Faculty of Engineering
Cairo University

Cairo University

Compilers Project Report

Team Members

Hossam Hassan

Khaled Osama

Omar Saeed

Omar Sayed

Project Overview

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

Tools and Technologies used

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.

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.

Language Rules

Variables and Constants
Variables and constants' types are integer (int), decimals (double) , characters (char) and strings(string).

Ex:
```
 int x=10;
int y = x+20;
char temp ='a';
double d =10.0;
string name = "omar";
```

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
Ex:
If statement:
```
if(k==1)
{
    int temp =1;
}
else
{
    if(y==1)
    {
                 h=12;
    }
    else
    {
         g=12;
         }
    s=12;
}
```

switch-case statement:

```
  switch (b+3) {
        case 'a':
                 c = b+d;
                break ;
        case 3:
                a=a+b-2*4/2;
                b=c+d*3;
        case 4:
                a=b+1;
                b=d*c+6;
        case 5:
                t=4+1;
                Break;
        case 2:
                c = b;
                d = 1+2;
                break;
        default:
                c = d;
        }
```

Loops

While, for and do-while loops

Ex:

For Loop:

```
for(int y=10; y<20;i=i+1)
{
    if(x==20)
    {
                break;
    }
     y = x+10;
}
for(int y=10; y<20;i=i+1)
{
    if(x==20)
    {
                continue;
    }
     y = x+10;
}
```

While loop:

```
while(X== 2)
   {
        if(X==2)
        {
                omar=22;
        }
        else
        {    omar =44;
        }
   }
```

Do-While loop:

```
do
{
   x=x+10;
}   while(y<20)
```

List of Tokens

DIGIT   [0-9]
NZDIGIT [1-9]
ALPHA   [a-zA-Z_]

Token	Description
Data Types	
"char"	Variable type for Characters
"int"	Variable type for integers
"double"	Variable type for doubles
"bool"	Variable type for booleans
"string"	Variable type for strings
Assignment	
"="	Equal Assignment operator
Values	
'{ALPHA}'	Character value
{DIGIT}{DIGIT}*	Integer value
({DIGIT}{DIGIT}*)"."({DIGIT})+	Double value
\"{ALPHA}+\"	String value

true	True value
false	False value
Reserved Words	
break	Break Statement
case	Case Statement
continue	Continue Statement
default	Default value for Case statement
do	Do in do-while loop statement
else	Else statement
for	For loop statement
if	If statement
return	Return statement
switch	Switch statement
while	While loop statement
cout	Standard Output
const	Constant for constant values
"<<"	Cout operator
\"({ALPHA}|{DIGIT}|[ ])*\"	Cout value
endl	New line
cin	Standard Input

">>"	Cin operator
"&&"	Logical And
"||"	Logical OR
"!"	Logical NOT
"<="	Less than or equal
">="	Greater than or equal
"=="	Equal in comparison operations
"<"	Less than
">"	Greater than
"!="	Not Equal
";"	Semicolon
("{")	Open bracket
("}")	Closed bracket
","	Comma
":"	Colon
"("	Open parentheses
")"	Closed parentheses

("[")	Open bracket
("]")	Closed bracket
"&"	And
"~"	Not
"-"	Subtraction
"+"	Addition
"*"	multiplication
"/"	division
[\n]	endline
"^"	Xor
{ALPHA}({ALPHA}|{DIGIT})*	Identifier
(" "|\t)*	
.	

List of language production rules

program : Stmt

program : program Stmt

/* Declaration block */

Declaration :   type IDENTIFIER '=' expressions

Declaration : type IDENTIFIER

ConstDeclaration   : CONST type IDENTIFIER '=' expressions

Assignment   : IDENTIFIER '=' expressions

type   : INT

type   : DOUBLE

type   : BOOL

type   : CHAR

type   : STRING

/* MathExpr block */

MathExpr: MathExpr '+' MathExpr

MathExpr: MathExpr '-' MathExpr

MathExpr: MathExpr '*' MathExpr

MathExpr: MathExpr '/' MathExpr

MathExpr: VALUES

expressions   : Expr

expressions   :MathExpr

expressions   :LogicExpr

Expr : MathExpr LE_OP MathExpr

Expr : MathExpr GE_OP MathExpr

Expr : MathExpr EQ_OP MathExpr

Expr :MathExpr NG_EQ MathExpr

Expr : MathExpr '>' MathExpr

Expr : MathExpr '<' MathExpr

```
LogicExpr:VALUES AND_OP VALUES
LogicExpr: VALUES OR_OP VALUES
LogicExpr: NE_OP    VALUES
LogicExpr: Bstart LogicExpr Bend
VALUES: IDENTIFIER
VALUES:INTVAL
VALUES:DOUBLEVAL
VALUES:CHARVAL
VALUES:TRUE1
VALUES:FALSE1
VALUES:STRINGVAL
```

