1. **Write a C++ program for implementing Singly Linked list.**

```
#include<iostream>
#include<cstdlib>
using namespace std;
struct node
{
        int info;
        struct node *next;
}*start;
class single_llist
{
        public:
                node* create_node(int);
                void insert_begin();
                void insert_last();
                void insert_pos();
                void delete_begin();
                void delete_last();
                void delete_pos();
                void update_begin();
                void update_last();
                void update_pos();
                void sort();
                void reverse();
                void search();
                void display();
        single_list()
        {
                start=NULL;
                 }
};
int main()
{
        int choice;
        single_llist s1,s2;
        start=NULL;
        do
        {
                cout<<"-----------------"<<endl;
                cout<<"Operattions on singly linked list"<<endl;
                cout<<"-----------------"<<endl;
                cout<<"1.Insert at first"<<endl;
                cout<<"2.Insert at last"<<endl;
                cout<<"3.Insert at position"<<endl;
                cout<<"4.Delete at first"<<endl;
                cout<<"5.Delete at Last"<<endl;
                cout<<"6.Delete at position"<<endl;
                cout<<"7.Update at first"<<endl;
                cout<<"8.Update at last"<<endl;
```

```
cout<<"9.Update at position"<<endl;
cout<<"10.Ascending order"<<endl;
cout<<"11.Descending order"<<endl;
cout<<"12.Search"<<endl;
cout<<"13.Display"<<endl;
cout<<"14.Exit"<<endl;
cout<<"Enter your choice:";
cin>>choice;
switch(choice)
{
case 1: s1.insert_begin();
            s1.display();
            break;
case 2: s1.insert_last();
            s1.display();
            break;
case 3: s1.insert_pos();
            s1.display();
            break;
case 4: s2.delete_begin();
            s1.display();
            break;
case 5: s2.delete_last();
            s1.display();
            break;
case 6: s1.delete_pos();
            s1.display();
            break;
case 7: s1.update_begin();
            s1.display();
            break;
case 8: s1.update_last();
            s1.display();
            break;
case 9: s1.update_pos();
            s1.display();
            break;
case 10: s1.sort();
            s1.display();
            break;
case 11: s1.reverse();
            s1.display();
            break;
case 12: s1.search();
            s1.display();
            break;
case 13:
            s1.display();
            break;
case 14: exit(0);
```

```
                break;
        default:cout<<"wrong choice...???"<<endl;
                break;
        }
    }
    while(choice!=14);
}
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, *s;
    temp = create_node(value);
    if(start==NULL)
    {
        start=temp;
        start->next=NULL;
        cout<<temp->info<<"is inserted at first in the empty list"<<endl;
    }
    else
    {
        s=start;
        start=temp;
        start->next=s;
        cout<<temp->info<<"is inserted at first"<<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);
    if(start==NULL)
    {
        start=temp;
        start->next=NULL;
        cout<<temp->info<<"is inserted at last in the empty list"<<endl;
    }
    else
    {
        s=start;
        while(s->next!=NULL)
        {
            s=s->next;
        }
        temp->next=NULL;
        s->next=temp;
        cout<<temp->info<<"is inserted at last"<<endl;
    }
}
void single_llist::insert_pos()
{
    int value, pos, counter = 0, loc = 1;
    struct node *temp, *s, *ptr;
    s = start;
    while (s != NULL)
    {
        s = s->next;
        counter++;
    }
    if (counter == 0){}
    else
    {
        cout<<"Enter the postion from "<<loc<<" to "<<counter+1<<" : ";
        cin>>pos;
        s = start;
        if(pos == 1)
        {
        cout<<"Enter the value to be inserted:";
        cin>>value;
        temp=create_node(value);
        start=temp;
        start->next=s;
        cout<<temp->info<<"is inserted at first"<<endl;
}
else if(pos>1 && pos<=counter)
{
    cout<<"Enter the value to be inserted:";
        cin>>value;
        temp=create_node(value);
        for(int i=1;i<pos;i++)
```

```
            {
                ptr=s;
                s=s->next;
            }
            ptr->next=temp;
            temp->next=s;
            cout<<temp->info<<"is inserted at position"<<pos<<endl;
        }
        else if(pos== counter+1)
        {
            cout<<"Enter the value to be inserted:";
            cin>>value;
            temp=create_node(value);
            while(s->next!=NULL)
            {
                s=s->next;
            }
            temp->next=NULL;
            s->next=temp;
            cout<<temp->info<<"is inserted at last"<<endl;
        }
        else
        {
            cout<<"Position out of range...!!!"<<endl;
        }
    }
}
void single_llist::delete_begin()
{
    if(start==NULL){}
    else
    {
        struct node *s, *ptr;
        s=start;
        start=s->next;
        cout<<s->info<<"deleted from first"<<endl;
        free(s);
    }
}
void single_llist::delete_last()
{
    int i, counter=0;
    struct node *s, *ptr;
    if(start==NULL){}
    else
    {
        s=start;
        while(s!=NULL)
        {
            s=s->next;
            counter++;
        }
        s=start;
        if(counter==1)
        {
            start=s->next;
            cout<<s->info<<"Deleted from last"<<endl;
            free(s);
        }
        else
        {
            for(i=1;i<counter;i++)
            {
                ptr=s;
                s=s->next;
            }
            ptr->next=s->next;
            cout<<s->info<<"deleted from last"<<endl;
            free(s);
        }
    }
}
void single_llist::delete_pos()
{
    int pos, i, counter = 0, loc = 1;
    struct node *s, *ptr;
    s = start;
    while (s != NULL)
    {
        s = s->next;
        counter++;
    }
    if(counter==0){}
    else
    {
        if(counter==1)
        {
            cout<<"Enter the postion [SAY"<<loc<<"]:";
            cin>>pos;
            s=start;
            if(pos==1)
            {
                start=s->next;
                cout<<s->info<<"Deleted from first"<<endl;
                free(s);
            }
            else
            cout<<"Position out of range....!!!"<<endl;
        }
        else
```

```
                {
                        cout<<"Enter the position from"<<loc<<"to"<<counter<<":";
                        cin>>pos;
                        s=start;
                        if(pos==1)
                        {
                                start=s->next;
                                cout<<s->info<<"deleted from first"<<endl;
                                free(s);
                        }
                        else if(pos>1 && pos<=counter)
                        {
                                for(i=1;i<pos;i++)
                                {
                                        ptr=s;
                                        s=s->next;
                                }
                                ptr->next=s->next;
                                if(pos==counter)
                                {cout<<s->info<<"deleted from last"<<endl;
                                free(s);}
                                else
                                {cout<<s->info<<"deleted from postion"<<pos<<endl;
                                free(s);}
                        }
                           else
                           cout<<"Position out of range...!!!"<<endl;
                }
        }
}

void single_llist::update_begin()
{
        int value, pos=1,i,counter=0;
        struct node *s, *ptr;
        s=start;
        while(s!=NULL)
        {
                s=s->next;
                counter++;
        }
        if(counter==0){}
        else if(pos==1)
        {
                cout<<"Enter the new node:";
                cin>>value;
                start->info=value;
                cout<<"Node updated at first position : "<<pos<<" = "<<start->info<<endl;
        }
}
```

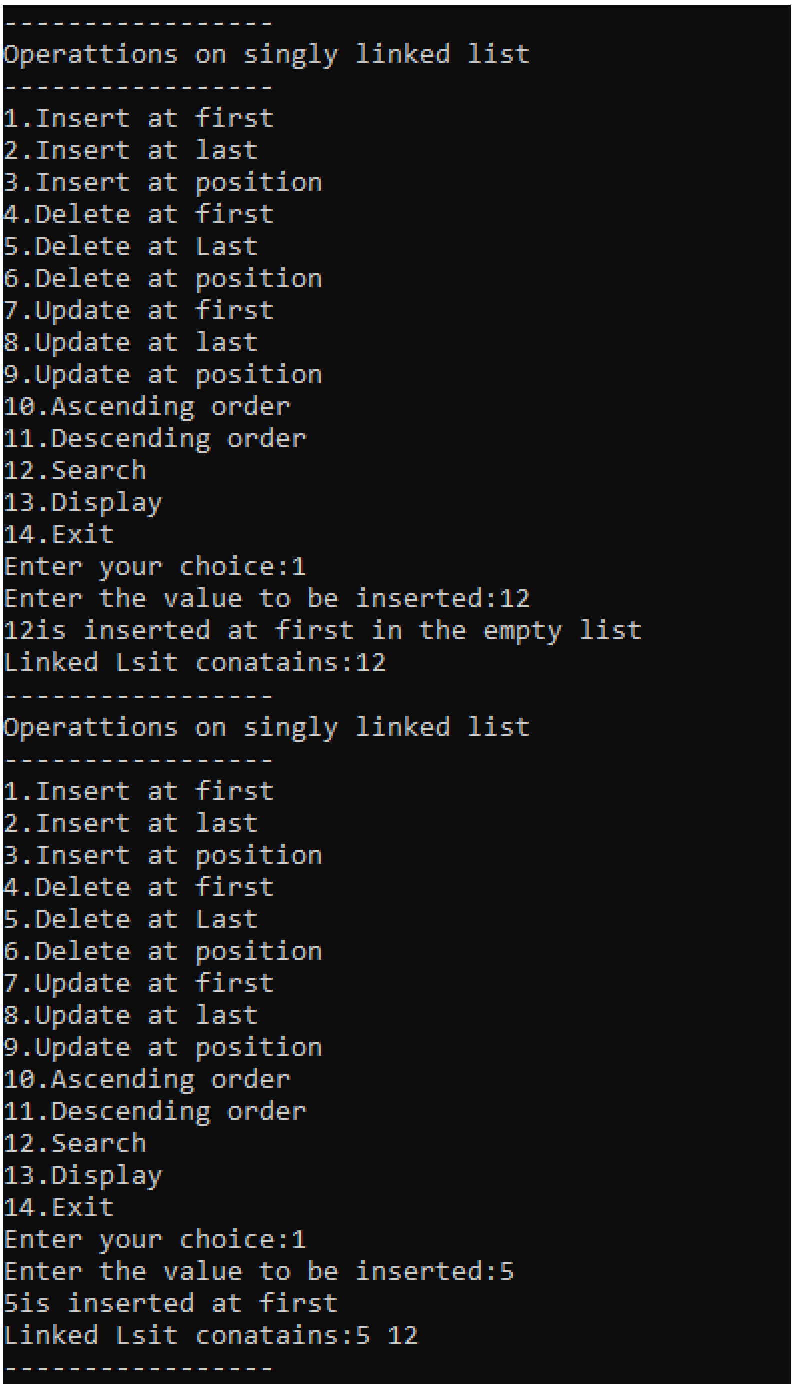
```
void single_llist::update_last()
{
    int value, pos, i, counter=0;
    struct node *s, *ptr;
    s=start;
    while(s!=NULL)
    {
        s=s->next;
        counter++;
    }
    s=start;
    if(counter==0){}
    else
    {
        cout<<"Enter the new node:";
        cin>>value;
        for(i=1;i<counter;i++)
        {
            s=s->next;
        }
        s->info=value;
        cout<<"Node updated at last position:"<<counter<<"="<<s->info<<endl;
    }
}
void single_llist::update_pos()
{
    int value, pos, i,counter = 0, loc = 1;
    struct node *s, *ptr;
    s = start;
    while (s != NULL)
    {
    s=s->next;
    counter++;
    }
    if(counter==0){}
     else
     {
        if(counter==1)
        {
            cout<<"Enter the position[SAY"<<loc<<"];";
            cin>>pos;
            s = start;
            if (pos == 1)
            {
                cout<<"Enter the new node : ";
                cin>>value;
                start->info = value;
                cout<<"Node updated at position : "<<pos<<" = "<<start->info<<endl;
            }
```

```
                                else
                                cout<<"Position out of range...!!!"<<endl;
                        }
                        else
                        {
                                cout<<"Enter the position from "<<loc<<"to"<<counter<<":";
                                cin>>pos;
                                s=start;
                                if(pos==1)
                                {
                                        cout<<"Enter the new node:";
                                        cin>>value;
                                        start->info=value;
                                        cout<<"Node updated at position:"<<pos<<"="<<start-
>info<<endl;
                                 }
                                 else if(pos>1 && pos<=counter)
                                 {
                                        cout<<"Enter the new node:";
                                        cin>>value;
                                        for (i = 1; i < pos ; i++)
                                        {
                                                s = s->next;
                                   }
                                        s->info = value;
                                        cout<<"Node updated at position : "<<pos<<" = "<<s-
>info<<endl;
                                 }
                                 else
                                 cout<<"Position out of range...!!!"<<endl;
                        }
                }
}
void single_llist::sort()
{
        struct node *ptr, *s;
        int value;
        if (start == NULL){}
        else
        {
                ptr = start;
                while (ptr != NULL)
      {
                for (s = ptr->next;s !=NULL;s = s->next)
                {
                        if (ptr->info > s->info)
                        {
                                value = ptr->info;
                                ptr->info = s->info;
                                s->info = value;
```

