Practical NO: 01

### Implementation of different sorting techniques.

1. **Bubble Sort Program**

#include <iostream> using namespace std; int main()

{

int k,s,x,flag=0,mid ,temp ;

cout << "Enter size of Array : "; cin>>s;

int a[s];

cout << "Enter Numbers :" << endl; for(int i=0;i<s;i++)

{

cin>>a[i];

}

for(int i=0;i<s;i++)

{

for(int j=0;j<s-i-1;j++)

{

if(a[j]>a[j+1])

{

temp=a[j]; a[j]=a[j+1]; a[j+1]=temp;

}

}

cout<<"\n\n pass"<<i+1<<":"<<" "; for(k=0;k<s;k++)

{

cout<<a[k]<<" ";

}

}

cout<<endl; cout<<endl;

cout<<" Sorted Array is :"<<endl;

for(int i=0;i<s;i++)

{

cout<<a[i]<<"\t"; cout<<" ";

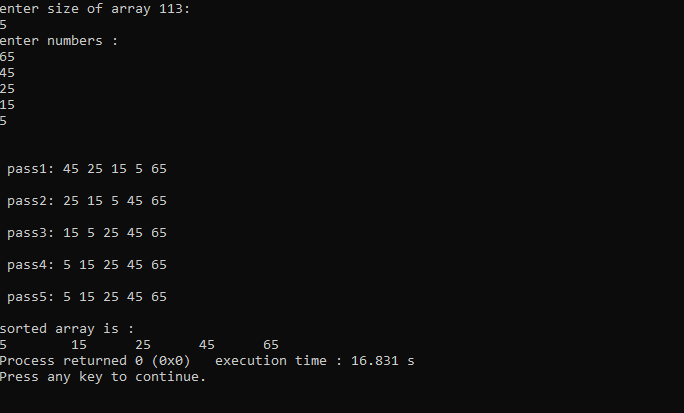
}

cout<<"\n";

return 0;

}

## Output:



### Insertion Sort Program

#include <iostream> using namespace std;

int main() {

int index; int smallest; int temp; int j,k;

cout<<"How many elements do you want "; cin>>index;

int number\_list [index]; for (int i=0;i<index;i++)

{

cin>>number\_list[i];

}

for(int i=1;i<index;i++)

{

temp=number\_list[i]; j=i-1;

while(j>=0&&temp<number\_list[j]){

number\_list[j+1]=number\_list[j]; j=j-1;

}

number\_list[j+1]=temp;

cout<<"\n\n pass"<<i+1<<":"<<" "; for(k=0;k<index;k++)

{

cout<<number\_list[k]<<" ";

}

}

cout<<endl; cout<<endl;

cout<<" Sorted Array is :"<<endl; for(int i=0;i<index;i++)

{

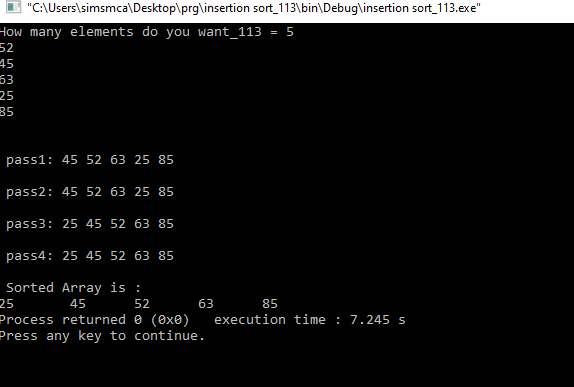
cout<<number\_list[i]<<"\t"; cout<<" ";

}

return 0;

}

### Output:



1. **Selection Sort Program**

#include <iostream> using namespace std; int main()

{

int k,s,x,flag=0,mid ,temp ;

cout << "Enter size of Array -->: "; cin>>s;

int a[s];

cout << "Enter Numbers :" << endl; for(int i=0;i<s;i++)

{

cin>>a[i];

}

for(int i=0;i<s;i++)

{

int min=i;

for(int j=i+1;j<s;j++)

{

if(a[min]>a[j])

{

min=j;

}

}

if(i!=min)

{

int temp; temp=a[i]; a[i]=a[min]; a[min]=temp;

}

cout<<"\n\n pass"<<i+1<<":"<<" "; for(k=0;k<s;k++)

{

cout<<a[k]<<" ";

}

}

cout<<endl; cout<<endl;

cout<<" Sorted Array is :"<<endl;

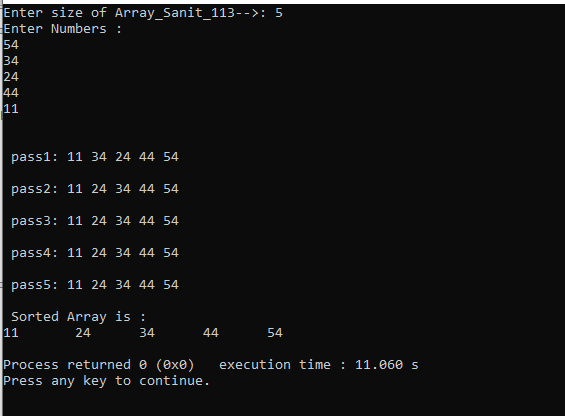
for(int i=0;i<s;i++){ cout<<a[i]<<"\t"; cout<<" ";

