PRACTICAL FILE

Department:	**Computer Science and Engineering**
Session:	**July - December**
Subject:	**Data Structure Lab**
Subject Code:	**BTCS306**
Semester:	**3rd**

Syllabus

1. Write a menu driven program that implements following operations (using separate functions) on a linear array:
   a) Insert a new element at end as well as at a given position
   b) Delete an element from a given whose value is given or whose position is given
   c) To find the location of a given element
   d) To display the elements of the linear array
2. Write a menu driven program that maintains a linear linked list whose elements are stored in on ascending order and implements the following operations (using separate functions):
   a) Insert a new element
   b) Delete an existing element
   c) Search an element
   d) Display all the elements
3. Write a program to demonstrate the use of stack (implemented using linear array) in converting arithmetic expression from infix notation to postfix notation.
4. Program to demonstrate the use of stack (implemented using linear linked lists) in evaluating arithmetic expression in postfix notation.
5. Program to demonstration the implementation of various operations on a linear queue represented using a linear array.
6. Program to demonstration the implementation of various operations on a circular queue represented using a linear array.
7. Program to demonstration the implementation of various operations on a queue represented using a linear linked list (linked queue).
8. Program to illustrate the implementation of different operations on a binary search tree.
9. Program to illustrate the traversal of graph using breadth-first search.
10. Program to illustrate the traversal of graph using depth-first search.
11. Program to sort an array of integers in ascending order using bubble sort.
12. Program to sort an array of integers in ascending order using selection sort.
13. Program to sort an array of integers in ascending order using insertion sort.
14. Program to sort an array of integers in ascending order using radix sort.
15. Program to sort an array of integers in ascending order using merge sort.
16. Program to sort an array of integers in ascending order using quick sort.
17. Program to sort an array of integers in ascending order using heap sort.
18. Program to sort an array of integers in ascending order using shell sort.
19. Program to demonstrate the use of linear search to search a given element in an array.
20. Program to demonstrate the use of binary search to search a given element in a sorted array in ascending order.

List of Practical

Sr. No.	Topic
1.	Write a menu driven program that implements following operations (using separate functions) on a linear array: a) Insert a new element at end as well as at a given position b) Delete an element from a given whose value is given or whose position is given c) To find the location of a given element d) To display the elements of the linear array
2.	Write a menu driven program that maintains a linear linked list whose elements are stored in on ascending order and implements the following operations (using separate functions): a) Insert a new element b) Delete an existing element c) Search an element d) Display all the elements
3.	Write a program to demonstrate the use of stack (implemented using linear array) in converting arithmetic expression from infix notation to postfix notation.
4.	Program to demonstrate the use of stack (implemented using linear linked lists) in evaluating arithmetic expression in postfix notation.
5.	Program to demonstration the implementation of various operations on a linear queue represented using a linear array.

6.	Program to demonstration the implementation of various operations on a circular queue represented using a linear array.
7.	Program to demonstration the implementation of various operations on a queue represented using a linear linked list (linked queue).
8.	Program to illustrate the implementation of different operations on a binary search tree.
9.	Program to illustrate the traversal of graph using breadth-first search.
10.	Program to illustrate the traversal of graph using depth-first search.
11	Program to sort an array of integers in ascending order using bubble sort.
12	Program to sort an array of integers in ascending order using selection sort.
13	Program to sort an array of integers in ascending order using insertion sort
14	Program to sort an array of integers in ascending order using radix sort.
15	Program to sort an array of integers in ascending order using merge sort.
16	Program to sort an array of integers in ascending order using quick sort.
17	Program to sort an array of integers in ascending order using heap sort.
18	Program to sort an array of integers in ascending order using shell sort.

19	Program to demonstrate the use of linear search to search a given element in an array.
20	Program to demonstrate the use of binary search to search a given element in a sorted array in ascending order.
21.	*Program to demonstrate the implement of Circular Linked List
22.	*Program to demonstrate the implement of Doubly Linked List

*Learning Beyond Syllabus Program to demonstrate the implement of Circular Linked List

*Learning Beyond Syllabus Program to demonstrate the implement of Doubly Linked List

Experiment 1

AIM : Write a menu driven program that implements following operations (using separate functions) on a linear array:

a) Insert a new element at end as well as at a given position
b) Delete an element from a given whose value is given or whose position is given
c) To find the location of a given element
d) To display the elements of the linear array

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<iostream.h>
void a_insert(int,int);
void a_delete(int);
void a_location(double);
void a_traverse();
int arr[10], n;
void main()
{
int k;
cout<<"enter length";
cin>>n;
cout<<"array is:";
for(k=0;k<n;k++)
{cin>>arr[k];
}
          int ch;
           int num,pos;
           double element;
          while(ch!=5)
   {
    cout<<"1> Insert";
    cout<<"\n2> Delete";
          cout<<"\n3> location";
    cout<<"\n4> Show";
    cout<<"\n5> Quit\n";
    cin>>ch;
          switch(ch)
                    {
                    case 1:
                              cout<<"enter element:";
                               cin>>num;
```

```
                        cout<<"enter pos.:";
                        cin>>pos;
                        a_insert(num,pos);
                        break;
                case 2:
                        cout<<"enter pos.:";
                        cin>>pos;
                        a_delete(pos);
                        break;
                case 3:
                        cout<<"enter element";
                        a_location(element);
                        break;

                case 4:

                        cout<<"\nArray:";
                        a_traverse();
                        break;
                case 5:
                        break;

                default:
                        cout << "Invalid input" << endl;
                }
                }


getch();

 }

void a_insert(int num, int pos)
 {
  for(int i=5; i>=pos;i--)
    arr[i]=arr[i-1];
  arr[i]=num;
  n++;
 }
 void a_delete(int pos)
 {
  for(int i=pos; i<=n;i++)
  arr[i-1]=arr[i];
  arr[i-1]=0;
 }
 void a_traverse()
 {  int i;
            cout<<"array is:"<<endl;
            for(i=0;i<n;i++)
            cout<<arr[i]<<endl;
 }
 void a_location(double element)
 {
            cin>>element;
 int Index;

for (int i=0;i<5; i++)
{
     if (element == arr[i])
     {
         Index = i;
     }
            cout<<Index;
}}
```

Experiment 2

AIM: Write a menu driven program that maintains a linear linked list whose elements are stored in on ascending order and implements the following operations (using separate functions):

a) Insert a new element
b) Delete an existing element
c) Search an element
d) Display all the elements

PROGRAM:

```
#include <iostream.h>
#include <stdlib.h> //for exit(1)
#include <conio.h>
#define MAX 10
struct node{
       int data;
       struct node *next;
};

class sngllist{
    node *A;
    int cnt;
    public:
    sngllist(){
       cnt=0;
       A=NULL;
    }
        void create(int); //initial data assignment
    void display(int); //process is display (assumed)
    void insert();
    void del();
    void search();
};


void sngllist :: create(int check){
     node *start=NULL,*newl=NULL,*end=NULL;
     int takedata;
    clrscr();
    cout<<"\n\t\t*****Create List*****\n";
    while(1){
cout<<"\t\t-999 to Quit\n";
cout<<"\t\tEnter data : ";
cin>>takedata;
if(takedata == -999)
   break;
else{
      //create memory for new node
      newl = new node;
      if(newl == NULL){
     cout<<"\n\t\tNot Enough Memory";
     getch();
     exit(1);
      }
      newl->data = takedata;
      newl->next = NULL;
      cnt++;

