Ex. No. : 01 Date :	IMPLEMENTATION OF SYMBOL TABLE

AIM :

The main objective is to write a C program to implement a symbol table which can create, insert, modify, search and display the character.

OUTCOME:

This program takes a source program as input, and it is expected to produce a functions which can create, insert, modify, search and display the character as output.

PRE-REQUESTIE/THEME:

It is easy to create, modify, search and display the character with the help of case statement. It is achieved by extracting character from the expression using built in functions in C, such as is switch(),while() etc.

SAMPLE PROGRAM TITLE:

Implement a C program for file manipulation.

ALGORITHM:

Step 1: Print the five choices namely create, insert, modify, search and display.

Step 2: Get the choice value.

Step 3: If choice is 1, then get total numbers of symbol to be added in the symbol table and call Create ( ) to create the symbol table and call the display ( ) to display the symbol table with the following contents: symbol name, location and symbol type.

Step 4: If choice is 2, call insert ( ) get a symbol table detail to insert into the symbol table and Increment the current position the insert the given detail.

Step 5: If choice is 3, then read the symbol to be modified and call the modified ( ) to modify the location of particular symbol.

Step 6: If choice is 4, then read the symbol to be searched and call the search ( ) to display the symbol table for particular symbol.

Step 7: If choice is 5, to display the symbol table.

PROGRAM :

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
struct source
{
        char la[25];
        char oc[25];
        char o[25];
        int addr;
}s[100];

struct stable
{
        char sym[25];
        int val;
}st[20];
        int n,m=0;
        int locctr=0;
void input();
void create();
void display();
void search(char key[10]);
void modify();
void main()
{
        int ch,a;
        char k[10];
do
{
printf("\n\tSYMBOL TABLE IMPLEMENTATION");
printf("\n1.CREATE(INSERT)\n\n2.SEARCH\n\n3.MODIFY\n\n4.DISPLAY\n\n5.EXIT\n\n
ENTER YOUR CHOICE\n");
scanf("%d",&ch);
switch(ch)
{
case 1:
        input();
        create();
        break;
case 2:
        printf("ENTER THE LABEL TO SEARCH\n");
        scanf("%s",k);
        search(k);
        break;
case 3:
        modify();
        break;
```

```
        case 4:
        display();
        break;
case 5:
        exit(0);
}
}
while(ch<=6);
}
void input()
{
int i=0;
printf("ENTER CODE & TO STOP TYPE END \n");
printf("\nLABEL\tOPCODE\tOPERAND");
do
{
        scanf("%s",s[i].la);
        if(strcmp(s[i].la,"END")==0)
        break;
        scanf("%s",s[i].oc);
        scanf("%s",s[i].o);
        i++;
        }while(1);
        n=i;
}
void create()
{
int i;
if(strcmp(s[0].oc,"START")==0)
{
        locctr=atoi(s[0].o);
        s[1].addr=locctr;
}
for(i=1;i<n;i++)
{
        if(strcmp(s[i].oc,"WORD")==0)
        locctr=locctr+3;
        else if(strcmp(s[i].oc,"RESW")==0)
        locctr=locctr+(3*atoi(s[i].o));
        else if(strcmp(s[i].oc,"RESB")==0)
        locctr=locctr+(atoi(s[i].o));
        else if(strcmp(s[i].oc,"BYTE")==0)
        locctr=locctr+(strlen(s[i].o)-3);
        else
        locctr=locctr+3;
        s[i+1].addr=locctr;
}
m=0;
for(i=1;i<n;i++)
{

