SEMBODAI RUKMANI VARATHARAJAN ENGINEERING COLLEGE

SEMBODAI – 614809

(Approved By AICTE,Newdelhi – Affiliated To ANNA UNIVERSITY::Chennai)

CS 6612 COMPILER LABORATORY

(REGULATION-2013)

LAB MANUAL

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Prepared By,
Ezhilvathani.A
AP/CSE/SRVEC

Approved By,
Radha.D
HOD/CSE/SRVEC

(REGULATION 2013)
AS PER ANNA UNIVERSITY SYLLABUS

SYLLABUS

1. Implementation of Symbol Table
2. Develop a lexical analyzer to recognize a few patterns in C.
   (Ex. identifiers, constants, comments, operators etc.)
3. Implementation of Lexical Analyzer using Lex Tool
4. Generate YACC specification for a few syntactic categories.
   a. Program to recognize a valid arithmetic expression that usesoperator +, - , * and /.
   b. Program to recognize a valid variable which starts with a letterfollowed by any number of letters or digits.
   c. Implementation of Calculator using LEX and YACC
5. Convert the BNF rules into Yacc form and write code to generate Abstract Syntax Tree.
6. Implement type checking
7. Implement control flow analysis and Data flow Analysis
8. Implement any one storage allocation strategies(Heap,Stack,Static)
9. Construction of DAG
10. Implement the back end of the compiler which takes the three address code and produces the 8086 assembly language instructions that can be assembled and run using a 8086 assembler. The target assembly instructions can be simple move, add, sub, jump. Also simple addressing modes are used.
11. Implementation of Simple Code Optimization Techniques (Constant Folding., etc.)

TABLE OF CONTENT

IMPLEMENTATION OF SYMBOL TABLE

Ex. No: 1 **Date:**

AIM:-

To write a program for implementing Symbol Table using C.

ALGORITHM:-

STEP 1: Start the program for performing insert, display, delete, search and modify option in symbol table

STEP 2: Define the structure of the Symbol Table

STEP 3: Enter the choice for performing the operations in the symbol Table

STEP 4: If the entered choice is 1, search the symbol table for the symbol to be inserted. If the symbol is already present, it displays "Duplicate Symbol". Else, insert the symbol and the corresponding address in the symbol table.

STEP 5: If the entered choice is 2, the symbols present in the symbol table are displayed.

STEP 6: If the entered choice is 3, the symbol to be deleted is searched in the symbol table.

STEP 7: If it is not found in the symbol table it displays "Label Not found". Else, the symbol is deleted.

STEP 8: If the entered choice is 5, the symbol to be modified is searched in the symbol table. The label or address or both can be modified.

PROGRAM:-

```
# include <stdio.h>
# include <conio.h>
# include <alloc.h>
# include <string.h>
# define null 0
int size=0;
void insert();
void del();
int search(char lab[]);
void modify();
void display();
struct symbtab
{
char label[10];
int addr;
struct symtab *next;
};
struct symbtab *first,*last;
```

```
void main()
{
int op;
int y;
char la[10];
clrscr();
do
{
printf("\nSYMBOL TABLE IMPLEMENTATION\n");
printf("1. INSERT\n");
printf("2. DISPLAY\n");
printf("3. DELETE\n");
printf("4. SEARCH\n");
printf("5. MODIFY\n");
printf("6. END\n");

printf("Enter your option : ");
scanf("%d",&op);
switch(op)
{
case 1:
insert();
display();
break;
case 2:
display();
break;
case 3:
del();
display();
break;
case 4:
printf("Enter the label to be searched : ");
scanf("%s",la);
y=search(la);
if(y==1)
{
printf("The label is already in the symbol Table");
}
else
{
printf("The label is not found in the symbol table");
```

```
}
break;
case 5:
modify();
display();
break;
case 6:
break;
}

}
while(op<6);
getch();
}
void insert()
{
int n;
char l[10];
printf("Enter the label : ");
scanf("%s",l);
n=search(l);
if(n==1)
{
printf("The label already exists. Duplicate cant be inserted\n");
}
else
{
struct symbtab *p;
p=malloc(sizeof(struct symbtab));
strcpy(p->label,l);
printf("Enter the address : ");
scanf("%d",&p->addr);
p->next=null;
if(size==0)
{
first=p;
last=p;
}
else
{
last->next=p;
last=p;
```

```
}
size++;
}

}
void display()
{
int i;
struct symbtab *p;
p=first;
printf("LABEL\tADDRESS\n");
for(i=0;i<size;i++)
{
printf("%s\t%d\n",p->label,p->addr);
p=p->next;
}
}
int search(char lab[])
{
int i,flag=0;
struct symbtab *p;
p=first;
for(i=0;i<size;i++)
{
if(strcmp(p->label,lab)==0)
{
flag=1;
}
p=p->next;
}
return flag;
}
void modify()
{
char l[10],nl[10];
int add, choice, i, s;
struct symbtab *p;
p=first;

printf("What do you want to modify?\n");
printf("1. Only the label\n");
printf("2. Only the address of a particular label\n");
```

```
printf("3. Both the label and address\n");
printf("Enter your choice : ");
scanf("%d",&choice);
switch(choice)
{
case 1:
printf("Enter the old label\n");
scanf("%s",l);
printf("Enter the new label\n");
scanf("%s",nl);
s=search(l);
if(s==0)
{
printf("NO such label");
}
else
{
for(i=0;i<size;i++)
{
if(strcmp(p->label,l)==0)
{
strcpy(p->label,nl);
}
p=p->next;
}
}
break;
case 2:
printf("Enter the label whose address is to modified\n");
scanf("%s",l);
printf("Enter the new address\n");
scanf("%d",&add);

s=search(l);
if(s==0)
{
printf("NO such label");
}
else
{
for(i=0;i<size;i++)
{
```

```
if(strcmp(p->label,l)==0)
{
p->addr=add;
}
p=p->next;
}
}
break;
case 3:
printf("Enter the old label : ");
scanf("%s",l);
printf("Enter the new label : ");
scanf("%s",nl);
printf("Enter the new address : ");
scanf("%d",&add);
s=search(l);
if(s==0)
{
printf("NO such label");
}
else
{
for(i=0;i<size;i++)
{
if(strcmp(p->label,l)==0)
{

strcpy(p->label,nl);
p->addr=add;
}
p=p->next;
}
}
break;
}
}
void del()
{
int a;
char l[10];
struct symbtab *p,*q;
p=first;
```

```
printf("Enter the label to be deleted\n");
scanf("%s",l);
a=search(l);
if(a==0)
{
printf("Label not found\n");
}
else
{
if(strcmp(first->label,l)==0)
{
first=first->next;
}
else if(strcmp(last->label,l)==0)
{
q=p->next;
while(strcmp(q->label,l)!=0)
{
p=p->next;
q=q->next;

}
p->next=null;
last=p;
}
else
{
q=p->next;
while(strcmp(q->label,l)!=0)
{
p=p->next;
q=q->next;
}
p->next=q->next;
}
size--;
}
}
```

