INDEX

1. Write a C++ program that uses functions to perform the following:

a. Create a singly linked list of integers.

b. Delete a given integer from the above linked list.

Display the contents of the above list after deletion.

```
#include<iostream.h>
#include<stdio.h>
#include<stdlib.h>
struct node
{
   int info;
   struct node *next;
}*start;
class single_llist
{
   public:
      node* create_node(int);
      void insert_begin();
      void insert_pos();
      void insert_last();
      void delete_pos();
      void display();
      single_llist()
      {
         start = NULL;
      }
};
main()
{
   int choice, nodes, element, position, i;
   single_llist sl;
   start = NULL;
   while (1)
   {
      cout<<endl<<"---------------------------------"<<endl;
      cout<<endl<<"Operations on singly linked list"<<endl;
      cout<<endl<<"---------------------------------"<<endl;
      cout<<"1.Insert Node at beginning"<<endl;
      cout<<"2.Insert node at last"<<endl;
      cout<<"3.Insert node at position"<<endl;
      cout<<"4.Delete a Particular Node"<<endl;
      cout<<"5.Display Linked List"<<endl;
      cout<<"6.Exit "<<endl;
      cout<<"Enter your choice : ";
      cin>>choice;
      switch(choice)
      {
      case 1:
         cout<<"Inserting Node at Beginning: "<<endl;
         sl.insert_begin();
         cout<<endl;
         break;
      case 2:
         cout<<"Inserting Node at Last: "<<endl;
         sl.insert_last();
```

```
                                        cout<<endl;
            break;
        case 3:
            cout<<"Inserting Node at a given position:"<<endl;
            sl.insert_pos();
            cout<<endl;
            break;
        case 4:
            cout<<"Delete a particular node: "<<endl;
            sl.delete_pos();
            break;
        case 5:
            cout<<"Display elements of link list"<<endl;
            sl.display();
            cout<<endl;
            break;
        case 6:
            cout<<"Exiting..."<<endl;
            exit(1);
            break;
        default:
            cout<<"Wrong choice"<<endl;
        }
    }
}
 node *single_llist::create_node(int value)
{
    struct node *temp, *s;
    temp = new(struct node);
    if (temp == NULL)
    {
        cout<<"Memory not allocated "<<endl;
        return 0;
    }
    else
    {
        temp->info = value;
        temp->next = NULL;
        return temp;
    }
}
void single_llist::insert_begin()
{
    int value;
    cout<<"Enter the value to be inserted: ";
    cin>>value;
    struct node *temp, *p;
    temp = create_node(value);
    if (start == NULL)
    {
        start = temp;
        start->next = NULL;
    }
    else
    {
        p = start;
```

```
                                        start = temp;
        start->next = p;
    }
    cout<<"Element Inserted at beginning"<<endl;
}
void single_llist::insert_last()
{
    int value;
    cout<<"Enter the value to be inserted: ";
    cin>>value;
    struct node *temp, *s;
    temp = create_node(value);
    s = start;
    while (s->next != NULL)
    {
        s = s->next;
    }
    temp->next = NULL;
    s->next = temp;
    cout<<"Element Inserted at last"<<endl;
}
void single_llist::insert_pos()
{
    int value, pos, counter = 0;
    cout<<"Enter the value to be inserted: ";
    cin>>value;
    struct node *temp, *s, *ptr;
    temp = create_node(value);
    cout<<"Enter the postion at which node to be inserted: ";
    cin>>pos;
    int i;
    s = start;
    while (s != NULL)
    {
        s = s->next;
        counter++;
    }
    if (pos == 1)
    {
        if (start == NULL)
        {
            start = temp;
            start->next = NULL;
        }
        else
        {
            ptr = start;
            start = temp;
            start->next = ptr;
        }
    }
    else if (pos > 1  && pos <= counter)
    {
        s = start;
        for (i = 1; i < pos; i++)
        {
```

```
                                        ptr = s;
            s = s->next;
        }
        ptr->next = temp;
        temp->next = s;
    }
    else
    {
        cout<<"Positon out of range"<<endl;
    }
}

void single_llist::delete_pos()
{
    int pos, i, counter = 0;
    if (start == NULL)
    {
        cout<<"List is empty"<<endl;
        return;
    }
    cout<<"Enter the position of value to be deleted: ";
    cin>>pos;
    struct node *s, *ptr;
    s = start;
    if (pos == 1)
    {
        start = s->next;
    }
    else
    {
        while (s != NULL)
        {
            s = s->next;
            counter++;
        }
        if (pos > 0 && pos <= counter)
        {
            s = start;
            for (i = 1;i < pos;i++)
            {
                ptr = s;
                s = s->next;
            }
            ptr->next = s->next;
        }
        else
        {
            cout<<"Position out of range"<<endl;
        }
        free(s);
        cout<<"Element Deleted"<<endl;
    }
}
void single_llist::display()
{
    struct node *temp;
```

```
if (start == NULL)
   {
      cout<<"The List is Empty"<<endl;
      return;
   }
   temp = start;++++++
   cout<<"Elements of list are: "<<endl;
   while (temp != NULL)
   {
      cout<<temp->info<<"->";
      temp = temp->next;
   }
   cout<<"NULL"<<endl;
}
```