        if(s[i].la[0]!='-')
```

```
        {
                strcpy(st[m].sym,s[i].la);
                st[m].val=s[i].addr;
                m++;
        }
}
printf("\nSymbol table created");
}
void display()
{
int i;
printf("\t\tSYMBOL TABLE \n\n");
printf("\nVALUE\tSYMBOL\n");
for(i=0;i<m;i++)
{
        printf("%d",st[i].val);
        printf("\t%s\n\n\n",st[i].sym);
}
}
void search(char key[10])
{
int i,a;
for(i=0;i<m;i++)
{
        if(strcmp(st[i].sym,key)==0)
        {
        printf("\nThe label is found at address: %d ",st[i].val);
        break;
        }
else
printf("\n the label is not found");
}
}
void modify()
{
int a,i;
char key[25];
printf("\n\tSOURCE CODE \t\n");
printf("\nADDRESS\tLABEL\tOPCODE\tOPERAND\n");
for(i=1;i<n;i++)
{
        printf("\n%d",s[i].addr);
        printf("\t%s",s[i].la);
        printf("\t%s",s[i].oc);
        printf("\t%s\n\n",s[i].o);
}
printf("\nENTER THE LABEL TO MODIFY\n");
scanf("%s",key);
for(i=0;i<m;i++)

{
        if(strcmp(st[i].sym,key)==0)
```

```
        {
        a=1;
        break;
        }
}
if(a==1)
{
printf("\nENTER THE NEW LABEL\n");
scanf("%s",key);
strcpy(st[i].sym,key);
}
else
        printf("\nTHE LABEL IS NOT FOUND\n");
}
```

OUTPUT :

```
linus@linus:~$ gcc symbol.c -o sym
linus@linus:~$ ./sym

SYMBOL TABLE IMPLEMENTATION
1.CREATE(INSERT)
2.SEARCH
3.MODIFY
4.DISPLAY
5.EXIT

ENTER YOUR CHOICE
1

ENTER CODE & TO STOP TYPE END
LABEL       OPCODE      OPERAND
SAMPLE      START       1000
-           LDA         ALPHA
ALPHA       RESB        1
END
Symbol table created

SYMBOL TABLE IMPLEMENTATION
1.CREATE(INSERT)
2.SEARCH
3.MODIFY
4.DISPLAY
5.EXIT

ENTER YOUR CHOICE
4

SYMBOL TABLE

VALUE       SYMBOL
1003        ALPHA

SYMBOL TABLE IMPLEMENTATION
1.CREATE(INSERT)
2.SEARCH
3.MODIFY
4.DISPLAY
5.EXIT

ENTER YOUR CHOICE
2
ENTER THE LABEL TO SEARCH
ALPHA
The label is found at address: 1003
```

```
SYMBOL TABLE IMPLEMENTATION
1.CREATE(INSERT)
2.SEARCH
3.MODIFY
4.DISPLAY
5.EXIT

ENTER YOUR CHOICE
3
SOURCE CODE
ADDRESS    LABEL      OPCODE     OPERAND
1000        -         LDA        ALPHA
1003       ALPHA      RESB       1

ENTER THE LABEL TO MODIFY
ALPHA

ENTER THE NEW LABEL
GAMMA

SYMBOL TABLE IMPLEMENTATION
1.CREATE(INSERT)
2.SEARCH
3.MODIFY
4.DISPLAY
5.EXIT


ENTER YOUR CHOICE
4

SYMBOL TABLE

VALUE      SYMBOL
1003       GAMMA

SYMBOL TABLE IMPLEMENTATION
1.CREATE(INSERT)
2.SEARCH
3.MODIFY
4.DISPLAY
5.EXIT

ENTER YOUR CHOICE
5
linus@linus:~$
```

RESULT:

Thus the C program to implement symbol table has been executed successfully.

Ex. No. : 02 Date :	**DEVELOP A LEXICAL ANALYZER TO RECOGNIZE THE PATTERNS IN C (Ex. identifiers, constants, comments, operators etc.)**

AIM :

The main objective is to develop a lexical analyzer to recognize the patterns (Ex. identifiers, constants, comments, operators etc.) using C program.

OUTCOME:

This program reads the file input and displays the tokens and its type as output.

PRE-REQUESTIE/THEME: Should be familiar in Lexical analyzer.

SAMPLE PROGRAM TITLE:

Implement a C program for lexical analyzer for token separation.

ALGORITHM:

1. Start the program.
2. Open the file.
3. Declare the variable and functions.
4. Input: Programming language 'if' statement

   Output: A sequence of tokens.
5. Tokens have to be identified and its respective attributes have to be printed.
6. Execute the program
7. Stop the program.

PROGRAM :