      //check for first node
      if(start == NULL)
      start = newl;
      else
      end->next = newl;
      end = newl;
      end->next = NULL;
}//end else
    }//end while


    //check to create which list
    if(check==1){
A->next = start;
A = start;
    }
    if(check==2){
B->next = start;
B = start;
    }
    if(check==3){
MRG->next = start;
MRG = start;
    }
```

```
}

void sngllist :: display(int check){
    node *tmp;

    //check to print which list
    if(check==1)
  tmp=A;
    if(check==2)
  tmp=B;
    if(check==3)
  tmp=MRG;

    cout<<"\n\t\t*****Display*****\n";
    cout<<"\t\t";
    for( ; tmp!=NULL; tmp=tmp->next){
  cout<<tmp->data;
  if(tmp->next != NULL)
      cout<<"-->";
    }//end for
    getch();
}


void sngllist :: insert(){
    node *newl=NULL,*tmp=NULL;
    int choice,takedata,pos,i;
    while(1){
 clrscr();
 cout<<"\n\t\t*****Insertion*****\n";
 cout<<"\t\t1) Begining\n";
 cout<<"\t\t2) In Between\n";
 cout<<"\t\t3) End\n";
 cout<<"\t\t4) Return to Main Menu\n";
 cout<<"\t\tEnter your choice : ";
 cin>>choice;
 if(choice==1 || choice==2 || choice==3){
 //create memory for new node
   newl = new node;
   if(newl == NULL){
      cout<<"\n\t\tNot Enough Memory";
      getch();
      exit(1);
   }
```

```
          cout<<"\n\t\tEnter data : ";
          cin>>takedata;
          newl->data = takedata;
          newl->next = NULL;
        }
        else
           return;

        switch(choice){
             case 1 :   newl->next = A;
                  A = newl;
                  break;

             case 2 :   cout<<"\n\t\tEnter Position : ";
                  cin>>pos;
                  if(pos <=1 || pos >= count(2)){
                       cout<<"\n\t\tInvalid Position";
                       getch();
                       break;
                  }
                  else{
                     //to points to previous node from where
                     //to insert
                     for(i=1,tmp=A; i < (pos-1); tmp=tmp->next,i++);

                     newl->next = tmp->next;
                     tmp->next  = newl;
                     break;
                  }
             case 3 :   //For pointing to last node
                  for(tmp=A; tmp->next != NULL; tmp=tmp->next);

                  tmp->next = newl;
        }//end switch
            }//end while
      }



      void sngllist :: del(){
            node *delnode=NULL,*tmp=NULL;
            int choice,deldata,pos,i;
            while(1){
        clrscr();
```

```
 cout<<"\n\t\t*****Deletion*****\n";
 cout<<"\t\t1) Begining\n";
 cout<<"\t\t2) In Between\n";
 cout<<"\t\t3) End\n";
 cout<<"\t\t4) Return to Main Menu\n";
 cout<<"\t\tEnter your choice : ";
 cin>>choice;
 switch(choice){
     case 1 :   delnode->next = A;
         delnode = A;
         A = A->next;
         break;

     case 2 :   cout<<"\n\t\tEnter Position : ";
         cin>>pos;
         if(pos <=1 || pos >= count(2)){
             cout<<"\n\t\tInvalid Position";
             getch();
             break;
         }
         else{
            //to points to previous node from where
            //to insert
            for(i=1,tmp=A; i < (pos-1); tmp=tmp->next,i++);

            delnode = tmp->next;
            tmp->next = tmp->next->next;
            break;
         }
     case 3 :   //For pointing to last node
         for(tmp=A; tmp->next->next != NULL; tmp=tmp->next);
         delnode = tmp->next;
         tmp->next = NULL;
         break;
     case 4  :  return;
     default :   cout<<"\n\t\tInvalid Position";
          getch();
 }//end switch
 delete(delnode);
    }//end while
}


void sngllist :: search(){
    node *tmp;
    int item,n;
    cout<<"\n\t\t*****Search*****\n";
    cout<<"\t\tEnter data to be searched : ";
    cin>>item;
    for(tmp=A,n=1; tmp!=NULL; tmp=tmp->next,n++){
  if(tmp->data == item){
     cout<<"\n\t\tSearch is located at "<<n<<" location";
     getch();
     return;
  }
    }
    cout<<"\n\t\tSearch Not Found";
    getch();
}

void main()
{
   int choice;
   sngllist obj;
   while(1){
   clrscr();
   cout<<"\n\t\tSINGLY LINK-LIST OPERATIONS\n\n";
   cout<<"\t\t1)  Create List\n";
   cout<<"\t\t2)  Display List\n";
   cout<<"\t\t3)  List Insertion\n";
   cout<<"\t\t4)  List Deletion\n";
   cout<<"\t\t5)  Search List\n";
   cout<<"\t\t6) Exit\n";
   cout<<"\t\tEnter your Choice :  ";
   cin>>choice;
   switch(choice){
  case 1 :  obj.create(1); // 1 for A list
        break;
  case 2 :  obj.display(1);// 1 for A list
        break;
  case 3 :  obj.insert();
        break;
  case 4 :  obj.del();
        break;
  case 5 :  obj.search();
        break;

  case 6 :  goto out;
```

```
   default:  cout<<"\n\n\t\tInvalid Choice\n\n";
          getch();
          break;
    }
  }
  out:
 }
```

Experiment 3

AIM: Write a program to demonstrate the use of stack (implemented using linear array) in converting arithmetic expression from infix notation to postfix notation.

PROGRAM

```
# include <iostream.h>
 # include   <string.h>
 # include   <stdlib.h>
 # include    <conio.h>

 struct node
 {
    char data;

    node *next;
 };

 node *top=NULL;
 node *bottom=NULL;
 node *entry;
 node *last_entry;
 node *second_last_entry;

 void push(constchar);
 constchar pop( );

 void infix_to_postfix(constchar *);

 int main( )
    {
       clrscr( );

       char Infix_expression[100]={NULL};

       cout<<"\n\n Enter the Infix Expression : ";
       cin>>Infix_expression;

       infix_to_postfix(Infix_expression);

       getch( );
       return 0;
    }
 /*************************************************************************///-----------------------------
push(const char)  ---------------------
///*************************************************************************/void push(constchar
Symbol)
   {
      entry=new node;

      if(bottom==NULL)
     {
        entry->data=Symbol;
        entry->next=NULL;
        bottom=entry;
        top=entry;
     }

      else
     {
        entry->data=Symbol;
        entry->next=NULL;
        top->next=entry;
        top=entry;
     }
   }

 /*************************************************************************///------------------------------
-  pop( )  ----------------------------
///*************************************************************************/constchar pop( )
   {
      char Symbol=NULL;

      if(bottom==NULL)
     cout<<"\n\n\n\t ***  Error : Stack is empty. \n"<<endl;

      else
     {
        for(last_entry=bottom;last_entry->next!=NULL;
                    last_entry=last_entry->next)
       second_last_entry=last_entry;

        if(top==bottom)
       bottom=NULL;

        Symbol=top->data;

        delete top;

        top=second_last_entry;
        top->next=NULL;
     }

      return Symbol;
```

```
    }

   /*********************************************************************///---------------------
  infix_to_postfix(const char *)  ----------------
  ///*********************************************************************/void
  infix_to_postfix(constchar *Infix)
    {
      char Infix_expression[100]={NULL};
      char Postfix_expression[100]={NULL};

      strcpy(Infix_expression,"(");
      strcat(Infix_expression,Infix);
      strcat(Infix_expression,")");

      char Symbol[5]={NULL};
      char Temp[5]={NULL};

      for(int count=0;count<strlen(Infix_expression);count++)
     {
       Symbol[0]=Infix_expression[count];

       if(Symbol[0]=='(')
      push(Symbol[0]);

       elseif(Symbol[0]==')')
      {
        Symbol[0]=pop( );

        while(Symbol[0]!='(')
          {
         strcat(Postfix_expression,Symbol);