Output:

```
-----------------------------------
Operations on single linked list
-----------------------------------
    1. Insert node at beginning
    2. Insert node at last
    3. Insert node at position
    4. Delete a particular node
    5. Display linked list
    6. Exit
Enter your choice : 1
Inserting node at beginning
Enter the value to be inserted:12
Element inserted at beginning
-----------------------------------
Operations on single linked list
-----------------------------------
    1. Insert node at beginning
    2. Insert node at last
    3. Insert node at position
    4. Delete a particular node
    5. Display linked list
    6. Exit
Enter your choice : 5
Display elements of linked list
Elements of list are:12
-----------------------------------
Operations on single linked list
-----------------------------------
    1. Insert node at beginning
    2. Insert node at last
    3. Insert node at position
    4. Delete a particular node
    5. Display linked list
    6. Exit
Enter your choice : 6Exiting…
```

2. Write a template based C++ program that uses functions to perform the following:
Create a doubly linked list of elements.
Delete a given element from the above doubly linked list.
Display the contents of the above list after deletion.

```
#include<iostream.h>
#include<stdio.h>
#include<stdlib.h>
struct node
{
   int info;
   struct node *next;
   struct node *prev;
}*start;

class double_llist
{
   public:
      void create_list(int value);
      void add_begin(int value);
      void add_after(int value, int position);
      void delete_element(int value);
      void display_dlist();
      double_llist()
      {
         start = NULL;
      }
};
int main()
{
   int choice, element, position;
   double_llist dl;
   while (1)
   {
      cout<<endl<<"----------------------------"<<endl;
      cout<<endl<<"Operations on Doubly linked list"<<endl;
      cout<<endl<<"----------------------------"<<endl;
      cout<<"1.Create Node"<<endl;
      cout<<"2.Add at begining"<<endl;
      cout<<"3.Add after position"<<endl;
      cout<<"4.Delete"<<endl;
      cout<<"5.Display"<<endl;
      cout<<"6.Quit"<<endl;
      cout<<"Enter your choice : ";
      cin>>choice;
      switch ( choice )
      {
      case 1:
         cout<<"Enter the element: ";
         cin>>element;
         dl.create_list(element);
         cout<<endl;
         break;
      case 2:
         cout<<"Enter the element: ";
         cin>>element;
```

```
                                        dl.add_begin(element);
            cout<<endl;
            break;
        case 3:
            cout<<"Enter the element: ";
            cin>>element;
            cout<<"Insert Element after postion: ";
            cin>>position;
            dl.add_after(element, position);
            cout<<endl;
            break;
        case 4:
            if (start == NULL)
            {
               cout<<"List empty,nothing to delete"<<endl;
               break;
            }
            cout<<"Enter the element for deletion: ";
            cin>>element;
            dl.delete_element(element);
            cout<<endl;
            break;
        case 5:
            dl.display_dlist();
            cout<<endl;
            break;
        case 6:
            exit(1);
        default:
            cout<<"Wrong choice"<<endl;
        }
    }
    return 0;
}
void double_llist::create_list(int value)
{
    struct node *s, *temp;
    temp = new(struct node);
    temp->info = value;
    temp->next = NULL;
    if (start == NULL)
    {
        temp->prev = NULL;
        start = temp;
    }
    else
    {
        s = start;
        while (s->next != NULL)
            s = s->next;
        s->next = temp;
        temp->prev = s;
    }
}
void double_llist::add_begin(int value)
{
```

```
if (start == NULL)
   {
      cout<<"First Create the list."<<endl;
      return;
   }
   struct node *temp;
   temp = new(struct node);
   temp->prev = NULL;
   temp->info = value;
   temp->next = start;
   start->prev = temp;
   start = temp;
   cout<<"Element Inserted"<<endl;
}
void double_llist::add_after(int value, int pos)
{
   if (start == NULL)
   {
      cout<<"First Create the list."<<endl;
      return;
   }
   struct node *tmp, *q;
   int i;
   q = start;
   for (i = 0;i < pos - 1;i++)
   {
      q = q->next;
      if (q == NULL)
      {
         cout<<"There are less than ";
         cout<<pos<<" elements."<<endl;
         return;
      }
   }
   tmp = new(struct node);
   tmp->info = value;
   if (q->next == NULL)
   {
      q->next = tmp;
      tmp->next = NULL;
      tmp->prev = q;
   }
   else
   {
      tmp->next = q->next;
      tmp->next->prev = tmp;
      q->next = tmp;
      tmp->prev = q;
   }
   cout<<"Element Inserted"<<endl;
}
void double_llist::delete_element(int value)
{
   struct node *tmp, *q;
   /*first element deletion*/
   if (start->info == value)
```

```
                                        {
        tmp = start;
        start = start->next;
        start->prev = NULL;
        cout<<"Element Deleted"<<endl;
        free(tmp);
        return;
    }
    q = start;
    while (q->next->next != NULL)
    {
        /*Element deleted in between*/
        if (q->next->info == value)
        {
            tmp = q->next;
            q->next = tmp->next;
            tmp->next->prev = q;
            cout<<"Element Deleted"<<endl;
            free(tmp);
            return;
        }
        q = q->next;
    }
     /*last element deleted*/
    if (q->next->info == value)
    {
        tmp = q->next;
        free(tmp);
        q->next = NULL;
        cout<<"Element Deleted"<<endl;
        return;
    }
    cout<<"Element "<<value<<" not found"<<endl;
}

void double_llist::display_dlist()
{
    struct node *q;
    if (start == NULL)
    {
        cout<<"List empty,nothing to display"<<endl;
        return;
    }
    q = start;
    cout<<"The Doubly Link List is :"<<endl;
    while (q != NULL)
    {
        cout<<q->info<<" <-> ";
        q = q->next;
    }
    cout<<"NULL"<<endl;
}
```