```
#include<stdio.h>
int main()
{
FILE *fp;
char op[20]={'+','-','*','/','%'};
int i,j=0,num=0,flag,f;
char file[20],ch,id[30];
printf("\n\t\t Token Seperation \n\n");
printf("Enter the file name:");
scanf("%s",file);
fp=fopen(file,"r");
while(ch!=EOF)
{
        printf("%c",ch);
        ch=getc(fp);
}
fclose(fp);
fp=fopen(file,"r");
printf("\n\t Line number \t Token \t\t Type \n");
printf("\n\t _____");
while(!feof(fp))
{
        flag=0;
        ch=fgetc(fp);
        i=0;
        if(isalpha(ch)||isdigit(ch))
        {
                f=0;
                while(isalpha(ch)||isdigit(ch))
                {
                        id[j++]=ch;
                        ch=fgetc(fp);
                }
                id[j]='\0';
                j=0;
                if(f==0)
                printf("\n\t %d \t\t %s \t\t Identifier",num,id);
        }
        i=0;
        while(i<5&&flag!=1)
        {
                if(ch==op[i])
                {
                        printf("\n\t %d \t\t %c \t\t Operator",num,ch);
                        flag=1;
                }
                        i++;
        }
```

```
        i=0;
        if(ch=='\n')
        num++;
}
fclose(fp);
}
```

OUTPUT :

INPUT FILE:

```
[compiler@localhost ~]$ vi sat.txt
a=b*c-d/e

linus@linus:~/Desktop$ gcc lexical.c -o cse
linus@linus:~/Desktop$ ./cse
```

Token Separation

```
Enter the file name: sat.txt
a=b*c-d/e
```

Line number	Token	Type
0	a	Identifier
0	b	Identifier
0	*	Operator
0	c	Identifier
0	-	Operator
0	d	Identifier
0	/	Operator
0	e	Identifier

```
linus@linus:~/Desktop$
```

RESULT:

Thus the C program to develop a lexical analyzer to recognize the patterns has been executed successfully.

Ex. No. : 03 Date :	IMPLEMENTATION OF LEXICAL ANALYZER USING LEX TOOL

AIM :

The main objective is to implement a lexical analyzer using LEX tool.

OUTCOME:

This program reads the lex program rules and displays the expressions as output.

PRE-REQUESTIE/THEME: Should be familiar in LEX rules and tools.

SAMPLE PROGRAM TITLE: Implement a lex program for identifying hyperlink.

ALGORITHM:

1. Start the program.
2. Lex program consists of three parts.
   - Declaration %%
   - Translation rules %%
   - Auxilary procedure.
3. The declaration section includes declaration of variables, maintest, constants and regular definitions.
4. Translation rule of lex program are statements of the form
   - P1 {action}
   - P2 {action}
   - Pn {action}
5. Write a program in the vi editor and save it with .l extension.
6. Compile the lex program with lex compiler to produce output file as lex.yy.c.

   eg $ lex filename.l and $ cc lex.yy.c -ll
7. Compile that file with C compiler and verify the output.

PROGRAM :