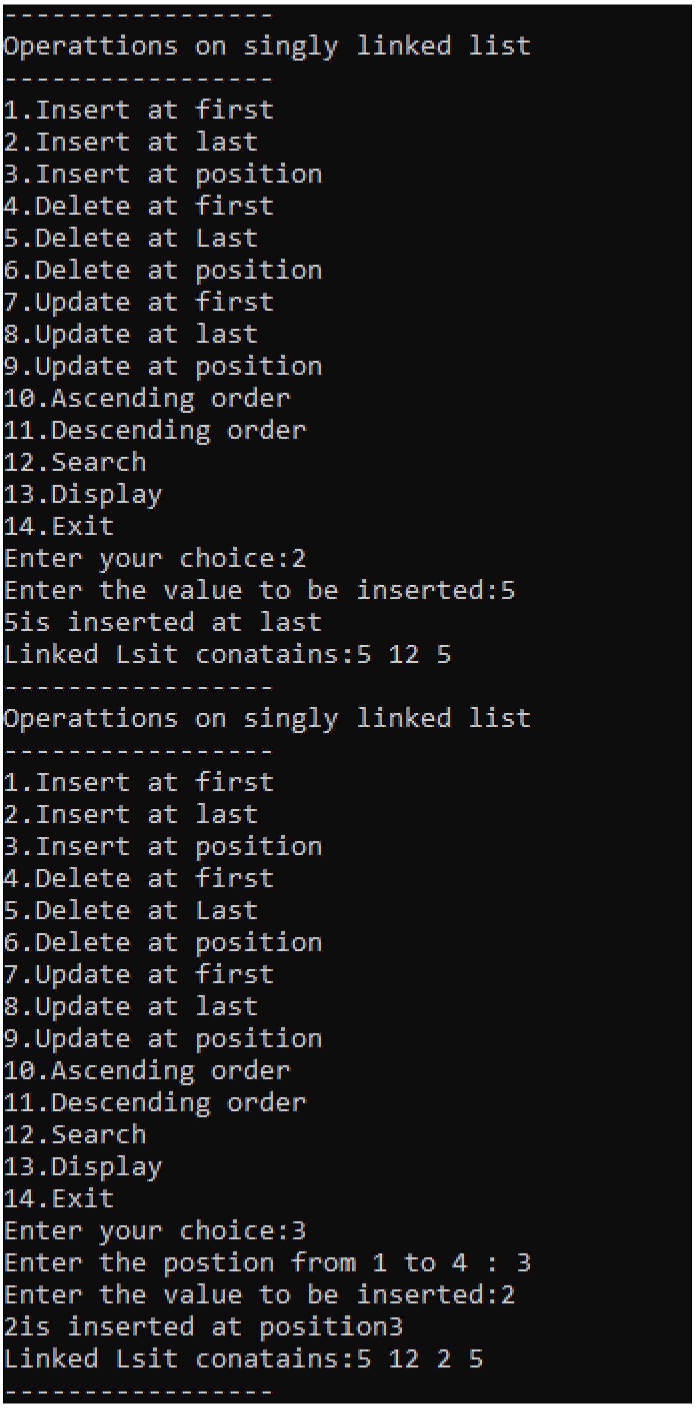
```
            }
        }
        ptr = ptr->next;
    }
    }
}
void single_llist::reverse()
{
    struct node *ptr, *s;
    int value;
    if (start == NULL){}
    else
    {
        ptr = start;
        while (ptr != NULL)
   {
        for (s = ptr->next;s !=NULL;s = s->next)
        {
            if (ptr->info < s->info)
            {
                value = ptr->info;
                ptr->info = s->info;
                s->info = value;
            }
        }
        ptr = ptr->next;
    }
    }
}
void single_llist::search()
{
 int value, loc = 0, pos = 0, counter = 0;
    struct node *s;
    s = start;
    while (s != NULL)
    {
        s = s->next;
        counter++;
    }
    if (start == NULL){}
    else
    {
        cout<<"Enter the value to be searched : ";
        cin>>value;
        struct node *s;
        s = start;
        while (s != NULL)
        {
            pos++;
            if (s->info == value)
```

```
                        {
                                loc++;
                                if(loc == 1)
                                cout<<"Element "<<value<<" is found at position "<<pos;
                                else if(loc <= counter)
                                cout<<" , "<<pos;
                        }
                        s = s->next;
                }
                cout<<endl;
                if (loc == 0)
                cout<<"Element "<<value<<" not found in the list"<<endl;
        }
}
void single_llist::display()
{
        struct node *temp;
        if (start == NULL)
        cout<<"Linked list is empty....!!!"<<endl;
        else
        {
                cout<<"Linked Lsit conatains:";
                temp = start;
                while (temp != NULL)
     {
     cout<<temp->info<<" ";
     temp= temp->next;
        }

                cout<<endl;
        }
}
```

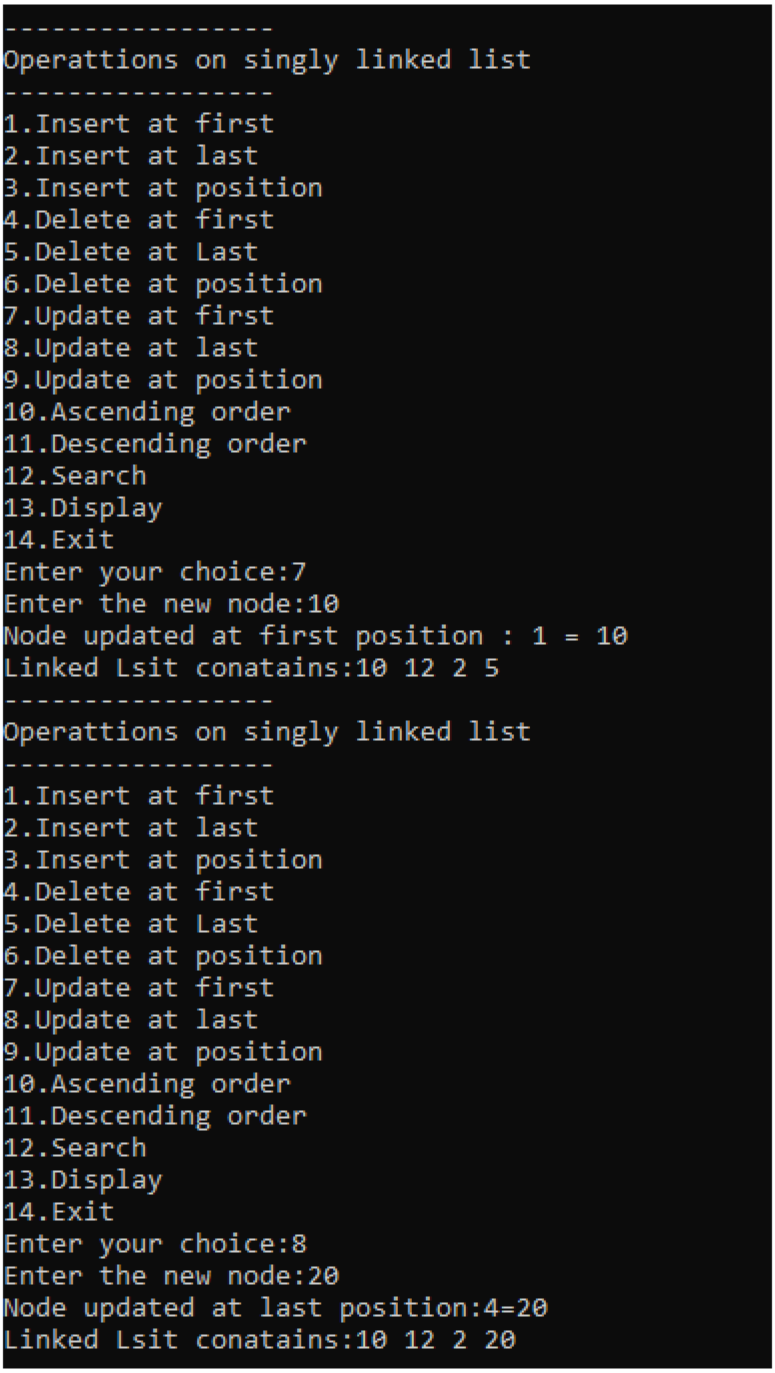
OUTPUT:



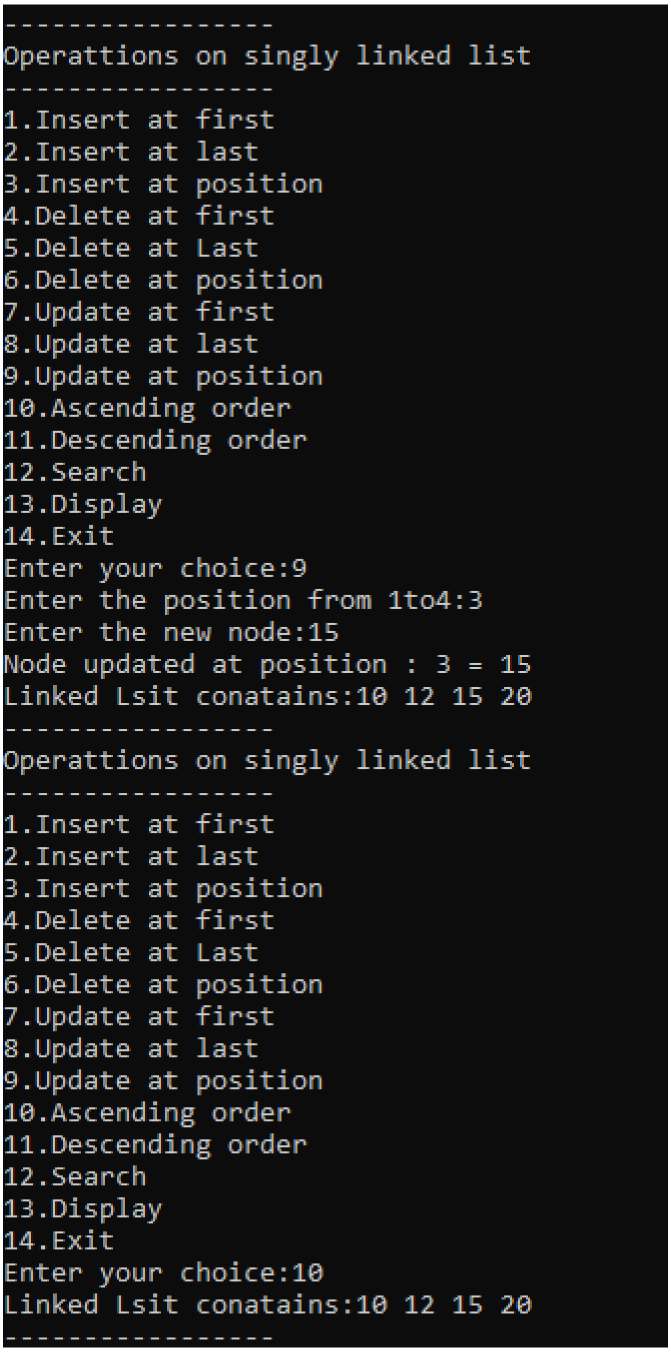
```
-----------------
Operattions on singly linked list
-----------------
1.Insert at first
2.Insert at last
3.Insert at position
4.Delete at first
5.Delete at Last
6.Delete at position
7.Update at first
8.Update at last
9.Update at position
10.Ascending order
11.Descending order
12.Search
13.Display
14.Exit
Enter your choice:1
Enter the value to be inserted:12
12is inserted at first in the empty list
Linked Lsit conatains:12
-----------------
Operattions on singly linked list
-----------------
1.Insert at first
2.Insert at last
3.Insert at position
4.Delete at first
5.Delete at Last
6.Delete at position
7.Update at first
8.Update at last
9.Update at position
10.Ascending order
11.Descending order
12.Search
13.Display
14.Exit
Enter your choice:1
Enter the value to be inserted:5
5is inserted at first
Linked Lsit conatains:5 12
-----------------
```



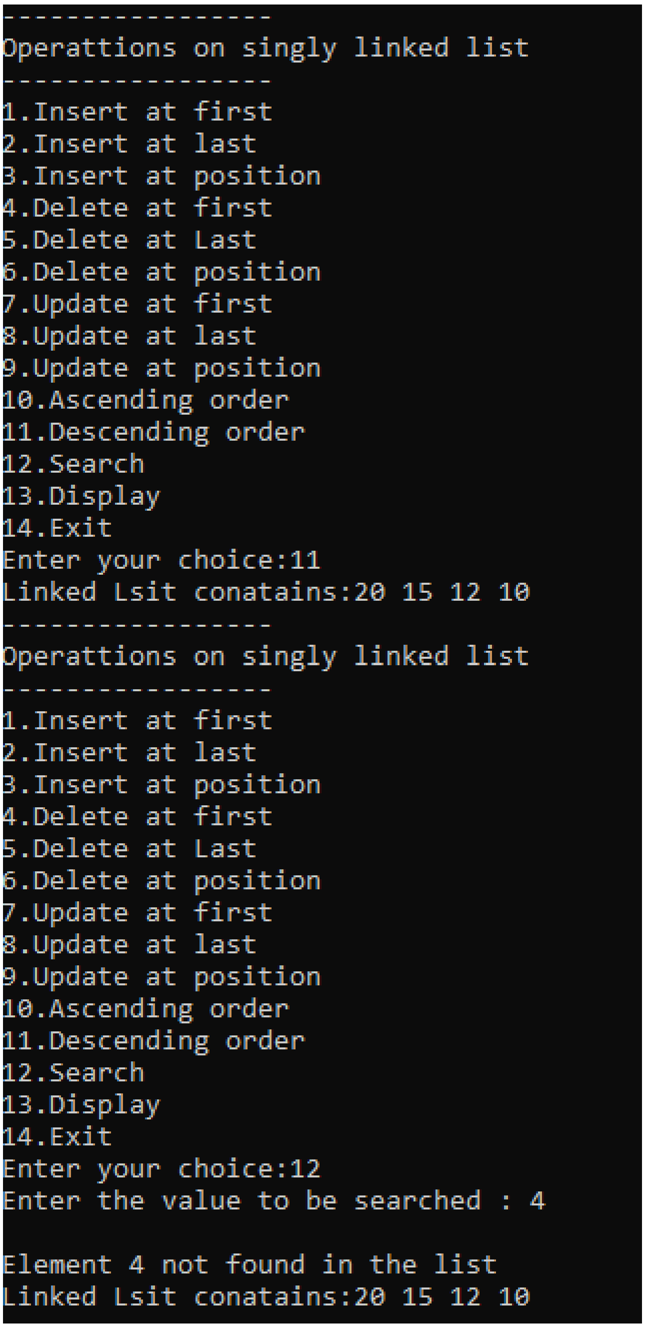
```
-----------------
Operattions on singly linked list
-----------------
1.Insert at first
2.Insert at last
3.Insert at position
4.Delete at first
5.Delete at Last
6.Delete at position
7.Update at first
8.Update at last
9.Update at position
10.Ascending order
11.Descending order
12.Search
13.Display
14.Exit
Enter your choice:2
Enter the value to be inserted:5
5is inserted at last
Linked Lsit conatains:5 12 5
-----------------
Operattions on singly linked list
-----------------
1.Insert at first
2.Insert at last
3.Insert at position
4.Delete at first
5.Delete at Last
6.Delete at position
7.Update at first
8.Update at last
9.Update at position
10.Ascending order
11.Descending order
12.Search
13.Display
14.Exit
Enter your choice:3
Enter the postion from 1 to 4 : 3
Enter the value to be inserted:2
2is inserted at position3
Linked Lsit conatains:5 12 2 5
-----------------
```



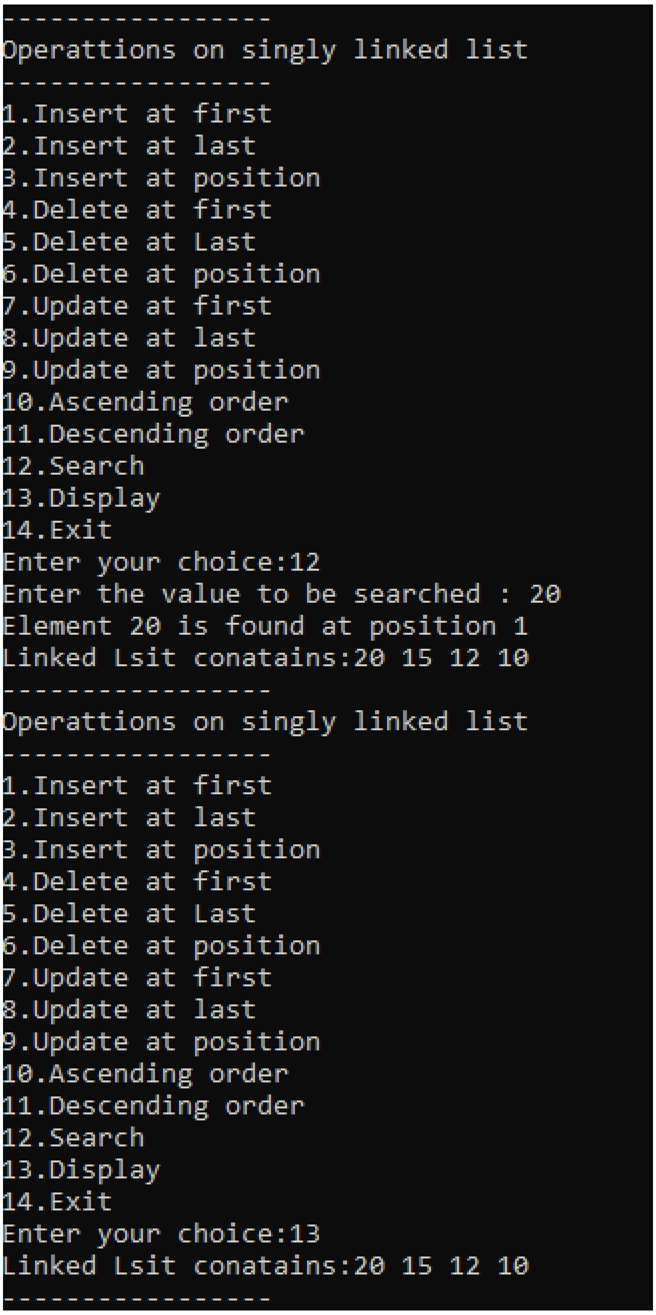
```
-----------------
Operattions on singly linked list
-----------------
1.Insert at first
2.Insert at last
3.Insert at position
4.Delete at first
5.Delete at Last
6.Delete at position
7.Update at first
8.Update at last
9.Update at position
10.Ascending order
11.Descending order
12.Search
13.Display
14.Exit
Enter your choice:7
Enter the new node:10
Node updated at first position : 1 = 10
Linked Lsit conatains:10 12 2 5
-----------------
Operattions on singly linked list
-----------------
1.Insert at first
2.Insert at last
3.Insert at position
4.Delete at first
5.Delete at Last
6.Delete at position
7.Update at first
8.Update at last
9.Update at position
10.Ascending order
11.Descending order
12.Search
13.Display
14.Exit
Enter your choice:8
Enter the new node:20
Node updated at last position:4=20
Linked Lsit conatains:10 12 2 20
```