Output:

```
-------------------------------------
Operations on double linked list
-------------------------------------
    1. Create node
    2. Insert at beginning
    3. Add after position
    4. Delete
    5. Display
    6. Quit
Enter your choice : 1
Enter the element:4
-------------------------------------
Operations on double linked list
-------------------------------------
    1. Create node
    2. Insert at beginning
    3. Add after position
    4. Delete
    5. Display
    6. Quit
Enter your choice : 5
The double linked list is:4
-------------------------------------
Operations on double linked list
-------------------------------------
    1. Create node
    2. Insert at beginning
    3. Add after position
    4. Delete
    5. Display
    6. Quit
Enter your choice : 6
```

3. Write a C++ program that uses stack operations to convert a given infix expression into its postfix equivalent, Implement the stack using an array.

```
#include<iostream.h>
#include<string.h>
#include<stack.h>
int getWeight(char ch) {
  switch (ch) {
    case '/':
    case '*': return 2;
    case '+':
    case '-': return 1;
    default : return 0;
  }
}
void infix2postfix(char infix[], char postfix[], int size) {
  stack<char> s;
  int weight;
  int i = 0;
  int k = 0;
  char ch;
  while (i < size) {
    ch = infix[i];
    if (ch == '(') {
      s.push(ch);
      i++;
      continue;
    }
    if (ch == ')') {
      while (!s.empty() && s.top() != '(') {
        postfix[k++] = s.top();
        s.pop();

      }
      if (!s.empty()) {
        s.pop();
      }
      i++;
      continue;
    }
    weight = getWeight(ch);
    if (weight == 0) {
      postfix[k++] = ch;
    }
    else {
```

```
                                    if (s.empty()) {
          s.push(ch);
        }
        else {
          while (!s.empty() && s.top() != '(' &&
               weight <= getWeight(s.top())) {
            postfix[k++] = s.top();
            s.pop();

          }
          s.push(ch);
        }
      }
      i++;
    }
    while (!s.empty()) {
      postfix[k++] = s.top();
      s.pop();
    }
    postfix[k] = 0; // null terminate the postfix expression
}
int main() {
    char infix[] = "A*(B+C)/D";
    int size = strlen(infix);
    char postfix[size];
    infix2postfix(infix,postfix,size);
    cout<<"\nInfix Expression :: "<<infix;
    cout<<"\nPostfix Expression :: "<<postfix;
    cout<<endl;
    return 0;
}
```

Output :

```
Infix expression: A*(B+C)*D
POSTFIX EXPRESSION:ABC+D**
```

4. **Write a C++ program to implement a double ended queue ADT using an array, using a doubly linked list.**

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

class node
{
public:
int data;
class  node *next;
class node *prev;
};

class dqueue: public node
{
  node *head,*tail;
  int top1,top2;
  public:
   dqueue()
   {
   top1=0;
   top2=0;
   head=NULL;
   tail=NULL;
   }
  void push(int x){
              node *temp;
              int ch;
              if(top1+top2 >=5)
              {
                cout <<"dqueue overflow";
                return ;
              }
              if( top1+top2 == 0)
                {
                  head = new node;
                  head->data=x;
                  head->next=NULL;
                  head->prev=NULL;
                  tail=head;
                  top1++;
                }
               else
               {
                 cout <<" Add element 1.FIRST 2.LAST\n enter ur choice:";
                 cin >> ch;
```