}

cout<<"\n"; return 0;

}

## Output:



### Shell Sort Program

#include <iostream> #include<math.h> using namespace std; int main()

{

int i,s,a[10],flag ,temp,k ; cout << "Enter size of Array "; cin>>s;

cout << "Enter Numbers :" << endl; for(i=0;i<s;i++)

{

cin>>a[i];

}

for(i=floor(s/2); i>0; i=floor(i/2))

{

flag=1; while(flag==1)

{

for( k=0; k<s-i; k++)

{

flag=0; if(a[k]>a[i+k])

{

temp=a[k]; a[k]=a[i+k]; a[i+k]=temp; flag=1;

}

}

}

}

cout<<" Sorted Array is :"<<endl; for(i=0;i<s;i++)

{

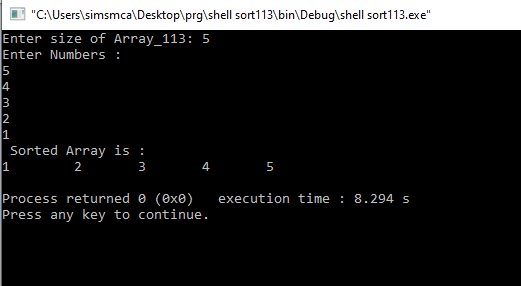
cout<<a[i]<<"\t"; cout<<" ";

}

cout<<"\n"; return 0;

}

### Output:



1. **Radix Sort Program**

#include<iostream> using namespace std;

int getMax(int arr[],int n)

{

int max=arr[0]; for(int i=1; i<n; i++)

if(arr[i]>max) max=arr[i];

return max;

}

void countSort(int arr[],int n,int exp)

{

int output[n],i,count[10]= {0}; for(i=0; i<n; i++)

count[(arr[i]/exp)%10]++; for(i=1; i<10; i++)

count[i]+=count[i-1]; for(i=n-1; i>=0; i--)

{

output[count[(arr[i]/exp)%10]-1]=arr[i]; count[(arr[i]/exp)%10]--;

}

for(i=0; i<n; i++) arr[i]=output[i];

}

void radixsort(int arr[],int n)

{

int exp,m; m=getMax(arr,n);

for(exp=1; m/exp>0; exp\*=10) countSort(arr,n,exp);

}

int main()

{

int n,i;

cout<< " FYMCA Roll No:100 DIV :- B";

cout<<"\nEnter 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];

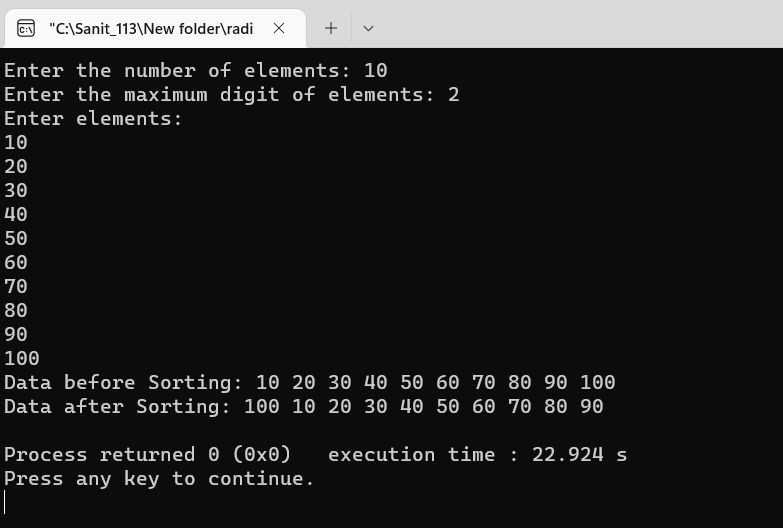
}

radixsort(arr,n); cout<<"\nSorted Data "; for(i=0; i<n; i++)

cout<<"->"<<arr[i]; return 0;

}

**Output:**



# Practical 2

### Implementation of Searching Algorithm and Hashing

1. **Linear Search Program**

#include<iostream> using namespace std;

main ()

{

int a[20],i,n,x,flag=0;

cout<<"how many elements do you want"; cin>>n;

cout<<"enter elements"; for(i=0; i<n; i++)

{

cin>>a[i];

}

cout<<"enter the element do you want to search"; cin>>x;

for(i=0; i<n; i++)

{

if(a[i]==x)

{

flag =1; break;

}

}

if(flag ==1)