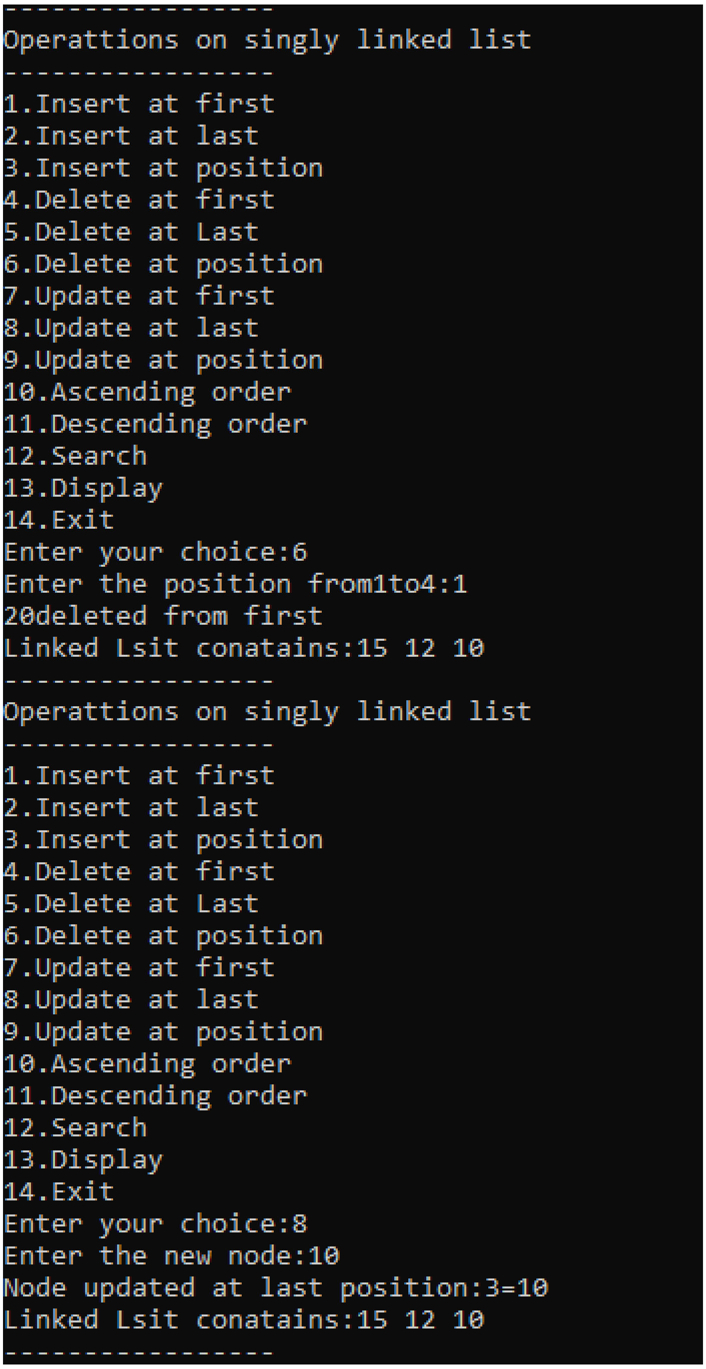
```
-----------------
Operattions on singly linked list
-----------------
1.Insert at first
2.Insert at last
3.Insert at position
4.Delete at first
5.Delete at Last
6.Delete at position
7.Update at first
8.Update at last
9.Update at position
10.Ascending order
11.Descending order
12.Search
13.Display
14.Exit
Enter your choice:9
Enter the position from 1to4:3
Enter the new node:15
Node updated at position : 3 = 15
Linked Lsit conatains:10 12 15 20
-----------------
Operattions on singly linked list
-----------------
1.Insert at first
2.Insert at last
3.Insert at position
4.Delete at first
5.Delete at Last
6.Delete at position
7.Update at first
8.Update at last
9.Update at position
10.Ascending order
11.Descending order
12.Search
13.Display
14.Exit
Enter your choice:10
Linked Lsit conatains:10 12 15 20
-----------------
```



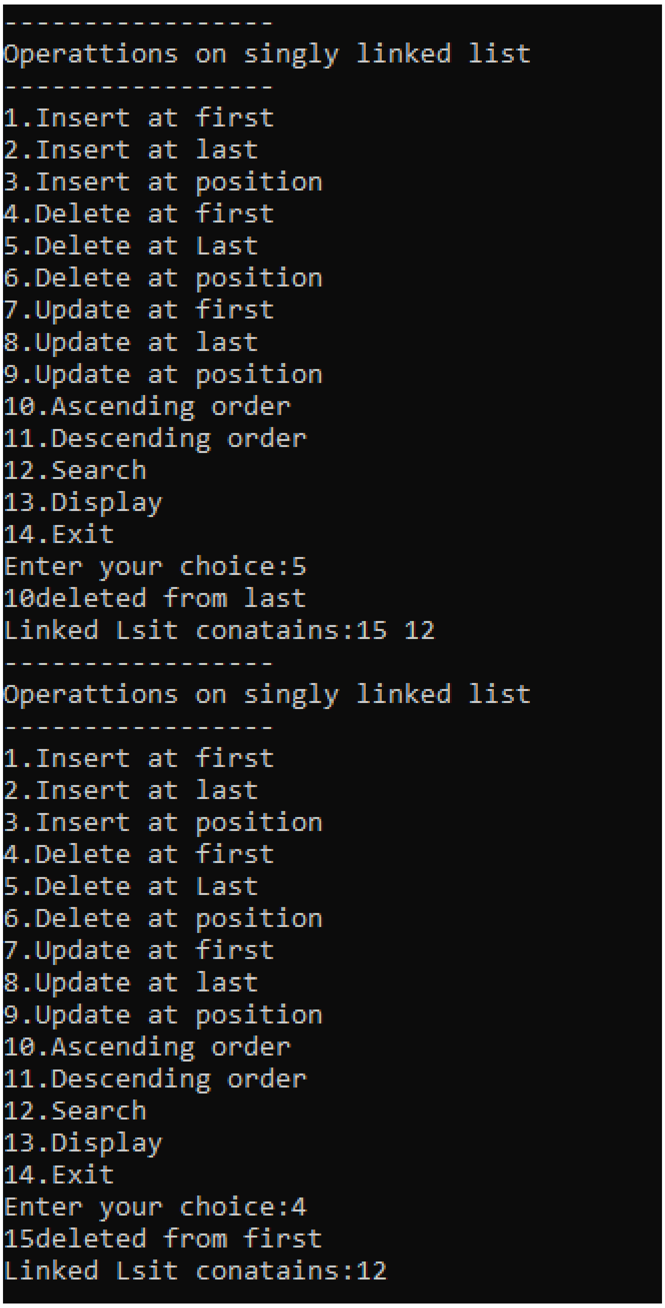
```
-----------------
Operattions on singly linked list
-----------------
1.Insert at first
2.Insert at last
3.Insert at position
4.Delete at first
5.Delete at Last
6.Delete at position
7.Update at first
8.Update at last
9.Update at position
10.Ascending order
11.Descending order
12.Search
13.Display
14.Exit
Enter your choice:11
Linked Lsit conatains:20 15 12 10
-----------------
Operattions on singly linked list
-----------------
1.Insert at first
2.Insert at last
3.Insert at position
4.Delete at first
5.Delete at Last
6.Delete at position
7.Update at first
8.Update at last
9.Update at position
10.Ascending order
11.Descending order
12.Search
13.Display
14.Exit
Enter your choice:12
Enter the value to be searched : 4

Element 4 not found in the list
Linked Lsit conatains:20 15 12 10
```



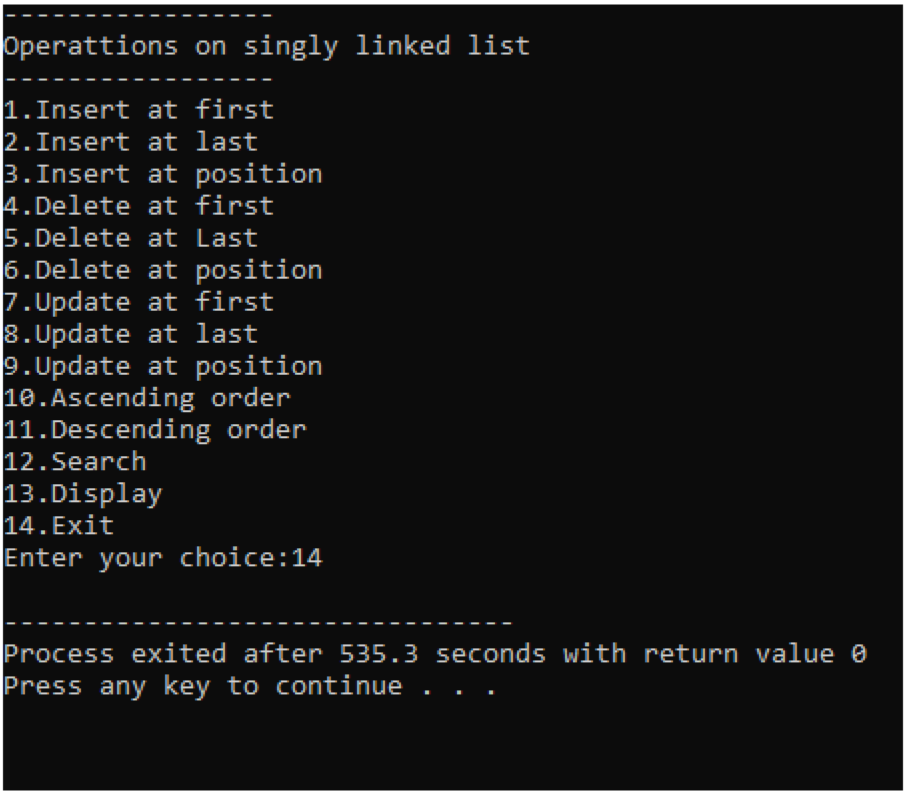
```
-----------------
Operattions on singly linked list
-----------------
1.Insert at first
2.Insert at last
3.Insert at position
4.Delete at first
5.Delete at Last
6.Delete at position
7.Update at first
8.Update at last
9.Update at position
10.Ascending order
11.Descending order
12.Search
13.Display
14.Exit
Enter your choice:12
Enter the value to be searched : 20
Element 20 is found at position 1
Linked Lsit conatains:20 15 12 10
-----------------
Operattions on singly linked list
-----------------
1.Insert at first
2.Insert at last
3.Insert at position
4.Delete at first
5.Delete at Last
6.Delete at position
7.Update at first
8.Update at last
9.Update at position
10.Ascending order
11.Descending order
12.Search
13.Display
14.Exit
Enter your choice:13
Linked Lsit conatains:20 15 12 10
-----------------
```



```
-----------------
Operattions on singly linked list
-----------------
1.Insert at first
2.Insert at last
3.Insert at position
4.Delete at first
5.Delete at Last
6.Delete at position
7.Update at first
8.Update at last
9.Update at position
10.Ascending order
11.Descending order
12.Search
13.Display
14.Exit
Enter your choice:6
Enter the position from1to4:1
20deleted from first
Linked Lsit conatains:15 12 10
-----------------
Operattions on singly linked list
-----------------
1.Insert at first
2.Insert at last
3.Insert at position
4.Delete at first
5.Delete at Last
6.Delete at position
7.Update at first
8.Update at last
9.Update at position
10.Ascending order
11.Descending order
12.Search
13.Display
14.Exit
Enter your choice:8
Enter the new node:10
Node updated at last position:3=10
Linked Lsit conatains:15 12 10
-----------------
```



```
-----------------
Operattions on singly linked list
-----------------
1.Insert at first
2.Insert at last
3.Insert at position
4.Delete at first
5.Delete at Last
6.Delete at position
7.Update at first
8.Update at last
9.Update at position
10.Ascending order
11.Descending order
12.Search
13.Display
14.Exit
Enter your choice:5
10deleted from last
Linked Lsit conatains:15 12
-----------------
Operattions on singly linked list
-----------------
1.Insert at first
2.Insert at last
3.Insert at position
4.Delete at first
5.Delete at Last
6.Delete at position
7.Update at first
8.Update at last
9.Update at position
10.Ascending order
11.Descending order
12.Search
13.Display
14.Exit
Enter your choice:4
15deleted from first
Linked Lsit conatains:12
```