OUTPUT:-

```
SYMBOL TABLE IMPLEMENTATION
INSERT
DISPLAY
DELETE
MODIFY
END
Enter the option:1
Enter the label:data
Enter the address:123
LABEL                ADDRESS
Data                 123
SYMBOL TABLE IMPLEMENTATION
INSERT
DISPLAY
DELETE
MODIFY
END
Enter your option:1
Enter the label:numbers
Enter the address:456
SYMBOL TABLE IMPLEMENTATION
INSERT
DISPLAY
DELETE
MODIFY
END
Enter your option:3
Enter the label to be deleted
Data
LABEL                ADDRESS
Numbers              456
SYMBOL TABLE IMPLEMENTATION
INSERT
DISPLAY
DELETE
MODIFY
END
Enter your option:5
What do you want to modify?
1.only the label
```

2. only the address of a particular label
3. Both the label and address
Enter your choice:1
Enter the old label
Number
Enter the new label
Hello

RESULT:-

Thus the program for implementation of Symbol Table is executed and verified.

IMPLEMENTATION OF LEXICAL ANALYZER USING C

Ex. No: 2 **Date:**

AIM:-

To develop a lexical analyzer to recognize a few patterns using c.

ALGORITHM:-

STEP 1: Read the input Expression
STEP 2: Check whether input is alphabet or digits then store it as identifier
STEP 3: If the input is is operator store it as symbol
STEP 4: Check the input for keywords

PROGRAM:-

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
void removeduplicate();
void final();
int Isiden(char ch);
int Isop(char ch);
int Isdel(char ch);
int Iskey(char * str);
void removeduplicate();

char op[8]={'+','-','*','/','=','<','>','%'};
char del[8]={'}','{',';','(',')','[',']',','};
char *key[]={"int","void","main","char","float"};

//char *operato[]={"+","-","/","*","<",">","=","%","<=",">=","++"};

int idi=0,idj=0,k,opi=0,opj=0,deli=0,uqdi=0,uqidi=0,uqoperi=0,kdi=0,liti=0,ci=0;
int uqdeli[20],uqopi[20],uqideni[20],l=0,j;
char uqdel[20],uqiden[20][20],uqop[20][20],keyword[20][20];
char iden[20][20],oper[20][20],delem[20],litral[20][20],lit[20],constant[20][20];

void lexanalysis(char *str)
 {
int i=0;
while(str[i]!='\0')
   {
```

```
if(Isiden(str[i]))     //for identifiers
      {
while(Isiden(str[i]))
       {
iden[idi][idj++]=str[i++];
       }
iden[idi][idj]='\0';
idi++;idj=0;
      }
else
if(str[i]=="")         //for literals
       {
lit[l++]=str[i];
for(j=i+1;str[j]!="";j++)
         {
lit[l++]=str[j];
         }
lit[l++]=str[j];lit[l]='\0';
strcpy(litral[liti++],lit);
        i=j+1;
        }
else
if(Isop(str[i]))       // for operators
        {
while(Isop(str[i]))
          {
oper[opi][opj++]=str[i++];
          }
oper[opi][opj]='\0';
opi++;opj=0;
        }
else
if(Isdel(str[i]))     //for delemeters
        {
while(Isdel(str[i]))
          {
delem[deli++]=str[i++];
          }
         }
else
         {
i++;
```

```
        }
    }

removeduplicate();
final();
}

int Isiden(char ch)
 {
if(isalpha(ch)||ch=='_'||isdigit(ch)||ch=='.')
return 1;
else
return 0;
 }

int Isop(char ch)
 {
int f=0,i;
for(i=0;i<8&&!f;i++)
  {
if(ch==op[i])
    f=1;
  }
return f;
}

int Isdel(char ch)
 {
int f=0,i;
for(i=0;i<8&&!f;i++)
  {
if(ch==del[i])
    f=1;
  }
return f;
 }

int Iskey(char * str)
{
int i,f=0;
for(i=0;i<5;i++)
  {
```

```
if(!strcmp(key[i],str))
     f=1;
   }
return f;
}

void removeduplicate()
 {
int i,j;
for(i=0;i<20;i++)
   {
uqdeli[i]=0;
uqopi[i]=0;
uqideni[i]=0;
   }
for(i=1;i<deli+1;i++)  //removing duplicate delemeters
    {
if(uqdeli[i-1]==0)
    {
uqdel[uqdi++]=delem[i-1];
for(j=i;j<deli;j++)
        {
if(delem[i-1]==delem[j])
uqdeli[j]=1;
        }
    }
    }

for(i=1;i<idi+1;i++)  //removing duplicate  identifiers
     {
if(uqideni[i-1]==0)
        {
strcpy(uqiden[uqidi++],iden[i-1]);
for(j=i;j<idi;j++)
        {
if(!strcmp(iden[i-1],iden[j]))
uqideni[j]=1;
        }
        }
   }

for(i=1;i<opi+1;i++)  //removing duplicate  operators
```

```
    {
if(uqopi[i-1]==0)
         {
strcpy(uqop[uqoperi++],oper[i-1]);
for(j=i;j<opi;j++)
          {
if(!strcmp(oper[i-1],oper[j]))
uqopi[j]=1;
          }
         }
    }

 }
void final()
 {
int i=0;
idi=0;
for(i=0;i<uqidi;i++)
  {
if(Iskey(uqiden[i]))       //identifying keywords
strcpy(keyword[kdi++],uqiden[i]);
else
if(isdigit(uqiden[i][0]))    //identifying constants
strcpy(constant[ci++],uqiden[i]);
else
strcpy(iden[idi++],uqiden[i]);
  }

