DEPARTMENT OF MATHEMATICS
UNIVERSITY COLLEGE OF SCIENCE
OSMANIA UNIVERSITY

CERTIFICATE

This is to Certify that this is the Record of Bonafide work done by _____Of **M.SC MATHS WITH COMPUTER SCIENCE ,SEMESTER - 4** in MCS**402L – COMPILER DESIGN LAB**, Bearing Hall Ticket No.:_____ as part of His / Her CurriculumDuring Academic Year 20 - 20 .

Date:

Internal Examiner External Examiner

INDEX

SR.NO.	PROGRAM	PAGE
1	Write a Program to Design Token Separator for the Given Expression.	1
2	Write a Program to Implement a Symbol Table.	4
3	Write a Program to Develop a Lexical Analyzer to Recognize a Few Patterns.	7
4	Write a Program to Develop a Lexical Analyzer using Lex Tool.	14
5	Write a Program to Recognize a Valid Arithmetic Expression using YACC.	17
6	Write a Program to Recognize a Valid Variable Name using YACC.	19
7	Write a Program to Implement Calculator Using Lex and YACC.	21
8	Write a program for Implementing Type Checking for Given Expression.	25
9	Write a Program to Convert the BNF Rules into YACC.	27
10	Write a Program to Implement Data Flow and Control Flow Analysis.	35
11	Write a Program to Implement Stack Storage Allocation Strategies.	42
12	Write a Program to Implement Heap Storage Allocation Strategies.	45
13	Write a Program to Construct a Directed Acyclic Graph (DAG).	52
14	Write a Program to Implement the Back End of the Compiler.	54
15	Write a Program to Implement Simple Code Optimization Technique.	56
16	Write a Program to Implement Simple Code Optimization Technique using do-while.	60

[1] Write a Program to Design Token Separator for the Given Expression.

```
#include<stdio.h>
#include<string.h>
#include<ctype.h>

void main()
{
char exp[50]="\0",con[50]="\0",kwd[50]="\0",id[50]="\0",sym[50]="\0";
char opr[50]="\0";
char key[6][10]={"if","for","do","while","int","float"};
char ch; char ptr[10][10]={"\0"};
int i=0,j=0,k=-1,n=-1,p=-1,s=-1;

puts("Enter The Expression for Lexical Analysis");
gets(exp);
puts("\n The Tokens are : ");
do{ ch=exp[i];
if(isalpha(ch))
{ k=-1;
ptr[++n][++k]=ch;
i++;
ch=exp[i];
if(isalpha(ch)||isdigit(ch))
{
while(isalpha(ch)||isdigit(ch))
{ ptr[n][++k]=ch;
i++;
ch=exp[i]; }
while(j<6)
{ if(strcmp(key[j],ptr[n])==0)
{ ptr[n][++k]=' ';
strcat(kwd,ptr[n]);
break; }
if(j==5)
{ ptr[n][++k]=' ';
strcat(id,ptr[n]);
}
j++; }}else
```

```
{ ptr[n][++k]=' ';
strcat(id,ptr[n]);}
i--;
ch=exp[i];
j=0; }
else if(isdigit(ch))
{ k=-1;
ptr[++n][++k]=ch;
i++;
ch=exp[i];

if(isdigit(ch))
{
while(isdigit(ch))
{ ptr[n][++k]=ch;
i++;
ch=exp[i]; }}
i--;
ch=exp[i];
ptr[n][++k]=' ';
strcat(con,ptr[n]); }

else if
(ch=='+'||ch=='-'||ch=='*'||ch=='/'||ch=='%'||ch=='>'||ch=='<'||ch=='='||ch=='!')
{ opr[++p]=ch;
i++;
ch=exp[i];
if
(ch=='+'||ch=='-'||ch=='*'||ch=='/'||ch=='%'||ch=='>'||ch=='<'||ch=='='||ch=='!')
{ opr[++p]=ch; }
else
{ i--;
ch=exp[i];
opr[++p]=' ';
}}
else
{ sym[++s]=ch;
sym[++s]=' ';} i++;
```

```
}
while(exp[i]!='\0');
puts("\nKeyword(s) :");
puts(kwd);
puts("\nIdentifier(s) :");
puts(id);
puts("\nConstant(s) :");
puts(con);
puts("\nOperator(s) :");
puts(opr);
puts("\nSymbol(s) :");
puts(sym);
}
```

OUTPUT:

```
unicomplexity@UCS-OU: ~/OUCS/CPL_DSG
unicomplexity@UCS-OU:~/OUCS/CPL_DSG$ cc P1.c 2>/dev/null
unicomplexity@UCS-OU:~/OUCS/CPL_DSG$ ./a.out
Enter The Expression for Lexical Analysis
if(a+b>=100)

 The Tokens are :

Keyword(s) :
if

Identifier(s) :
a b

Constant(s) :
100

Operator(s) :
+ >=

Symbol(s) :
( )
unicomplexity@UCS-OU:~/OUCS/CPL_DSG$
```

***Note: use “ 2>/dev/null ” command only to Suppress Warnings.**

[2] Write a Program to Implement a Symbol Table.

```
#include<stdio.h>
#include<ctype.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>

void main()
{
 int i=0,j=0,x=0,n, flag=0;
 void *p,*add[5];
 char ch,srch,b[15],d[15],c;
 printf("\nEnter Expression Terminated by $ : ");
 while((c=getchar())!='$')
 {
  b[i]=c;
  i++;
 }
 n=i-1;
 printf("\nGiven Expression : ");
 i=0;
 while(i<=n)
 {
  printf("%c",b[i]);
  i++;
 }
 printf("\n");
 printf("\n-------------------------------------");
 printf("\n\t  Symbol Table\t\t   ");
 printf("\n-------------------------------------");
 printf("\n");
 printf(" Symbol ");
 printf("    Address   ");
 printf("      Type    ");
 printf("\n-------------------------------------");

 while(j<=n)
 {
  c=b[j];
  if(isalpha(toascii(c)))
  {
   p=malloc(c);
   add[x]=p;
   d[x]=c;
   printf("\n   %c \t   %d     Identifier",c,p);
```

```
    x++;
    j++;
   }
   else
   {
    ch=c;
    if(ch=='+'||ch=='-'||ch=='*'||ch=='=')
    {
     p=malloc(ch);
     add[x]=p;
     d[x]=ch;
     printf("\n   %c \t   %d      Operator ",ch,p);
     x++;
     j++;
    }}}
 printf("\n");
 printf("-------------------------------------\n");
printf("\n Enter the Symbol to be Searched : ");
scanf("%s", &c);
for(i=0;i<=x;i++)
{ if(c==d[i])
 {
         printf("\n Symbol Found\n");
         printf("%c %s %d \n",c," @Address ",add[i]);

         flag=1;
  }
 }
if(flag==0)
printf("Symbol Not Found\n");

}
```

OUTPUT:

```
unicomplexity@UCS-OU: ~/OUCS/CPL_DSG
unicomplexity@UCS-OU:~/OUCS/CPL_DSG$ cc P2.c -w
unicomplexity@UCS-OU:~/OUCS/CPL_DSG$ ./a.out

Enter Expression Terminated by $ : a+b=c*d$

Given Expression : a+b=c*d

-------------------------------------
          Symbol Table
-------------------------------------
 Symbol     Address         Type
-------------------------------------
   a       -2003756352     Identifier
   +       -2003756240      Operator
   b       -2003756176     Identifier
   =       -2003756064      Operator
   c       -2003755984     Identifier
   *       -2003755872      Operator
   d       -2003755808     Identifier
-------------------------------------

 Enter the Symbol to be Searched : d

 Symbol Found
d  @Address  -2003755808
unicomplexity@UCS-OU:~/OUCS/CPL_DSG$
```

[3] Write a Program to Develop a Lexical Analyzer to Recognize a Few Patterns.