         Symbol[0]=pop( );
          }
      }

       elseif(Symbol[0]=='^' || Symbol[0]=='*' || Symbol[0]=='/'
                || Symbol[0]=='+' || Symbol[0]=='-')
      {
        if(Symbol[0]=='*' || Symbol[0]=='/')
          {
         Temp[0]=pop( );

         while(Temp[0]=='^' || Temp[0]=='*' || Temp[0]=='/')
           {
             strcat(Postfix_expression,Temp);

             Temp[0]=pop( );
           }

         push(Temp[0]);
```

```
          }

        elseif(Symbol[0]=='+' || Symbol[0]=='-')
          {
         Temp[0]=pop( );

         while(Temp[0]!='(')
           {
              strcat(Postfix_expression,Temp);

              Temp[0]=pop( );
           }

         push(Temp[0]);
          }

        push(Symbol[0]);
      }

       else
      strcat(Postfix_expression,Symbol);
    }

     cout<<"\n\n Postfix Expression : "<<Postfix_expression;
  }
```

Experiment 4

AIM: Program to demonstrate the use of stack (implemented using linear linked lists) in evaluating arithmetic expression in postfix notation.

PROGRAM

```
#include <iostream.h>
 #include   <string.h>
 #include   <stdlib.h>
 #include    <conio.h>
 #include     <math.h>

 const max_length=10;

 class Evaluator
    {
       private:
      char infix_expression[25][6];
      char postfix_expression[25][6];

      char char_stack[max_length];
      long int_stack[max_length];

      int int_top;
      int char_top;

      int postfix_rows;
      int infix_rows;

      int input_characters_count;

       public:
      char char_pop( );

      long int_pop( );

      void set_initial_values( );
      void get_infix_expression( );
      void char_push(char);
      void show_infix_to_postfix_convertion( );
      void int_push(long);
      void show_evaluation_of_postfix_expression( );
    };

 /*********************************************************************///----------------------
set_initial_values( )  ------------------------
```

```
///*********************************************************************/void
Evaluator::set_initial_values( )
   {
      int_top=-1;
      char_top=-1;
      infix_rows=0;
      postfix_rows=0;

      for(int count_1=0;count_1<max_length;count_1++)
     {
        int_stack[count_1]=0;
        char_stack[count_1]=NULL;
     }

      for(int count_2=0;count_2<25;count_2++)
     {
        strset(infix_expression[count_2],NULL);
        strset(postfix_expression[count_2],NULL);
     }
   }

 /********************************************************************///-----------------------
get_infix_expression( )  ---------------------
///*********************************************************************/void
Evaluator::get_infix_expression( )
   {
      Input:

      clrscr( );
      cout<<"\n\n\t **********   Arithmetic Expression Evaluator   ********* "<<endl;

      set_initial_values( );

      int flag=0;
      int count_1=-1;
      int number_length=0;
      int operand_count=0;
      int operator_count=0;
      int expression_right_wrong=1;
      int input_characters_count=0;
      int left_parenthesis_count=0;
      int right_parenthesis_count=0;

      char temp_expression[5]={NULL};

      cout<<"\n\n\n\t Enter the Infix Expression : ";

      do
     {
        temp_expression[0]=getch( );
 if(int(temp_expression[0])!=8)
cout<<temp_expression;

 if((int(temp_expression[0])>=48 &&
        int(temp_expression[0])<=57) && number_length<5)
{
   if(flag==0)
      {
     count_1++;
     flag=1;
     operand_count++;

     strcpy(infix_expression[count_1],temp_expression);
      }

   elseif(flag==1)
      strcat(infix_expression[count_1],temp_expression);

   number_length++;
}

 elseif( temp_expression[0]=='^' || temp_expression[0]=='/'
   || temp_expression[0]=='*' || temp_expression[0]=='-'
   || temp_expression[0]=='+' || temp_expression[0]=='('
                  || temp_expression[0]==')')
{
   flag=0;
   count_1++;
   number_length=0;

   strcpy(infix_expression[count_1],temp_expression);

   if(temp_expression[0]=='(')
      left_parenthesis_count++;

   elseif(temp_expression[0]==')')
      right_parenthesis_count++;

   else
      operator_count++;
}

 if(int(temp_expression[0])==8 && count_1>=0)
{
   int length=strlen(infix_expression[count_1]);

   for(int count_2=0;count_2<length;count_2++)
      cout<<char(8)<<" "<<char(8);

   if(int(infix_expression[count_1][0])>=48 &&
              int(infix_expression[count_1][0])<=57)
```

```
         operand_count--;

      elseif(infix_expression[count_1][0]=='^' ||
              infix_expression[count_1][0]=='/' ||
             infix_expression[count_1][0]=='*' ||
               infix_expression[count_1][0]=='-' ||
              infix_expression[count_1][0]=='+' )
         operator_count--;

      elseif(infix_expression[count_1][0]=='(')
         left_parenthesis_count--;

      elseif(infix_expression[count_1][0]==')')
         right_parenthesis_count--;

      strset(infix_expression[count_1],NULL);

      flag=0;
      count_1--;
      input_characters_count--;
   }

    if(int(temp_expression[0])==13)
   break;

    elseif(int(temp_expression[0])==27)
   {
      infix_expression[25][6]='$';

      break;
   }

    elseif(operand_count<operator_count)
   {
      infix_expression[25][6]='$';

      break;
   }

    elseif(!left_parenthesis_count && right_parenthesis_count)
   {
      count_1++;

      break;
   }

    elseif(temp_expression[0]!='^' && temp_expression[0]!='/' &&
         temp_expression[0]!='*' && temp_expression[0]!='-' &&
         temp_expression[0]!='(' && temp_expression[0]!=')' &&
         temp_expression[0]!='+' && temp_expression[0]!='0' &&
         temp_expression[0]!='1' && temp_expression[0]!='2' &&
              temp_expression[0]!='3' && temp_expression[0]!='4' &&
              temp_expression[0]!='5' && temp_expression[0]!='6' &&
              temp_expression[0]!='7' && temp_expression[0]!='8' &&
              temp_expression[0]!='9' && int(temp_expression[0])!=8
                   && int(temp_expression[0])!=27 )
           {
             infix_expression[25][6]='$';

             break;
           }

            input_characters_count++;
          }
           while(count_1<=22 && input_characters_count<=24);

           if(operator_count!=(operand_count-1))
          expression_right_wrong=0;

           elseif(left_parenthesis_count!=right_parenthesis_count)
          expression_right_wrong=0;

           elseif(count_1<2)
          expression_right_wrong=0;

           elseif(infix_expression[25][6]=='$')
          expression_right_wrong=0;

           if(expression_right_wrong==0)
          {
            cout<<"\n\n\t Input Expression is not correct."<<endl;

            goto Input;
          }

           infix_rows=(count_1+1);
        }

  /**********************************************************************///--------------------------
 int_push( )  ----------------------------
 ///**********************************************************************/void
 Evaluator::int_push(long item)
    {
      int_top++;
      int_stack[int_top]=item;
    }