```
            if(ch==1)
            {
              top1++;
              temp=new node;
              temp->data=x;
              temp->next=head;
              temp->prev=NULL;
              head->prev=temp;
              head=temp;
            }
            else
            {
              top2++;
              temp=new node;
              temp->data=x;
              temp->next=NULL;
              temp->prev=tail;
              tail->next=temp;
              tail=temp;
            }

          }
}
void pop()
{
 int ch;
 cout <<"Delete 1.First Node 2.Last Node\n Enter ur choice:";
 cin >>ch;
 if(top1 + top2 <=0)
 {
   cout <<"\nDqueue under flow";
   return;
 }
 if(ch==1)
 {
   head=head->next;
   head->prev=NULL;
   top1--;
 }
 else
 {
   top2--;
   tail=tail->prev;
   tail->next=NULL;
 }
}

void display()
{
 int ch;
 node *temp;
```

```
cout <<"display from 1.Staring 2.Ending\n Enter ur choice";
  cin >>ch;
  if(top1+top2 <=0)
  {
   cout <<"under flow";
   return ;
  }
  if (ch==1)
  {
   temp=head;
   while(temp!=NULL)
   {
    cout << temp->data <<" ";
    temp=temp->next;
   }
  }
  else
  {
   temp=tail;
   while( temp!=NULL)
   {
    cout <<temp->data << " ";
    temp=temp->prev;
   }
  }
   }
 };

 main()
 {
  dqueue d1;
  int ch;
  while (1){
          cout <<"1.INSERT  2.DELETE  3.DISPLAU  4.EXIT\n Enter ur choice:";
  cin >>ch;
  switch(ch)
  {
  case 1:    cout <<"enter element";
                  cin >> ch;
                  d1.push(ch); break;
  case 2: d1.pop(); break;
  case 3: d1.display(); break;
  case 4: exit(1);
  }
 }}
```

OUTPUT:

1. Insert
2. Delete
3. Display

4. Exit

Enter your choice:1
Enter element:4
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice:1
Enter element:5
Add element 1. First 2. Last
Enter your choice:1
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice:4

5.Write a C++ program that uses functions to perform the following:

a. Create a binary search tree of characters.

b. Traverse the above Binary search tree recursively in preorder, in order and post order

```
#include<iostream>


struct Node {
        char data;
        struct Node *left;
        struct Node *right;
};

void Preorder(struct Node *root) {
        // base condition for recursion
        // if tree/sub-tree is empty, return and exit
        if(root == NULL) return;

        printf("%c ",root->data); // Print data
        Preorder(root->left);     // Visit left subtree
        Preorder(root->right);    // Visit right subtree
}

//Function to visit nodes in Inorder
void Inorder(Node *root) {
        if(root == NULL) return;

        Inorder(root->left);       //Visit left subtree
        printf("%c ",root->data);  //Print data
        Inorder(root->right);      // Visit right subtree
}

//Function to visit nodes in Postorder
void Postorder(Node *root) {
        if(root == NULL) return;

        Postorder(root->left);    // Visit left subtree
        Postorder(root->right);   // Visit right subtree
        printf("%c ",root->data); // Print data
}

// Function to Insert Node in a Binary Search Tree
Node* Insert(Node *root,char data) {
        if(root == NULL) {
                root = new Node();
                root->data = data;
                root->left = root->right = NULL;
        }
        else if(data <= root->data)
                root->left = Insert(root->left,data);
        else
                root->right = Insert(root->right,data);
        return root;
}
```

```
int main() {
        Node* root = NULL;
        root = Insert(root,'M'); root = Insert(root,'B');
root = Insert(root,'Q'); root = Insert(root,'Z');
        root = Insert(root,'A'); root = Insert(root,'C');
        //Print Nodes in Preorder.
        cout<<"Preorder: ";
        Preorder(root);
        cout<<"\n";
        //Print Nodes in Inorder
        cout<<"Inorder: ";
        Inorder(root);
        cout<<"\n";
        //Print Nodes in Postorder
        cout<<"Postorder: ";
        Postorder(root);
        cout<<"\n";
}
```

Output:

```
 PREORDER:CABMZQ
INORDER:ABCMQZ
POSTORDER:MACBZQ
```

6.Write a C++ program that uses function templates to perform the following:
i.Search for a key element in a list of elements using linear search.
ii.Search for a key element in a list of sorted elements using binary search.