//Program.c

```
#include<stdio.h>
#include<ctype.h>
#include<string.h>

void main()
{
FILE *fi,*fo,*fop,*fk;
int flag=0,i=1;
char c;
char t,a[15],ch[15],file[20];

printf("\n Enter the File Name : ");
scanf("%s",file);
fi=fopen(file,"r");
fo=fopen("inter.c","w");
fop=fopen("oper.c","r");
fk=fopen("key.c","r");
c=getc(fi);
while(!feof(fi))
{ if(isalpha(c)||isdigit(c)||(c=='['||c==']'||c=='.'==1))
  fputc(c,fo);
else
{ if(c=='\n')
fprintf(fo,"\t$\t");
else
fprintf(fo,"\t%c\t",c); }
c=getc(fi);
}
fclose(fi);
fclose(fo);
fi=fopen("inter.c","r");
printf("\n Lexical Analysis ");
fscanf(fi,"%s",a);
printf("\n Line : %d\n",i++);
```

```
while(!feof(fi))
{ if(strcmp(a,"$")==0)
{
printf("\n Line : %d \n",i++);
fscanf(fi,"%s",a);
}
fscanf(fop,"%s",ch);
while(!feof(fop))
{
if(strcmp(ch,a)==0)
{
fscanf(fop,"%s",ch);
printf("\t\t%s\t:\t%s\n",a,ch);
flag=1;
} fscanf(fop,"%s",ch);
}
rewind(fop);
fscanf(fk,"%s",ch);
while(!feof(fk))
{
if(strcmp(ch,a)==0)
{
fscanf(fk,"%k",ch);
printf("\t\t%s\t:\tKeyword\n",a);
flag=1;
}
fscanf(fk,"%s",ch);
}
rewind(fk);
if(flag==0)
{
if(isdigit(a[0]))
printf("\t\t%s\t:\tConstant\n",a);
else
printf("\t\t%s\t:\tIdentifier\n",a);
}
flag=0;
fscanf(fi,"%s",a); }
```

```
//input.c

  #include<stdio.h>
  #include<conio.h>
  void main()
  {
  int a=10,b,c;
  a=b*c;
  gets();
  }
```

//key.c

```
int
void
main
char
if
for
while
else
printf
scanf
FILE
include
stdio.h
conio.h
iostream.h
```

//oper.c

```
( open para
) close para
{ open brace
} close brace
< lesser
> greater
" doublequote
' singlequote
: colon
; semicolon
# preprocessor
== equal
= asign
% percentage
^ bitwise
& reference
* star
+ add
- sub
\ backslash
/ slash
```

OUTPUT:

```
unicomplexity@UCS-OU: ~/OUCS/CPL_DSG
unicomplexity@UCS-OU:~/OUCS/CPL_DSG$ cc Program.c -w
unicomplexity@UCS-OU:~/OUCS/CPL_DSG$ ./a.out

 Enter the File Name : input.c

 Lexical Analysis
 Line : 1
                #       :       preprocessor
                include :       Keyword
                <       :       lesser
                stdio.h :       Keyword
                >       :       greater

 Line : 2
                #       :       preprocessor
                include :       Keyword
                <       :       lesser
                conio.h :       Keyword
                >       :       greater

 Line : 3
                void    :       Keyword
                main    :       Keyword
                (       :       open
                )       :       close

 Line : 4
                {       :       open
```

```
unicomplexity@UCS-OU: ~/OUCS/CPL_DSG
 Line : 5
                int     :       Keyword
                a       :       Identifier
                =       :       asign
                10      :       Constant
                ,       :       Identifier
                b       :       Identifier
                ,       :       Identifier
                c       :       Identifier
                ;       :       semicolon

 Line : 6
                a       :       Identifier
                =       :       asign
                b       :       Identifier
                *       :       star
                c       :       Identifier
                ;       :       semicolon

 Line : 7
                gets    :       Identifier
                (       :       open
                )       :       close
                ;       :       semicolon

 Line : 8
                }       :       close
```

```
unicomplexity@UCS-OU: ~/OUCS/CPL_DSG
unicomplexity@UCS-OU:~/OUCS/CPL_DSG$ ./a.out

 Enter the File Name : key.c

 Lexical Analysis
 Line : 1
                int     :       Keyword

 Line : 2
                void    :       Keyword

 Line : 3
                main    :       Keyword

 Line : 4
                char    :       Keyword

 Line : 5
                if      :       Keyword

 Line : 6
                for     :       Keyword

 Line : 7
                while   :       Keyword

 Line : 8
                else    :       Keyword
```

```
unicomplexity@UCS-OU: ~/OUCS/CPL_DSG

 Line : 8
                else    :       Keyword

 Line : 9
                printf  :       Keyword

 Line : 10
                scanf   :       Keyword

 Line : 11
                FILE    :       Keyword

 Line : 12
                include :       Keyword

 Line : 13
                stdio.h :       Keyword

 Line : 14
                conio.h :       Keyword

 Line : 15
                iostream.h      :       Keyword

 Line : 16
                $       :       Identifier
unicomplexity@UCS-OU:~/OUCS/CPL_DSG$
```

```
unicomplexity@UCS-OU: ~/OUCS/CPL_DSG
Enter the File Name : oper.c

Lexical Analysis
Line : 1
              (       :       open
              open    :       para
              open    :       brace
              para    :       )
              para    :       {

Line : 2
              )       :       close
              close   :       para
              close   :       brace
              para    :       )
              para    :       {

Line : 3
              {       :       open
              open    :       para
              open    :       brace
              brace   :       }
              brace   :       <

Line : 4
              }       :       close
              close   :       para
              close   :       brace
              brace   :       }
              brace   :       <
```

```
unicomplexity@UCS-OU: ~/OUCS/CPL_DSG

Line : 5
              <       :       lesser
              lesser  :       >

Line : 6
              >       :       greater
              greater :       "

Line : 7
              "       :       doublequote
              doublequote     :       '

Line : 8
              '       :       singlequote
              singlequote     :       :

Line : 9
              :       :       colon
              colon   :       ;

Line : 10
              ;       :       semicolon
              semicolon       :       #

Line : 11
              #       :       preprocessor
              preprocessor    :       ==

Line : 12
```

```
unicomplexity@UCS-OU: ~/OUCS/CPL_DSG
                preprocessor    :       ==

Line : 12
                =       :       asign
                =       :       asign
                Equal   :       =

Line : 13
                =       :       asign
                asign   :       %

Line : 14
                %       :       percentage
                percentage      :       ^

Line : 15
                ^       :       bitwise
                bitwise :       &

Line : 16
                &       :       reference
                reference       :       *

Line : 17
                *       :       star
                star    :       +

Line : 18
                +       :       add
                add     :       -
```

```
unicomplexity@UCS-OU: ~/OUCS/CPL_DSG
                ^       :       bitwise
                bitwise :       &

Line : 16
                &       :       reference
                reference       :       *

Line : 17
                *       :       star
                star    :       +

Line : 18
                +       :       add
                add     :       -

Line : 19
                -       :       sub
                sub     :       \

Line : 20
                \       :       backslash
                backslash       :       /

Line : 21
                /       :       slash
                slash   :       slash

Line : 22
                $       :       Identifier
unicomplexity@UCS-OU:~/OUCS/CPL_DSG$
```