  /**********************************************************************///--------------------------
 int_pop( )  -----------------------------
 ///**********************************************************************/long Evaluator::int_pop(
 )
    {
      long item=0;

      item=int_stack[int_top];
      int_stack[int_top]=0;
      int_top--;

      return item;
   }

 /*************************************************************************///--------------------------
char_push( )  ----------------------------
///*************************************************************************/void
Evaluator::char_push(char item)
   {
      char_top++;
      char_stack[char_top]=item;
   }

 /*************************************************************************///--------------------------
char_pop( )  -----------------------------
///*************************************************************************/char
Evaluator::char_pop( )
   {
      char item=0;

      item=char_stack[char_top];
      char_stack[char_top]=NULL;
      char_top--;

      return item;
   }

 /*************************************************************************///-----------------
show_infix_to_postfix_convertion( )  --------------
///*************************************************************************/void
Evaluator::show_infix_to_postfix_convertion( )
   {
      char_push('(');
      strcpy(infix_expression[infix_rows],")");

      int count_1=0;
      int count_2=0;

      do
     {
       char symbol_scanned[10]={NULL};

       strcpy(symbol_scanned,infix_expression[count_1]);

       if(symbol_scanned[0]=='(')
      char_push(symbol_scanned[0]);
 elseif(symbol_scanned[0]==')')
{
   char temp[5]={NULL};

   while(char_stack[char_top]!='(')
      {
     temp[0]=char_pop( );

     strcpy(postfix_expression[count_2],temp);

     count_2++;
      }

    temp[0]=char_pop( );
}

 elseif(symbol_scanned[0]=='/' || symbol_scanned[0]=='*' ||
       symbol_scanned[0]=='-' || symbol_scanned[0]=='+'
                  || symbol_scanned[0]=='^')
{
   if(symbol_scanned[0]=='^')
      {
      }

   elseif(symbol_scanned[0]=='*' || symbol_scanned[0]=='/')
      {
     while(char_stack[char_top]=='^' ||
            char_stack[char_top]=='*' ||
                   char_stack[char_top]=='/')
        {
        char temp[5]={NULL};

        temp[0]=char_pop( );
        strcpy(postfix_expression[count_2],temp);

        count_2++;
        }
      }

   elseif(symbol_scanned[0]=='+' || symbol_scanned[0]=='-')
      {
     while(char_stack[char_top]!='(')
        {
        char temp[5]={NULL};

        temp[0]=char_pop( );
        strcpy(postfix_expression[count_2],temp);

        count_2++;
         }
```

```
              }

            char_push(symbol_scanned[0]);
          }

           elseif(symbol_scanned[0]!='/' || symbol_scanned[0]!='*' ||
               symbol_scanned[0]!='-' || symbol_scanned[0]!='+' ||
               symbol_scanned[0]!='(' || symbol_scanned[0]!=')' ||
                                symbol_scanned[0]!='^')
          {
            strcpy(postfix_expression[count_2],symbol_scanned);

            count_2++;
          }

           count_1++;
        }
         while(char_stack[char_top]!=NULL);

         postfix_rows=count_2;

         cout<<"\n\n\t Postfix Expression is : ";

         for(int count_3=0;count_3<postfix_rows;count_3++)
        cout<<postfix_expression[count_3]<<" ";

         cout<<endl;
      }

 /*************************************************************************///------------
show_evaluation_of_postfix_expression( )  ---------------
///*************************************************************************/void
Evaluator::show_evaluation_of_postfix_expression( )
    {
       int count=0;

       char symbol_scanned[10]={NULL};

       strcat(postfix_expression[postfix_rows],"=");

       do
      {
         strset(symbol_scanned,NULL);
         strcpy(symbol_scanned,postfix_expression[count]);

         count++;

         if(symbol_scanned[0]=='/' || symbol_scanned[0]=='*' ||
        symbol_scanned[0]=='-' || symbol_scanned[0]=='+' ||
                           symbol_scanned[0]=='^')
        {
            long value_1=0;
            long value_2=0;
            long result=0;

            value_1=int_pop( );
            value_2=int_pop( );

            switch(symbol_scanned[0])
              {
             case'+': result=value_2+value_1;
                  break;

             case'/': result=value_2/value_1;
                  break;

             case'*': result=value_2*value_1;
                  break;

             case'-': result=value_2-value_1;
                  break;

             case'^': result=powl(value_2,value_1);
                  break;
              }

            int_push(result);
          }

           elseif((symbol_scanned[0]!='/' || symbol_scanned[0]!='*' ||
               symbol_scanned[0]!='-' || symbol_scanned[0]!='+' ||
                symbol_scanned[0]!='^' )&& symbol_scanned[0]!='=')

          {
            long number=atol(symbol_scanned);

            int_push(number);
          }
        }
         while(symbol_scanned[0]!='=');

         cout<<"\n\n\t Result of Postfix Expression Evaluation : ";
         cout<<int_stack[0];

         getch( );
      }


    int main( )
      {
         clrscr( );
        Evaluator obj;

        obj.get_infix_expression( );
        obj.show_infix_to_postfix_convertion( );
        obj.show_evaluation_of_postfix_expression( );

        return 0;
    }
```

Experiment 5

AIM: Program to demonstration the implementation of various operations on a linear queue represented using a linear array

PROGRAM

```
************************************************/
/*PROGRAM TO IMPLEMENT LINEAR QUEUE USING ARRAYS*/
/***********************************************/

#include<stdio.h>
#include<conio.h>
#include<malloc.h>
#include<process.h>
#include<ctype.h>

#define SIZE 5

void menu();
void display();
int underflow();
int overflow();
void enqueue(int);
void dequeue();

int queue[SIZE];
int front=-1;
int rear=-1;

void main()
{
        clrscr();
        menu();
}

void menu()
{
        int choice,item;
        printf("MENU");
        printf("\n1. Insert into the queue");
        printf("\n2. Delete from queue");
        printf("\n3. Display");
        printf("\n4. Exit");
        printf("\nEnter your choice: ");
        scanf("%d",&choice);
        switch(choice)
        {
                case 1:
                        clrscr();
                        if(overflow()==0)
                        {
                                printf("\nEnter the item tobe inserted: ");
                                scanf("%d",&item);
                                enqueue(item);
                                clrscr();
                                printf("\nAfter inserting queue is:\n");
                        }
                        display();
                        getch();
                        clrscr();
                        menu();
                        break;

                case 2:
                        clrscr();
                        if(underflow()==1)
                        {
                                dequeue();
                                if(underflow()==1)
                                {
                                        printf("\nAfter deletion queue is:\n");
                                        display();
                                }
                        }
                        getch();
                        clrscr();
                        menu();
                        break;
                case 3:
                        clrscr();
                        if(underflow()==1)
                        {
                                printf("The queue is:\n");
                                display();
                        }
                        getch();
                        clrscr();
                        menu();
                        break;
                case 4:
                        exit(1);
                default:
                        clrscr();
                        printf("Your choice is wrong\n\n");
                        menu();
        }
}

int underflow()
```

```
{
        if((front==-1)&&(rear==-1))
        {
                printf("\nQueue is empty");
                return(0);
        }
        else
        {
                return(1);
        }
}

int overflow()
{
        if(rear==SIZE-1)
        {
                printf("\nQueue is full\n");
                return(1);
        }
        else
        {
                return(0);
        }
}

void enqueue(int item)
{
        if((front==-1)&&(rear==-1))
        {
                front=0;
                rear=0;
        }
        else
        {
                rear=rear+1;
        }
        queue[rear]=item;
}

void dequeue()
{
        if(front==rear)
        {
                front=-1;
                rear=-1;
        }
        else
        {
                front=front+1;
        }
}
```