```
#include<iostream>
using namespace std;
template <class T>
void Lsearch(T *a, T item, int n)
{
 int z=0;
 for(int i=0;i<n;i++)
 {
  if(a[i]== item)
  {
   z=1;
   cout<<"\n Item found at position = "<<i+1<<"\n\n";
  }
  else
   if(z!=1)
  {
   z=0;
  }
 }

 if(z==0)
  cout<<"\n Item not found in the list\n\n";
}

template <class T>
void Bsearch(T *a, T item, int n)
{
 int beg=0,end=n-1;
 int mid=beg+end/2;

 while((a[mid]!=item) && (n>0))
 {
  if(item>a[mid])
   beg=mid;
  else
   end=mid;

  mid=(beg+end)/2;

  n--;
 }

 if(a[mid]==item)
  cout<<"\n Item found at position = "<<mid+1<<"\n\n";
 else
  cout<<"\n Item not found in the list\n\n";
```

```
}
  void main()
{
int iarr[10] = {2,42,56,86,87,99,323,546,767,886};
double darr[6]= {2.4, 5.53,44.4, 54.45, 65.7,89.54};
int iele;
double dele;

cout<<"\n Elements of Integer Array \n";
for(int i=0;i<10;i++)
{
 cout<<" "<<iarr[i]<<" ,";
}
cout<<"\n\n Enter an item to be search: ";
cin>>iele;

cout<<"\n\n Linear Search Method\n";
Lsearch(iarr,iele,10);
cout<<"\n\n Binary Search method\n";
Bsearch(iarr,iele,10);

cout<<"\n Elements of double Array \n";
for(int i=0;i<6;i++)
{
 cout<<" "<<darr[i]<<" ,";
}
cout<<"\n\n Enter an item to be search: ";
cin>>dele;

cout<<"\n\n Linear Search Method\n";
Lsearch(darr,dele,6);
cout<<"\n\n Binary Search method\n";
Bsearch(darr,dele,6);

system("pause");
}
```

OUTPUT:

```
Elements of Integer Array
2,42,56,86,87,99,323,546,767,886
Enter an item to be search:86
Linear Search Method
Item found at position:4
Binary Search method
Item found at position:4
Elements of double Array
2.4, 5.53,44.4, 54.45, 65.7,89.54
Enter an item to be search:54.45
Linear Search Method
Item found at position:4
Binary Search method
Item found at position:4
```

7.Write a C++ program that implements Insertion sort algorithm to arrange a list of integers in ascending order.

```
#include <iostream.h>
// A structure to represent a node.
struct list
{
        int data;
        list *next;
};

// Function implementing insertion sort.
list* InsertinList(list *head, int n)
{
        // Creating newnode and temp node.
        list *newnode = new list;
        list *temp = new list;

        // Using newnode as the node to be inserted in the list.
        newnode->data = n;
        newnode->next = NULL;

        // If head is null then assign new node to head.
        if(head == NULL)
        {
                head = newnode;
                return head;
        }
        else
        {
                temp = head;

                // If newnode->data is lesser than head->data, then insert newnode before
head.
                if(newnode->data < head->data)
                {
                        newnode->next = head;
                        head = newnode;
                        return head;
                }

                // Traverse the list till we get value more than newnode->data.
                while(temp->next != NULL)
                {
                        if(newnode->data < (temp->next)->data)
                                break;

                        temp=temp->next;
                }

                // Insert newnode after temp.
```

```
                newnode->next = temp->next;
                temp->next = newnode;
                return head;
        }
}

int main()
{
        int n, i, num;
        // Declaring head of the linked list.
        list *head = new list;
        head = NULL;

        cout<<"\nEnter the number of data element to be sorted: ";
        cin>>n;

        for(i = 0; i < n; i++)
        {
                cout<<"Enter element "<<i+1<<": ";
                cin>>num;
                // Inserting num in the list.
                head = InsertinList(head, num);
        }

        // Display the sorted data.
        cout<<"\nSorted Data ";
        while(head != NULL)
        {
                cout<<"->"<<head->data;
                head = head->next;
        }

        return 0;
}
```

Output:

Enter the number of data element to be sorted: 5
Enter element 1
23
Enter element 2 15
Enter element 3 24
Enter element 4
50
Enter element 5
42
Sorted data→15→23→24→42→50

8. Write a template based C++ program that implements selection sort algorithm to arrange a list of elements in descending order.