```
/* program name is lexp.l */
%{
/* program to recognize a c program */
int COMMENT=0;
%}

identifier [a-zA-Z][a-zA-Z0-9]*
%%
#.* { printf("\n%s is a PREPROCESSOR DIRECTIVE",yytext);}

int |
float |
char |
double |
while |
for |
do |
if |
break |
continue |
void |
switch |
case |
long |
struct |
const |
typedef |
return |
else |
goto {printf("\n\t%s is a KEYWORD",yytext);}
"/*" {COMMENT = 1;}
"*/" {COMMENT = 0;}
{identifier}\( {if(!COMMENT)printf("\n\nFUNCTION\n\t%s",yytext);}
\{ {if(!COMMENT) printf("\n BLOCK BEGINS");}
\} {if(!COMMENT) printf("\n BLOCK ENDS");}
{identifier}(\[[0-9]*\])? {if(!COMMENT) printf("\n %s IDENTIFIER",yytext);}
\".*\" {if(!COMMENT) printf("\n\t%s is a STRING",yytext);}
[0-9]+ {if(!COMMENT) printf("\n\t%s is a NUMBER",yytext);}
\)(\;)? {if(!COMMENT) printf("\n\t");ECHO;printf("\n");}
\( ECHO;
= {if(!COMMENT)printf("\n\t%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 0;
}
```

hi.c

```
#include<stdio.h>
int main()
{
int a,b;
}
```

OUTPUT :

```
linus@linus:~/Desktop$ lex lex.l
linus@linus:~/Desktop$ gcc lex.
lex.l lex.l~ lex.yy.c
linus@linus:~/Desktop$ gcc lex.yy.c -o aa
linus@linus:~/Desktop$ ./aa hi.c

#include<stdio.h> is a PREPROCESSOR DIRECTIVE
int is a KEYWORD

FUNCTION
main()

BLOCK BEGINS
int is a KEYWORD
a IDENTIFIER,
b IDENTIFIER;

BLOCK ENDS
```

RESULT:

Thus the above program is compiled and executed successfully and output is verified.

Ex. No. : 04. (a) Date :	PROGRAM TO RECOGNIZE A VALID ARITHMETIC EXPRESSION THAT USES OPERATOR +, - , * AND /.

AIM:

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

ALGORITHM:

1. Start the program
2. Create the program using YACC tool
3. Create the input file arith_id.y and the file is used to print the valid arithmetic expression.
4. Display the arithmetic expression using yyparse() function
5. Stop the program

PROGRAM :

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

%token num let
%left '+' '-'
%left '*' '/'
%%

Stmt : Stmt '\n' { printf("\n..valid expresssion..\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;
        if(isalpha(ch))
        return let;
        return ch;
}
yyerror(char *s)
{
        printf("%s",s);
}
```

OUTPUT :

```
linus@linus:~/Desktop$ yacc -d operator.y
linus@linus:~/Desktop$ gcc y.tab.c -o aa
linus@linus:~/Desktop$ ./aa

enter an expression to validate:4+t
..valid expresssion..

linus@linus:~/Desktop$ ./aa

enter an expression to validate:a+*5
syntax error
..invalid..

linus@linus:~/Desktop$
```

RESULT:

Thus the above program is compiled and executed successfully and output is verified.

Ex. No. : 04.(b) Date :	PROGRAM TO RECOGNIZE A VALID VARIABLE WHICH STARTS WITH A LETTER FOLLOWED BY ANY NUMBER OF LETTERS OR DIGITS

AIM:

To write a Program to recognize a valid variable which starts with a letter followed by any number of letters or digits.

PROGRAM :

```
%{
#include<stdio.h>
#include<ctype.h>
%}
%token let dig
%%

TERM : XTERM '\n' { printf("\n Accepted\n"); }
| error { yyerror("Rejected\n"); } ;
XTERM : XTERM let
| XTERM dig
| let
;
%%

yylex()
{
char ch;
while((ch=getchar())==' ');
if(isalpha(ch))
return let;
if(isdigit(ch))
return dig;
return ch;
}

main()
{
printf("enter a variable:");
yyparse();
}
yyerror(char *s)
{
printf("%s",s);
}
```

OUTPUT :

```
linus@linus:~/Desktop$ yacc -d var.y
linus@linus:~/Desktop$ gcc y.tab.c -o aa
linus@linus:~/Desktop$ ./aa

enter a variable:abc12
Accepted
^C

linus@linus:~/Desktop$ ./aa
enter a variable:12abc

syntax error Rejected
Rejected
Rejected
Rejected
Rejected
Rejected
Rejected
```

RESULT:

Thus the above program is compiled and executed successfully and output is verified.