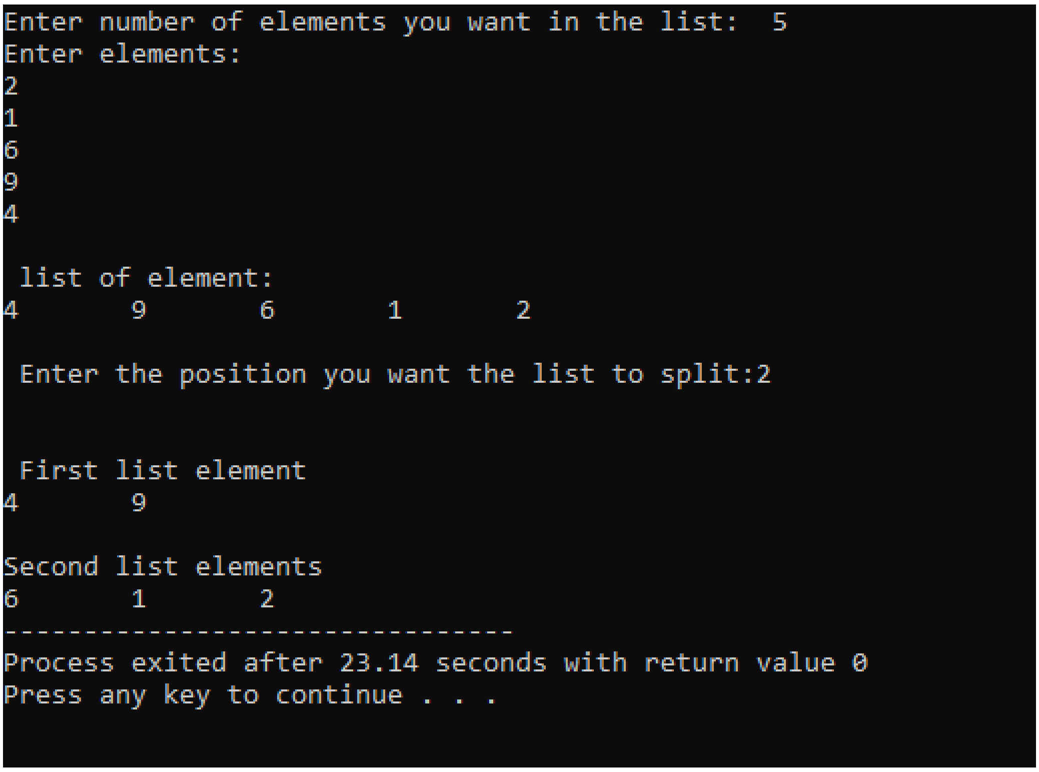


```
-----------------
Operattions on singly linked list
-----------------
1.Insert at first
2.Insert at last
3.Insert at position
4.Delete at first
5.Delete at Last
6.Delete at position
7.Update at first
8.Update at last
9.Update at position
10.Ascending order
11.Descending order
12.Search
13.Display
14.Exit
Enter your choice:14

--------------------------------
Process exited after 535.3 seconds with return value 0
Press any key to continue . . .
```

2. Write a C++ program to split the linked list into two halves such that the element ‘e’ should be the first element of second list.

```
#include<iostream>
using namespace std;
struct Node{
        int value;
        struct Node*next;
};
struct Node*head=NULL;
struct Node*sHead=NULL;
struct Node*temp=NULL;
void insert(int new_data){
        struct Node *new_node=new Node();
        new_node->value=new_data;
        new_node->next=head;
        head=new_node;
}
int n;
int ele;
int splitIndex;
int main(){
        int i;
        cout<<"Enter number of elements you want in the list:\t";
        cin>>n;
        cout<<"Enter elements:"<<endl;
        for(i=0;i<n;i++){
                cin>>ele;
                insert(ele);
        }
        cout<<"\n list of element:"<<endl;
        Node *t;
        t=head;
        while(t!=NULL){
                cout<<t->value<<"\t";
                t=t->next;
        }
        cout<<"\n\n Enter the position you want the list to split:";
        cin>>splitIndex;
        while(splitIndex<0 || splitIndex>n-1){
                cout<<"Invalid position.Try again."<<endl;
                cin>>splitIndex;
        }
        temp=head;
        for(i=0;i<=splitIndex;i++){
                if(i==splitIndex-1){
                        Node *tN;
                        tN=temp->next;
                        sHead=tN;
                        temp->next=NULL;
                        break;
                }
                temp=temp->next;
        }
        temp=head;
        if(temp==NULL){
                cout<<"\n FIrst list is empty"<<endl;
        }else{
                cout<<"\n \n First list element"<<endl;
                while(temp!=NULL){
                        cout<<temp->value<<"\t";
                        temp=temp->next;
                }
        }
        temp=sHead;
        if(temp==NULL){
        cout<<"\nSecond list is empty"<<endl;
        }else{
        cout<<"\n\nSecond list elements "<<endl;
        while(temp != NULL){
        cout<<temp->value<<"\t";
        temp = temp->next;
        }
}
return 0;
}
```

OUTPUT:



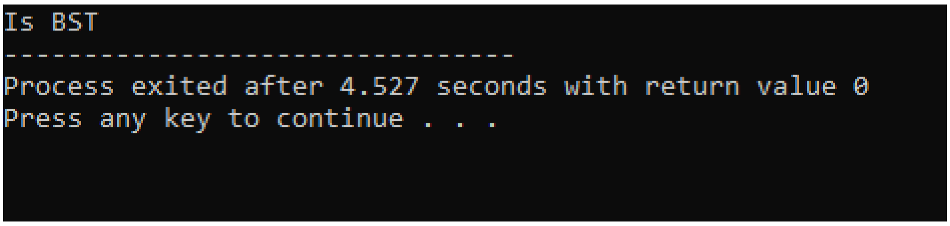
```
Enter number of elements you want in the list:  5
Enter elements:
2
1
6
9
4

 list of element:
4       9       6       1       2

 Enter the position you want the list to split:2


 First list element
4       9

Second list elements
6       1       2
--------------------------------
Process exited after 23.14 seconds with return value 0
Press any key to continue . . .
```

3. Write a C++ program to check if a binary tree is BST or not.

```
#include<bits/stdc++.h>
using namespace std;
struct Node
{
	int data;
	struct Node *left, *right;
	Node(int data)
	{
		this->data=data;
		left=right=NULL;
	}
};
bool isBSTUtil(struct Node*root, Node*&prev)
{
	if(root)
	{
		if(!isBSTUtil(root->left, prev))
		return false;
		if(prev !=NULL && root->data<=prev->data)
		return false;
		prev=root;
		return isBSTUtil(root->right, prev);
	}
	return true;
}
bool isBST(Node *root)
{
	Node *prev=NULL;
	return isBSTUtil(root,prev);
}
int main()
{

// struct Node *root=new Node(200);
// root->left=new Node(6);
//	root->left->right=new Node(80);
//	root->left->right->left=new Node(9);
//	root->left->right->right=new Node(100);
//	root->left->right->right->right=new Node(150);
//  root->left->right->left->left=new Node(7);
//	root->left->right->left->right=new Node(30);
//	root->left->right->left->right->left=new Node(17);
//	root->left->right->left->right->right=new Node(65);
//	root->left->right->left->right->right->left=new Node(58);

	struct Node *root = new Node(7);
	root->left = new Node(5);
	root->right = new Node(8);
	root->left->left = new Node(3);
```

```
            root->left->right = new Node(6);
            if(isBST(root))
            cout<<"Is BST";
            else
            cout<<"Not a BST";
            return 0;
}
```

OUTPUT:



```
Is BST
--------------------------------
Process exited after 4.527 seconds with return value 0
Press any key to continue . . .
```

4. Construct a binary search tree (BST) to support the following operations.

Given a key, perform a search in the BST. If the key is found then display “key found”.

- **Insert an element into a binary search tree.**
- **Delete an element from a binary search tree.**

Display the tree using inorder, preorder and post order traversal methods(a).

```
#include<iostream>
#include<cstdlib>
using namespace std;
struct node
{
        int info;
        struct node*left;
        struct node*right;
}*root;
class BST
{
        public:
        void find(int, node **,node **);
        void insert(node *,node *);
        void del(int);
        void case_a(node *,node *);
        void case_b(node *,node *);
        void case_c(node *,node *);
        void preorder(node *);
        void inorder(node *);
        void postorder(node *);
        void display(node *,int);
        BST()
        {
                root=NULL;
        }
};
int main()
{
        int choice,num;
        BST bst;
        node *temp;
        while(1)
        {
                cout<<"------"<<endl;
                cout<<"Operations on BST"<<endl;
                cout<<"------"<<endl;
                cout<<"1.Insert Element"<<endl;
                cout<<"2.Delete Element "<<endl;
                cout<<"3.Inorder Traversal"<<endl;
                cout<<"4.Preorder Traversal"<<endl;
                cout<<"5.Postorder Traversal"<<endl;
                cout<<"6.Display"<<endl;
                cout<<"7.Quit"<<endl;
                cout<<"Enter your choice:";
                cin>>choice;
                switch(choice)
                {
                        case 1:
                                temp=new node;
                                cout<<"Enter the number to be inserted:";
                                cin>>temp->info;
                                bst.insert(root,temp);
                                break;
                        case 2:
                                if(root==NULL)
                                {
                                        cout<<"Tree is empty,nothing to delete"<<endl;
                                        continue;
                                 }
                                cout<<"Enter the number to be deleted:";
                                cin>>num;
                                bst.del(num);
                                break;
                        case 3:
                                cout<<"Inorder Traversal of BST:"<<endl;
                                bst.inorder(root);
                                cout<<endl;
                                break;
                        case 4:
                                cout<<"Preorder Traversal of BST:"<<endl;
                                bst.preorder(root);
                                cout<<endl;
                                break;
                        case 5:
                                cout<<"Postorder Traversal of BST:"<<endl;
                                bst.postorder(root);
                                cout<<endl;
                                break;
                        case 6:
                                cout<<"Display BST:"<<endl;
                                bst.display(root,1);
                                cout<<endl;
                                break;
                        case 7:
                                exit(1);
                        default:
                                cout<<"Wrong choice"<<endl;
                }
        }
}
void BST::find(int item, node **par,node **loc)
{
        node *ptr, *ptrsave;
```

