LAB MANUAL

Subject : COMPILER DESIGN LAB

Code : CSE-411-F

Department of Computer Science Engineering

RPS College of Engineering and Technology

Mahendergarh , Haryana – 123029

BEFORE ENTERING IN THE LAB

- All the students are supposed to prepare the theory regarding the next experiment.
- Students are supposed to bring the practical file and the lab copy.
- Previous programs should be written in the practical file.
- All the students must follow the instructions, failing which he/she may not be allowed in the lab.

WHILE WORKING IN THE LAB

- Adhere to experimental schedule as instructed by the lab in-charge
- Get the previously executed program signed by the instructor
- Get the output of the current program checked by the instructor in the lab copy
- Each student should work on his/her assigned computer at each turn of the lab
- Take responsibility of valuable accessories
- Concentrate on the assigned practical and do not play games
- If anyone caught red handed carrying any equipment of the lab, then he/she will have to face serious consequences.

LIST OF PROGRAMS

S.No	Program
1	To study compiler and LEX tools
2	Write a program to check whether the string belongs to grammar or not
3	Write a program to find leading terminals
4	Write a program to find trailing terminals in a grammar
5	Write a Program to find the FIRST of non-terminals
6	Write a program to check whether a grammar is Left Recursive and remove left recursion
7	Write a program to remove Left Factoring
8	Write a program to show whether the grammar is Operator Grammar or not
9	Write a program to show all operations on a Stack
10	Write a program to show File operations i.e read , write , modify in a text file .

SOFTWARE REQUIREMENT

Platform used for Programming	**:**	UBUNTU
Editor used for writing programs	**:**	gedit
Compiler used for execution of Programs	**:**	g++

Steps to be followed

- Open command prompt (Terminal) on UBUNTU
- Use command **gedit** hello.cpp[hello.c is the name of program] to open the Editor
- Write the instructions as per your program
- Save the program in the specified folder
- Use **gcc hello.cpp - o hh** [to compile the program ……..][to create object file of name "hh"]
- Use Command **./hh** to run the program

Program 1 : To study compiler and LEX tools.