```
#include<iostream.h>
template <class T>
void s_sort(T a[],int n)
{
    int i,j,t;
    for(i=0;i<n;i++)
    {
                for(j=i+1;j<n;j++)
                {
                            if(a[j]>a[i]) //for descending order use if(a[j]>a[i])
                            {
                                        t=a[i];
                                        a[i]=a[j];
                                        a[j]=t;
                            }
                }
    }
}
int main()
{
   int a[100],i,n;
   cout<<"Enter The number of Element:\n";
   cin>>n;
   cout<<"\nEnter Elements:\n";
   for(i=0;i<n;i++)
   {

                cout<<"\nEnter:";
               cin>>a[i];
   }
   s_sort(a,n);
   cout<<"\nAfter Sorting\n";
   for(i=0;i<n;i++)
   {
               cout<<a[i]<<"\t";
   }

   return 0;
}
```

Output:

Enter the number of element:3
Enter elements:12
34
22
After sorting: 34 22 12

9. Write a template based C++ program that implements Quick sort algorithm to arrange a list of elements in ascending order.

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

//Function for partitioning array
int part(int low,int high,int *a)
{
    int i,h=high,l=low,p,t;  //p==pivot
    p=a[low];
    while(low<high)
    {
                while(a[l]<p)
                {
                        l++;
                }
                while(a[h]>p)
                {
                        h--;
                }
                if(l<h)
                {
                        t=a[l];
                        a[l]=a[h];
                        a[h]=t;
                }
                else
                {
                   t=p;
                   p=a[l];
                   a[l]=t;
                   break;
                }
    }
    return h;
}

void quick(int l,int h,int *a)
{
  int index,i;
  if(l<h)
  {
        index=part(l,h,a);
        quick(l,index-1,a);
        quick(index+1,h,a);
  }
}

int main()
{
```

```
                                        int a[100],n,l,h,i;
        cout<<"Enter number of elements:";
        cin>>n;
        cout<<"Enter the elements (Use Space As A Separator):";
        for(i=0;i<n;i++)
        cin>>a[i];
        cout<<"\nInitial Array:\n";
        for(i=0;i<n;i++)
        {
                cout<<a[i]<<"\t";
        }
        h=n-1;
        l=0;
        quick(l,h,a);
        cout<<"\nAfter Sorting:\n";
        for(i=0;i<n;i++)
        {
                cout<<a[i]<<"\t";
        }
        getch();
        return 0;
}
```

Output:

Enter the no of elements :3
Enter the elements(use space as a separator):
56
34
12
Initial array: 56 34 12
After sorting: 12 34 56

10. Write a C++ program that implements Heap sort algorithm for sorting a list of integers in ascending order.

```
#include <iostream.h>
const int MAX = 10 ;
class array
{
    private :
            int arr[MAX] ;
            int count ;
    public :
            array( ) ;
            void add ( int num ) ;
            void makeheap(int ) ;
            void heapsort( ) ;
            void display( ) ;
} ;
array :: array( )
{
    count = 0 ;
    for ( int i = 0 ; i < MAX ; i++ )
            arr[MAX] = 0 ;
}
void array :: add ( int num )
{
    if ( count < MAX )
    {
            arr[count] = num ;
            count++ ;
    }
    else
            cout << "\nArray is full" << endl ;
}
void array :: makeheap(int c)
{

    for ( int i = 1 ; i < c ; i++ )
    {
            int val = arr[i] ;
            int s = i ;
            int f = ( s - 1 ) / 2 ;
            while ( s > 0 && arr[f] < val )
            {
                    arr[s] = arr[f] ;
                    s = f ;
                    f = ( s - 1 ) / 2 ;
            }
            arr[s] = val ;
    }
}
void array :: heapsort( )
```

```
                                        {
        for ( int i = count - 1 ; i > 0 ; i-- )
        {
                int ivalue = arr[i] ;
                arr[i] = arr[0] ;
                arr[0]=ivalue;
                makeheap(i);

        }
    }
    void array :: display( )
    {
        for ( int i = 0 ; i < count ; i++ )
                cout << arr[i] << "\t" ;
        cout << endl ;
    }
    void main( )
    {
        array a ;

        a.add ( 11 ) ;
        a.add ( 2 ) ;
        a.add ( 9 ) ;
        a.add ( 13 ) ;
        a.add ( 57 ) ;
        a.add ( 25 ) ;
        a.add ( 17 ) ;
        a.add ( 1 ) ;
        a.add ( 90 ) ;
        a.add ( 3  ) ;
        a.makeheap(10) ;
        cout << "\nHeap Sort.\n" ;
        cout << "\nBefore Sorting:\n"  ;
        a.display( ) ;
        a.heapsort( ) ;
        cout << "\nAfter Sorting:\n" ;
        a.display( ) ;
    }
```

Output:

Heap sort
Before sorting:
90 57 25 13 11 9 17 1 2 3
After sorting:
1 2 3 9 11 13 17 25 57 90

11. Write a C++ program that implements Merge sort algorithm for sorting a list of integers in ascending order

```
#include <iostream.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 < 10; i++)
     {
      cin>>a[i];
     }
     mergesort(a, 0, 9);
     cout<<"sorted array\n";
     for (i = 0; i < 10; i++)
     {
      cout<<a[i]<<endl;
     }
     cout<<"enter  the elements\n";
     for (i = 0; i < 10; i++)
     {
      cin>>b[i];
     }
     mergesort(b, 0, 9);
     cout<<"sorted array\n";
     for (i = 0; i < 10; i++)
     {
      cout<<b[i]<<endl;
     }
  }
```