[4] Write a Program to Develop a Lexical Analyzer using Lex Tool.

//P4.l

```
%{
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 |

struct |

typedef |

do |

if |

break |

continue |

void |

switch |

return |

else |

goto { printf("\t\n%s is a keyword \n ",yytext);}

"/*" { COMMENT=1;}{ printf("\t %s is a COMMENT \n",yytext);}

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

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

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

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

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

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

\)(\:)? { if(!COMMENT) printf("\n\t");ECHO;printf("\n");}

\( ECHO;

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

\<= |

\>= |

\< |

== |

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

%%

int main(int argc, char **argv)
{
FILE *file;
file=fopen("var.c","r"); //Create a C Program File Named var.c in Same Directory
if(!file)
{
printf("could not open the file");
exit(0);
}
yyin=file;
yylex();
printf("\n");
return(0);
}
int yywrap()
{
return(1);
}
```

OUTPUT :

```
unicomplexity@UCS-OU: ~/OUCS/CPL_DSG/LEX-YACC
unicomplexity@UCS-OU:~/OUCS/CPL_DSG/LEX-YACC$ lex P4.l
unicomplexity@UCS-OU:~/OUCS/CPL_DSG/LEX-YACC$ cc lex.yy.c
unicomplexity@UCS-OU:~/OUCS/CPL_DSG/LEX-YACC$ ./a.out

#include<stdio.h> is a preprocessor directive

#include<conio.h> is a preprocessor directive

void is a keyword
   FUNCTION
        main(
        )

 BLOCK BEGINS


int is a keyword
  a IDENTIFIER
,b IDENTIFIER
,c IDENTIFIER
;
a IDENTIFIER
         = is an ASSIGNMENT OPERATOR
1 is a NUMBER
;
b IDENTIFIER
         = is an ASSIGNMENT OPERATOR
2 is a NUMBER
```

```
unicomplexity@UCS-OU: ~/OUCS/CPL_DSG/LEX-YACC
int is a keyword
  a IDENTIFIER
,b IDENTIFIER
,c IDENTIFIER
;
a IDENTIFIER
         = is an ASSIGNMENT OPERATOR
1 is a NUMBER
;
b IDENTIFIER
         = is an ASSIGNMENT OPERATOR
2 is a NUMBER
;
c IDENTIFIER
         = is an ASSIGNMENT OPERATOR
a IDENTIFIER
+b IDENTIFIER
;
 FUNCTION
        printf(  "Sum:%d" is a STRING
,c IDENTIFIER

        )
;
 BLOCK ENDS


unicomplexity@UCS-OU:~/OUCS/CPL_DSG/LEX-YACC$
```

[5] Write a Program to Recognize a Valid Arithmetic Expression using YACC.

//P5.l

```
%{
#include<stdio.h>
#include "y.tab.h"
%}

%%
[a-zA-Z]+ return VARIABLE;
[0-9]+ return NUMBER;
[\t] ;
[\n] return 0;
. return yytext[0];
%%

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

//P5.y

```
%{
    #include<stdio.h>
%}
%token NUMBER
%token VARIABLE

%left '+' '-'
%left '*' '/' '%'
%left '(' ')'

%%

S: VARIABLE'='E {
       printf("\nEntered arithmetic expression is Valid\n\n");
       return 0;
     }

E:E'+'E
 |E'-'E
 |E'*'E
 |E'/'E
 |E'%'E
 |'('E')'
 | NUMBER
 | VARIABLE
;

%%
```

```
void main()
{

printf("\n Enter Arithmetic Expression Having Operations [+,-,*,/,%] and Round
Brackets : \n");
   yyparse();
}

void yyerror()
{
   printf("\nEntered arithmetic expression is Invalid\n\n");

}
```

OUTPUT:

```
unicomplexity@UCS-OU: ~/OUCS/CPL_DSG/LEX-YACC
unicomplexity@UCS-OU:~/OUCS/CPL_DSG/LEX-YACC$ lex P5.l
unicomplexity@UCS-OU:~/OUCS/CPL_DSG/LEX-YACC$ yacc -d P5.y
unicomplexity@UCS-OU:~/OUCS/CPL_DSG/LEX-YACC$ cc lex.yy.c y.tab.c -w
unicomplexity@UCS-OU:~/OUCS/CPL_DSG/LEX-YACC$ ./a.out

 Enter Arithmetic Expression Having Operations [+,-,*,/,%] and Round Brackets :
a=b+c-d*e/f%g

Entered arithmetic expression is Valid

unicomplexity@UCS-OU:~/OUCS/CPL_DSG/LEX-YACC$ ./a.out

 Enter Arithmetic Expression Having Operations [+,-,*,/,%] and Round Brackets :
a=(2+3)*(5-3)/(20%30)

Entered arithmetic expression is Valid

unicomplexity@UCS-OU:~/OUCS/CPL_DSG/LEX-YACC$ ./a.out

 Enter Arithmetic Expression Having Operations [+,-,*,/,%] and Round Brackets :
a=b+c-10+(i/10)

Entered arithmetic expression is Valid

unicomplexity@UCS-OU:~/OUCS/CPL_DSG/LEX-YACC$
```

[6] Write a Program to Recognize a Valid Variable Name using YACC.

//P6.l

```
%{
    #include "y.tab.h"
%}

%%

[a-zA-Z_][a-zA-Z_0-9]* return letter;

[0-9] return digit;

.  return yytext[0];

\n return 0;

%%

int yywrap()

{
return 1;
}
```

//P6.y

```
%{
    #include<stdio.h>

    int valid=1;
%}

%token digit letter

%%

start : letter s

s :     letter s

      | digit s

      |

      ;

%%
```

```
int yyerror()
{
    printf("\nIts not a identifier!\n");

    valid=0;

    return 0;
}

int main()

{
    printf("\n Enter a name to tested for identifier \n");

    yyparse();

    if(valid)

    {
        printf("\nIt is a identifier!\n");
    }

}
```

OUTPUT:

```
unicomplexity@UCS-OU: ~/OUCS/CPL_DSG/LEX-YACC
unicomplexity@UCS-OU:~/OUCS/CPL_DSG/LEX-YACC$ lex P6.l
unicomplexity@UCS-OU:~/OUCS/CPL_DSG/LEX-YACC$ yacc -d P6.y
unicomplexity@UCS-OU:~/OUCS/CPL_DSG/LEX-YACC$ cc lex.yy.c y.tab.c -w
unicomplexity@UCS-OU:~/OUCS/CPL_DSG/LEX-YACC$ ./a.out

 Enter a name to tested for identifier
A

It is a identifier!
unicomplexity@UCS-OU:~/OUCS/CPL_DSG/LEX-YACC$ ./a.out

 Enter a name to tested for identifier
id

It is a identifier!
unicomplexity@UCS-OU:~/OUCS/CPL_DSG/LEX-YACC$ ./a.out

 Enter a name to tested for identifier
E

It is a identifier!
unicomplexity@UCS-OU:~/OUCS/CPL_DSG/LEX-YACC$ ./a.out

 Enter a name to tested for identifier
26

Its not a identifier!
unicomplexity@UCS-OU:~/OUCS/CPL_DSG/LEX-YACC$
```

[7] Write a Program to Implement Calculator using Lex and YACC.

//P7-LEX.l (Program to Implement Calculator using Lex)

```
%{
int op = 0,i;
float a, b;
%}

dig [0-9]+|([0-9]*)"."([0-9]+)
add "+"
sub "-"
mul "*"
div "/"
pow "^"
ln \n

%%

{dig} { digi();}
{add} { op=1;}
{sub} { op=2;}
{mul} { op=3;}
{div} { op=4;}
{pow} { op=5;}
{ln} { printf("\n The Answer : %f\n\n",a);}

%%
digi()
{
if(op==0)


a=atof(yytext);

else
{
b=atof(yytext);


switch(op)
{
case 1:a=a+b;
    break;

case 2:a=a-b;
        break;

case 3:a=a*b;
        break;

case 4:a=a/b;
        break;
```

```
case 5:for(i=a;b>1;b--)
            a=a*i;
             break;
}
op=0;
}
}

main(int argv,char *argc[])
{
printf("\n Enter Two Values using Operators [+,-,*,/] : \n");
yylex();
}

yywrap()
{
return 1;
}
```

OUTPUT:

```
unicomplexity@UCS-OU: ~/OUCS/CPL_DSG/LEX-YACC
unicomplexity@UCS-OU:~/OUCS/CPL_DSG/LEX-YACC$ lex P7-LEX.l
unicomplexity@UCS-OU:~/OUCS/CPL_DSG/LEX-YACC$ cc lex.yy.c -w
unicomplexity@UCS-OU:~/OUCS/CPL_DSG/LEX-YACC$ ./a.out

 Enter Two Values using Operators [+,-,*,/] :
2022+2023

 The Answer : 4045.000000

2023-3

 The Answer : 2020.000000

2020+2021+2022+2023

 The Answer : 8086.000000

2020+2021-2019/2023*2000

 The Answer : 1999.011353
```

/*Program to Implement Calculator using YACC*/

//P7-Y.l

```
%{
#include<stdio.h>
#include "y.tab.h"
extern int yylval;
%}

%%

[0-9]+ { yylval=atoi(yytext);
          return NUMBER; }
[\t] ;

[\n] return 0;

. return yytext[0];

%%

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

//P7-Y.y

```
%{
#include<stdio.h>
int flag=0;
%}

%token NUMBER
%left '+' '-'
%left '*' '/' '%'
%left '(' ')'


%%
ArithmeticExpression: E{ printf("\n Result of Operation : %d\n", $$); return 0;
                      };
E:E'+'E {$$=$1+$3;}
|E'-'E {$$=$1-$3;}
|E'*'E {$$=$1*$3;}
|E'/'E {$$=$1/$3;}
|E'%'E {$$=$1%$3;}
|'('E')' {$$=$2;}
```

```
| NUMBER {$$=$1;}
;
%%

void main()
{
printf("\nENTER ARITHMETIC OPERATION HAVING OPERATORS [ +,-,*,/,% ] AND ROUND
BRACKETS : \n");

yyparse();
if(flag==0)
printf("\n Entered Arithmetic Expression is Valid\n\n");
}

void yyerror()
{
printf("\n Entered Arithmetic Expression is Invalid\n\n");
flag=1;
}
```

OUTPUT:

```
unicomplexity@UCS-OU: ~/OUCS/CPL_DSG/LEX-YACC
unicomplexity@UCS-OU:~/OUCS/CPL_DSG/LEX-YACC$ lex P7-Y.l
unicomplexity@UCS-OU:~/OUCS/CPL_DSG/LEX-YACC$ yacc -d P7-Y.y
unicomplexity@UCS-OU:~/OUCS/CPL_DSG/LEX-YACC$ cc lex.yy.c y.tab.c -w
unicomplexity@UCS-OU:~/OUCS/CPL_DSG/LEX-YACC$ ./a.out

ENTER ARITHMETIC OPERATION HAVING OPERATORS [ +,-,*,/,% ] AND ROUND BRACKETS :
10+20-5*30/5%2

 Result of Operation : 30

 Entered Arithmetic Expression is Valid

unicomplexity@UCS-OU:~/OUCS/CPL_DSG/LEX-YACC$ ./a.out

ENTER ARITHMETIC OPERATION HAVING OPERATORS [ +,-,*,/,% ] AND ROUND BRACKETS :
(10+20)-(5*30)/(5%2)

 Result of Operation : -120

 Entered Arithmetic Expression is Valid

unicomplexity@UCS-OU:~/OUCS/CPL_DSG/LEX-YACC$ ./a.out

ENTER ARITHMETIC OPERATION HAVING OPERATORS [ +,-,*,/,% ] AND ROUND BRACKETS :
((10+20)*(40-10)/(121/11)%100)

 Result of Operation : 81

 Entered Arithmetic Expression is Valid
```

[8] Write a Program for Implementing Type Checking for Given Expression.

```
#include<stdio.h>
#include<stdlib.h>

int main()
{
int n,i,k,flag=0;
char vari[15],typ[15],b[15],c;
printf("\nEnter the number of variables : ");
scanf(" %d",&n);
for(i=0;i<n;i++)
{
printf("\nEnter the variable[%d] : ",i);
scanf(" %c",&vari[i]);
printf("Enter the variable-type[%d](float-f,int-i) : ",i);
scanf(" %c",&typ[i]);
if(typ[i]=='f')
flag=1;
}
printf("\nEnter the Expression(end with $) : ");
i=0;
getchar();
while((c=getchar())!='$')
{
b[i]=c;
i++;  }
k=i;
for(i=0;i<k;i++)
{
if(b[i]=='/')
{
flag=1;
break;  }  }
for(i=0;i<n;i++)
{
if(b[0]==vari[i])
{
```

```
if(flag==1)
{
if(typ[i]=='f')
{ printf("\nThe Datatype is Correctly Defined..!\n");
break; }
else
{
printf("Identifier %c must be a Float Type..!\n",vari[i]);
break;
}
}
else
{ printf("\nThe Datatype is Correctly Defined..!\n");
break; }
}
}
return 0;
}
```