Compiler:-

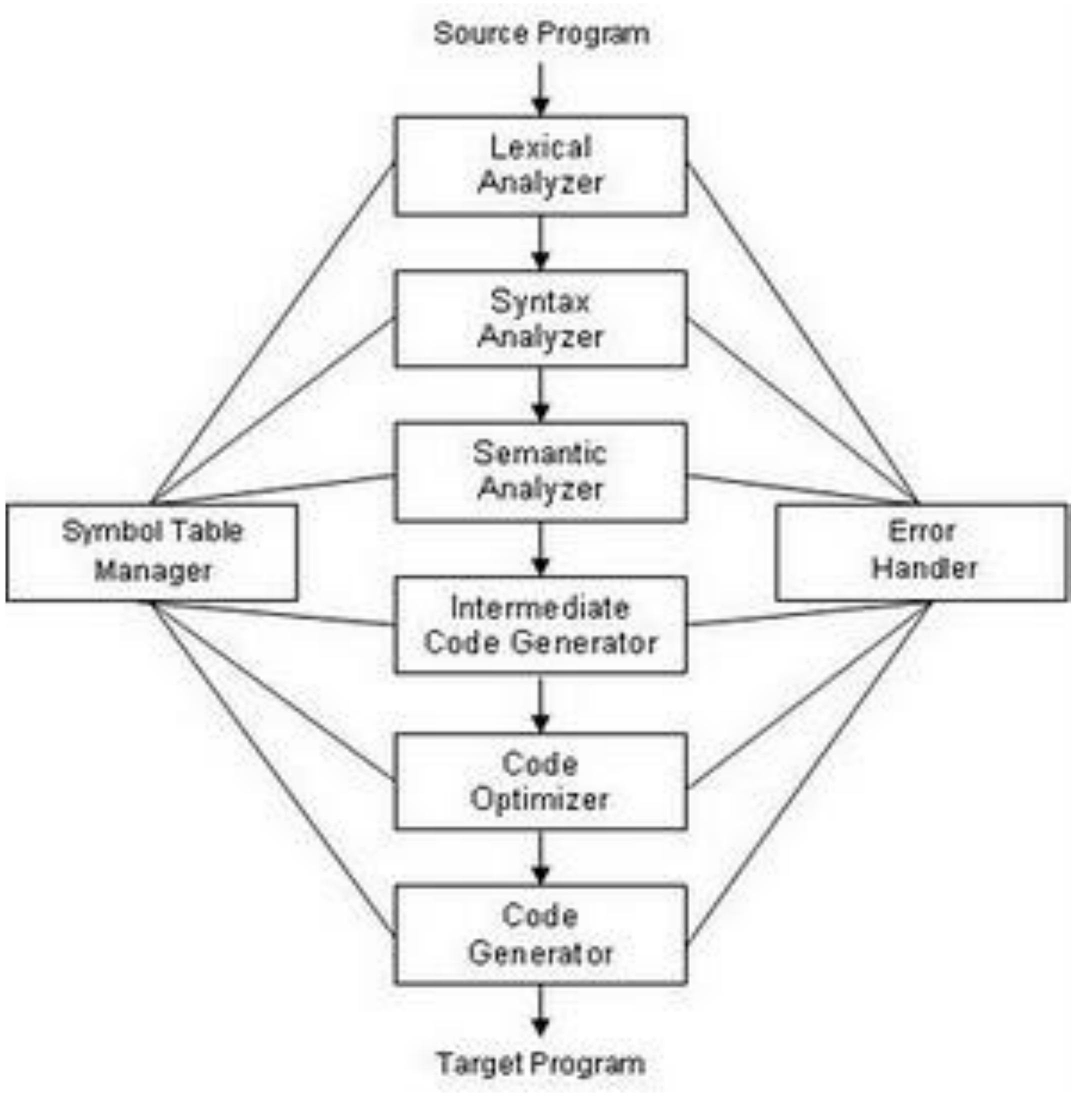
A compiler is a computer program that transforms source code written in a programming language into another computer language. The most common reason for wanting to transform source code is to create an executable program. The name "compiler" is primarily used for programs that translate source code from a high-level programming language to a lower level language. If the compiled program can only run on a computer whose CPU or operating system is different from the one on which the compiler runs the compiler is known as a cross-compiler. A program that translates from a low level language to a higher level one is a decompiler. A program that translates between high-level languages is usually called a language translator, source to source translator, or language converter. A compiler is likely to perform many or all of the following operations: lexical analysis, preprocessing, parsing, semantic analysis, code generation, and code optimization.

Structure of a compiler:-

The two major parts of a compiler are the Analysis phase and the Synthesis phase. The analysis part breaks up the source program into constant piece and creates an intermediate representation of the source program. The Lexical Analyzer, Syntax Analyzer and Semantic Analyzer form the bulk of this part. On the other hand, the synthesis part constructs the desired target program from the intermediate representation. The Intermediate Code Generator, Code Generator, and Code Optimizer are the parts of this synthesis portion. Each phase transforms the source program from one representation into another representation. They all communicate with error handlers and so with the symbol table.

Phases of a compiler:- **Lexical Analyser:-**

In the first phase of a compiler, we define lexical rules by regular expression. Lexical Analysis is also called Scanning. The Lexical Analyzer reads the source program character by character and returns the tokens of the source program, which are pattern of characters having same meaning in the source program. In this phase, the source program is checked to have valid characters and words. After detailed analysis, input stream of tokens are stored in a buffer.



