Assignment no.A5
Roll no. 4202

TITLE :

Intermediate code generation for sample language using LEX and YACC.

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 (If,else,while)

OBJECTIVE :

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

S/W AND HARDWARE REQUIREMENT:

1. Linux OS (Fedora 20)
2. 64-bit Intel-i5/ i7 processor computers,
3. 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 language 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 save a considerable amount of effort: m x n compilers can be built by writing just m front ends and n back ends.

Benefits of using a 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 post-fix notation.

Three-address code:

Each statements generally consists of 3 addresses, 2 for operands and 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 and of nested flow-of-control statements makes three-address code desirable for target code generation and optimization.

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

Three-address code is a liberalized representation of a syntax tree or a dag in which explicit names correspond to the interior nodes of the graph. The syntax tree and dag are represented by the three-address code sequences. Variable names can appear directly in three address statements.

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 operators, and conversion operators that, for example, 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, and executes the statement with label L next if x stands in relation rel op to y. If not, the three-address statement following if x relop y goto L is executed next, as in the usual sequence.
6. param x and call p, n for procedure calls and return y, where y representing a returned value is optional. For example,

```
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 and pointer assignments of the form x : = &y , x : = *y, and *x : = y.
9. Implementation of Three-Address Statements: A three-address statement is an abstract form of intermediate code. In a compiler, these statements can be implemented as records with fields for the operator and the operands. Three such representations are: Quadruples, Triples, Indirect triples.

A. Quadruples:

- A quadruple is a record structure with four fields, which are, op, arg1, arg2 and result.
- The op field contains an internal code for the operator. The 3 address statement x = y op z is represented by placing y in arg1, z in arg2 and x in result.
- The contents of fields arg1, arg2 and result are normally pointers to the symbol-table entries for the names represented by these fields. If so, temporary names must be entered into the symbol table as they are created.
- Fig a) shows quadruples for the assignment a : b * - c + b *- c

B. Triples:

- To avoid entering temporary names into the symbol table, we might refer to a temporary value by the position of the statement that computes it.
- If we do so, three-address statements can be represented by records with only three fields: op, arg1 and arg2.
- The fields arg1 and arg2, for the arguments of op, are either pointers to the symbol table or pointers into the triple structure (for temporary values).
- Since three fields are used, this intermediate code format is known as triples.

	op	arg1	arg2	Result
(0)	uminus	c		t1
(1)	*	b	t1	t2
(2)	uminus	c		t3
(3)	*	b	t3	t4
(4)	+	t2	t4	t5
(5)	:=	t5		a

Fig(a)

Mathematical Model :

Let S be the solution for the system.

S= { St , E , I , O , F, DD , NDD}

Where,

St is an initial state such that

St={Fl,Fy | Fl is lex file and Fy is a yacc file}. At initial state, system consists of two files, each with three sections.

E is a End state.

E = {Sc, Fc | Sc = success case which generate intermediate three-address code. and Fc = failure case which mentions that unable to generate three-address code}.

I = set of inputs

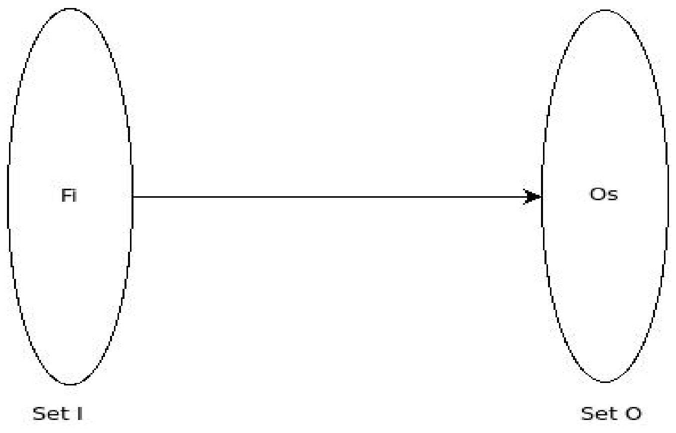
I={Fi | Fi is an input file which consists of High Level Language code}.

O =set of outputs

O={Os, Oc | Os is symbol table, Oc is console output which mention that three-address code of input program.}.

F = A set of Functions.

F={f1 | f1:I → O, function f1 generates the three-address code from input program}



Function f1: I → O

DD=Deterministic Data

DD={x | x is HLL's syntax which includes brackets, keywords, variables, functions definitions etc.}

NDD= Non-Deterministic Data

NDD={y | y is Semantic of the statements of Language}

ALGORITHM:

Write a LEX and YACC program to generate Intermediate Code for arithmetic expression

LEX program

1. Declaration of header files specially 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 and 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 contains „.‟ then return yytext[0]

8. End rule-action section by %%

10. Declare main function

    a. open file given at command line

    b.if any error occurs then print error and exit

    c. assign file pointer fp to yyin

    d.call function yylex until file ends

11. End.

YACC program:

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 three address code.

9. If input type is expression of the form.

    a. exp‟+‟exp then add to table the argument1, argument2, operator.

    b. exp‟-‟exp then add to table the argument1, argument2, operator.

    c. exp‟*‟exp then add to table the argument1, argument2, operator.

    d. exp‟/‟exp then add to table the argument1, argument2, operator.

    e. „(„exp‟)‟ then assign $2 to $$.

    f. Digit OR Letter then assigns $1 to $$.

10. End the section by %%.

11. Declare file *yyin externally.

12. Declare main function and call yyparse function untill yyin ends

13. Declare yyerror for if any error occurs.

14. Declare char pointer s to print error.

15. Print error message.

16. End of the program

.

Conclusion:

In this way the Intermediate code generated for sample language using LEX and YACC.