```
void display()
{
        int i;
        for(i=front;i<=rear;i++)
        {
                printf("\nElement %d : %d",i+1,queue[i]);
        }
}
```

Experiment 6

AIM: Program to demonstration the implementation of various operations on a circular queue represented using a linear array.

PROGRAM

```
/*************************************************/
/*PROGRAM TO IMPLEMENT CIRCULAR QUEUE USING ARRAYS*/
/*************************************************/

#include<stdio.h>
#include<conio.h>
#include<malloc.h>
#include<process.h>
#include<ctype.h>

#define SIZE 5

void menu();
void display();
int underflow();
int overflow();
void enqueue(int);
void dequeue();

int cqueue[SIZE];
int front=-1;
int rear=-1;

void main()
{
	clrscr();
	menu();
}

void menu()
{
	int choice,item;
	printf("MENU");
	printf("\n1. Insert into the queue");
	printf("\n2. Delete from queue");
	printf("\n3. Display");
	printf("\n4. Exit");
	printf("\nEnter your choice: ");
	scanf("%d",&choice);
	switch(choice)
	{
		case 1:
			clrscr();
			if(overflow()==0)
			{
				printf("\nEnter the item tobe inserted: ");
				scanf("%d",&item);
				enqueue(item);
				clrscr();
				printf("\nAfter inserting queue is:\n");
			}
			display();
			getch();
			clrscr();
			menu();
			break;
		case 2:
			clrscr();
			if(underflow()==1)
			{
				dequeue();
				if(underflow()==1)
				{
					printf("\nAfter deletion queue is:\n");
					display();
				}
			}
			getch();
			clrscr();
			menu();
			break;
		case 3:
			clrscr();
			if(underflow()==1)
			{
				printf("The queue is:\n");
				display();
			}
			getch();
			clrscr();
			menu();
			break;
		case 4:
			exit(1);
		default:
			clrscr();
			printf("Your choice is wrong\n\n");
			menu();
	}
}
```

```
int underflow()
{
	if((front==-1)&&(rear==-1))
	{
		printf("\nQueue is empty");
		return(0);
	}
	else
	{
		return(1);
	}
}

int overflow()
{
	if(((front==0)&&(rear==SIZE-1))||(front==rear+1))
	{
		printf("\nQueue is full\n");
		return(1);
	}
	else
	{
		return(0);
	}
}
void enqueue(int item)
{
	if((front==-1)&&(rear==-1))
	{
		front=0;
		rear=0;
	}
	else if(rear==SIZE-1)
	{
		rear=0;
	}
	else
	{
		rear=rear+1;
	}
	cqueue[rear]=item;
}

void dequeue()
{
	if(front==rear)
	{
		front=-1;
		rear=-1;
	}
	else if(front==SIZE-1)
```

```
		{
			front=0;
		}
		else
		{
			front=front+1;
		}
	}

	void display()
	{
		int i;
		if(front<=rear)
		{
			for(i=front;i<=rear;i++)
			{
				printf("\nElement %d : %d",i+1,cqueue[i]);
			}
		}
		else
		{
			for(i=front;i<=SIZE-1;i++)
			{
				printf("\nElement %d : %d",i+1,cqueue[i]);
			}
			for(i=0;i<=rear;i++)
			{
				printf("\nElement %d : %d",i+1,cqueue[i]);
			}
		}
	}
```

Experiment 7

AIM: Program to demonstration the implementation of various operations on a queue represented using a linear linked list (linked queue).

PROGRAM

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

using namespace std;

struct node
{
        int data;
        struct node *next;
}*front=NULL,*rear,*temp;


void ins()
{
        temp=new node;
        cout<<"Enter data:";
        cin>>temp->data;
        temp->next=NULL;

        if(front==NULL)
                front=rear=temp;
        else
        {
                rear->next=temp;
                rear=temp;
        }
        cout<<"Node has been inserted\n";
}


void del()
{
        if(front==NULL)
                cout<<"Queue is empty\n";
        else
        {
                temp=front;
                front=front->next;
                cout<<"Deleted node is "<<temp->data<<"\n";
                delete(temp);
        }
}
```

```
void dis()
{
        if(front==NULL)
                cout<<"Queue is empty\n";
        else
        {
                temp=front;
                while(temp->next!=NULL)
                {
                        cout<<temp->data<<"->";
                        temp=temp->next;
                }

                cout<<temp->data;
        }
}


main()
{
        int ch;
        while(1)
        {
                cout<<"\n*** Menu ***"<<"\n1.Insert\n2.Delete\n3.Display\n4.Exit";
                cout<<"\n\nEnter your choice(1-4):";
                cin>>ch;
                cout<<"\n";

                switch(ch)
                {
                        case 1:  ins();
                                break;
                        case 2: del();
                                break;
                        case 3: dis();
                                break;
                        case 4: exit(0);
                                break;
                        default: cout<<"Wrong Choice!!!";
                }
        }

        return 0;
}
```

Experiment 8

AIM: Program to illustrate the implementation of different operations on a binary search tree.