// printing the outputs

printf("\n\tDelemeter are : \n");
for(i=0;i<uqdi;i++)
printf("\t%c\n",uqdel[i]);

printf("\n\tOperators are : \n");
for(i=0;i<uqoperi;i++)
 {
printf("\t");
puts(uqop[i]);
 }

printf("\n\tIdentifiers are : \n");
```

```
for(i=0;i<idi;i++)
 {
printf("\t");
puts(iden[i]);
 }

printf("\n\tKeywords are : \n");
for(i=0;i<kdi;i++)
 {
printf("\t");
puts(keyword[i]);
 }

printf("\n\tConstants are :\n");
for(i=0;i<ci;i++)
  {
printf("\t");
puts(constant[i]);
  }

printf("\n\tLiterals are :\n");
for(i=0;i<liti;i++)
 {
printf("\t");
puts(litral[i]);
 }
}
void main()
{
char str[50];
  //clrscr();
printf("\nEnter the string : ");
scanf("%[^\n]c",str);
lexanalysis(str);
  //getch();
}
```

OUTPUT:-

```
Enter the string:#include<stdio.h>
Delemeter are:
Operators are:
<
>
Identifiers are:
Include
Stdio.h
Keywords are:
Constants are:
Literals are:
Enter the string: void main()
Delemeter are:
(
)
Operators are:
Identifiers are:
Keywords are:
Void
Main
Constants are:
Literals are:
Enter the string: int a=5;
Delemeter are:
;
Operators are:
=
Identifiers are:
a
Keywords are:
Int
Constants are:
5
Literals are:
```

RESULT:-

Thus the c program for lexical analyzer has been executed successfully.

LEXICAL ANALYSIS USING LEX

Ex. No: 3 **Date:**

AIM:-

To write a C program to implement lexical analysis using LEX tool.

GENERAL FORMAT:-

Lex program consists of three parts.
Declaration %{ …. }%
Translation rules %% ….. %%
Auxilary procedure.
The declaration section includes declaration of C variables, constants and so on.
Translation rule of lex program are statements of the form:
Pattern 1 {action}
Pattern 2 {action}
…
…
Pattern N {action}
Auxiliary procedure includes definition of C variables, functions, and so on.

ALGORITHM:-

STEP 1: In the Auxiliary procedure section, write the main function which should get a file as the command line argument and start the scanning process.
STEP 2: In the declaration section, declare the global variables.
STEP 3: In the rules section, write down the appropriate patterns and its actions to produce the tokens.

PROGRAM:-

```
%{
int COMMENT=0;
%}

identifier [_a-zA-Z][_a-zA-Z0-9]*

%%


#.*                {printf("\n%s is a PREPROCESSOR DIRECTIVE",yytext);}

int |
float |
```

```
char |
while |
for |
do |
if |
break |
continue |
void |
switch |
case |
long |
struct |
const |
typedef |
return |
else |
goto                    {printf("\n%s is a KEYWORD",yytext);}

"/*"{identifier}        {COMMENT=1;
printf("\n\n\t COMMENT BIGNS\n");}

"*/"                    {COMMENT=0;
printf("\n\n\t COMMENT ENDS\n");}

{identifier}\(          { if(!COMMENT)
printf("\n%s FUNCTION ",yytext);}

\{                      {if(!COMMENT)
                        printf(" BLOCK BEGINS");}

\}                      {if(!COMMENT)
printf(" BLOCK ENDS");}

{identifier}(\[[0-9]*\])?    {if(!COMMENT)
                        printf("\n%s IDENTIFIER",yytext);}

\".*\"                      {if(!COMMENT)
printf("\n %s is a STRING",yytext);}

[0-9]+                  {if(!COMMENT)
                        printf("\n %s is a NUMBER",yytext);}
```

```
\)(\;)?                    {if(!COMMENT)
printf(""); ECHO; }

\(                          ECHO ;

=                          {if(!COMMENT)
printf("\n%s is an ASSIGNMENT OPERATOR",yytext);}

\<= |
\>= |
\< |
== |
\>                  {if(!COMMENT)
printf("\n\t%s is a RELATIONAL OPERATOR",yytext);}


%%

int main(int argc,char *argv[])
{
if(argc>1)
{
FILE *file;
file=fopen(argv[1],"r");
if(!file)
{
printf("could not open %s\n",argv[1]);
exit(0);
}
yyin=file;
}
yylex();
printf("\n\n");
return 0;
}

int yywrap()
{
return 1;
}
```

SAMPLE INPUT PROGRAM:

```
main()
{
int a=5,b=10;
printf("\n a=%d, b=%d",a,b);
}
```

OUTPUT:-

```
$lex lex1.l
$cc lex.yy.c –ll
$./a.out sample.c
Main<FUNCTION>
BLOCK BEGINS
A IDENTIFIER
= is an ASSIGNMENT OPERATOR
is NUMBER,
b IDENTIFIER
= is an ASSIGNMENT OPERATOR
is NUMBER;
printf<FUNCTION
“\n a=%d,b=%d” is a STRING,
a IDENTIFIER,
b IDENTIFIER>;
BLOCK ENDS
```

RESULT:-

Thus the C program to implement lexical analysis using LEX tool has been executed successfully.

RECOGNIZE A VALID ARITHMETIC EXPRESSION THAT USES OPERATOR +, - , * AND /.

Ex. No: 4(a) **Date:**

AIM:-

To write a program that recognize a valid arithmetic expression that uses operator +,-,* and /.

PROGRAM:-

```
%{      #include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
%}
%token      num  let
%left   '+'   '-'
%left    '*'   '/'
%%
Stmt :          Stmt   '\n'          {   printf ("\n.. Valid Expression.. \n");   exit(0);  }
  |    expr
  |
  |    error    '\n'        {    printf ("\n..Invalid ..\n");     exit(0); }
  ;
expr :          num
           |    let
  |    expr  ' + '   expr
           |    expr   ' - '   expr
           |    expr   ' * '   expr
           |    expr   ' / '   expr
           |    '('     expr    ')'
           ;
```

```
%%

main ( )

{

printf (“Enter an expression to validate : ”);

yyparse ( );

}


yylex()