OUTPUT:

```
unicomplexity@UCS-OU: ~/OUCS/CPL_DSG
unicomplexity@UCS-OU:~/OUCS/CPL_DSG$ cc P8.c
unicomplexity@UCS-OU:~/OUCS/CPL_DSG$ ./a.out

Enter the number of variables : 3

Enter the variable[0] : a
Enter the variable-type[0](float-f,int-i) : f

Enter the variable[1] : b
Enter the variable-type[1](float-f,int-i) : i

Enter the variable[2] : c
Enter the variable-type[2](float-f,int-i) : f

Enter the Expression(end with $) : c=a+b$

The Datatype is Correctly Defined..!
unicomplexity@UCS-OU:~/OUCS/CPL_DSG$ ./a.out

Enter the number of variables : 2

Enter the variable[0] : a
Enter the variable-type[0](float-f,int-i) : i

Enter the variable[1] : b
Enter the variable-type[1](float-f,int-i) : f

Enter the Expression(end with $) : a=b+10$
Identifier a must be a Float Type..!
unicomplexity@UCS-OU:~/OUCS/CPL_DSG$
```

[9] Write a Program to Convert the BNF Rules into YACC.

//P9.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];

%%
```

//P9.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\tPos
Operator\tArg1\tArg2 \tResult" "\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);
}
```

OUTPUT:

```
unicomplexity@UCS-OU: ~/OUCS/CPL_DSG/LEX-YACC
unicomplexity@UCS-OU:~/OUCS/CPL_DSG/LEX-YACC$ lex P9.l
unicomplexity@UCS-OU:~/OUCS/CPL_DSG/LEX-YACC$ yacc -d P9.y
unicomplexity@UCS-OU:~/OUCS/CPL_DSG/LEX-YACC$ cc lex.yy.c y.tab.c -ll -lm -w
unicomplexity@UCS-OU:~/OUCS/CPL_DSG/LEX-YACC$ ./a.out Test.c


                --------------------------------------
                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
                --------------------------------------
```

[10] Write a Program to Implement Data Flow and Control Flow Analysis.

DATA FLOW ANALYSIS

//P10-DFA.c

```
#include<stdio.h>
#include<string.h>
#include<ctype.h>

void input();
void output();
void change(int p,int q,char *res);
void constant();
void expression();

struct expr
{
char op[2],op1[5],op2[5],res[5];
int flag;
} arr[10]; int n;

int main()
{
int ch=0;

input();
constant();
expression();
output();
}

void input()
{
int i;
printf("\n\nEnter the maximum number of expressions:");
scanf("%d",&n);
printf("\nEnter the input : \n");
```

```
for(i=0;i<n;i++)
{
scanf("%s",arr[i].op);
scanf("%s",arr[i].op1);
scanf("%s",arr[i].op2);
scanf("%s",arr[i].res);

arr[i].flag=0;
}
}

void constant()
{
int i;
int op1,op2,res;
char op,res1[5];
for(i=0;i<n;i++)
{
if(isdigit(arr[i].op1[0]) && isdigit(arr[i].op2[0]))
{
op1=atoi(arr[i].op1);
op2=atoi(arr[i].op2);
op=arr[i].op[0];

switch(op)
{
case '+':
res=op1+op2;
break;

case '-':
res=op1-op2;
break;

case '*':
res=op1*op2;
```

```
break;

case '/':
res=op1/op2;
break;

}

sprintf(res1,"%d",res);
arr[i].flag=1;
change(i,i,res1);

}
}
}
void expression()
{
int i,j;
for(i=0;i<n;i++)
{
for(j=i+1;j<n;j++)
{
if(strcmp(arr[i].op,arr[j].op)==0)
{
if(strcmp(arr[i].op,"+")==0||strcmp(arr[i].op,"*")==0)
{
if(strcmp(arr[i].op1,arr[j].op1)==0&&strcmp(arr[i].op2,arr[j].op2)==0 ||
strcmp(arr[i].op1,arr[j].op2)==0&&strcmp(arr[i].op2,arr[j].op1)==0)
{
arr[j].flag=1;
change(i,j,NULL);
}
}
else
{
```

```
if(strcmp(arr[i].op1,arr[j].op1)==0 && strcmp(arr[i].op2,arr[j].op2)==0)
{
arr[j].flag=1;
change(i,j,NULL);
}          }

}          }

}          }

void output()
{
int i=0;
printf("\nOptimized code is : \n");
for(i=0;i<n;i++)
{
if(!arr[i].flag)
{
printf("\n%s %s %s %s\n",arr[i].op,arr[i].op1,arr[i].op2,arr[i].res);
}
}
}
void change(int p,int q,char *res)
{
int i;

for(i=q+1;i<n;i++)
{
if(strcmp(arr[q].res,arr[i].op1)==0)
if(res == NULL)
strcpy(arr[i].op1,arr[p].res);
else
strcpy(arr[i].op1,res);
else if(strcmp(arr[q].res,arr[i].op2)==0)
if(res == NULL)
strcpy(arr[i].op2,arr[p].res);
```

```
else
strcpy(arr[i].op2,res);
}
}
```

OUTPUT:

```
unicomplexity@UCS-OU: ~/OUCS/CPL_DSG
unicomplexity@UCS-OU:~/OUCS/CPL_DSG$ cc P10-DFA.c -w
unicomplexity@UCS-OU:~/OUCS/CPL_DSG$ ./a.out


Enter the Maximum Number of Expressions : 5

Enter the Input :
+ 4 2 T1
+ A T1 T2
- B A T3
+ A 6 T4
+ T3 T4 T5

Optimized Code is :

+ A 6 T2

- B A T3

+ T3 T2 T5
unicomplexity@UCS-OU:~/OUCS/CPL_DSG$
```