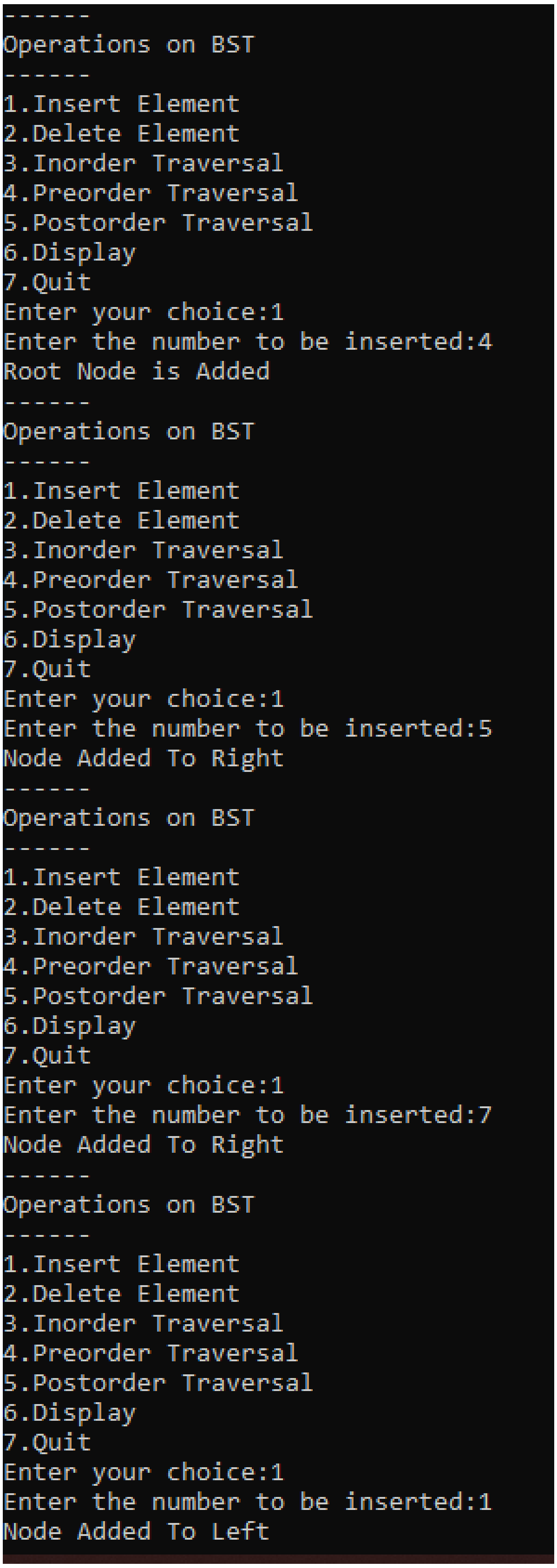
```
    if(root==NULL)
    {
        *loc=NULL;
        *par=NULL;
        return;
    }
    if(item==root->info)
    {
        *loc=root;
        *par=NULL;
        return;
    }
    if(item<root->info)
    ptr=root->left;
    else
    ptr=root->right;
    ptrsave=root;
    while(ptr!=NULL)
    {
        if(item==ptr->info)
        {
            *loc=ptr;
            *par=ptrsave;
            return;
        }
        ptrsave = ptr;
        if (item < ptr->info)
        ptr = ptr->left;
        else
        ptr = ptr->right;
    }
*loc=NULL;
*par=ptrsave;
}
void BST::insert(node *tree,node*newnode)
{
    if(root==NULL)
    {
        root=new node;
        root->info=newnode->info;
        root->left=NULL;
        root->right=NULL;
        cout<<"Root Node is Added"<<endl;
        return;
    }
    if(tree->info==newnode->info)
    {
        cout<<"Element already in the tree"<<endl;
        return;
    }
```

```
        if(tree->info>newnode->info)
        {
                if(tree->left!=NULL)
                {
                        insert(tree->left,newnode);
                }
                else
                {
                        tree->left=newnode;
                        (tree->left)->left=NULL;
                        (tree->left)->right=NULL;
                        cout<<"Node Added To Left"<<endl;
                        return;
                }
        }
        else
        {
                if(tree->right!=NULL)
                {
                        insert(tree->right,newnode);
                }
                else
                {
                tree->right = newnode;
                (tree->right)->left = NULL;
                (tree->right)->right = NULL;
                cout<<"Node Added To Right"<<endl;
                return;
                }
        }
}
void BST::del(int item)
{
        node *parent,*location;
        if(root==NULL)
        {
                cout<<"Tree empty"<<endl;
                return;
        }
        find(item, &parent,&location);
        if(location==NULL)
        {
                cout<<"Item not present in tree"<<endl;
                return;
        }
        if(location->left==NULL && location->right==NULL)
        case_a(parent,location);
        if (location->left != NULL && location->right == NULL)
        case_b(parent, location);
        if (location->left == NULL && location->right != NULL)
```

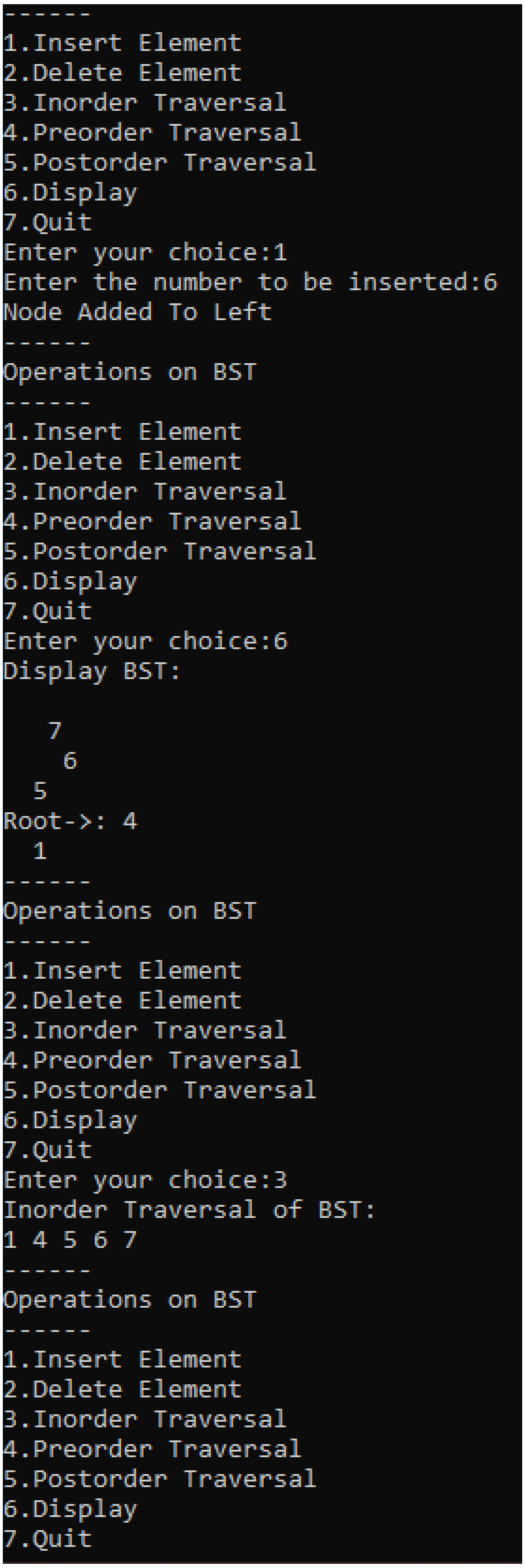
```
    case_b(parent, location);
    if (location->left != NULL && location->right != NULL)
    case_c(parent, location);
    free(location);
}

void BST::case_a(node *par, node *loc )
{
    if (par == NULL)
    {
        root = NULL;
    }
    else
    {
        if (loc == par->left)
        par->left = NULL;
    else
        par->right = NULL;
    }
}

void BST::case_b(node *par, node *loc)
{
    node *child;
    if (loc->left != NULL)
    child = loc->left;
 else
    child = loc->right;
    if(par==NULL)
    {
        root=child;
    }
    else
    {
        if(loc==par->left)
        par->left=child;
        else
        par->right=child;
    }
}

void BST::case_c(node *par, node *loc)
{
    node *ptr, *ptrsave, *suc, *parsuc;
    ptrsave = loc;
    ptr = loc->right;
    while (ptr->left != NULL)
    {
        ptrsave = ptr;
        ptr = ptr->left;
    }
    suc = ptr;
    parsuc = ptrsave;
    if (suc->left == NULL && suc->right == NULL)
    case_a(parsuc, suc);
    else
    case_b(parsuc, suc);
    if(par==NULL)
    {
        root=suc;
    }
    else
    {
        if(loc==par->left)
        par->left=suc;
        else
        par->right=suc;
    }
    suc->left=loc->left;
    suc->right=loc->right;
}

void BST::preorder(node *ptr)
{
    if (root == NULL)
    {
        cout<<"Tree is empty"<<endl;
        return;
    }
    if (ptr != NULL)
    {
        cout<<ptr->info<<" ";
        preorder(ptr->left);
        preorder(ptr->right);
    }
 }

void BST::inorder(node *ptr)
 {
    if(root==NULL)
    {
        cout<<"Tree is empty"<<endl;
        return;
 }
 if(ptr!=NULL)
 {
        inorder(ptr->left);
        cout<<ptr->info<<" ";
        inorder(ptr->right);
 }
```

```
}

void BST::postorder(node *ptr)
{
        if(root==NULL)
        {
                cout<<"Tree is empty"<<endl;
                return;
        }
        if(ptr!=NULL)
        {
                postorder(ptr->left);
                postorder(ptr->right);
                cout<<ptr->info<<" ";
        }
}

void BST::display(node *ptr, int level)
{
        int i;
        if (ptr != NULL)
        {
                display(ptr->right, level+1);
                cout<<endl;
                if (ptr == root)
                cout<<"Root->: ";
                 else
                {
                        for (i = 0;i < level;i++)
                        cout<<" ";
                }
                cout<<ptr->info;
                display(ptr->left, level+1);
        }
}
```