{

      int ch;

      while ( ( ch = getchar() ) == '  ' );

      if ( isdigit(ch) )

        return  num;                    // return token num

      if ( isalpha(ch) )

                return  let;              // return token let

      return ch;

}


yyerror (char *s)

{

   printf ( “%s”, s );

}
```

OUTPUT:-

```
$ yacc -d prog.y
$ cc y.tab.c -ll
./a.out
 4 + t                          Accepted
( 5 * d + 5 )                   Accepted
 a + * 5                        Rejected
```

RESULT:-

Thus the C program to valid arithmetic expression using LEX tool has been executed successfully.

YACC PROGRAM TO RECOGNIZE AN VALID VARIABLE WHICH STARTS WITH LETTER FOLLOWED BY A DIGIT.

Ex. No: 4(b) Date:

AIM:-

To write a program that recognizesa valid variable which starts with letter followed by a digit using yacc tool.

PROGRAM:-

```
%{
#include "y.tab.h"
%}
%%
[0-9] return digit;
[a-z] return letter;
[\n] return yytext[0];
. return 0;
%%
/* Yacc program to validate the given variable */
%{
#include<type.h>
%}
% token digit letter;
%%
ident : expn '\n' { printf ("valid\n"); exit (0); }
;
expn : letter
| expn letter
| expn digit
| error { yyerror ("invalid \n"); exit (0); }
;
%%
main()
{
yyparse();
}
yyerror (char *s)
{
printf("%s", s);
}
```

```
/* Yacc program which has c program to pass tokens */
%{
#include <stdio.h>
#include <ctype.h>
%}
%token LETTER DIGIT
%%
st:st LETTER DIGIT '\n' {printf("\nVALID");}
| st '\n'
|
| error '\n' {yyerror("\nINVALID");yyerrok;}
;
%%
main()
{
yyparse();
}
yylex()
{
char c;
while((c=getchar())==' ');
if(islower(c)) return LETTER;
if(isdigit(c)) return DIGIT;
return c;
}
yyerror(char *s)
{
printf("%s",s);
}
```

OUTPUT:-

a(a+0)
Valid
0(0+b)
Invalid

RESULT:-

Thus the C program to recognize an valid variable which starts with letter followed by a digit YACC tool has been executed successfully.

IMPLEMENTATION OF CALCULATOR USING YACC

Ex. No: 4.c **Date:**

AIM:-

To write a program for implementing a calculator for computing the given expression using semantic rules of the YACC tool.

ALGORITHM:-

STEP 1: A Yacc source program has three parts as follows:

Declarations
%%
translation rules
%%
supporting C routines

STEP 2: Declarations Section:

This section contains entries that:
i. Include standard I/O header file.
ii. Define global variables.
iii. Define the list rule as the place to start processing.
iv. Define the tokens used by the parser.
v. Define the operators and their precedence.

STEP 3: Rules Section:

The rules section defines the rules that parse the input stream. Each rule of a grammar production and the associated semantic action.

STEP 4: Programs Section:

The programs section contains the following subroutines. Because these subroutines are included in this file, it is not necessary to use the yacc library when processing this file.

STEP 5: Main- The required main program that calls the yyparse subroutine to start the program.

STEP 6: yyerror(s) -This error-handling subroutine only prints a syntax error message.

STEP 7: yywrap -The wrap-up subroutine that returns a value of 1 when the end of input occurs. The calc.lex file contains include statements for standard input and output, as programmar file information if we use the -d flag with the yacc command. The y.tab.h file contains definitions for the tokens that the parser program uses.

STEP 8: calc.lex contains the rules to generate these tokens from the input stream.

PROGRAM:-

Cal.l

```
%{
#include<stdio.h>
#include<math.h>
#include"y.tab.h"
%}
%%
([0-9]+|([0-9]*\.[0-9]+)([eE][-+]?[0-9]+)?)
{yylval.dval=atof(yytext);
return NUMBER;}
MEM {return MEM;}
[\t];
\$ {return 0;}
\n {return yytext[0];}
. {return yytext[0];}
%%
```

Cal.y

```
%{
#include<stdio.h>
#include<math.h>
double memvar;
%}
%union
{
double dval;
}
%token<dval> NUMBER
%token<dval> MEM
%left '-' '+'
%left '*' '/'
%nonassoc UMINUS
%type<dval> expression
%%
start:statement '\n'
|start statement '\n'
statement:MEM '=' expression {memvar=$3;}
|expression {printf("answer=%g\n",$1);}
;
expression:expression'+'expression {$$=$1+$3;}
|expression'-'expression {$$=$1-$3;}
|expression'*'expression {$$=$1*$3;}
|expression'/'expression {if($3==0)
yyerror("divide by zero");
else
```

```
$$=$1/$3;
};
expression:'-'expression %prec UMINUS {$$= -$2;}
|'('expression')' {$$=$2;}
|NUMBER {$$=$1;}
|MEM {$$=memvar;};
%%
int main(void)
{
printf("Enter the expression");
yyparse();
printf("\n\n");
return 0;
}
int yywrap()
{

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

OUTPUT:-

```
[CSE@localhost ~]$ lex cal.l
[CSE@localhost ~]$ yacc -d cal.y
[CSE@localhost ~]$ cc lex.yy.c y.tab.c -ll
[CSE@localhost ~]$ ./a.out
Enter the expression5+3
answer=8
[cse@NFSSERVER ~]$ ./a.out
Enter the expression5+-5
answer=0
[cse@NFSSERVER ~]$ ./a.out
Enter the expression+5/
syntax error
```

RESULT:-

Thus the program for implementation of Calculator using YACC toolis executed and verified.

CONVERT THE BNF RULES INTO YACC FORM AND WRITE CODE TO GENERATE ABSTRACT SYNTAX TREE.

Ex. No: 5 **Date:**

AIM:-

To Design and develop the BNF rules into Yacc form and write code to generate abstract syntax tree.

ALGORITHM:-

STEP1:Start
STEP2: declare the declarations as a header file
{include<ctype.h>}
STEP3:token digit
STEP4:define the translations rules like line, expr, term, factor
Line:exp „\n‟ {print(“\n %d \n”,$1)}
Expr:expr‟+‟ term ($$=$1=$3}
Term:term „+‟ factor($$ =$1*$3}
Factor
Factor:‟(„enter‟) „{$$ =$2)
% %
STEP5:define the supporting C routines
STEP6:Stop.