<u>Syntax Analyser</u>:-

The second phase of a compiler is the Syntax Analysis. Generally, a Syntax Analyzer creates the syntactic structure (generally a parse tree) of the given program. In this phase, we define syntax rules by Context Free Grammar (CFG). Syntax Analysis is also called parser. During parsing, syntax rules are checked. Inputs of this phase are tokens and output is a parse tree or a syntax tree.

Semantic Analyser:-

The third phase is called Semantic Analysis. Basically the word semantic means the "meaning". A semantic analyzer is the one which checks the source program for semantic errors and collects the type information for the code generation. In this phase, we define semantic rules by attribute grammars. Semantic Analysis has two types: Declaration Checking and Type Checking. The input of this phase is a syntax tree and the output is an attribute grammar.

Intermediate Code Generator:-

On a logical level the output of the syntax analyser is some representation of a parse tree. The intermediate code generation phase transforms this parse tree into an intermediate language representation of the source program.

One popular type of intermediate language is what is called "three-address code". A typical three-address code statement is

A:=B op C

A,B,C are operands and op is binary operator.

Code optimizer:-

Object program that are frequently executed should be fast and small. The phase in which there is an attempt to produce an intermediate-language program can ultimately be produced. This phase is popularly called the optimization phase. A good optimizing compiler can improve the target program by perhaps a factor of two in overall speed. Optimization can be done in two ways:-

- **Local optimization:-** There are local optimization that can be applied to a program to attempt an improvement. Such as :-

  ```
  if A>B goto L2
   goto L3
  ```

  the sequence can be written as single statement:-

  ```
  if A>B goto L3
  ```

- **Loop optimization:-** Loops are especially good targets for optimization because programs spend most of their time in inner loops. A typical loop improvement is to move a computation that produces the same result each time the loop is entered. Then this computation is done only each time the loop is entered, rather than once for each iteration of the loop. Such a computation is called loop invariant.

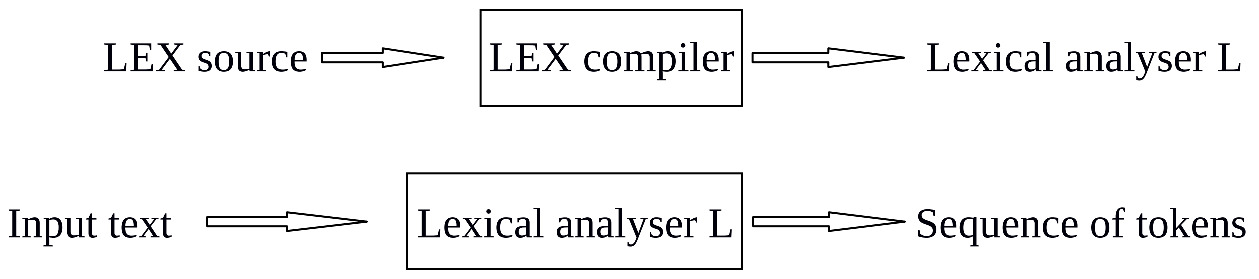
Code Generator:-

The last phase is the Target Code Generation. After analysis of source code is completed, then last step is converting it into a target language. The Code Generator produces the target language in a specific architecture in which the target program is normally a relocatable object file containing the machine codes. A simple minded code generator might map the statement A:=B+C into the machine code sequence.

```
LOAD B
ADD C
STORE A
```

LEX Tool:-

LEX is a program that generates lexical analyzers ("scanners" or "lexers"). Lex is commonly used with the yacc parser generator. Lex reads an input stream specifying the lexical analyzer and outputs source code implementing the lexer in the C programming language. The role of lex can be defined by a diagram as:-



- **Definition** section is the place to define macros and to import header files written in C. It is also possible to write any C code here, which will be copied verbatim into the generated source file.

- **Rules** section is the most important section; it associates patterns with C statements. Patterns are simply regular expressions. When the lexer sees some text in the input matching a given pattern, it executes the associated C code. This is the basis of how lex operates.