Ex. No. : 04.(c) Date :	IMPLEMENTATION OF CALCULATOR USING LEX AND YACC

AIM :

To write a program to implement a Calculator using LEX and YACC.

ALGORITHM :

1. Start the program
2. Math.h is used to calculate the mathematical functions using LEX and YACC
3. The program is created then defines the possible symbol files in the calci.y file name.
4. Yyparse() is used to parse the input string
5. Yyerror() is used to report the error
6. Stop the program.

PROGRAM :

cal.l

```
%{
#include <stdlib.h>
#include <stdio.h>
#include "y.tab.h"
void yyerror(char*);
extern int yylval;
%}
%%
[ \t]+ ;
[0-9]+ {yylval = atoi(yytext);
return INTEGER;}
[-+*/] {return *yytext;}
"(" {return *yytext;}
")" {return *yytext;}
\n {return *yytext;}
. {char msg[25];
sprintf(msg,"%s <%s>","invalid character",yytext);
yyerror(msg);
}
```

cal.y

```
%{
#include <stdlib.h>
#include <stdio.h>
int yylex(void);
#include "y.tab.h"
%}
%token INTEGER
%%
program:
line program
| line
line:
expr '\n' { printf("%d\n",$1); }
| 'n'
expr:
expr '+' mulex { $$ = $1 + $3; }
| expr '-' mulex { $$ = $1 - $3; }
| mulex { $$ = $1; }
mulex:
mulex '*' term { $$ = $1 * $3; }
| mulex '/' term { $$ = $1 / $3; }
| term { $$ = $1; }
term:
'(' expr ')' { $$ = $2; }
| INTEGER { $$ = $1; }
%%
```

```
int yyerror(char *s)
{
fprintf(stderr,"%s\n",s);
return;
}
yywrap()
{
return(1);
}
int main(void)
{
/*yydebug=1;*/
yyparse();
return 0;
}
```

OUTPUT :

```
linus@linus:~$ cd Desktop/
linus@linus:~/Desktop$ lex cal.l
linus@linus:~/Desktop$ yacc -d cal.y
linus@linus:~/Desktop$ gcc lex.yy.c y.tab.c
linus@linus:~/Desktop$ ./a.out
10+10
20
4-2
2
2++2

syntax error

linus@linus:~/Desktop$
```

RESULT:

Thus the above program is compiled and executed successfully and output is verified.

Ex. No. : 05 Date :	CONVERT THE BNF RULES INTO YACC FORM AND WRITE CODE TO GENERATE ABSTRACT SYNTAX TREE

AIM :

The main objective is to convert the BNF rules into YACC form and write code to generate Abstract Syntax Tree.

OUTCOME:

This program converts the input file to abstract syntax tree and generates three address code in the form of quadruple tabular form.

PRE-REQUESTIE/THEME: Should be familiar in YACC rules and specifications.

SAMPLE PROGRAM TITLE: Implementation of syntax analyzer using lex and YACC.