PROGRAM:-

```
<int.l>
%
{
 #include"y.tab.h"
 #include<stdio.h>
 #include<string.h>
int LineNo=1;
 %
}
identifier [a-zA-Z][_a-zA-Z0-9]*
number [0-9]+|([0-9]*\.[0-9]+)
%%
main\(\) return MAIN;
if return IF;
else return ELSE;
while return WHILE;
int |
char |
float return TYPE;
{identifier} {strcpy(yylval.var,yytext);
```

```
return VAR;}
{number} {strcpy(yylval.var,yytext);
return NUM;}
< |> |>= |<= |==
{strcpy(yylval.var,yytext);
return RELOP;}
[ \t] ;
 \n LineNo++;
. return yytext[0];
%%
<int.y>
%{
#include<string.h>
 #include<stdio.h>
struct quad{
char op[5];
char arg1[10];
char arg2[10];
char result[10];
}QUAD[30];
struct stack{
int items[100];
int top; }stk;
int Index=0,tIndex=0,StNo,Ind,tInd;
extern int LineNo;
%}
 %union{
char var[10];
}
%token <var> NUM VAR RELOP
 %token MAIN IF ELSE WHILE TYPE
%type <var> EXPR ASSIGNMENT CONDITION IFST ELSEST WHILELOOP
%left '-' '+'
%left '*' '/'
%%
PROGRAM : MAIN BLOCK
;
BLOCK: '{' CODE '}'
;
CODE: BLOCK
 | STATEMENT CODE
 | STATEMENT
 ;
STATEMENT: DESCT ';'
 | ASSIGNMENT ';'
| CONDST
 | WHILEST
; DESCT: TYPE VARLIST
```

```
 ; VARLIST: VAR ',' VARLIST
 | VAR
;
ASSIGNMENT: VAR '=' EXPR{
strcpy(QUAD[Index].op,"=");
strcpy(QUAD[Index].arg1,$3);
strcpy(QUAD[Index].arg2,"");
strcpy(QUAD[Index].result,$1);
strcpy($$,QUAD[Index++].result); } ;
EXPR: EXPR '+' EXPR {AddQuadruple("+",$1,$3,$$);}
 | EXPR '-' EXPR {AddQuadruple("-",$1,$3,$$);}
 | EXPR '*' EXPR {AddQuadruple("*",$1,$3,$$);}
| EXPR '/' EXPR {AddQuadruple("/",$1,$3,$$);}
| '-' EXPR {AddQuadruple("UMIN",$2,"",$$);}
 | '(' EXPR ')' {strcpy($$,$2);}
 | VAR
| NUM
;
CONDST: IFST{
Ind=pop();
sprintf(QUAD[Ind].result,"%d",Index);
Ind=pop();
sprintf(QUAD[Ind].result,"%d",Index); } | IFST ELSEST ;
 IFST: IF '(' CONDITION ')'
{
strcpy(QUAD[Index].op,"==");
strcpy(QUAD[Index].arg1,$3);
strcpy(QUAD[Index].arg2,"FALSE");
strcpy(QUAD[Index].result,"-1");
push(Index);
Index++;
}
BLOCK {
strcpy(QUAD[Index].op,"GOTO");
strcpy(QUAD[Index].arg1,"");
strcpy(QUAD[Index].arg2,"");
strcpy(QUAD[Index].result,"-1");
push(Index);
Index++;
 };
ELSEST: ELSE{
tInd=pop();
Ind=pop();
push(tInd);
sprintf(QUAD[Ind].result,"%d",Index);
}
BLOCK{
 Ind=pop();
```

```
sprintf(QUAD[Ind].result,"%d",Index);
};
CONDITION: VAR RELOP VAR {AddQuadruple($2,$1,$3,$$);
StNo=Index-1;
}
| VAR
| NUM
;
WHILEST: WHILELOOP{
Ind=pop();
sprintf(QUAD[Ind].result,"%d",StNo);
Ind=pop();
sprintf(QUAD[Ind].result,"%d",Index); }
 ;
WHILELOOP: WHILE '(' CONDITION ')' {
strcpy(QUAD[Index].op,"==");
strcpy(QUAD[Index].arg1,$3);
strcpy(QUAD[Index].arg2,"FALSE");
strcpy(QUAD[Index].result,"-1");
push(Index);
Index++;
}
BLOCK {
strcpy(QUAD[Index].op,"GOTO");
strcpy(QUAD[Index].arg1,"");
strcpy(QUAD[Index].arg2,"");
strcpy(QUAD[Index].result,"-1");
push(Index);
Index++;
}
;
%%
extern FILE *yyin;
int main(int argc,char *argv[]) {
FILE *fp;
int i; if(argc>1){
fp=fopen(argv[1],"r");
if(!fp) {
printf("\n File not found");
exit(0);
}
yyin=fp;
 }
yyparse();
printf("\n\n\t\t ----------------------------""\n\t\t Pos Operator Arg1 Arg2 Result" "\n\t\t ----------
---------"); for(i=0;i<Index;i++)
{
```

```
printf("\n\t\t %d\t %s\t %s\t %s\t
%s",i,QUAD[i].op,QUAD[i].arg1,QUAD[i].arg2,QUAD[i].result); } printf("\n\t\t ----------
---------"); printf("\n\n"); return 0;
}
void push(int data)
{ stk.top++;
if(stk.top==100)
{ printf("\n Stack overflow\n");
exit(0);
}
stk.items[stk.top]=data;
} int pop()
{ int data;
if(stk.top==-1)
{ printf
("\n Stack underflow\n");
exit(0);}
data=stk.items[stk.top--];
return data;
} void AddQuadruple(char op[5],char arg1[10],char arg2[10],char result[10]) {
strcpy(QUAD[Index].op,op);
strcpy(QUAD[Index].arg1,arg1);
strcpy(QUAD[Index].arg2,arg2);
sprintf(QUAD[Index].result,"t%d",tIndex++);
strcpy(result,QUAD[Index++].result); }
yyerror()
{
printf("\n Error on line no:%d",LineNo);
}
```