Program 2 : Write a program to check whether the string belongs to grammar or not

```
#include<iostream.h>
#include<conio.h>
#include<ctype.h>
#include<string.h>
void main()
{
int a=0,b=0,c;
char str[20], tok[11];
int d;
clrscr();
cout<<"input the expression";
cin>>str;
while(str[a]!='\0')
{
 if((str[a]=='(') ||(str[a]=='{'))
{
 tok[b]='4';
 b++;
 }
 if((str[a]==')')||(str[a]=='}'))
 {
  tok[b]='5';
  b++;
  }
  if(isdigit(str[a]))
  {
  while(isdigit(str[a]))
  {
   a++;
   }
   a--;
   tok[b]='6';
   b++;
   }
   if(str[a]=='+')
   {
    tok[b]='2';
    b++;
   }
   if(str[a]=='*')
   {
    tok[b]='3';
    b++;
   }
   a++;
}
tok[b]='\0';
cout<<tok;
b=0;
while(tok[b]!='\0')
{
if(((tok[b]=='6')&&(tok[b+1]=='2')&&(tok[b+2]=='6'))||((tok[b]=='6')&&(tok[
b+1]=='3')&&(tok[b+2]=='6'))||((tok[b]=='4')&&(tok[b+1]=='6')
&&(tok[b+2]=='5')))
  {
   tok[b]='6';
   c=b+1;
   while(tok[c]!='\0')
    {
    tok[c]=tok[c+2];
    c++;
    }
   tok[c]='\0';
   cout<<endl<<tok;
   b=0;
  }
   else
  {
    b++;
    cout<<endl<<tok;
  }
}
d=strcmp(tok,"6");
if(d==0)
{
cout<<"it is in grammar";
}
else
{
cout<<" it is not in grammar";
}
getch();
}
```

Program 3 : Write a program to find leading terminals

```
#include<iostream.h>
#include<conio.h>
void main()
{
char t[5],nt[10],p[5][5],first[5][5],temp,e=238;
int i,j,not,nont,k=0,f=0;
clrscr();
cout<<"\nEnter the no. of Non-terminals in the grammer:";
cin>>nont;
cout<<"\nEnter the Non-terminals in the grammer:\n";
for(i=0;i<nont;i++)
{
   cin>>nt[i];
}
cout<<"\nEnter the no. of Terminals in the grammer: ( Enter e  for  "<< e<<" )
";
cin>>not;
cout<<"\nEnter the Terminals in the grammer:\n";
for(i=0;i<not;i++)
{
   cin>>t[i];
}
for(i=0;i<nont;i++)
{
  p[i][0]=nt[i];
  first[i][0]=nt[i];
}
for(i=0;i<nont;i++)
{
  cout<<"\nEnter the production for : "<<p[i][0];
  cout<<"( End the production with '$' sign ):";
  for(j=0;p[i][j]!='$';)
  {
   j+=1;
   cin>>p[i][j];
  }
}
for(i=0;i<nont;i++)
{
  cout<<"\nThe production for  "<< p[i][0]<<" -> ";
  for(j=1;p[i][j]!='$';j++)
  {
    cout<<p[i][j];
  }
}
for(i=0;i<nont;i++)
{
  f=0;
  for(j=1;p[i][j]!='$';j++)
  {
    for(k=0;k<not;k++)
    {
       if(f==1)
       break;
     if(p[i][j]==t[k])
     {
      first[i][j]=t[k];
      first[i][j+1]='$';
      f=1;
      break;
     }
     else if(p[i][j]==nt[k])
     {
      first[i][j]=first[k][j];
      if(first[i][j]=='e')
      continue;
      first[i][j+1]='$';
      f=1;
      break;
     }
    }
  }
}
for(i=0;i<nont;i++)
{
  cout<<"\nThe first of  "<<first[i][0]<<" is : ";
  for(j=1;first[i][j]!='$';j++)
  {
   cout<<first[i][j]<<",";
  }
}
getch();
}
```

