ASSIGNMENT NO: 5.

Title:

Implement the front end of a compiler that generates the three address code for a simple language.

Problem Statement:

1. Write a LEX and YACC program to generate Intermediate Code for arithmetic expression.
2. Write a LEX and YACC program to generate Intermediate Code for Subset of C.

Objective:

- To understand the fourth phase of a compiler Intermediate code generation (ICG).
- To learn and use compiler writing tools.
- Understand and learn how to write three address code for given statement.

S/W packages:

- Linux OS
- LEX
- YACC.

Theory:

Introduction:

In the analysis synthesis model of a compiler, the front end analyzes a source program and creates an intermediate representation, from which the back end generates target code. Ideally, details of the source languag are confined to the front end, and details of the target machine to the back end. The front end translates a source program into an intermediate representation from which the back end generates target code. With a suitably defined intermediate representation, a compiler for language i and machine j can then be built by combining the front end for language i with the back end for machine j. This approach to creating suite of compilers can be saved.

Benefits of using Machine independent Intermediate form are:

1. Compiler for a different machine can be created by attaching a back end for the new machine to an existing front end.

2. A machine independent code optimizer can be applied to the intermediate representation.

Three ways of Intermediate representation:

- Syntax tree
- Postfix Notation
- Three address code.

The semantic rules for generating three-address code from common programming language constructs are similar to those for constructing syntax trees or for generating postfix notation.

Three - address code:

Each statement generally consists of 3 addresses, 2 for operands & 1 for result. $x := y$ op z where x, y, z are variables, constants or compiler generated variables.

Advantages of three-address code

Complicated arithmetic expressions & of nested flow-of-control statements makes three-address code desirable for target code generation & optimization.

The use of names for the intermediate values computed by a program allows three address codes to be easily rearranged- unlike post-fix notation.

Three-address code is liberalized representation of a syntax tree or a dag in which explicit names correspond to the interior nodes of the graph.

Types of Three-Address Statements:

The common three-address statements are:

1. Assignment statements of the form $x := y$ op z, where op is a binary arithmetic or logical operation.

2. Assignment instructions of the form $x :=$ op y, where op is a unary operation. Essential unary operations include unary minus, logical negation, shift reduction, conversion operators that, for eg, convert a fixed-point number to a floating-point number.

3. Copy statements of the form $x:=y$ where the value of y is assigned to x.

4. The unconditional jump goto l. The three-address statement with label l is the next to be executed.

5. Conditional jumps such as if x rel op y goto l. This instruction applies a relational operator (<, <=, >=, etc) to x and y, & executes the statement with label l, next if x stands in relation rel op to y. If not, the three-address statement as to the foll^n if x relop y goto l is executed next usual sequence.

6. param x and call p, n for procedure calls & return y where y representing a returned value is optional for eg

param x1
param x2
param xn call p,n.

generated as part of a call of the procedure
p(x1, x2 ... xn)

7. Indexed assignments of the form $x:=y[i]$ and $x[i]:=y$

8. Address & pointer assignments of the form $x:=\&y$, $x:=*y$ and $*x:=y$.

9. Implementation of 3 Address Statements: A three address

is an abstract form of intermediate code. In a compiler these statements can be implemented as records.

Quadruples:-

- A quadruple is a record structure with four fields, which are Op1, op2, arg1, arg2 & result.

- The op field contains an internal code for the operator. The 3 address statement $x=y$ op z is implemented by placing y in arg1, z in arg2 & x in result.

- Triples:-

  1. To avoid entering temporary names into the symbol table, we might refer to temporary value by the position of the statement that computes it.

  2. If we do so, three-address statements can be represented by records with only three fields op, arg1 and arg2.

ALGORITHM:

LEX:-

1. Declaration of header file especially y.tab.h which contains declaration for Letter, Digit, expr.

2. End declaration section by %%.

3. Match regular expression.

4. If match found then convert it into char & store it in yylval.p where p is pointer declared in YACC.

5. Return token.

6. If input contains new line character (\n) then return 0.

7. If input contain ,," then return yytext[0].

8. End rule-action section by %%.

9. Declare main function
   a. open file given at command line.
   b. if any error occurs then print error & exit.
   c. assign file pointer fp to yyin.
   d. call function yylex until file ends.

10. Ends.

YACC:

1. Declaration of header files.

2. Declare structure for three address code representation having fields of argument1, argument2, operator, result.

3. Declare pointer of char type in union.

4. Declare token expr of type pointer p.

5. Give precedence to „*", "/".

6. Give precedence to „+", "-".

7. End of declaration section by %%.

8. If final expression evaluates then add it to the table of 3 address.

9. If input type is expression of the form.
   a. exp "+" exp then add to table the arg1, arg2, op.
   b. exp "-" exp then add to table the arg1, arg2, op.
   c. exp "*" exp then add to table the arg1, arg2, op.
   d. exp "/" exp then add to table the arg1, arg2, op.
   e. „(„exp")" then assign $2 to $$.
   f. Digit OR Letter then assign $1 to $$.

10. End the section by %%.
11. Declare file *yyin externally

12. Declare main funct^n & call yyparse funct^n until yyin ends.

13. Declare yyeror for any error.

14. Declare char pointer s to print error.

15. Print error message.

16. End of program.

Test input

```
main(){ int x,y,z,i;
        x=5;
        i=6;
        y=8;
        z=1;
        if(x<=7) {
            i=i+1+y*z;
            if(x==1){
                y=3;
                if(x<=2) { y=2;}
            }.
            else { x=5;}
        }
        else { i=i-1-y*z;}
        while(x>0){  x=x-1; }
        if(x>=7)
        { i=i+1+y*z;
        }
        else { i=i-1-y*z;}
        while(x>0){
            x=x-1;
        }
}
```

Steps to execute the program:

```
$ lex filename1 (eg: comp.l)
$ yacc -d filename.y (eg: comp.y)
$ gcc lex.yy.c y.tab.c.
$./a.out.
```

output :- The 3 address code.

0	=	5	x	
1	=	6	i	
2	=	8	y	
3	=	1	z	
4	<=	x	7	t0
5	IF	t0	0	20
6	+	i	1	t1
7	*	y	z	t2
8	+	t1	t2	t3
9	=	t3	i	
10	==	x	1	t4
11	IF	t4	0	17
12	=	3		y
13	<=	x	2	t5
14	IF	t5	0	t6
15	=	2		y
16	GOTO			19
17	ELSE			
18	=	5		x
19	GOTO			25
20	ELSE			
21	-	i	1	t6
22	*	y	z	t7
23	-	t6	t7	t8
24	=	t8		i
25	>	x	0	t9
26	WHILE	t9	0	30
27	-	x	1	t10

28	=	t10	x
29	GOTO		26
30	>=	x	t11

Symbol Table

i int

z int

y int

x int.

Conclusion:-

Hence, implemented the front end of a compiler that generates the 3 address code for a simple language.

Scanned by CamScanner