Input:

```
$vi test.c
main()
{
int a,b,c;
if(a<b)
{
a=a+b;
}
while(a<b)
{
a=a+b;
}
if(a<=b)
{
c=a-b;
 }
else
{
```

```
c=a+b;
 }
}case 1:
printf("Enter the old label\n");
scanf("%s",l);
printf("Enter the new label\n");
scanf("%s",nl);
s=search(l);
if(s==0)
{
printf("NO such label");
}
else
{
for(i=0;i<size;i++)
{
if(strcmp(p->label,l)==0)
{
strcpy(p->label,nl);
}
p=p->next;
}
}
break;
case 2:
printf("Enter the label whose address is to modified\n");
scanf("%s",l);
printf("Enter the new address\n");
scanf("%d",&add);

s=search(l);
if(s==0)
{
printf("NO such label");
}
else
{
for(i=0;i<size;i++)
{
if(strcmp(p->label,l)==0)
{
p->addr=add;
```

```
}
p=p->next;
}
}
break;
case 3:
printf("Enter the old label : ");
scanf("%s",l);
printf("Enter the new label : ");
scanf("%s",nl);
printf("Enter the new address : ");
scanf("%d",&add);
s=search(l);
if(s==0)
{
printf("NO such label");
}
else
{
for(i=0;i<size;i++)
{
if(strcmp(p->label,l)==0)
{

strcpy(p->label,nl);
p->addr=add;
}
p=p->next;
}
}
break;
}
}
void del()
{
int a;
char l[10];
struct symbtab *p,*q;
p=first;
printf("Enter the label to be deleted\n");
scanf("%s",l);
a=search(l);
```

```
if(a==0)
{
printf("Label not found\n");
}
else
{
if(strcmp(first->label,l)==0)
{
first=first->next;
}
else if(strcmp(last->label,l)==0)
{
q=p->next;
while(strcmp(q->label,l)!=0)
{
p=p->next;
q=q->next;

}
p->next=null;
last=p;
}
else
{
q=p->next;
while(strcmp(q->label,l)!=0)
{
p=p->next;
q=q->next;
}
p->next=q->next;
}
size--;
}
}
```

OUTPUT:-

$lex int.l
$yacc –d int.y
$gcc lex.yy.c y.tab.c –ll –lm
$./a.out test.c

Pos	Operator	Arg1	Arg2	Result
0	<	a	b	t0
1	==	t0	**FALSE**	**5**
2	+	**a**	**b**	**t1**
3	==	**t1**		**5**
4	**GOTO**			
5	<	**a**	**b**	**t2**
6	==	**t2**	**FALSE**	**10**
7	+	**a**	**b**	**t3**
8	=	**t3**		**a**
9	**GOTO**		**5**	
10	<=	**a**	**b**	**t4**
11	==	**t4**	**FALSE**	**15**
12	-	**a**	**b**	**t5**
13	=	**t5**		**c**
14	**GOTO**		**17**	
15	+	**a**	**b**	**t6**
16	=	**t6**		**c**

RESULT:-

Thus the program for implementing yacc code from bnf has been successfully implemented.

IMPLEMENTATION OF STACK STORAGE ALLOCATION

Ex. No: 8 **Date:**

AIM:-

To implement stack allocation using c program.

ALGORITHM:-

PUSH( )
STEP 1: t = newnode( )
STEP 2: Enter info to be inserted
STEP 3: Read n
STEP 4: t□ info = n
STEP 5: t□ next = top
STEP 6: top = t
STEP 7: Return
POP( )
STEP 1: If (top = NULL)
Print " underflow"
Return
STEP 2: x = top
STEP 3: top = top □ □ next
STEP 4: delnode(x)
STEP 5: Return

PROGRAM:-

```
#include <stdio.h>
#define MAXSIZE 5

struct stack
{
int stk[MAXSIZE];
int top;
};
typedef struct stack STACK;
STACK s;

void push(void);
int  pop(void);
void display(void);

void main ()
{
```

```
int choice;
int option = 1;
   s.top = -1;

printf ("STACK OPERATION\n");
while (option)
   {
printf ("-----------------------------------------\n");
printf ("     1   -->   PUSH          \n");
printf ("     2   -->   POP         \n");
printf ("     3   -->   DISPLAY           \n");
printf ("     4   -->   EXIT        \n");
printf ("-----------------------------------------\n");

printf ("Enter your choice\n");
scanf    ("%d", &choice);
switch (choice)
      {
case 1:
push();
break;
case 2:
pop();
break;
case 3:
display();
break;
case 4:
return;
      }
fflush (stdin);
printf ("Do you want to continue(Type 0 or 1)?\n");
scanf    ("%d", &option);
   }
}
/*  Function to add an element to the stack */
void push ()
{
int num;
if (s.top == (MAXSIZE - 1))
   {
printf ("Stack is Full\n");
```

```
return;
   }
else
   {
printf ("Enter the element to be pushed\n");
scanf ("%d", &num);
      s.top = s.top + 1;
s.stk[s.top] = num;
   }
return;
}
/*  Function to delete an element from the stack */
int pop ()
{
int num;
if (s.top == - 1)
   {
printf ("Stack is Empty\n");
return (s.top);
   }
else
   {
num = s.stk[s.top];
printf ("poped element is = %dn", s.stk[s.top]);
      s.top = s.top - 1;
   }
return(num);
}
/*  Function to display the status of the stack */
void display ()
{
int i;
if (s.top == -1)
   {
printf ("Stack is empty\n");
return;
   }
else
   {
printf ("\n The status of the stack is \n");
for (i = s.top; i >= 0; i--)
      {
```

```
printf ("%d\n", s.stk[i]);
        }
    }
printf ("\n");
}
```

OUTPUT:-

```
STACK OPERATION
---->PUSH
---->POP
---->DISPLAY
---->EXIT
ENTER YOUR CHOICE
1
ENTER THE ELEMENT TO BE PUSHED
75
DO YOU WANT TO CONTINUE(Type 0 or 1)?
1
---->PUSH
---->POP
---->DISPLAY
---->EXIT
ENTER YOUR CHOICE
3
THE STATUS OF THE STACK IS
75
DO YOU WANT TO CONTINUE(Type 0 or 1)?
1

---->PUSH
---->POP
---->DISPLAY
---->EXIT
ENTER YOUR CHOICE
2
Poped element is=75 Do you want to continue(Type 0 or 1)?
0
1
---->PUSH
---->POP
---->DISPLAY
---->EXIT
```

RESULT:-

Thus the c program for stack allocation has been implemented successfully.