Program 4 : Write a program to find trailing terminals in a grammar

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
#include<process.h>
void display();
void trail(char,char[]);
void disp_trail();
struct productions
{
  int np;
  char str[10][10];
  int ln[10];
}p[20];
struct trailing
{
  char str[20];
}t[20];
char non_term[20];
int nt;
void main()
{
  clrscr();
  char i,j,k;
  cout<<"Enter no. of non terminals:";
  cin>>nt;
  for(i=0;i<nt;i++)
  {
   cout<<"Enter the symbol of a non terminal no. "<<i+1<<"(in capitals
only):";
```

```
   cin>>non_term[i];
  }
  for(i=0;i<nt;i++)
  {
   cout<<"For non terminal "<<non_term[i]<<":";
   cout<<"\n\t\tEnter no. of productions: ";
   cin>>p[i].np;
   for(j=0;j<p[i].np;j++)
   {
     cout<<"\t\tEnter the production "<<j+1<<": ";
     cin>>p[i].str[j];
     p[i].ln[j]=strlen(p[i].str[j]);
   }
  }
  cout<<"\nProductions are:";
  display();
  for(i=nt-1;i>=0;i--)
  {
    trail(non_term[i],t[i].str);
  }
disp_trail();
getch();
exit(0);
}
void trail(char c1, char c2[20])
{
    char c3[10];
    int i,j,k;
    for(i=0;i<nt;i++)
         if(non_term[i]==c1)
         break;
    for(j=p[i].np-1;j>=0;j--)
    {
      c3[0]='\0';
      for(k=strlen(p[i].str[j])-1;k>=0;k--)
      {
         if(p[i].str[j][k]<'A' || p[i].str[j][k]>'Z')
         {
          c3[0]=p[i].str[j][k];
          c3[1]='\0';
          strcat(c2,c3);
          break;
         }
       else
         if(strlen(p[i].str[j])==1 && p[i].str[j][k]>='A' && p[i].str[j][k]<='Z' &&
p[i].str[j][k+1]=='\0')
         trail(p[i].str[j][0],c2);
      }
    }
}
void display()
{   int i,j;
    for(i=0;i<nt;i++)
    {
     cout<<"\n\t"<<non_term[i]<<"--->";
     for(j=0;j<p[i].np;j++)
      cout<<p[i].str[j];
    }
}
void disp_trail()
{   int i;
    for(i=0;i<nt;i++)
    cout<<"\n\t"<<"trail("<<non_term[i]<<")"<<"--->"<<t[i].str;
}
```

Program 5 : Write a Program to find the FIRST of non-terminals

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
void main()
{
char AR1[3][5],first,A[1][5],B[1][5],C[1][5],ch,ch1;
int pro,int i,n,j;
clrscr();
for(i=0;i<3;i++)
{   for(j=0;j<5;j++)
    {  if(i==0)
       {  cin>>AR1[i][j];
          A[0][j]=AR1[i][j];
        }
        if(i==1)
        {   cin>>AR1[i][j];
            B[0][j]=AR1[i][j];
         }
        if(i==2)
        {   cin>>AR1[i][j];
            C[0][j]=AR1[i][j];
         }
      }
   }
cout<<"\n";
for(i=0;i<3;i++)
{     for(j=0;j<5;j++)
       {     if(i==0)
             {   cout<<A[0][j];
              }
              if(i==1)
             {    cout<<B[0][j];
             }
             if(i==2)
```

```
           {   cout<<C[0][j];
            }
         }
      cout<<"\n\n";
  }
cout<<"Enter :";
cin>>pro;
if(pro==1)
{      if((A[0][3]>=97)&&(A[0][3]<=122))
       {    cout<<A[0][3];
        }
        else if(A[0][3]==B[0][0])