ALGORITHM:

1. Read an input file line by line.
2. Convert it in to abstract syntax tree using three address code.
3. Represent three address code in the form of quadruple tabular form.
4. Go to terminal .
5. Open vi editor, Lex lex.l , cc lex.yy.c ,
6. Execute the program by ./a.out

PROGRAM :

bnf.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];
%%
```

bnf.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");
}
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");

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

OUTPUT :

```
linus@linus:~/Desktop$ ./aa test.c
linus@linus:~/Desktop$ lex bnf.l
linus@linus:~/Desktop$ yacc -d bnf.y
linus@linus:~/Desktop$ gcc lex.yy.c y.tab.c -o aa
```

Pos	Operator	Arg1	Arg2	Result
0	<	a	b	t0
1	==	t0	FALSE	5
2	+	a	b	t1
3	=	t1	a	
4	GOTO			5
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 to Convert the BNF rules into YACC form and write code to generate Abstract Syntax Tree has been implemented successfully.

Ex. No. : 06 Date :	IMPLEMENT TYPE CHECKING

AIM :

The main objective of this C program is to implement type checking.

OUTCOME: implementation of type checking.

SAMPLE PROGRAM TITLE:

Write a program to check whether the string belongs to the grammar or not.

ALGORITHM:

1. Start the program.
2. Declare the variable and functions.
3. Input: Programming language structure is created.
4. Check the file type.
5. Execute the program
6. Stop the program

PROGRAM :

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
int main(int argc,char*argv[])
{
int i;
struct stat buf;
for(i=1;i<argc;i++)
{
        if(lstat(argv[i],&buf)<0)
        printf("lstat error!\n");
        else if(S_ISREG(buf.st_mode))
        printf("regular file!\n");
        else if(S_ISDIR(buf.st_mode))
        printf("Directory file!\n");
        else if(S_ISCHR(buf.st_mode))
        printf("character device file!\n");
        else if(S_ISFIFO(buf.st_mode))
        printf("FIFO file!\n");
        else
        printf("socket file\n");
}
}
```

OUTPUT :

```
linus@linus:~$ gcc typechecking.c -o qq
linus@linus:~$ ./qq
linus@linus:~$ ./qq aa.odt
regular file!
linus@linus:~$ ./qq Examples
lstat error!
linus@linus:~$ ./qq os
Directory file!
linus@linus:~$ ./qq ss.png
regular file!
linus@linus:~$
```

RESULT:

Thus the C program is to implement type checking has been implemented.

Ex. No. : 07 Date :	IMPLEMENTATION OF SIMPLE CODE OPTIMIZATION TECHNIQUES (Constant Folding.,)

AIM :

The main objective is to write a C program to implement code optimization techniques.

OUTCOME:

This program displays intermediate code as the output of constant folding.

SAMPLE PROGRAM TITLE:

Implement a C program to find the number of whitespaces and newline characters.

ALGORITHM:

1. Start the program.
2. Declare the variables and structures.
3. Write the generated code into output definition of the file.
4. Print the output.
5. Stop the program.

PROGRAM:

```
#include<stdio.h>
#include<unistd.h>
#include<string.h>
#include<termios.h>
struct op
{
        char l;
        char r[20];
}op[10],pr[10];
void main()
{
int a,i,k,j,n,z=0,m,q;
char *p,*l;
char temp,t;
char *tem;
printf("Enter the Number of Values:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
        printf("left: ");
        op[i].l=getchar();
        printf("\tright: ");
        scanf("%s",op[i].r);
}
printf("Intermediate Code\n") ;
for(i=0;i<n;i++)
{
        printf("%c=",op[i].l);
        printf("%s\n",op[i].r);
}
for(i=0;i<n-1;i++)
{
        temp=op[i].l;
        for(j=0;j<n;j++)
        {
                p=strchr(op[j].r,temp);
                if(p)
                {
                pr[z].l=op[i].l;
                strcpy(pr[z].r,op[i].r);
                z++;
        }
}
}
pr[z].l=op[n-1].l;
strcpy(pr[z].r,op[n-1].r);
z++;
printf("\nAfter Dead Code Elimination\n");
```

```
for(k=0;k<z;k++)
{
	printf("%c\t=",pr[k].l);
	printf("%s\n",pr[k].r);
}
for(m=0;m<z;m++)
{
	tem=pr[m].r;
	for(j=m+1;j<z;j++)
	{
		p=strstr(tem,pr[j].r);
		if(p)
		{
			t=pr[j].l;
			pr[j].l=pr[m].l;
			for(i=0;i<z;i++)
			{
				l=strchr(pr[i].r,t) ;
				if(l)
				{
					a=l-pr[i].r;
					printf("pos: %d",a);
					pr[i].r[a]=pr[m].l;
				}
			}
		}
	}
}
printf("Eliminate Common Expression\n");
for(i=0;i<z;i++)
{
	printf("%c\t=",pr[i].l);
	printf("%s\n",pr[i].r);
}
for(i=0;i<z;i++)
{
	for(j=i+1;j<z;j++)
	{
		q=strcmp(pr[i].r,pr[j].r);
		if((pr[i].l==pr[j].l)&&!q)
		{
			pr[i].l='\0';
			strcpy(pr[i].r,'\0');
		}
	}
}




printf("Optimized Code\n");
```

```
for(i=0;i<z;i++)
{
        if(pr[i].l!='\0')
        {
        printf("%c=",pr[i].l);
        printf("%s\n",pr[i].r);
        }
}
}
```