PROGRAM

```
//Binary Search Tree Operations

#include <iostream.h>
#include <process.h> //for exit(1)
#include <conio.h>

struct node{
 int data;
 struct node *left;
 struct node *right;
};

class BST{
 public:
 node *tree;
 BST(){
    tree=NULL;
 }
 void createTree(node **,int item);          //For Building Tree
 void preOrder(node *);      //For Tree Traversal
 void inOrder(node *);
 void postOrder(node *);

 void determineHeight(node *,int *);
 int totalNodes(node *);
 int internalNodes(node *); //no. of non-leaf nodes
 int externalNodes(node *); //no. of leaf nodes.
 void removeTree(node **); //Remove tree from memory.

 node **searchElement(node **,int);
 void findSmallestNode(node *);
 void findLargestNode(node *);
 void deleteNode(int);
};
```

```
//it is used for inseting an single element in
//a tree, but if calls more than once will create tree.
void BST :: createTree(node **tree,int item){
 if(*tree == NULL){
     *tree = new node;
     (*tree)->data = item;
     (*tree)->left = NULL;
     (*tree)->right = NULL;
 }
 else{
     if( (*tree)->data > item)
          createTree( &((*tree)->left),item);
     else
          createTree( &((*tree)->right),item);
 }
}


void BST :: preOrder(node *tree){
 if( tree!=NULL){
     cout<<"   "<< tree->data;
     preOrder(tree->left);
     preOrder(tree->right);
 }
}

void BST :: inOrder(node *tree){
 if( tree!=NULL){
     inOrder( tree->left);
     cout<<"   "<< tree->data;
     inOrder(tree->right);
 }
}


void BST :: postOrder(node *tree){
 if( tree!=NULL){
     postOrder( tree->left);
     postOrder( tree->right);
     cout<<"   "<<tree->data;
 }
}
```

```
void BST :: determineHeight(node *tree, int *height){
 int left_height, right_height;
 if( tree == NULL)
    *height = 0;
 else{
    determineHeight(tree->left, &left_height);
    determineHeight(tree->right, &right_height);
    if( left_height > right_height)
        *height = left_height + 1;
    else
        *height = right_height + 1;
 }
}


int BST :: totalNodes(node *tree){
 if( tree == NULL)
    return 0;
 else
    return( totalNodes(tree->left) + totalNodes(tree->right) + 1 );
}

int BST :: internalNodes(node *tree){
 if( (tree==NULL)  || (tree->left==NULL  && tree->right==NULL))
    return 0;
 else
    return( internalNodes(tree->left) + internalNodes(tree->right) + 1 );
}

int BST :: externalNodes(node *tree){
 if( tree==NULL )
    return 0;
 else if( tree->left==NULL  && tree->right==NULL)
    return 1;
 else
    return( externalNodes(tree->left) + externalNodes(tree->right));
}

void BST :: removeTree(node **tree){
 if( (*tree) != NULL){
    removeTree( &(*tree)->left );
    removeTree( &(*tree)->right );
    delete( *tree );
 }
}

node ** BST :: searchElement(node **tree, int item){
 if( ((*tree)->data == item) || ( (*tree) == NULL) )
     return tree;
 else if( item < (*tree)->data)
     return searchElement( &(*tree)->left, item);
 else
     return searchElement( &(*tree)->right, item);
}

void BST :: findSmallestNode(node *tree){
 if( tree==NULL || tree->left==NULL)
     cout<< tree->data;
 else
     findSmallestNode( tree->left);
}


//Finding In_order Successor of given node..
//for Delete Algo.
node * find_Insucc(node *curr)
{
 node *succ=curr->right; //Move to the right sub-tree.
 if(succ!=NULL){
     while(succ->left!=NULL)          //If right sub-tree is not empty.
          succ=succ->left; //move to the left-most end.
 }
 return(succ);
}


void BST :: findLargestNode(node *tree){
 if( tree==NULL || tree->right==NULL)
     cout<<tree->data;
 else
     findLargestNode(tree->right);
}


void BST :: deleteNode(int item){
 node *curr=tree,*succ,*pred;
 int flag=0,delcase;
```

```
//step to find location of node
while(curr!=NULL && flag!=1)
{
   if(item < curr->data){
       pred = curr;
       curr = curr->left;
   }
   else if(item > curr->data){
       pred = curr;
       curr = curr->right;
   }
   else{ //curr->data = item
       flag=1;
   }
}

if(flag==0){
   cout<<"\nItem does not exist : No deletion\n";
   getch();
   goto end;
}

//Decide the  case of deletion
if(curr->left==NULL && curr->right==NULL)
   delcase=1; //Node has no child
else if(curr->left!=NULL && curr->right!=NULL)
   delcase=3; //Node contains both the child
else
   delcase=2; //Node contains only one child


//Deletion Case 1
if(delcase==1){
   if(pred->left == curr) //if the node is a left child
       pred->left=NULL; //set pointer of its parent
   else
       pred->right=NULL;
   delete(curr); //Return deleted node to the memory bank.
}

//Deletion Case 2
if(delcase==2){
   if(pred->left==curr){ //if the node is a left child
       if(curr->left==NULL)
```

```
                pred->left=curr->right;
            else
                pred->left=curr->left;
        }
        else{ //pred->right=curr
            if(curr->left==NULL)
                pred->right=curr->right;
            else
                pred->right=curr->left;
        }
        delete(curr);
    }

    //Deletion case 3
    if(delcase==3){
        succ = find_Insucc(curr); //Find the in_order successor
                                  //of the node.
        int item1 = succ->data;
        deleteNode(item1);  //Delete the inorder successor
        curr->data = item1; //Replace the data with the data of
                            //in order successor.
    }
end:
}



void main(){
 BST obj;
 int choice;
 int height=0,total=0,n,item;
 node **tmp;

 while(1){
    clrscr();
    cout<<"*****BINARY SEARCH TREE OPERATIONS*****\n\n";
    cout<<"--Binary Tree and Binary Search Tree common operations--\n";
    cout<<"1) Create Tree\n";
    cout<<"2) Traversal\n";
    cout<<"3) Height of Tree\n";
    cout<<"4) Total Nodes\n";
    cout<<"5) Internal Nodes \n";
    cout<<"6) External Nodes \n";
    cout<<"7) Remove Tree\n";
```

```
cout<<"\n--Only Binary Search Tree Operations--\n";
cout<<"8)  Insert Node\n";
cout<<"9)  Search Node\n";
cout<<"10) Find Smallest Node\n";
cout<<"11) Find Largest Node\n";
cout<<"12) Delete Node\n";
cout<<"13) Exit\n";
cout<<"Enter your choice : ";
cin>>choice;
switch(choice){
    case 1 : //Create Tree
            cout<<"\n\n--Creating Tree--";
            cout<<"\nHow many nodes u want to enter : ";
            cin>>n;
            for(int i=0;i<n;i++){
                    cout<<"Enter value : ";
                    cin>>item;
                    obj.createTree(&obj.tree,item);
            }
            break;

    case 2 : //All Traversals
            cout<<"\n\nInorder Traversal : ";
            obj.inOrder(obj.tree);

            cout<<"\n\nPre-order Traversal : ";
            obj.preOrder(obj.tree);

            cout<<"\n\nPost-order Traversal : ";
            obj.postOrder(obj.tree);
            getch();
            break;

    case 3 : //Determining Height of Tree
            obj.determineHeight(obj.tree,&height);
            cout<<"\n\nHeight of Tree : "<<height;
            getch();
            break;

    case 4 : //Total nodes in a tree
            total=obj.totalNodes(obj.tree);
            cout<<"\n\nTotal Nodes : "<<total;
            getch();
            break;
```

```
case 5 : //Internal nodes in a tree
        total=obj.internalNodes(obj.tree);
        cout<<"\n\nInternal Nodes : "<<total;
        getch();
        break;

case 6 : //External nodes in a tree
        total=obj.externalNodes(obj.tree);
        cout<<"\n\nExternal Nodes : "<<total;
        getch();
        break;

case 7 : //Remove Tree from memory
        obj.removeTree(&obj.tree);
        cout<<"\n\nTree is removed from Memory";
        getch();
        break;

case 8 : //Inserting a node in a tree
        cout<<"\n\n--Inserting Node in a tree--\n";
        cout<<"Enter value : ";
        cin>>item;
        obj.createTree(&obj.tree,item);
        cout<<"\nItem is inserted\n";
        getch();
        break;

case 9 : //Search element
        cout<<"\n\n--Search Element--\n";
        cout<<"Enter item to searched : ";
        cin>>item;
        &(*tmp) = obj.searchElement(&obj.tree,item);
        if( (*tmp) == NULL)
                cout<<"\nSearch Element Not Found";
        else
                cout<<"\nSearch Element was Found";
        getch();
        break;

case 10 : //Find Smallest Node
        cout<<"\n\nSmallest Node is :  ";
        obj.findSmallestNode(obj.tree);
        getch();
```

```
                        break;

                case 11 : //Find Largest Node
                        cout<<"\n\nLargest Node is :  ";
                        obj.findLargestNode(obj.tree);
                        getch();
                        break;

                case 12 : //Deleting a node from a tree
                        cout<<"\n\n--Deleting a Node from a tree--\n";
                        cout<<"Enter value : ";
                        cin>>item;
                        obj.deleteNode(item);
                        break;

                case 13 : exit(1);
            }//end of switch
         }
        }
```

Experiment 9

AIM: Program to illustrate the traversal of graph using breadth-first search.