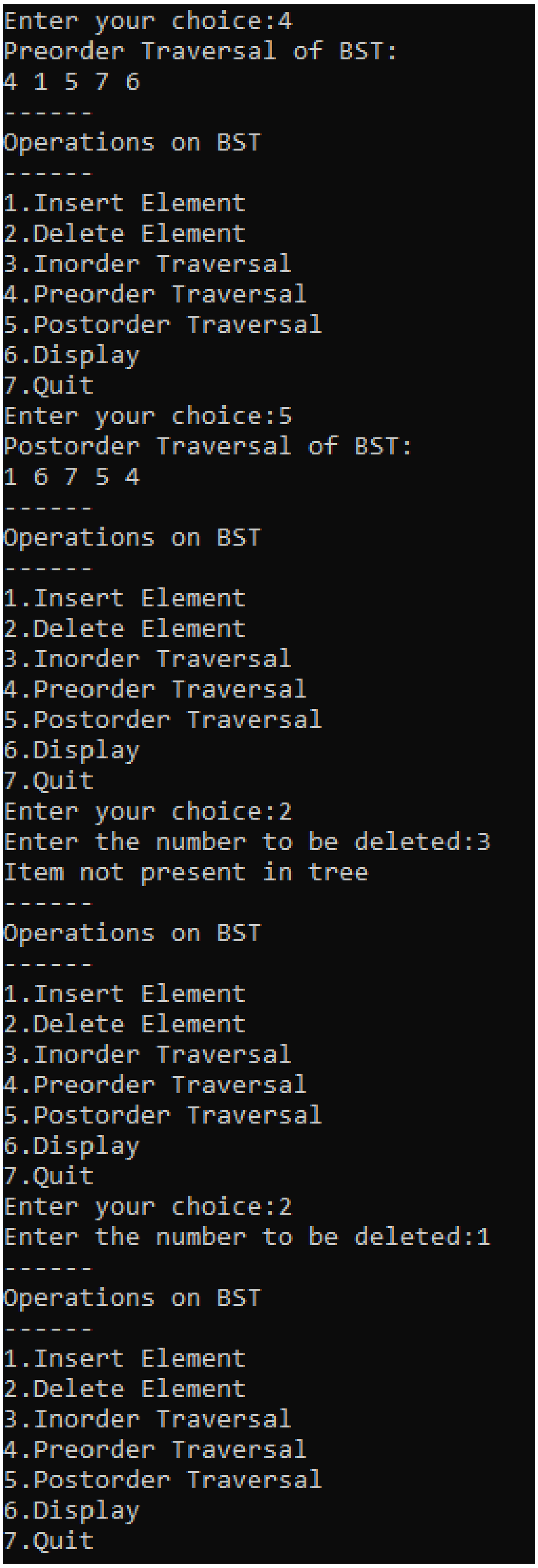
OUTPUT:



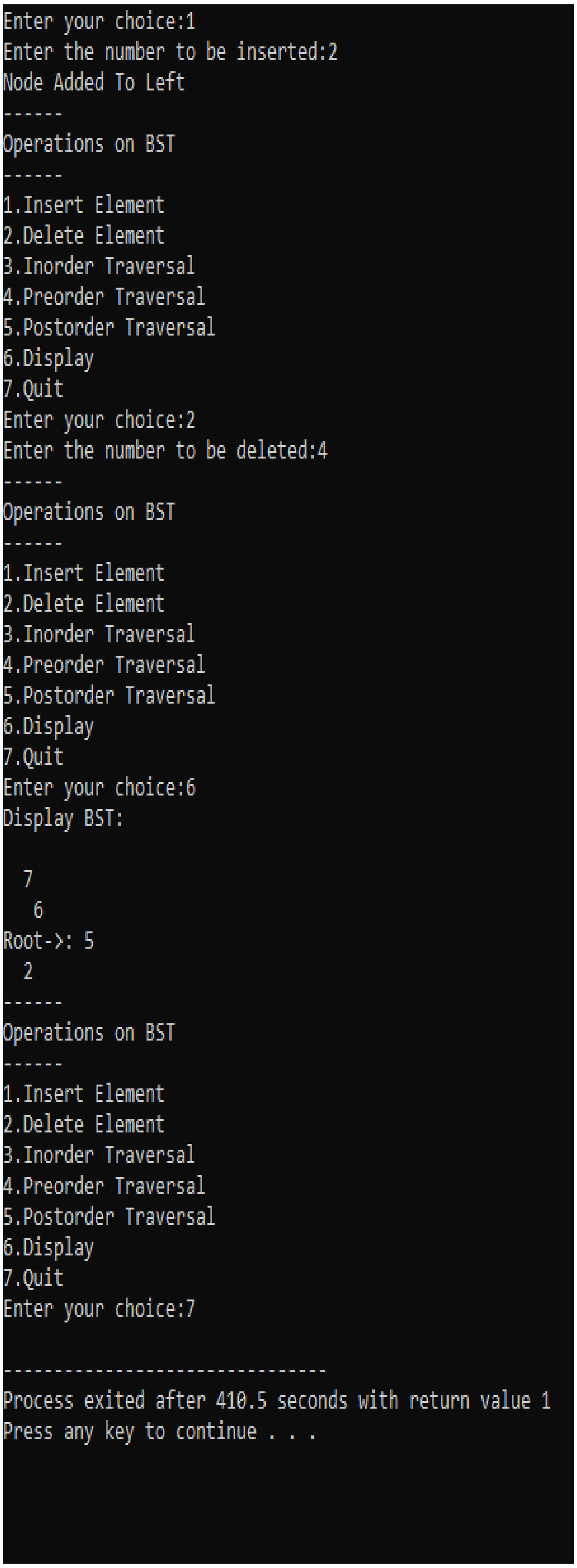
```
------
Operations on BST
------
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
Enter your choice:1
Enter the number to be inserted:4
Root Node is Added
------
Operations on BST
------
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
Enter your choice:1
Enter the number to be inserted:5
Node Added To Right
------
Operations on BST
------
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
Enter your choice:1
Enter the number to be inserted:7
Node Added To Right
------
Operations on BST
------
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
Enter your choice:1
Enter the number to be inserted:1
Node Added To Left
```



```
------
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
Enter your choice:1
Enter the number to be inserted:6
Node Added To Left
------
Operations on BST
------
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
Enter your choice:6
Display BST:

   7
    6
  5
Root->: 4
  1
------
Operations on BST
------
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
Enter your choice:3
Inorder Traversal of BST:
1 4 5 6 7
------
Operations on BST
------
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
```



```
Enter your choice:4
Preorder Traversal of BST:
4 1 5 7 6
------
Operations on BST
------
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
Enter your choice:5
Postorder Traversal of BST:
1 6 7 5 4
------
Operations on BST
------
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
Enter your choice:2
Enter the number to be deleted:3
Item not present in tree
------
Operations on BST
------
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
Enter your choice:2
Enter the number to be deleted:1
------
Operations on BST
------
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
```

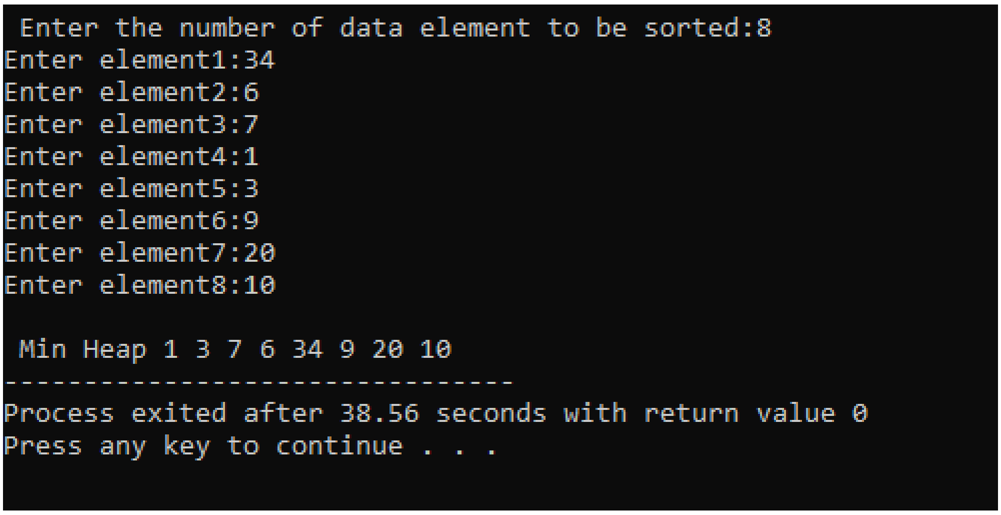


```
Enter your choice:1
Enter the number to be inserted:2
Node Added To Left
------
Operations on BST
------
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
Enter your choice:2
Enter the number to be deleted:4
------
Operations on BST
------
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
Enter your choice:6
Display BST:

  7
   6
Root->: 5
  2
------
Operations on BST
------
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
Enter your choice:7

--------------------------------
Process exited after 410.5 seconds with return value 1
Press any key to continue . . .
```

5. Write a program to implement Minheap.

```
#include<iostream>
using namespace std;
void MinHeapify(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 Build_MinHeap(int a[],int n)
{
	int i;
	for(i=n/2;i>=1;i--)
	MinHeapify(a,i,n);
}
int main()
{
	int n,i,arr[100];
	cout<<"\n Enter the number of data element to be sorted:";
	cin>>n;
	n++;
	for(i=1;i<n;i++)
	{
		cout<<"Enter element"<<i<<":";
		cin>>arr[i];
	}
	Build_MinHeap(arr,n-1);
	cout<<"\n Min Heap";
	for(i=1;i<n;i++)
	cout<<" "<<arr[i];
	return 0;

}
```

OUTPUT:

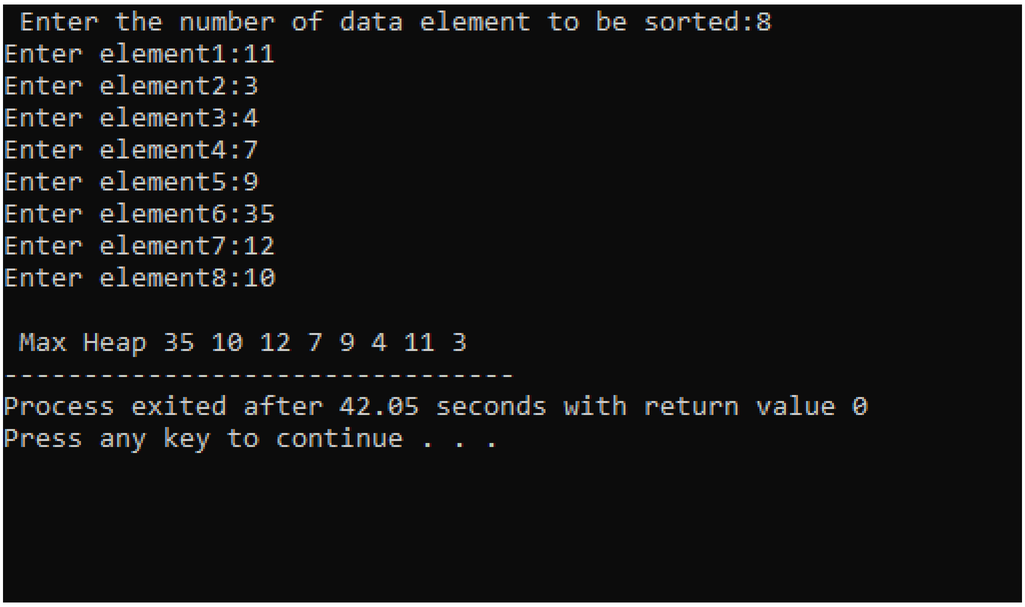


```
 Enter the number of data element to be sorted:8
Enter element1:34
Enter element2:6
Enter element3:7
Enter element4:1
Enter element5:3
Enter element6:9
Enter element7:20
Enter element8:10

 Min Heap 1 3 7 6 34 9 20 10
--------------------------------
Process exited after 38.56 seconds with return value 0
Press any key to continue . . .
```