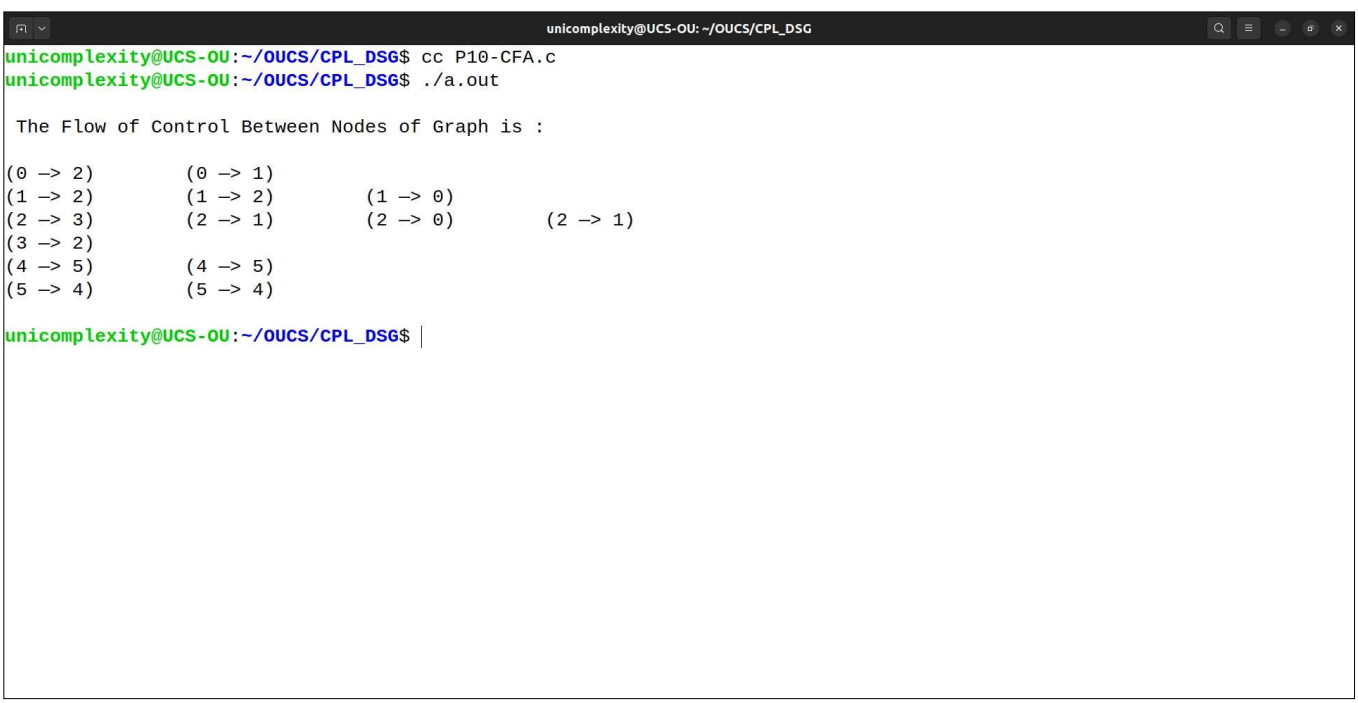
CONTROL FLOW ANALYSIS

//P10-CFA.c

```
#include <stdio.h>
#include <stdlib.h>

    // Define the maximum number of vertices in the graph
    #define N 6

    struct Graph
    { struct Node* head[N];
    };

    struct Node
    { int dest;
      struct Node* next;
    };

    struct Edge
    { int src, dest;
    };

    struct Graph* createGraph(struct Edge edges[], int n)
    { struct Graph* graph = (struct Graph*)malloc(sizeof(struct Graph));

        for (int i = 0; i < N; i++)
           { graph->head[i] = NULL;}

        for (int i = 0; i < n; i++)
        {   int src = edges[i].src;
            int dest = edges[i].dest;
            struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
            newNode->dest = dest;
            newNode->next = graph->head[src];
            graph->head[src] = newNode;
            newNode = (struct Node*)malloc(sizeof(struct Node));
            newNode->dest = src;
            newNode->next = graph->head[dest];
            graph->head[dest] = newNode; }
        return graph; }

    void printGraph(struct Graph* graph)
    { for (int i = 0; i < N; i++)
        { struct Node* ptr = graph->head[i];
            while (ptr != NULL)
            {   printf("(%d –> %d)\t", i, ptr->dest);
                ptr = ptr->next; }
```

```
            printf("\n"); }
    }

    int main(void)
    {   struct Edge edges[] =
        { {0, 1}, {1, 2}, {2, 0}, {2, 1}, {3, 2}, {4, 5}, {5, 4} };

        int n = sizeof(edges)/sizeof(edges[0]);
        struct Graph *graph = createGraph(edges, n);
        printf("\n The Flow of Control Between Nodes of Graph is : \n");
        printf("\n");
        printGraph(graph);
        printf("\n");
        return 0;
    }
```

OUTPUT:



[11] Write a Program to Implement Stack Storage Allocation Strategies.

```
#include<stdio.h>
#include<stdlib.h>
#define size 5

struct stack
{
int s[size];
int top;
} st;

int stfull()
{
if (st.top >= size - 1)
return 1;
else
return 0;
}

void push(int item)
{
st.top++;
st.s[st.top] = item;
}

int stempty()
{
if (st.top == -1)
return 1;
else
return 0;
}

int pop()
{
int item;
item = st.s[st.top];
st.top--;
return (item);
}

void display()
{
int i;
if (stempty())
printf("\nSTACK IS EMPTY !\n");
else
{
for (i = st.top; i >= 0; i--)
printf("\n%d", st.s[i]);
}
}
```

```
int main()
{
int item, choice;
char ans;
st.top = -1;
printf("\n\t IMPLEMENTATION OF STACK ");

do {
printf("\n MAIN MENU \n");
printf("\n1.PUSH \n2.POP \n3.DISPLAY \n4.EXIT");
printf("\n ENTER YOUR CHOICE : ");
scanf("%d", &choice);

switch (choice)
{
case 1:
printf("\nEnter The Item to be Pushed : ");
scanf("%d", &item);
if (stfull())
printf("\nStack is Full! \n");
else
push(item);
break;

case 2:
if (stempty())
printf("\nEmpty stack! Underflow !!");
else
{
item = pop();
printf("\nThe Popped Element is : %d", item);
}
break;

case 3:
display();
break;

case 4:
goto halt;
}

printf("\nDO YOU WANT TO CONTINUE (Y/N) ?\n");
scanf(" %c",&ans);

}
while (ans == 'Y' || ans == 'y');
halt:
return 0;

}
```

OUTPUT:

```
unicomplexity@UCS-OU: ~/OUCS/CPL_DSG
unicomplexity@UCS-OU:~/OUCS/CPL_DSG$ cc P11.c
unicomplexity@UCS-OU:~/OUCS/CPL_DSG$ ./a.out

         IMPLEMENTATION OF STACK
 MAIN MENU

1.PUSH
2.POP
3.DISPLAY
4.EXIT
 ENTER YOUR CHOICE : 1

Enter The Item to be Pushed : 2022

DO YOU WANT TO CONTINUE (Y/N) ?
Y

 MAIN MENU

1.PUSH
2.POP
3.DISPLAY
4.EXIT
 ENTER YOUR CHOICE : 1

Enter The Item to be Pushed : 2023

DO YOU WANT TO CONTINUE (Y/N) ?
```

```
unicomplexity@UCS-OU: ~/OUCS/CPL_DSG
Y

 MAIN MENU

1.PUSH
2.POP
3.DISPLAY
4.EXIT
 ENTER YOUR CHOICE : 1

Enter The Item to be Pushed : 2023

DO YOU WANT TO CONTINUE (Y/N) ?
Y

 MAIN MENU

1.PUSH
2.POP
3.DISPLAY
4.EXIT
 ENTER YOUR CHOICE : 3

2023
2022
DO YOU WANT TO CONTINUE (Y/N) ?
N
unicomplexity@UCS-OU:~/OUCS/CPL_DSG$
```