             {     if((B[0][3]>=97)&&(B[0][3]<=122))
                   {
                    cout<<B[0][3];
                    }
                   else if(B[0][3]==C[0][0])
                         {
                           if((C[0][3]>=97)&&(C[0][3]<=122))
                            {       cout<<C[0][3];
                            }
                         }
               }
                else if(A[0][3]==C[0][0])
                {          if((C[0][3]>=97)&&(C[0][3]<=122))
                           {
                            cout<<C[0][3];
                           }
                            else if(C[0][3]==B[0][0])
                             {
                                 if((B[0][3]>=97)&&(B[0][3]<=122))
                                {
                                 cout<<B[0][3];
                                 }
                              }
                  }
         }
        if(pro==2)
        {
                if((B[0][3]>=97)&&(B[0][3]<=122))
                {
                        cout<<B[0][3];
                }
                else if(B[0][3]==A[0][0])
                {
                        if((A[0][3]>=97)&&(A[0][3]<=122))
                         {
                                cout<<A[0][3];
                         }
                          else if(A[0][3]==C[0][0])
                         {
                               if((C[0][3]>=97)&&(C[0][3]<=122))
                                {
                                cout<<C[0][3];
                                 }
                         }
        }
        else if(B[0][3]==C[0][0])
        {
                if((C[0][3]>=97)&&(C[0][3]<=122))
                {
                        cout<<C[0][3];
                 }
                else if(C[0][3]==A[0][0])
                          {
                                    if((A[0][3]>=97)&&(A[0][3]<=122))
                                    {
                                       cout<<A[0][3];
                                     }
                           }
        }
}
if(pro==3)
{
        if((C[0][3]>=97)&&(C[0][3]<=122))
        {
                cout<<C[0][3];
        }
         else if(C[0][3]==A[0][0])
         {       if((A[0][3]>=97)&&(A[0][3]<=122))
                 {
                        cout<<A[0][3];
                  }
                 else if(A[0][3]==B[0][0]){ if((B[0][3]>=97)&&(B[0][3]<=122))
                 {
                                cout<<B[0][3];
                  }
          }
}
else if(C[0][3]==B[0][0])
{       if((B[0][3]>=97)&&(B[0][3]<=122))
        {
          cout<<B[0][3];
         }
       else    if(B[0][3]==A[0][0]){ if((A[0][3]>=97)&&(A[0][3]<=122))
                {
                        cout<<A[0][3];
                }
}
}
}
getch();
}
```