/* statement and block statements */

```
Stmt: Decleration
Stmt:Assignment
Stmt:ConstDecleration
Stmt:IfStmt
Stmt:coutstatement
Stmt:cinstatement
Stmt:expressions
Stmt:WhileStmt
Stmt:SwitchStatement
Stmt:DoWhileStmt
Stmt:ForStmt
Stmt:contStat
Stmt:breakStat
BLOCKStmt:    Kstart StmtList Kend
BLOCKStmt:    Stmt
contStat : CONTINUE
breakStat: BREAK
```

Kstart : '{'
ifStart :'{'
ifEnd1 : '}'
ifEnd : '}'
ELSEend : '}'
Kend : '}'
Bstart : '('
Bend :')'
StmtList: StmtList Stmt |
/* IfStmt Block between { } */
expressionsif : expressions
IfStmt : IF Bstart expressionsif Bend ifStart StmtList ifEnd1
IfStmt :IF Bstart expressionsif Bend ifStart StmtList ifEnd ELSE Kstart StmtList ELSEend
/* switch Block lazm ben { } */
SwitchStatement : SWITCH Bstart MathExpr Bend Kstart CaseBlock Kend
CaseBlock : CaseStatement CaseBlock
CaseStatement   : CASE INTVAL ':' StmtList
CaseStatement   : CASE CHARVAL ':' StmtList
CaseStatement   : DEFAULT ':' StmtList
/* WHILE STATEMENT */
WhileStmt: WHILE Bstart Expr Bend BLOCKStmt
/* do while statement */
DoWhileStmt : DO BLOCKStmt WHILE Bstart Expr Bend
/* FOR LOOP STATEMENT */
ENDexpr1   : ';'
ENDexpr2   : ';'
ENDFOR : ')'

ForStmt: FOR Bstart Expr ENDexpr1 Expr ENDexpr2 Assignment ENDFOR BLOCKStmt
ForStmt: FOR Bstart Decleration ENDexpr1 Expr ENDexpr2 Assignment ENDFOR BLOCKStmt
/* cin & cout statement */
Cinstatement: CIN cinstatement2 ';'
cinstatement2 : CINOP IDENTIFIER cinstatement3
cinstatement3 : cinstatement2
coutstatement:COUT coutstatement2 ';'
coutstatement2 :COUTOP COUTSTR endstatement
endstatement: COUTOP ENDL
endstatement: coutstatement2

List of quadruples

Quadruple	Description
MOV Rx,Ry	Move Ry to Rx (Rx=Ry)
MOV x,Ry	Identifier x=Ry
MOV Rx,y	Rx=y (y is identifier or value)
ADD Rx,Ry,Rz	Rx=Ry+Rz
SUB Rx,Ry,Rz	Rx=Ry-Rz
MUL Rx,Ry,Rz	Rx=Ry*Rz
DIV Rx,Ry,Rz	Rx=Ry/Rz
CMPLE Rx,Ry	Rx<=Ry
CMPLE x,Ry	x<=Ry (x is identifier or value)

CMPLE Rx,y	Rx<=y (y is identifier or value)
CMPLE x,y	x<=y (x,y are identifiers or values)
CMPGE Rx,Ry	Rx>=Ry
CMPGE x,Ry	x>=Ry (x is identifier or value)
CMPGE Rx,y	Rx>=y (y is identifier or value)
CMPGE x,y	x>=y (x,y are identifiers or values)
CMPE Rx,Ry	Rx==Ry
CMPE x,Ry	x==Ry (x is identifier or value)
CMPE Rx,y	Rx==y (y is identifier or value)
CMPE x,y	x==y (x,y are identifiers or values)
CMPNE Rx,Ry	Rx!=Ry
CMPNE x,Ry	x!=Ry (x is identifier or value)
CMPNE Rx,y	Rx!=y (y is identifier or value)
CMPNE x,y	x!=y (x,y are identifiers or values)
CMPG Rx,Ry	Rx>Ry
CMPG x,Ry	x>Ry (x is identifier or value)
CMPG Rx,y	Rx>y (y is identifier or value)
CMPG x,y	x>y (x,y are identifiers or values)
CMPL Rx,Ry	Rx<Ry
CMPL x,Ry	x<Ry (x is identifier or value)
CMPL Rx,y	Rx<y (y is identifier or value)

CMPL x,y	x<y (x,y are identifiers or values)
AND Rx,Ry,Rz	Rx=Ry&Rz
OR Rx,Ry,Rz	Rx=Ry or Rz
NOT Rx,Ry	Rx=!Ry
JMP L	Jump to Label L
JF L	Jump if False to Label L
JT L	Jump if True to Label L