }
        | TRUE                  { $$ = con("true", 4); }
        | TEXT                  { $$ = con($1, 3); };
```

```
expression: DataTypes                       {     $$ = $1; }
          | booleanExpression               {  $$ = $1;     } ;

caseExpression: DEFAULT COLON stmtlist BREAK SEMICOLON           { $$ = opr(DEFAULT, 2, $3, opr(BREAK, 0)); }
        | CASE INTEGERNUMBER COLON stmtlist BREAK SEMICOLON  caseExpression  { char c[] = {}; itoa($2, c, 10);
          ;
```

List of the quadruples and a short description of each:

Quadruple	Description
mov R1, v	Move value to R1
Jnz L	Jump to label L if equal zero
jmp Label	Unconditional jump to Label L
neg R1, R2	R1=-R2
add R3, R2, R1	R3=R1+R2
sub R3, R2, R1	R3=R1-R2
mul R3, R2, R1	R3=R1*R2
div R3, R2, R1	R3=R1/R2
power R3, R2, R1	R3=R1 POW R2
compGREATER R3, R2, R1	Compare if R1 > R2 and result in R3
compLESS R3, R2, R1	Compare if R1 < R2 and result in R3
compGE R3, R2, R1	Compare if R1 >= R2 and result in R3
compLE R3, R2, R1	Compare if R1 <= R2 and result in R3
compNE R3, R2, R1	Compare if R1 != R2 and result in R3
compEQ R3, R2, R1	Compare if R1 == R2 and result in R3
inc R1	Value in R1++
dec R1	Value in R1--
add R3, R2, R1	x+=2
mov v , R1	Assign x=2