OUTPUT:

```
Enter the Number of Values:5
Left: a right: 9
Left: b right: c+d
Left: e right: c+d
Left: f right: b+e
Left: r right: f

Intermediate Code
a=9
b=c+d
e=c+d
f=b+e
r=:f

After Dead Code Elimination
        =f
Eliminate Common Expression
        =f
Optimized Code
        =f
```

RESULT:

Thus the C program to implement code optimization technique has been executed successfully.

Ex. No. : 08 Date :	IMPLEMENTATION OF BACK END OF THE COMPILER

AIM :

The main objective is to write a C program to implement of 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.

OUTCOME:

This program takes a source program as input, and it is expected to produce a functions with intermediate code as the output.

PRE-REQUESTIE/THEME:

Should have the knowledge about processor programs.

ALGORITHM:

1- Start the program and open the file.

2- Get address code sequence.

3- Determine current location of 3 using address (for 1st operand).

4- If current location not already exist generate move (B,O).

5- Update address of operand and Import the files

6- Store the move instruction in memory

7- Stop

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>
void main()
{
int i=2,j=0,k=2,k1=0;
char ip[10],kk[10];
FILE*fp;
printf("\n enter the filename of the intermediate code");
scanf("%s",&kk);
fp=fopen(kk,"r");
if(fp==NULL)
{
        printf("\n Error in opening the file");
}
while(!feof(fp))
{
        fscanf(fp,"%s/n",ip);
        printf("\t\t%s\n",ip);
}
rewind(fp);
printf("\n- ---- -- - -- \n");
printf("\t statement \t\t target code\n");
while(!feof(fp))
{
        fscanf(fp,"%s",ip);
        printf("\t%s",ip);
        printf("\t\t MOV%c,R%d\n\t",ip[i+k],j);
        if(ip[i+1]=='+')
                printf("\t\tADD");
        else
                printf("\t\tSUB");

        if(islower(ip[i]))
                printf("%c,R%d\n\n",ip[i+k1],j);
        else
                printf("%c,%c\n",ip[i],ip[i+2]);
        j++;
        k1=2;
        k=0;
}
printf("\n- - - - --- - -- - \n");
fclose(fp);
}
```

OUTPUT :

Enter the filename of intermediate code : **b.txt**

```
x=a-b
y=a-c
z=a+b
c=A-B
c=A-B
```

Statement	Target code
x=a-b	Mov,R0 SUBa,R0
y=a-c	MOVa,R1 SUBc,R1
z=a+b	MOCa,R2 ADDb,R2
c=A-B	MOVa,R4 SUBA,B
c=A-B	MOVA,R5 SUBA,B

RESULT:

Thus the C program to implement Back end compiler using 8086 assembler has been executed successfully.