Assignment No: 5

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

Objective: To learn the function of compiler by:
1.To understand fourth phase of compiler: Intermediate code generation.
2.To learn and use compiler writing tools.
3.To learn how to write three address code for given statement.

Software Requirements: Lex,Yacc,GCC

Hardware Requirements: 4 GB RAM,500 GB HDD

Theory:

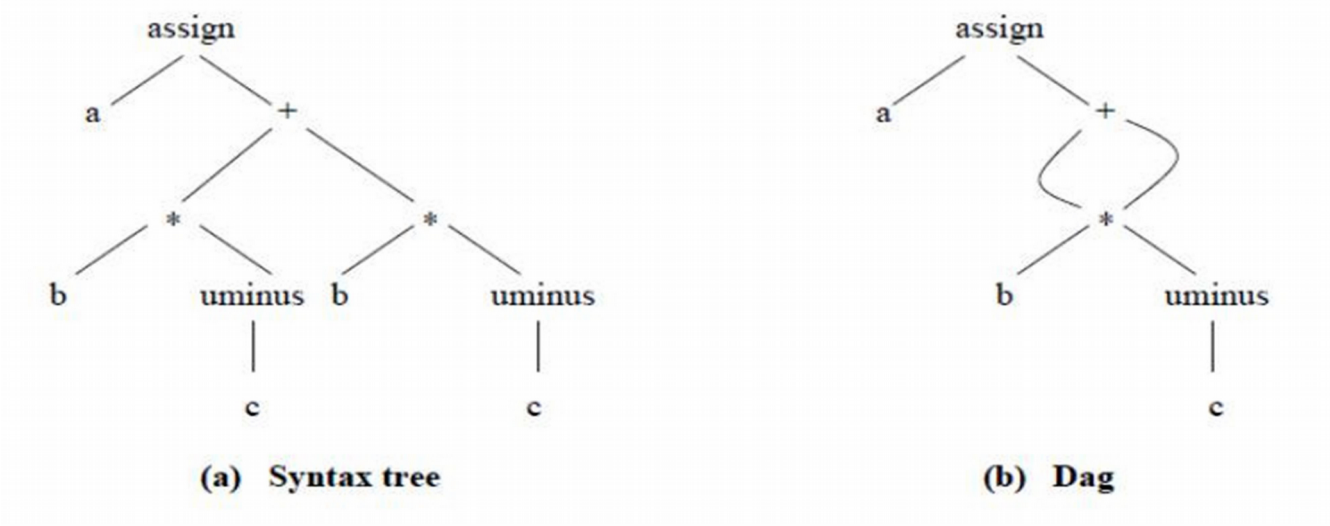
Introduction:
In the analysis - synthesis model of a compiler, the front end analyzes a source program and creates an intermediate representation, from which the back end generates target code. Ideally, details of the source language are confined to the front end, and details of the target machine to the back end. The front end translates a source program into an intermediate representation from which the back end generates target code. 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. Retargeting is facilitated. That is, a 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

Intermediate Languages:

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.
Graphical Representations:

1. Syntax tree:

A syntax tree depicts the natural hierarchical structure of a source program. A dag(Directed Acyclic Graph) gives the same information but in a more compact way because common sub expressions are identified. A syntax tree and dag for the assignment statement a : =b * -c + b * -c are as follows:



2. Postfix notation:

Postfix notation is a linearized representation of a syntax tree; it is a list of the nodes of the tree in which a node appears immediately after its children. The postfix notation for the syntax tree given above is

a b c uminus * b c uminus * + assign

3. Three-Address Code:

Three-address code is a sequence of statements of the general form x : = y op z

Advantages of three-address code:

-The unraveling of 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 postfix 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:

- Assignment statements of the form x : = y op z, where op is a binary arithmetic or logical operation.
- 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.
- Copy statements of the form x : = y where the value of y is assigned to x.
- The unconditional jump goto L. The three-address statement with label L is the next to be executed.
- Conditional jumps such as if x relop 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 relop to

y. If not, the three-address statement following if x relop y goto L is executed next, as in the usual sequence.