CONSTRUCTION OF DAG

Ex. No: 9 **Date:**

AIM:-

To construct dag using c++ program.

ALGORITHM:-

For Leaf Nodes

Assign label 1 to left child node and label 0 to right child node.

For Interior Nodes

Case 1: If Node's children's labels are different, then
Node's Label = Maximum among Children's Labels.
Case 2: If Node's children's labels are same, then
Node's Label = (Left Child's Label OR Right Child's Label) + 1.

PROGRAM:-

```
#include<stdlib.h>
#include<iostream>
using namespace std;

/* We will implement DAG as Strictly Binary Tree where each node has zero or two children
*/

struct bin_tree
{
char data;
int label;
struct bin_tree *right, *left;
};
typedef bin_tree node;

class dag
{
private:
/* R is stack for storing registers */
int R[10];
int top;

/* op will be used for opcode name w.r.t. arithmetic operator e.g. ADD for + */
char *op;
```

```
public:

void initializestack(node *root)
{
/* value of top = index of topmost element of stack R = label of Root of tree(DAG) minus one
*/
   top=root->label - 1;

   /* Allocating Stack Registers */
   int temp=top;
   for(int i=0;i<=top;i++)
     {
       R[i]=temp;
       temp--;
     }
}

/* insertnode() and insert() functions are for adding nodes to tree(DAG) */

void insertnode(node **tree,char val)
{
node *temp = NULL;

if(!(*tree))
   {
      temp = (node *)malloc(sizeof(node));
      temp->left = temp->right = NULL;
      temp->data = val;
      temp->label=-1;
      *tree = temp;
   }
}

void insert(node **tree,char val)
{
   char l,r;
   int numofchildren;

   insertnode(tree, val);

   cout << "\nEnter number of children of " << val <<" :";
   cin >> numofchildren;
```

```
 if(numofchildren==2)
  {
   cout << "\nEnter Left Child of " << val <<" :";
   cin >> l;
   insertnode(&(*tree)->left,l);

   cout << "\nEnter Right Child of " << val <<" :";
   cin >> r;
   insertnode(&(*tree)->right,r);

   insert(&(*tree)->left,l);
   insert(&(*tree)->right,r);
  }
}

/* findleafnodelabel() will find out the label of leaf nodes of tree(DAG) */

void findleafnodelabel(node *tree,int val)
{

if(tree->left != NULL && tree->right !=NULL)
{
findleafnodelabel(tree->left,1);
findleafnodelabel(tree->right,0);
}

else
{
tree->label=val;
}

}

/* findinteriornodelabel() will find out the label of interior nodes of tree(DAG) */

void findinteriornodelabel(node *tree)
{
if(tree->left->label==-1)
{
findinteriornodelabel(tree->left);
}
```

```
else if(tree->right->label==-1)
{
findinteriornodelabel(tree->right);
}

else
{

if(tree->left != NULL && tree->right !=NULL)
{

if(tree->left->label == tree->right->label)
{

tree->label=(tree->left->label)+1;
}

else
{

if(tree->left->label > tree->right->label)
{
tree->label=tree->left->label;
}
else
{
tree->label=tree->right->label;
}

}
}
}
}
```

/* function print_inorder() will print inorder of nodes. Here we are also printing label of each node of tree(DAG) */

```
void print_inorder(node * tree)
{
   if (tree)
   {
```

```
        print_inorder(tree->left);
        cout << tree->data <<" with Label "<< tree->label << "\n";
        print_inorder(tree->right);
    }
}

/* function swap() will swap the top and second top elements of Register stack R */

void swap()
{
int temp;
temp=R[0];
R[0]=R[1];
R[1]=temp;
}

/* function pop() will remove and return topmost element of stack */

int pop()
{
int temp=R[top];
top--;
return temp;
}

/* function push() will increment top by one and will insert element at top position of Register
stack */

void push(int temp)
{
top++;
R[top]=temp;
}

/* nameofoperation() will return opcode w.r.t. arithmetic operator */

void  nameofoperation(char temp)
{
switch(temp)
{
case '+': op =(char *)"ADD"; break;
case '-': op =(char *)"SUB"; break;
```

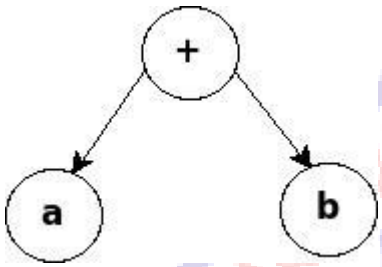
```
case '*': op =(char *)"MUL"; break;
case '/': op =(char *)"DIV"; break;
}
}

/* gencode() will generate Assembly code w.r.t. labels of tree(DAG) */

void gencode(node * tree)
{
if(tree->left != NULL && tree->right != NULL)
{
if(tree->left->label == 1 && tree->right->label == 0 && tree->left->left==NULL && tree->left->right==NULL && tree->right->left==NULL && tree->right->right==NULL)
{
cout << "MOV "<< tree->left->data << "," << "R[" << R[top] << "]\n";
nameofoperation(tree->data);
cout << op << " " << tree->right->data << ",R[" << R[top] << "]\n";
}

else if(tree->left->label >= 1 && tree->right->label == 0)
{
gencode(tree->left);
nameofoperation(tree->data);
cout << op << " " << tree->right->data << ",R[" << R[top] << "]\n";
}

else if(tree->left->label < tree->right->label)
{
int temp;
swap();
gencode(tree->right);
temp=pop();
gencode(tree->left);
push(temp);
swap();
nameofoperation(tree->data);
cout << op << " " << "R[" << R[top-1] <<"],R[" << R[top] << "]\n";
}

else if(tree->left->label >= tree->right->label)
{
int temp;
```

```
gencode(tree->left);
temp=pop();
gencode(tree->right);
push(temp);
nameofoperation(tree->data);
cout << op << " " << "R[" << R[top-1] << "],R[" << R[top] <<"]\n";
}

}

else if(tree->left == NULL && tree->right == NULL && tree->label == 1)
{
cout << "MOV " << tree->data << ",R[" << R[top] << "]\n";
}

}

/* deltree() will free the memory allocated for tree(DAG) */

void deltree(node * tree)
{
   if (tree)
   {
      deltree(tree->left);
      deltree(tree->right);
      free(tree);
   }
}

};

/* Program execution will start from main() function */

int main()
{
   node *root;
   root = NULL;
   node *tmp;
   char val;
   int i,temp;

   dag d;
```

```
    /* Inserting nodes into tree(DAG) */

    cout << "\nEnter root of tree:";
    cin >> val;

    d.insert(&root,val);

    /* Finding Labels of Leaf nodes */
    d.findleafnodelabel(root,1);

    /* Finding Labels of Interior nodes */
    while(root->label == -1)
      d.findinteriornodelabel(root);
    /* Initializing Stack contents and top variable */
    d.initializestack(root);

    /* Printing inorder of nodes of tree(DAG) */
    cout << "\nInorder Display:\n";
    d.print_inorder(root);

    /* Printing assembly code w.r.t. labels of tree(DAG) */
    cout << "\nAssembly Code:\n";
    d.gencode(root);

    /* Deleting all nodes of tree */
    d.deltree(root);


    return 0;
}
```