PROGRAM

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
int cost[10][10],i,j,k,n,queue[10],front,rear,v,visit[10],visited[10];
void main()
{
        int m;
        clrscr();
        cout <<"enterno of vertices";
        cin >> n;
        cout <<"ente no of edges";
        cin >> m;
        cout <<"\nEDGES \n";
        for(k=1;k<=m;k++)
        {
                cin >>i>>j;
                cost[i][j]=1;
        }
        cout <<"enter initial vertex";
        cin >>v;
        cout <<"Visitied vertices\n";
        cout << v;
        visited[v]=1;
        k=1;
        while(k<n)
        {
                for(j=1;j<=n;j++)
                        if(cost[v][j]!=0 && visited[j]!=1 && visit[j]!=1)
                        {
                                visit[j]=1;
                                queue[rear++]=j;
                        }
                v=queue[front++];
                cout<<v << " ";
                k++;
                visit[v]=0;
                visited[v]=1;
        }
        getch();
}
```

Experiment 10

AIM: Program to illustrate the traversal of graph using depth-first search.

PROGRAM

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
int cost[10][10],i,j,k,n,stack[10],top,v,visit[10],visited[10];
void main()
{
        int m;
        cout <<"enterno of vertices";
        cin >> n;
        cout <<"ente no of edges";
        cin >> m;
        cout <<"\nEDGES \n";
        for(k=1;k<=m;k++)
        {
                cin >>i>>j;
                cost[i][j]=1;
        }
        cout <<"enter initial vertex";
        cin >>v;
        cout <<"ORDER OF VISITED VERTICES";
        cout << v <<" ";
        visited[v]=1;
        k=1;
        while(k<n)
        {
                for(j=n;j>=1;j--)
                        if(cost[v][j]!=0 && visited[j]!=1 && visit[j]!=1)
                        {
                                visit[j]=1;
                                stack [top]=j;
                                top++;
                        }
                v= stack [--top];
                cout<<v << " ";
                k++;
                visit[v]=0; visited[v]=1;
        }
        getch();
}
```

Experiment 11

AIM: Program to sort an array of integers in ascending order using bubble sort.

PROGRAM

Bubble Sort

```
#include<iostream.h>
#include<conio.h>
void main()
{
clrscr();
int a[100],i,n,p,j,temp;

cout<<"\n------------ BUBBLE SORT ------------ \n\n";

cout<<"Enter No. of Elements : ";
cin>>n;

cout<<"\nEnter Elements : \n";
for(i=1;i<=n;i++)
{
cin>>a[i];
}


for(p=1;p<=n-1;p++)             // Loop for Pass
{

for(j=1;j<=n-1;j++)
{
if(a[j]>a[j+1])
{
temp=a[j];                      // Interchange Values
a[j]=a[j+1];
a[j+1]=temp;
}
}

}

cout<<"\nAfter Sorting : \n";
for(i=1;i<=n;i++)
{
cout<<a[i]<<endl;
}
getch();
}
```

AIM: Program to sort an array of integers in ascending order using selection sort

PROGRAM

Selection Sort

```
#include<iostream.h>
#include<conio.h>
void main()
{
clrscr();
int a[100],i,n,p,k,min,loc,temp;

cout<<"\n------------ SELECTION SORT ------------ \n\n";
cout<<"Enter No. of Elements=";
cin>>n;

cout<<"\nEnter Elements=\n";
for(i=1;i<=n;i++)
{
cin>>a[i];
}

for(p=1;p<=n-1;p++)            // Loop for Pass
{
min=a[p];                      // Element Selection
loc=p;

for(k=p+1;k<=n;k++)            // Finding Min Value
{
if(min>a[k])
{
min=a[k];
loc=k;
}
}

temp=a[p];                     // Swap Selected Element and Min Value
a[p]=a[loc];
a[loc]=temp;

}

cout<<"\nAfter Sorting : \n";

for(i=1;i<=n;i++)
{
cout<<a[i]<<endl;
}
getch();
}
```

Experiment 13

AIM: Program to sort an array of integers in ascending order using insertion sort.

PROGRAM

Insertion Sort

```
#include<iostream.h>
#include<conio.h>
void main()
{
clrscr();
int a[100],i,n,p,ptr,temp;

cout<<"\n------------ INSERTION SORT ------------ \n\n";

cout<<"Enter No. of Elements : ";
cin>>n;

cout<<"\nEnter Elements : \n";
for(i=1;i<=n;i++)
{
cin>>a[i];
}

a[0]=0;

for(p=2;p<=n;p++)
{
temp=a[p];
ptr=p-1;

while(temp<a[ptr])
{
a[ptr+1]=a[ptr];               // Move Element Forward
ptr--;
}

a[ptr+1]=temp;                 // Insert Element in Proper Place
}

cout<<"\nAfter Sorting : \n";
for(i=1;i<=n;i++)
{
cout<<a[i]<<endl;
}

getch();
}
```

SRI SUKHMANI INSTITUTE OF ENGINEERING & TECHNOLOGY

Affiliated to PTU, & Approved by AICTE

Experiment 14

AIM: Program to sort an array of integers in ascending order using radix sort.

PROGRAM

RADIX SORT

```
#include <iostream>

using namespace std;

void print(int *input, int n)
{
 for (int i = 0; i < n; i++)
      cout << input[i] << "\t";
}

void radixsort(int *input, int n)
{
  int i;

  // find the max number in the given list.
  // to be used in loop termination part.
  int maxNumber = input[0];
  for (i = 1; i < n; i++)
  {
    if (input[i] > maxNumber)
      maxNumber = input[i];
  }

  // run the loop for each of the decimal places
  int exp = 1;
  int *tmpBuffer = new int[n];
  while (maxNumber / exp > 0)
  {
    int decimalBucket[10] = {  0 };
    // count the occurences in this decimal digit.
    for (i = 0; i < n; i++)
      decimalBucket[input[i] / exp % 10]++;

    // Prepare the position counters to be used for re-ordering the numbers
    // for this decimal place.
    for (i = 1; i < 10; i++)
      decimalBucket[i] += decimalBucket[i - 1];

    // Re order the numbers in the tmpbuffer and later copy back to original buffer.
    for (i = n - 1; i >= 0; i--)
      tmpBuffer[--decimalBucket[input[i] / exp % 10]] = input[i];
    for (i = 0; i < n; i++)
      input[i] = tmpBuffer[i];

    // Move to next decimal place.
```

```
      exp *= 10;

        cout << "\nPASS   : ";
        print(input, n);
  }
}

const int INPUT_SIZE = 10;

int main()
{
  int input[INPUT_SIZE] = {143, 123, 222, 186, 235, 9, 905, 428, 543, 373};
  cout << "Input: ";
  print(input, INPUT_SIZE);
  radixsort(input,INPUT_SIZE);
  cout << "\nOutput: ";
  print(input, INPUT_SIZE);
  cout << "\n";
  return 0;
}
```

AIM: Program to sort an array of integers in ascending order using merge sort.

PROGRAM

MERGE SORT

```
#include <iostream>
using namespace std;
#include <conio.h>
void merge(int *,int, int , int );
void mergesort(int *a, int low, int high)
{
    int mid;
    if (low < high)
    {
        mid=(low+high)/2;
        mergesort(a,low,mid);
        mergesort(a,mid+1,high);
        merge(a,low,high,mid);
    }
    return;
}
void merge(int *a, int low, int high, int mid)
{
    int i, j, k, c[50];
    i = low;
    k = low;
    j = mid + 1;
    while (i <= mid && j <= high)
    {
        if (a[i] < a[j])
        {
            c[k] = a[i];
            k++;
            i++;
        }
        else
                    {
            c[k] = a[j];
            k++;
            j++;
        }
    }
    while (i <= mid)
    {
        c[k] = a[i];
        k++;
        i++;
    }
    while (j <= high)
    {
            c[k] = a[j];
            k++;
            j++;
        }
        for (i = low; i < k; i++)
        {
            a[i] = c[i];
        }
    }
    int main()
    {
        int a[20], i, b[20];
        cout<<"enter  the elements\n";
        for (i = 0; i < 5; i++)
        {
            cin>>a[i];
        }
        mergesort(a, 0, 4);
        cout<<"sorted array\n";
        for (i = 0; i < 5; i++)
        {
            cout<<a[i];
        }
        cout<<"enter  the elements\n";
        for (i = 0; i < 5; i++)
        {
            cin>>b[i];
        }
        mergesort(b, 0, 4);
        cout<<"sorted array\n";
        for (i = 0; i < 5; i++)
        {
            cout<<b[i];
        }
        getch();
    }
```

AIM: Program to sort an array of integers in ascending order using quick sort.