- Indexed assignments of the form x : = y[i] and x[i] : = y.
- Address and pointer assignments of the form x : = &y , x : = *y, and *x : = y.

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.

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.

Three fields are used, this intermediate code format is known as triples.

Fig shows the triples for the assignment statement a: = b * c + b * c.

	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): Quadruples

	op	Arg1	arg2
(0)	uminus	c	
(1)	*	b	(0)
(2)	uminus	c	
(3)	*	b	(2)
(4)	+	(1)	(3)
(5)	assign	a	(4)

Fig b): Triple

Quadruples & Triple representation of three-address statement

C. Indirect triples:

Indirect triple representation is the listing pointers to triples rather-than listing the triples themselves.

-Let us use an array statement to list pointers to triples in the desired order.

Fig shows the indirect triple representation.

	Statement
(0)	(14)
(1)	(15)
(2)	(16)
(3)	(17)
(4)	(18)
(5)	(19)

	op	arg1	arg2
(14)	uminus	c	
(15)	*	b	(14)
(16)	uminus	c	
(17)	*	b	(16)
(18)	+	(15)	(17)
(19)	assign	a	(18)

Fig c): Indirect triples representation of three address statements

Steps to execute the program
$ lex filename.l
$ yacc -d filename.y
B.E.C.E(Sem-II) [2018-19]
(eg: comp.l)
(eg: comp.y)
$cc lex.yy.c y.tab.c –ll –ly -lm
$./a .out

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 %%
9. 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
10. End
   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.

Three address code function will print the values from the structure in the form first temporary variable, argument1, operator, argument2.

Quadruple Function will print the values from the structure in the form first operator, argument1, argument2, result field.

Triple Function will print the values from the structure in the form first argument1, argument2, and operator. The temporary variables in this form are integer / index instead of variables.

Conclusion:
Thus we have studied how to generate intermediate code. 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 that we have studied.

Code:

icg1.y

```
%{
#include<string.h>
#include<stdio.h>
#include<ctype.h>
%}

%token ID NUM
%right '='
%left '+' '-'
%left '*' '/'
%left UMINUS

%%
S:ID{push();} '='{push();} E{codegen_assign();}
;
E:E '+'{push();} T{codegen();}
|E '-'{push();} T{codegen();}
|T
;
T:T '*'{push();} F{codegen();}
|T '/'{push();} F{codegen();}
|F
;
F: '(' E ')'
|'-'{push();} F{codegen_umin();} %prec UMINUS
|ID{push();}
|NUM{push();}
;
%%
#include"lex.yy.c"
char st[100][10];
int top=0;
char i_[2]="0";
char temp[2]="t";

int main()
{
printf("enter the expression:");
yyparse();
return 0;
}
push()
{
strcpy(st[++top],yytext);
}
```

```
int codegen()
{
strcpy(temp,"t");
strcat(temp,i_);
printf("%s=%s %s %s \n",temp,st[top-2],st[top-1],st[top]);
top-=2;
strcpy(st[top],temp);
i_[0]++;
return 0;
}

int codegen_umin()
{
strcpy(temp,"t");
strcat(temp,i_);
printf("%s=-%s\n",temp,st[top]);
top--;
strcpy(st[top],temp);
i_[0]++;
return 0;
}
int codegen_assign()
{
printf("%s=%s\n",st[top-2],st[top]);
top-=2;
return 0;
}

int yyerror(char *s)
{
printf("%s\n",s);
return 0;
}
```

icg1.l

```
ALPHA [A-Za-z]
DIGIT [0-9]
%{
 #include"y.tab.h"
%}

%%
{ALPHA}({ALPHA}|{DIGIT})* return ID;
{DIGIT}+ {yylval=atoi(yytext);return NUM;}
[\n\t] yyterminate();
. return yytext[0];
%%
int yywrap()
{
return 1;
}
```

Output:

```
/*

(base) dell@dell-Inspiron-15-3567:~/BEALL2/Compiler/Compiler/Assignment5$ ./a.out
enter the expression:a=b+c*d
t0=c * d
t1=b + t0
a=t1
(base) dell@dell-Inspiron-15-3567:~/BEALL2/Compiler/Compiler/Assignment5$

*/
```