OUTPUT:-

>>>g++ codegeneration.cpp
>>>./a.out



RESULT:-

Thus the c++ program for DAG generation has been implemented successfully.

GENERATION OF CODE FOR A GIVEN INTERMEDIATE CODE

Ex. No: 10 **Date:**

AIM:-

To write a program for the generation of assembly language code ofrelational operator.

ALGORITHM:-

STEP 1: The three address code using the relational operator is get from the user.
STEP2: Identifying the addressing mode of the given three address code.
STEP3: Identify the relational operator used in the statement.
STEP 4: Generate and display the assembly language code.

PROGRAM:-

```
#include<iostream.h>
#include<stdlib.h>
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<ctype.h>
void forswitch(char,int);
void conv(int);
char arg[10][10],op[5][2],ch[10],go[3][3],c[10];
void main()
{
int i=-1,m=0,k=10;
clrscr();
cout<<"\t\t\tTHREE ADDRESS CODE";
gotoxy(15,7);
cout<<"OPERATOR";
gotoxy(30,7);
cout<<"ARGUMENT-1";
gotoxy(45,7);
cout<<"ARGUMENT-2";
gotoxy(60,7);
cout<<"GOTO";
gotoxy(15,8);
cout<<"~~~~~~~~~~~~~~~~~~~~~~~~~~";
do
{
i++;
gotoxy(2,k);
```

```
printf("[%d]",i);
gotoxy(18,k);
scanf("%s",&op[i]);
forswitch(op[i][0],i);
gotoxy(33,k);

scanf("%s",&arg[m+i]);
gotoxy(48,k);
scanf("%s",&arg[m+1+i]);
gotoxy(61,k);
scanf("%s",&go[i]);
conv(m+i);
conv(m+1+i);
k++;
m++;
}while(i!=3);
clrscr();
printf("ASSEMBLY LANGUAGE CODE");
printf("\n\n100\tMOV %s,RO",arg[0]);
printf("\n101\tMOV %s,R1",arg[1]);
printf("\n102\tCMP R0,R1");
printf("\n103\t%s 107",ch);
printf("\n104\tMOV%s,R2",arg[3]);
printf("\n105\tMOV R2,%s",arg[2]);
printf("\n106\tJUMP 109");
printf("\n107\tMOV %s,R2",arg[5]);
printf("\n109\tend");
getch();
}
void conv(int x)
{
if(isdigit(arg[x][0]))
{
strcpy(c,"#");
strcat(c,arg[x]);
strcpy(arg[x],c);
}
}
void forswitch(char sh,int t)
{
if(t<1)
switch(sh)
```

```
{
case '<':
strcpy(ch,"JC");
break;
case '>':
strcpy(ch,"JNC");
break;
case '=':
strcpy(ch,"JZ");
break;
case '-':
break;
default:
gotoxy(8,40);
cout<<"\n\t\tINVALID ENTRY";
getch();
exit(0);
break;
}
}
```

OUTPUT:-

THREE ADDRESS CODE
OPERATOR ARGUMENT1 ARGUMENT2 GOTO
[0] > date 5 [2]
[1] = fine 50 [3]
[2] = fine 0 ----
[3] - ---- ---- end
ASSEMBLY LANGUAGE CODE
100 MOV date,R0
101 MOV #5,R1
102 CMP R0,R1
103 JNC 107
104 MOV #50,R2
105 MOV R2,fine
106 JUMP 109
107 MOV #0,R2
108 MOV R2,fine
109 END

RESULT:-

Thus the program for generation of Machine Code for the givenIntermediate code is executed and verified.

IMPLEMENTATION OF CODE OPTIMIZATION TECHNIQUES

Ex. No: 11 **Date:**

AIM:-

To write a program for implementation of Code Optimization Technique infor and do-while loop using C++.

ALGORITHM:-

STEP 1: Generate the program for factorial program using for and do-while loop to specify optimization technique.

STEP 2: In for loop variable initialization is activated first and the condition is checked next. If the condition is true the corresponding statements are executed and specified increment / decrement operation is performed.

STEP 3:The for loop operation is activated till the condition failure.

STEP4: In do-while loop the variable is initialized and the statements are executed then the condition checking and increment / decrement operation is performed.

STEP5: When comparing both for and do-while loop for optimization dowhile is best because first the statement execution is done then only the condition is checked. So, during the statement execution itself we can fin the inconvenience of the result and no need to wait for the specified condition result.

STEP6: Finally when considering Code Optimization in loop do-while is best with respect to performance.

PROGRAM:-

```
#include<iostream.h>
#include <conio.h>
int main()
{
int i, n;
int fact=1;
cout<<"\nEnter a number: ";
cin>>n;
for(i=n ; i>=1 ; i--)
fact = fact * i;
cout<<"The factoral value is: "<<fact;
getch();
return 0;}
```

OUTPUT:-

Enter the number: 5
The factorial value is: 120

After:
Using do-while:

```
#include<iostream.h>
#include<conio.h>
void main()
{
clrscr();
int n,f;
f=1;
cout<<"Enter the number:\n";
cin>>n;
do
{
f=f*n;
n--;
}while(n>0);
cout<<"The factorial value is:"<<f;
getch();
}
```

OUTPUT:-

Enter the number: 3
The factorial value is: 6

RESULT:-

Thus the program for implementation of Code Optimizationtechnique is executed and verified.