[12] Write a Program to Implement Heap Storage Allocation Strategies.

```
#include<stdio.h>
#include<stdlib.h>
#define TRUE 1
#define FALSE 0

typedef struct Heap
{
int data;
struct Heap *next;
}
node;
node *create();

void main()
{
int choice,val;
char ans;
node *head;
void display(node *);
node *search(node *,int);
node *insert(node *);
void dele(node **);
head=NULL;
do{
printf("\n PROGRAM TO PERFORM VARIOUS OPERATIONS ON HEAP USING DYNAMIC MEMORY
MANAGEMENT");
printf("\n1.CREATE");
printf("\n2.DISPLAY");
printf("\n3.INSERT AN ELEMENT IN A LIST");
printf("\n4.DELETE AN ELEMENT FROM LIST");
printf("\n5.QUIT");
printf("\nENTER YOUR CHIOCE (1-5) : ");
scanf("%d",&choice);
switch(choice){
case 1:head=create();
break;
case 2:display(head);
break;
case 3:head=insert(head);
break;
case 4:dele(&head);
break;
case 5:exit(0);
default:
```

```
printf("INVALID CHOICE, TRY AGAIN");
}
}
while(choice!=5);
}
node* create()
{
node *temp,*New,*head;
int val,flag;
char ans='y';
node *get_node();
temp=NULL;
flag=TRUE;
do
{
printf("\n ENTER THE ELEMENT : ");
scanf("%d",&val);
New=get_node();
if(New==NULL)
printf("\n MEMORY IS NOT ALLOCATED \n");
New->data=val;
if(flag==TRUE)
{
head=New;
temp=head;
flag=FALSE;
}
else
{
temp->next=New;
temp=New;
}
printf("\n DO YOU WANT TO ENTER MORE ELEMENTS ? (Y/N) : ");
scanf(" %c",&ans);
}
while(ans=='y'||ans=='Y');
printf("\nTHE LIST IS CREATED\n");
return head;
}
node *get_node()
{
node *temp;
temp=(node*)malloc(sizeof(node));
temp->next=NULL;
return temp;
}
void display(node *head){
```

```
node *temp;
temp=head;
if(temp==NULL)
{
printf("\nTHE LIST IS EMPTY\n");
return;
}
while(temp!=NULL)
{
printf("%d->",temp->data);
temp=temp->next;
}
printf("NULL");
printf("\n");
}
node *search(node *head,int key)
{
node *temp;
int found;
temp=head;
if(temp==NULL)
{
printf("THE LINKED LIST IS EMPTY\n");
return NULL;
}
found=FALSE;
while(temp!=NULL && found==FALSE)
{
if(temp->data!=key)
temp=temp->next;
else
found=TRUE;
}
if(found==TRUE)
{
printf("\nTHE ELEMENT IS PRESENT IN THE LIST\n");
return temp;
}
else{
printf("THE ELEMENT IS NOT PRESENT IN THE LIST\n");
return NULL;
}
}
node *insert(node *head)
{
int choice;
node *insert_head(node *);
```

```
void insert_after(node *);
void insert_last(node *);
printf("\n1. INSERT A NODE AS A HEAD NODE");
printf("\n2. INSERT A NEW NODE AS A HEAD NODE");
printf("\n3. INSERT A NODE AT INTERMEDIATE POSITION IN THE LIST");
printf("\n ENTER YOUR CHOICE FOR INSERTION OF NODE : \n");
scanf("%d",&choice);
switch(choice){
case 1:head=insert_head(head);
break;
case 2:insert_last(head);
break;
case 3:insert_after(head);
break;
}
return head;
}
node *insert_head(node *head)
{
node *New,*temp;
New=get_node();
printf("\nENTER THE ELEMENT WHICH YOU WANT TO INSERT\n");
scanf("%d",&New->data);
if(head==NULL)
head=New;
else
{
temp=head;
New->next=temp;
head=New;
}
return head;
}
void insert_last(node *head)
{
node *New,*temp;
New=get_node();
printf("\n ENTER THE ELEMENT WHICH YOU WANT TO INSERT \n ");
scanf("%d",&New->data);
if(head==NULL)
head=New;
else
{
temp=head;
while(temp->next!=NULL)
temp=temp->next;
temp->next=New;
```

```
New->next=NULL;
}
}
void insert_after(node *head)
{
int key;
node *New,*temp;
New=get_node();
printf("\n ENTER THE ELEMENTS WHICH YOU WANT TO INSERT \n");
scanf("%d",&New->data);
if(head==NULL)
{
head=New;
}
else
{
printf("\ENTER THE ELEMENT WHICH YOU WANT TO INSERT IN THE NODE \n");
scanf("%d",&key);
temp=head;
do
{
if(temp->data==key)
{
New->next-temp->next;
temp->next=New;
return;
}
else
temp=temp->next;
}
while(temp!=NULL);
}
}
node *get_prev(node *head,int val)
{
node *temp,*prev;
int flag;
temp=head;
if(temp==NULL)
return NULL;
flag=FALSE;
prev=NULL;
while(temp!=NULL && ! flag)
{
if(temp->data!=val)
{
prev=temp;
temp=temp->next;
```

```
}
else
flag=TRUE;
}
if(flag)
return prev;
else
return NULL;
}
void dele(node **head)
{
node *temp,*prev;
int key;
temp=*head;
if(temp==NULL)
{
printf("\nTHE LIST IS EMPTY\n");
return;
}
printf("\n ENTER THE ELEMENT YOU WANT TO DELETE : ");
scanf("%d",&key);
temp=search(*head,key);
if(temp!=NULL)
{
prev=get_prev(*head,key);
if(prev!=NULL)
{
prev->next=temp->next;
free(temp);
}
else
{
*head=temp->next;
free(temp);
}
printf("\nTHE ELEMENT IS DELETED\n");

}
}
```

[1]

[1]