Program 6 : Write a program to check whether a grammar is Left Recursive and remove left recursion

```
#include<stdio.h>
#include<iostream.h>
#include<conio.h>
void main()
{
char str[10],alpha[10],beta[10], x=238;
int i=0, j=0;
clrscr();
cout<<"Enter the grammar";
cin>>str;
if(str[0]==str[3])
{
for(i=4;str[i]!='/';i++)
{
alpha[j]=str[i];
j++;
}
alpha[j]='\0';
j=0;
cout<<"Left Recursion removed "<<endl;
for(++i;str[i]!='\0';i++)
{
beta[j]=str[i];
j++;
}
beta[j]='\0';
for(i=0;i<=2;i++)
cout<<str[i];
for(j=0;beta[j]!='\0';j++)
{
cout<<beta[j];
}
cout<<"S'"<<endl;
cout<<"S'->";
for(j=0;alpha[j]!='\0';j++)
{
cout<<alpha[j];
}
cout<<"S'/"<<x;
}
else
cout<<"Not recursive";
getch();
}
```

Program 7 : Write a program to remove Left Factoring

```
#include<iostream.h>
#include<conio.h>
void main()
{
char a[16],b[4],c[6],d[6],x=238;
int i,j=0,aa,bb,l=0,k[2];
clrscr();
cout<<"Enter the production\n\n";
cin>>a;
for(i=0;i<16;i++)
{
  if(a[i]=='/')
  {
    k[j]=i;
    j++;
  }
}
aa=k[0];
bb=k[1];
for(i=0;i<aa-3;i++)
{
 b[i]=a[i+3];
}
for(i=aa+1,j=0;i<bb;j++,i++)
{
  c[j]=a[i];
}
for(i=0;i<6;i++)
{
if(c[i]!=b[i])
{
  d[l]=c[i];
  l++;
}
}
cout<<"\n\nThe result is: \n";
cout<<"S->";
for(i=0;i<4;i++)
{
cout<<b[i];
}
cout<<"Z"<<endl;
cout<<"Z->";
for(i=0;i<l;i++)
{
cout<<d[i];
}
cout<<"/"<<x;
cout<<"\nS->";
cout<<a[15];
getch();
}
```

Program 8 : Write a program to show whether the grammar is Operator Grammar or not

```
#include<iostream.h>
#include<conio.h>
#include<ctype.h>
#include<process.h>
void main()
{   char a[15],c,d;
  int i;
  clrscr();
  cout<<"Enter a Grammar ";
  cin>>a;
  for(i=3;a[i]!='\0';i++)
  {    c=a[i];
   d=a[i+1];
   if(isupper(c) && isupper(d))
   {
     cout<<"Not a Operator Grammar ";
     getch();
     exit(0);
   }  }
if(a[3]=='e')
{
 cout<<" Not a Operator Grammar ";
}
Else {  cout<<" Operator Grammar "; }
getch();
}
```

Program 9 : Write a program to show all operations on a Stack

```
#include<iostream.h>
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void push(struct stack *p, int x);
int pop(struct stack *p);
void traverse(struct stack *p);
struct stack
{
int a[100];
int top;
};
void main()
{
 struct stack s;
 int choice,m;
 char ch;
 clrscr();
 s.top=-1;
do
{
  cout<<"\n 1 push";
  cout<<"\n 2 pop";
  cout<<"\n 3 traverse";
  cout<<"\n enter the choice";
  cin>>choice;
  switch(choice)
  {
case 1: cout<<"Number you want to insert";
```

```
             cin>>m;
             push(&s,m);
             break;
    case 2: cout<<"Status of Stack : ";
             cout<<pop(&s);
             break;
    case 3: traverse(&s);
             break;
    default:cout<<"wrong choice";
   }
 cout<<"\n wish to continue(y/n)";
 fflush(stdin);
 cin>>ch;
}
while(ch=='Y'|| ch=='y');
getch();
}
void push(struct stack *p,int item)
{
if(p->top==99)
{
 cout<<"stack full\n";
 getch();
 exit(0);
}
else
{
p->a[++(p->top)]=item;
}
}
int pop(struct stack *p)
{   int item;
```

```
    if(p->top==-1)
    {
    cout<<"stack empty\n";
    getch();
    exit(0);
    }
    else
    {
    item=p->a[p->top--];
    }
   return(item);
}
void traverse(struct stack *p)
{int i;
if(p->top==-1)
{
cout<<"stack empty\n";
getch();
exit(0);
}
else
{
 cout<<" Numbers in the stack are : ";
 for(i=p->top;i>=0;i--)
  {
  cout<<"\t"<<p->a[i];
  } } }
```

Program 10 : Write a program to show File operations i.e read , write , modify in a text file .

```
#include <fstream.h>
#include <iostream.h>
#include<conio.h>
void main ()
{
  char data[100],opt;
  int i;
  clrscr();
  ofstream outfile;
  outfile.open("afile.dat",ios::app);
  cout << "Writing to the file" << endl;
  for(i=0;i<2;i++)
  {
  cout << "Enter your name: ";
  cin>>data;
  outfile << data << endl;
  cout << "Enter your age: ";
  cin >> data;
  outfile << data << endl;
  }
  outfile.close();
  ifstream infile;
  infile.open("afile.dat",ios::in);
  cout << "Reading from the file" << endl;
  while(infile)
  {
  infile >> data;
  cout << data << endl;
```

```
 }
 infile.close();
 getch();
}
```