6. Write a program to implement Max heap.

```
#include<iostream>
using namespace std;
void MaxHeapify(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 Build_MaxHeap(int a[],int n)
{
	int i;
	for(i=n/2;i>=1;i--)
	MaxHeapify(a,i,n);
}
int main()
{
	int n,i,arr[100];
	cout<<"\n Enter the number of data element to be sorted:";
	cin>>n;
	n++;
	for(i=1;i<n;i++)
	{
		cout<<"Enter element"<<i<<":";
		cin>>arr[i];
	}
	Build_MaxHeap(arr,n-1);
	cout<<"\n Max Heap";
	for(i=1;i<n;i++)
	cout<<" "<<arr[i];
	return 0;

}
```

OUTPUT:



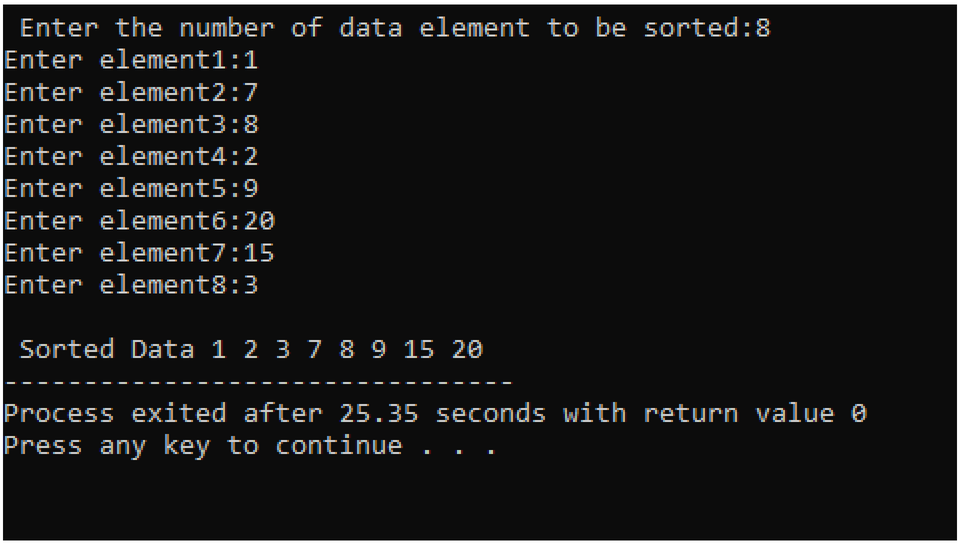
```
 Enter the number of data element to be sorted:8
Enter element1:11
Enter element2:3
Enter element3:4
Enter element4:7
Enter element5:9
Enter element6:35
Enter element7:12
Enter element8:10

 Max Heap 35 10 12 7 9 4 11 3
--------------------------------
Process exited after 42.05 seconds with return value 0
Press any key to continue . . .
```

7. Write a C++ program for implementing Heap sort technique.

```
#include<iostream>
using namespace std;
void MaxHeapify(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;
		MaxHeapify(a,1,i-1);
	}
}
void Build_MaxHeap(int a[],int n)
{
	int i;
	for(i=n/2;i>=1;i--)
	MaxHeapify(a,i,n);
}
int main()
{
	int n,i,arr[100];
	cout<<"\n Enter the number of data element to be sorted:";
	cin>>n;
	n++;
	for(i=1;i<n;i++)
	{
		cout<<"Enter element"<<i<<":";
		cin>>arr[i];
```

```
        }
        Build_MaxHeap(arr,n-1);
        HeapSort(arr,n-1);
        cout<<"\n Sorted Data";
        for(i=1;i<n;i++)
        cout<<" "<<arr[i];
        return 0;

}
```

OUTPUT:

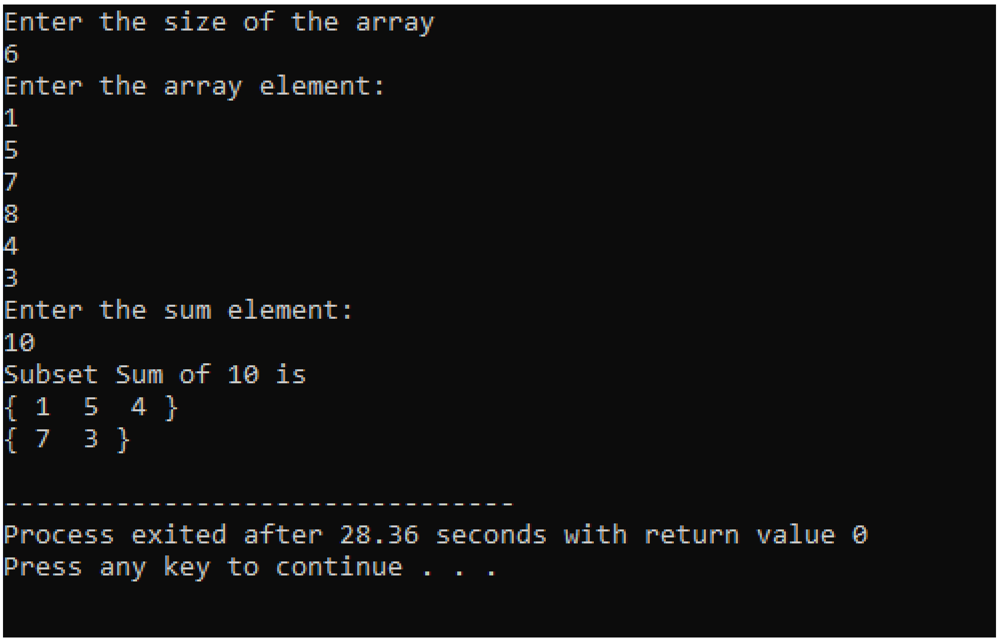


```
 Enter the number of data element to be sorted:8
Enter element1:1
Enter element2:7
Enter element3:8
Enter element4:2
Enter element5:9
Enter element6:20
Enter element7:15
Enter element8:3

 Sorted Data 1 2 3 7 8 9 15 20
--------------------------------
Process exited after 25.35 seconds with return value 0
Press any key to continue . . .
```

8. C++ program to implement sum of subsets using backtracking.

```
#include <iostream>
#include<limits.h>
using namespace std;
class Subset
{
    public:
  void printSum(int result[], int front, int tail)
  {
    cout << "{";
        for (int i = front; i < tail; ++i)
        {
            if (result[i] != INT_MAX)
            {
                cout << " " << result[i] << " ";
            }
        }
        cout << "}\n";
    }
    void subsetSum(int arr[], int result[], int sum, int size, int current_sum, int location)
    {

        if (location == -1)
        {
            return;
        }
        this->subsetSum(arr, result, sum, size, current_sum, location - 1);
        result[location] = arr[location];
        if (current_sum + arr[location] == sum)
        {
            this->printSum(result, location, size);
        }
        this->subsetSum(arr, result, sum, size, current_sum + arr[location], location -
1);
        result[location] = INT_MAX;
    }
    void findSubset(int arr[], int size, int sum)
    {
        if (size <= 0)
        {
            return;
        }
        int result[size];
        for (int i = 0; i < size; ++i)
        {
            result[i] = INT_MAX;
        }
        cout << "Subset Sum of " << sum << " is \n";
        this->subsetSum(arr, result, sum, size, 0, size - 1);
    }
};
int main()
{
    Subset task = Subset();
    int n;
    cout<<"Enter the size of the array\n";
    cin>>n;
    int arr[n]={};
    cout<<"Enter the array element:"<<endl;
    for(int i=0;i<n;i++)
    {
        cin>>arr[i];
    }
    int size = sizeof(arr) / sizeof(arr[0]);
int sum;
cout<<"Enter the sum element: "<<endl;
cin>>sum;
task.findSubset(arr, size, sum);
return 0;
}
```

OUTPUT:

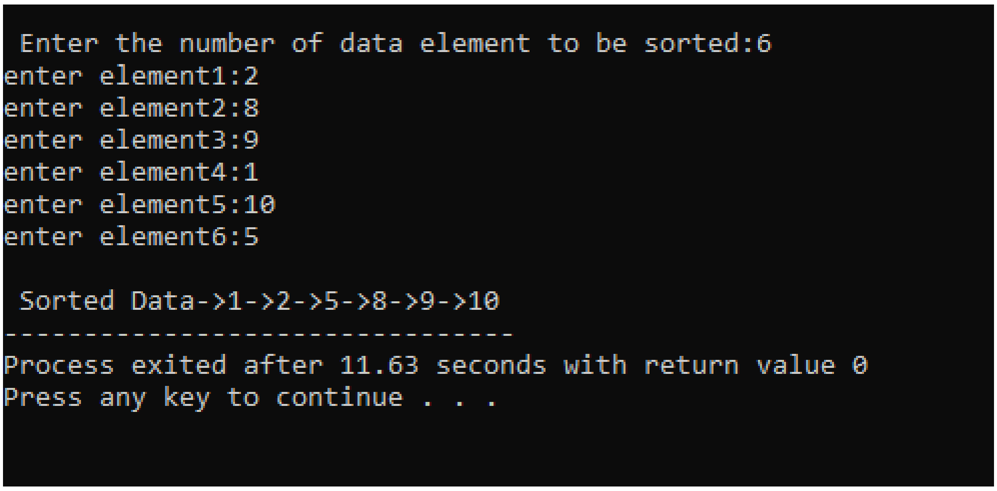


```
Enter the size of the array
6
Enter the array element:
1
5
7
8
4
3
Enter the sum element:
10
Subset Sum of 10 is
{ 1  5  4 }
{ 7  3 }

--------------------------------
Process exited after 28.36 seconds with return value 0
Press any key to continue . . .
```

9. Write a C++ program to implement merge sort technique using divide and conquer method.

```
#include<iostream>
using namespace std;
void Merge(int *a,int low, int high,int mid)
{
        int i,j,k,temp[high-low+1];
        i=low;
        k=0;
        j=mid+1;
        while(i<=mid && j<=high)
        {
                if(a[i]<a[j])
                {
                        temp[k]=a[i];
                        k++;
                        i++;
                }
                else
                {
                        temp[k]=a[j];
                        k++;
                        j++;
                }
        }
        while(i<=mid)
        {
                temp[k]=a[i];
                k++;
                i++;
        }
        while(j<=high)
        {
                temp[k]=a[j];
                k++;
                j++;
        }
        for(i=low;i<=high;i++)
        {
                a[i]=temp[i-low];
        }
}
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);
        }
}
int main()
{
        int n,i;
        cout<<"\n Enter the number of data element to be sorted:";
        cin>>n;
        int arr[n];
        for(i=0;i<n;i++)
        {
                cout<<"enter element"<<i+1<<":";
                cin>>arr[i];
        }
        MergeSort(arr,0,n-1);
        cout<<"\n Sorted Data";
        for(i=0;i<n;i++)
        cout<<"->"<<arr[i];
        return(0);
}
```

OUTPUT:



```
 Enter the number of data element to be sorted:6
enter element1:2
enter element2:8
enter element3:9
enter element4:1
enter element5:10
enter element6:5

 Sorted Data->1->2->5->8->9->10
--------------------------------
Process exited after 11.63 seconds with return value 0
Press any key to continue . . .
```