OUTPUT:

```
unicomplexity@UCS-OU: ~/OUCS/CPL_DSG
unicomplexity@UCS-OU:~/OUCS/CPL_DSG$ cc P12.c
unicomplexity@UCS-OU:~/OUCS/CPL_DSG$ ./a.out

 PROGRAM TO PERFORM VARIOUS OPERATIONS ON HEAP USING DYNAMIC MEMORY MANAGEMENT
1.CREATE
2.DISPLAY
3.INSERT AN ELEMENT IN A LIST
4.DELETE AN ELEMENT FROM LIST
5.QUIT
ENTER YOUR CHIOCE (1-5) : 1

 ENTER THE ELEMENT : 2020

 DO YOU WANT TO ENTER MORE ELEMENTS ? (Y/N) : Y

 ENTER THE ELEMENT : 2021

 DO YOU WANT TO ENTER MORE ELEMENTS ? (Y/N) : Y

 ENTER THE ELEMENT : 2022

 DO YOU WANT TO ENTER MORE ELEMENTS ? (Y/N) : Y

 ENTER THE ELEMENT : 2023

 DO YOU WANT TO ENTER MORE ELEMENTS ? (Y/N) : N

THE LIST IS CREATED
```

```
 PROGRAM TO PERFORM VARIOUS OPERATIONS ON HEAP USING DYNAMIC MEMORY MANAGEMENT
1.CREATE
2.DISPLAY
3.INSERT AN ELEMENT IN A LIST
4.DELETE AN ELEMENT FROM LIST
5.QUIT
ENTER YOUR CHIOCE (1-5) : 2
2020->2021->2022->2023->NULL

 PROGRAM TO PERFORM VARIOUS OPERATIONS ON HEAP USING DYNAMIC MEMORY MANAGEMENT
1.CREATE
2.DISPLAY
3.INSERT AN ELEMENT IN A LIST
4.DELETE AN ELEMENT FROM LIST
5.QUIT
ENTER YOUR CHIOCE (1-5) : 4

 ENTER THE ELEMENT YOU WANT TO DELETE : 2020

THE ELEMENT IS PRESENT IN THE LIST

THE ELEMENT IS DELETED

 PROGRAM TO PERFORM VARIOUS OPERATIONS ON HEAP USING DYNAMIC MEMORY MANAGEMENT
1.CREATE
2.DISPLAY
3.INSERT AN ELEMENT IN A LIST
4.DELETE AN ELEMENT FROM LIST
5.QUIT
ENTER YOUR CHIOCE (1-5) : 2
2021->2022->2023->NULL
```

[13] Write a Program to Construct a Directed Acyclic Graph (DAG).

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>

#define MIN_PER_RANK 1
#define MAX_PER_RANK 5
#define MIN_RANKS 3
#define MAX_RANKS 5
#define PERCENT 30

void main()
{
int i,j,k,nodes=0;
srand(time(NULL));
int ranks=MIN_RANKS+(rand()%(MAX_RANKS-MIN_RANKS+1));
printf("\n DIRECTED ACYCLIC GRAPH\n");
    for(i=1;i<ranks;i++)
    {
    int new_nodes=MIN_PER_RANK+(rand()%(MAX_PER_RANK-MIN_PER_RANK+1));
    for(j=0;j<nodes;j++)
    for(k=0;k<new_nodes;k++)
    if((rand()%100)<PERCENT)
    printf("%d->%d;\n",j,k+nodes);
    nodes+=new_nodes;
    }
}
```

OUTPUT:

```
unicomplexity@UCS-OU: ~/OUCS/CPL_DSG
unicomplexity@UCS-OU:~/OUCS/CPL_DSG$ cc P13.c
unicomplexity@UCS-OU:~/OUCS/CPL_DSG$ ./a.out

 DIRECTED ACYCLIC GRAPH
1->2;
0->4;
1->4;
1->5;
1->8;
2->7;
3->6;
3->8;
0->9;
0->11;
1->10;
3->12;
4->10;
5->10;
7->11;
7->12;
unicomplexity@UCS-OU:~/OUCS/CPL_DSG$
```

[14] Write a Program to Implement the Back End of the Compiler.

```
#include<stdio.h>
#include<stdio.h>
#include<string.h>

void main()
{
char icode[10][30],str[20],opr[10];
int i=0;

printf("\n Enter the Set of Intermediate Code and Terminate using exit : \n");
do
{
    scanf("%s",icode[i]);
}
while(strcmp(icode[i++],"exit")!=0);

printf("\n THE TARGET CODE / ASSEMBLY CODE GENERATED IS ");
printf("\n----------------------------------------------\n");
i=0;

do
{
    strcpy(str,icode[i]);
    switch(str[3])
    {
    case'+':
    strcpy(opr,"ADD ");
    break;
    case'-':
    strcpy(opr,"SUB ");
    break;
    case'*':
    strcpy(opr,"MUL ");
    break;
    case'/':
    strcpy(opr,"DIV ");
    break;
    }
    printf("\n\tMOV %c,R%d",str[2],i);
    printf("\n\t%s%c,R%d",opr,str[4],i);
    printf("\n\tMOV R%d,%c",i,str[0]);
    printf("\n");
    }
while(strcmp(icode[++i],"exit")!=0);
}
```

OUTPUT:

```
unicomplexity@UCS-OU: ~/OUCS/CPL_DSG
unicomplexity@UCS-OU:~/OUCS/CPL_DSG$ cc P14.c
unicomplexity@UCS-OU:~/OUCS/CPL_DSG$ ./a.out

 Enter the Set of Intermediate Code and Terminate using exit  :
A=B+C
D=A+C
F=X*Y
Y=M/N
exit

 THE TARGET CODE / ASSEMBLY CODE GENERATED IS
----------------------------------------------

        MOV B,R0
        ADD C,R0
        MOV R0,A

        MOV A,R1
        ADD C,R1
        MOV R1,D

        MOV X,R2
        MUL Y,R2
        MOV R2,F

        MOV M,R3
        DIV N,R3
        MOV R3,Y
```

[15] Write a Program to Implement Simple Code Optimization Technique.

```
  #include<stdio.h>

  int main() {
  int n,fact=1,i;
  printf("enter the n value");
  scanf("%d", & n);
  for(i=1;i<n;i++)
  {
       fact=fact*i;
       Printf("the factorial of a given number is fact=%d",fact);
  }
}
```

OUTPUT:

```
unicomplexity@UCS-OU: ~/OUCS/CPL_DSG
unicomplexity@UCS-OU:~/OUCS/CPL_DSG$ cc P15.c -w
unicomplexity@UCS-OU:~/OUCS/CPL_DSG$ ./a.out

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
B = C+D
E = C+D
F = B+E
R = F
```

```
unicomplexity@UCS-OU: ~/OUCS/CPL_DSG
Right: F

Intermediate Code
A = 9
B = C+D
E = C+D
F = B+E
R = F

After Dead Code Elimination
B = C+D
E = C+D
F = B+E
R = F

pos: 2

Eliminate Common Expression
B = C+D
B = C+D
F = B+B
R = F

Optimized Code
B = C+D
F = B+B
R = F
unicomplexity@UCS-OU:~/OUCS/CPL_DSG$
```

[16] Write a Program to Implement Simple Code Optimization Technique using do-while.

```
  #include <stdio.h>
  int main() {
  int n,fact=1,i;
  printf("enter the n value");
  scanf("%d", & n);

  do {
  fact=fact*n;
  n--;

  }while(n>0);
  Printf(“the factorial of a given number is fact=%d”,fact);
}
```

OUTPUT:

```
unicomplexity@UCS-OU: ~/OUCS/CPL_DSG
unicomplexity@UCS-OU:~/OUCS/CPL_DSG$ cc P16.c
unicomplexity@UCS-OU:~/OUCS/CPL_DSG$ ./a.out

 ENTER SIZE OF LIST : 10

 ENTER ELEMENTS OF LIST
08
15
10
18
12
19
14
21
20
22

 LIST AFTER SORTING : 8 10 12 14 15 18 19 20 21 22
unicomplexity@UCS-OU:~/OUCS/CPL_DSG$
```