Output:

Enter the elements
3
12
56
32
42
85
69
14
15
1
Sorted array
1 3 12 14 15 32 42 56 69 85

12. Write a C++ program to implement all the functions of a dictionary (ADT) using hashing.

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
# define max 10
 typedef struct list
{
int data;
struct list  *next;
}node_type;
node_type *ptr[max],*root[max],*temp[max];

class Dictionary
{
public:
  int index;

  Dictionary();
  void insert(int);
  void search(int);
  void delete_ele(int);
};
Dictionary::Dictionary()
{
index=-1;
for(int i=0;i<max;i++)
{
root[i]=NULL;
ptr[i]=NULL;
temp[i]=NULL;
}
}
 void Dictionary::insert(int key)
{
index=int(key%max);
ptr[index]=(node_type*)malloc(sizeof(node_type));
ptr[index]->data=key;
if(root[index]==NULL)
{
root[index]=ptr[index];
root[index]->next=NULL;
temp[index]=ptr[index];
}

else
{
temp[index]=root[index];
while(temp[index]->next!=NULL)
temp[index]=temp[index]->next;
```

```
                                temp[index]->next=ptr[index];
}
}
void Dictionary::search(int key)
{
int flag=0;
index=int(key%max);
temp[index]=root[index];
while(temp[index]!=NULL)
{
if(temp[index]->data==key)
{
cout<<"\nSearch key is found!!";
flag=1;
break;
}
else temp[index]=temp[index]->next;
}
if (flag==0)
cout<<"\nsearch key not found.....";
}

void Dictionary::delete_ele(int key)
{
index=int(key%max);
temp[index]=root[index];
while(temp[index]->data!=key && temp[index]!=NULL)
{
ptr[index]=temp[index];
temp[index]=temp[index]->next;
}
ptr[index]->next=temp[index]->next;
cout<<"\n"<<temp[index]->data<<" has been deleted.";
temp[index]->data=-1;
temp[index]=NULL;
free(temp[index]);
}
main()
{
int val,ch,n,num;
char c;
Dictionary d;

do
{
cout<<"\nMENU:\n1.Create";
cout<<"\n2.Search for a value\n3.Delete an value";
cout<<"\nEnter your choice:";
cin>>ch;
switch(ch)
{
```

```
case 1:cout<<"\nEnter the number of elements to be inserted:";
        cin>>n;
        cout<<"\nEnter the elements to be inserted:";
        for(int i=0;i<n;i++)
        {
        cin>>num;
        d.insert(num);
        }
        break;
case 2:cout<<"\nEnter the element to be searched:";
        cin>>n;
        d.search(n);
case 3:cout<<"\nEnter the element to be deleted:";
        cin>>n;
        d.delete_ele(n);
        break;
default:cout<<"\nInvalid choice....";
}
cout<<"\nEnter y to continue.....";
cin>>c;
}while(c=='y');
getch();
}
```

Output:

MENU:

1. Create
2. Search for a value
3. Delete a value

Enter your choice:1
Enter the number of elements to be inserted:3
Enter the elements to be inserted:12 23 56
Enter y to continue:y

1. Create
2. Search for a value
3. Delete a value

Enter your choice:2
Enter the element searched :23
Search key is found!!
Enter the element to be deleted 23
23 has been deleted

13. Write a C++ program that implements Radix sort algorithm for sorting a list of integers in ascending order

```
#include <iostream.h>

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;
}
```

Output :