PROGRAM

Algorithm of Quick Sort

```
QuickSort(a,beg,end)            // a is Array
{
if(beg<end)
{
p=Partition(a,beg,end);         //Calling Procedure to Find Pivot

QuickSort(a,beg,p-1);                //QuickSort Calls Itself
QuickSort(a,p+1,end);           //(Divides the List into two Sub Lists)
}
}
```

Procedure to Find Pivot :-

```
Partition(a,beg,end)
{
p=beg, pivot=a[beg];

for loc=beg+1 to end;
{
if(pivot>a[loc])
{
a[p]=a[loc];
a[loc]=a[p+1];
a[p+1]=pivot;

p=p+1;
}
}
return p;
}
```

Program in C++ --- Quick Sort

```
#include<iostream.h>
#include<conio.h>

int Partition(int a[], int beg, int end)          //Function to Find Pivot Point
{
int p=beg, pivot=a[beg], loc;

for(loc=beg+1;loc<=end;loc++)
{
if(pivot>a[loc])
{
a[p]=a[loc];
a[loc]=a[p+1];
a[p+1]=pivot;

p=p+1;
}
}
return p;
}

void QuickSort(int a[], int beg, int end)
{
if(beg<end)
{
int p=Partition(a,beg,end);                    //Calling Procedure to Find Pivot

QuickSort(a,beg,p-1);                          //Calls Itself (Recursion)
QuickSort(a,p+1,end);                                    //Calls Itself (Recursion)
}
}

void main()
{
clrscr();
int a[100],i,n,beg,end;

cout<<"\n------- QUICK SORT -------\n\n";
cout<<"Enter the No. of Elements : ";
cin>>n;

for(i=1;i<=n;i++)
{
cin>>a[i];
}
beg=1;
end=n;

QuickSort(a,beg,end);                              //Calling of QuickSort Function

cout<<;"\nAfter Sorting : \n";
for(i=1;i<=n;i++)
{
cout<<a[i]<<endl;
}
getch();
}
```

Experiment 17

AIM: Program to sort an array of integers in ascending order using heap sort.

PROGRAM

HEAP SORT

```
#include <iostream>
#include <conio.h>
using namespace std;
void max_heapify(int *a, int i, int n)
{
    int j, temp;
    temp = a[i];
    j = 2*i;
    while (j <= n)
    {
        if (j < n && a[j+1] > a[j])
            j = j+1;
        if (temp > a[j])
            break;
        else if (temp <= a[j])
        {
            a[j/2] = a[j];
            j = 2*j;
        }
    }
    a[j/2] = temp;
    return;
}
void heapsort(int *a, int n)
{
    int i, temp;
    for (i = n; i >= 2; i--)
    {
        temp = a[i];
        a[i] = a[1];
        a[1] = temp;
        max_heapify(a, 1, i - 1);
    }
}
void build_maxheap(int *a, int n)
{
    int i;
    for(i = n/2; i >= 1; i--)
    {
        max_heapify(a, i, n);
    }
}
int main()
```

```
{
    int n, i, x;
    cout<<"enter no of elements of array\n";
    cin>>n;
    int a[20];
    for (i = 1; i <= n; i++)
    {
        cout<<"enter element"<<(i)<<endl;
        cin>>a[i];
    }
    build_maxheap(a,n);
    heapsort(a, n);
    cout<<"sorted output\n";
    for (i = 1; i <= n; i++)
    {
        cout<<a[i]<<endl;
    }
    getch();
}
```

Experiment 18

AIM: Program to sort an array of integers in ascending order using shell sort.

PROGRAM

```
SHELL SORT
#include <iostream.h>
#include <conio.h>
#define MAX 10

class shellsort{
   int arr[MAX],n;
   public:
   void getdata();
   void showdata();
   void sortLogic();
};

void shellsort :: getdata(){
   cout<<"How many elements you require : ";
   cin>>n;
   for(int i=0;i<n;i++)
      cin>>arr[i];
}

void shellsort :: showdata(){
   cout<<"\n--Display--\n";
   for(int i=0;i<n;i++)
      cout<<arr[i]<<"   ";
}

void shellsort :: sortLogic(){
   int i,j,temp,increment;

   for(increment=n/2; increment>0; increment /= 2){
      for(i=increment; i<n; i++){
         temp=arr[i];
         for(j=i; j>=increment; j -= increment){
            if(temp < arr[j-increment])
               arr[j] = arr[j-increment];
            elsebreak;
         }
         arr[j] = temp;
      }
   }
}

void main(){
   clrscr();
   cout<<"\n*****Shell Sort*****\n";
   shellsort obj;
```

```
        obj.getdata();
        obj.sortLogic();
        obj.showdata();
        getch();
    }
```

SRI SUKHMANI INSTITUTE OF ENGINEERING & TECHNOLOGY

Affiliated to PTU, & Approved by AICTE

Experiment 19

AIM: Program to demonstrate the use of linear search to search a given element in an array.

PROGRAM

Linear Search

```
#include<iostream.h>
#include<conio.h>

void main()
{
clrscr();

int a[100],i,n,item,s=0;

cout<<"\n------------ LINEAR SEARCH ------------ \n\n";
cout<<"Enter No. of Elements=";
cin>>n;

cout<<"\nEnter Elements=\n";
for(i=1;i<=n;i++)
{
cin>>a[i];
}

cout<<"\nEnter Element you want to Search=";
cin>>item;

for(i=1;i<=n;i++)                    //Array Elements Comparsion with Item
{
if(a[i]==item)
{
cout<<"\nData is Found at Location : "<<i;
s=1;
break;
}
}

if(s==0)
{
cout<<"Data is Not Found";
}


getch();
}
```

AIM: Program to demonstrate the use of binary search to search a given element in a sorted array in ascending order.

PROGRAM

Binary Search

```
#include<iostream.h>
#include<conio.h>
void main()
{
clrscr();
int a[100],n,i,beg,end,mid,item;

cout<<"\n------------ BINARY SEARCH ------------ \n\n";
cout<<"Enter No. of Elements= ";
cin>>n;

cout<<"\nEnter Elements in Sorted Order=\n";
for(i=1;i<=n;i++)
{
cin>>a[i];
}

cout<<"\nEnter Item you want to Search= ";
cin>>item;

beg=1;
end=n;

mid=(beg+end)/2;                    // Find Mid Location of Array

while(beg<=end && a[mid]!=item)     // Compare Item and Value of Mid
{
if(a[mid]<item)
beg=mid+1;
else
end=mid-1;

mid=(beg+end)/2;
}

if(a[mid]==item)
{
cout<<"\nData is Found at Location : "<<mid;
}
else
{
cout<<"Data is Not Found";
}
getch();
```

```
    }
```