```
Input: 143 123 222 186 235 9 905 428 543 373
PASS:222 143 123 543 373 235 905 186 428 9
PASS:905 9  222 123 428 235 143 543 373 186
PASS:9 123 143 186 222 235 373 428 543 905
Output:9 123 143 186 222 235 373 428 543 905
```

14.Write a C++ program that uses functions to perform the following:
i.Create a binary search tree of integers.
ii.Traverse the above Binary search tree non recursively in inorder.

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
class node
{
public:
class node *left;
class node *right;
int data;
};

class tree: public node
{
public:
int stk[50],top;
node *root;
tree()
{
root=NULL;
top=0;
}
void insert(int ch)
{
        node *temp,*temp1;
        if(root== NULL)
        {
                root=new node;
                root->data=ch;
                root->left=NULL;
                root->right=NULL;
                return;
        }
        temp1=new node;
        temp1->data=ch;
        temp1->right=temp1->left=NULL;
        temp=search(root,ch);
        if(temp->data>ch)
                temp->left=temp1;
        else
                temp->right=temp1;

}

node *search(node *temp,int ch)
{
        if(root== NULL)
        {
```

```
                cout <<"no node present";
                return NULL;
        }
        if(temp->left==NULL && temp->right== NULL)
                return temp;

        if(temp->data>ch)
            {  if(temp->left==NULL) return temp;
                search(temp->left,ch);}
        else
             { if(temp->right==NULL) return temp;
             search(temp->right,ch);

}            }

void display(node *temp)
{
        if(temp==NULL)
           return ;
        display(temp->left);
        cout<<temp->data;
        display(temp->right);
}
void inorder( node *root)
{
        node *p;
        p=root;
        top=0;
        do
        {
                while(p!=NULL)
                {
                        stk[top]=p->data;
                        top++;
                        p=p->left;
                }
                if(top>0)
                {

                        p=pop(root);
                        cout << p->data;
                        p=p->right;
                }
        }while(top!=0 || p!=NULL);
}


node * pop(node *p)
{
        int ch;
        ch=stk[top-1];
```

```
                                        if(p->data==ch)
        {
                top--;
                return p;
        }
        if(p->data>ch)
                pop(p->left);
        else
                pop(p->right);
}
};

main()
{
        tree t1;
        int ch,n,i;
        while(1)
        {
                cout                    <<"\n1.INSERT\n2.DISPLAY                3.INORDER
TRAVERSE\n4.EXIT\nEnter your choice:";
                cin >> ch;
                switch(ch)
                {
                case 1:  cout <<"enter no of elements to insert:";
                         cin >> n;
                         for(i=1;i<=n;i++)
                         {  cin >> ch;
                           t1.insert(ch);
                         }
                          break;
                case 2:  t1.display(t1.root);break;
                case 3:  t1.inorder(t1.root); break;
                case 4:  exit(1);
                }
        }
}
```

Output:

1. Insert
2. Display
3. Inorder
4. Exit

Enter your choice:1

Enter no of elements to insert:3

23

24

54

1. Insert
2. Display

3. Inorder
4. Exit

Enter your choice:3
23 24 54

1. Insert
2. Display
3. Inorder
4. Exit

Enter your choice:4

15.Write a C++ program that uses functions to perform the following:

i.Create a binary search tree of integers.

ii.Search for an integer key in the above binary search tree non recursively.

iii.Search for an integer key in the above binary search tree recursively.

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

void insert(int,int );
void display(int);
int search(int);
int search1(int,int);
int tree[40],t=1,s,x,i;

main()
{
        int ch,y;
        for(i=1;i<40;i++)
        tree[i]=-1;
        while(1)
        {
cout <<"1.INSERT\n2.DISPLAY\n3.SEARCH\n4.EXIT\nEnter your choice:";
                cin >> ch;
                switch(ch)
                {
                case 1:
                        cout <<"enter the element to insert";
                        cin >> ch;
                        insert(1,ch);
                        break;
                case 2:
                        display(1);
                        cout<<"\n";
                        for(int i=0;i<=32;i++)
                        cout <<i;
                        cout <<"\n";
                        break;
case 3:
                        cout <<"enter the element to search:";
                        cin >> x;
                        y=search(1);
                        if(y == -1) cout <<"no such element in tree";
                        else cout <<x << "is in" <<y <<"position";
                        break;
                case 4:
                        exit(0);
                }
        }
}
```

```
void insert(int s,int ch )
{
        int x;
        if(t==1)
        {
                tree[t++]=ch;
                return;
        }
        x=search1(s,ch);
        if(tree[x]>ch)
                tree[2*x]=ch;
        else
                tree[2*x+1]=ch;
        t++;
}


int search(int s)
{
if(t==1)
{
cout <<"no element in tree";
return -1;
}
if(tree[s]==-1)
return tree[s];
if(tree[s]>x)
search(2*s);
else if(tree[s]<x)
search(2*s+1);
else
return s;
}

void display(int s)
{
if(t==1)
{cout <<"no element in tree:";
return;}
for(int i=1;i<40;i++)
if(tree[i]==-1)
cout <<" ";
else cout <<tree[i];
return ;
}

int search1(int s,int ch)
{
if(t==1)
{
```

```
cout <<"no element in tree";
return -1;
}
if(tree[s]==-1)
return s/2;
if(tree[s] > ch)
search1(2*s,ch);
else search1(2*s+1,ch);
}
```

Output:

1. Insert
2. Display
3. Search
4. Exit

Enter your choice:1
Enter the element to insert 54

1. Insert
2. Display
3. Search
4. Exit

Enter your choice:1
Enter the element to insert 23

1. Insert
2. Display
3. Search
4. Exit

Enter your choice:1
Enter the element to insert 45

1. Insert
2. Display
3. Search
4. Exit

Enter your choice:2
54 23 45

1. Insert
2. Display
3. Search
4. Exit

Enter your choice:3
Enter element to search: 45
45 is in 3 position

1. Insert
2. Display
3. Search
4. Exit

Enter your choice:4