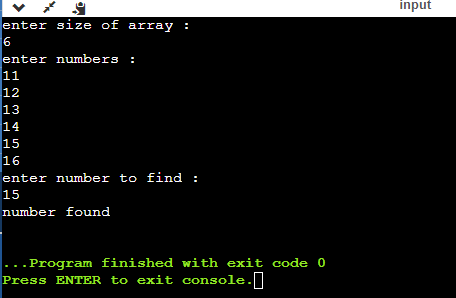
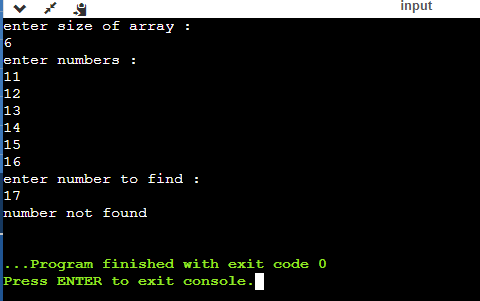
else

}

cout<<"element found";

cout<<"element not found";

**Output:**

* Case 1: Element Found
  + Case 2: Element Not Found

**Data Structure**

### Binary Search Program

#include<iostream> using namespace std;

main ()

{

int a[20],n,mid,x,lower,upper,flag=0;

cout<<"how many elements do you want"; cin>>n;

cout<<"enter elements"; for(int i=0; i<n; i++)

{

cin>>a[i];

}

cout<<"enter the element do you want to search"; cin>>x;

while(lower <= upper)

{

mid=(lower + upper)/2; if(x==a[mid]){

flag=1; break;

}

if(x>a[mid])

lower=mid + 1; else

upper=mid -1;

if(flag ==1) cout<<"element found";

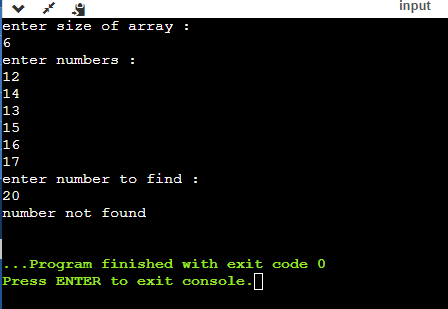
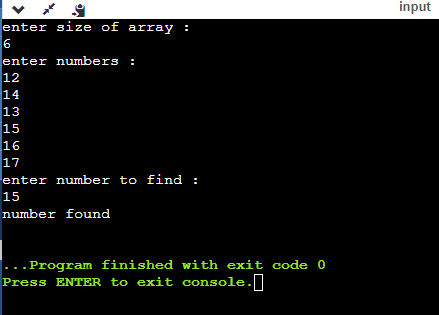
else

cout<<"element not found";

}

}

## Output:



### program to implement various method of hashing

#include<iostream> #include<conio.h> #include<stdio.h> #include<iomanip> using namespace std; const int Size=10; static int coll; class hashing

{

long key; long index; long arr[10];

public:

void directHash(); void subHash(); void modDivision(); void linProbe(); void digitExHash();

void foldShiftHash(); void foldBoundHash(); void display();

void display1();

};

void hashing::modDivision()

{

for(int i=0; i<10; i++) arr[i]=-1;

for(int i=0; i<10; i++)

{

int x;

cout<<"\nEnter a number:"; cin>>x;

index=x%10; while(arr[index]!=-1)

index=(index+1)%10; arr[index]=x;

}

}

void hashing::display()

{

cout<<"\nHash Table\n"; for(int i=0; i<10; i++)

cout<<setw(8)<<i; cout<<"\n";

for(int i=1; i<=10; i++) cout<<setw(8)<<arr[i];

cout<<"\n";

}

void hashing::display1()

{

cout<<"\nHash Table\n"; for(int i=0; i<10; i++)

cout<<setw(8)<<i; cout<<"\n";

for(int i=0; i<10; i++) cout<<setw(8)<<arr[i];

cout<<"\n";

}

void hashing::directHash()

{

for(int i=0; i<10; i++) arr[i]=-1;

for(int i=0; i<10; i++)

{

int x;

cout<<"\nEnter numbers from 1 to 10:"; cin>>x;

int index=x; arr[index]=x;

}

}

void hashing::subHash()

{

for(int i=0; i<10; i++) arr[i]=-1;

for(int i=1; i<=10; i++)

{

int x;

cout<<"\nEnter numbers from 1001 to 1010:"; cin>>x;

int index=x-1000; arr[index]=x;

}

}

void hashing::digitExHash()

{

for(int i=0; i<10; i++) arr[i]=-1;

for(int i=0; i<10; i++)

{

int x;

cout<<"\nEnter a number of 6 digits:"; cin>>x;

int index=0;

long r,inc=100000,incr=1000; for(int i=1; i<=6; i++)

{

if(i==1||i==3||i==5)

{

incr=incr/10; r=(x/inc)%10; index=index+(r\*incr);

}

inc=inc/10;

}

index=index%10; while(arr[index]!=-1)

index=(index+1)%10; arr[index]=x;

}

}

void hashing::foldShiftHash()

{

for(int i=0; i<10; i++) arr[i]=-1;

for(int i=0; i<10; i++)

{

int x;

cout<<"\nEnter a number of 4 digits:"; cin>>x;

index=0;

long no,no1,no2,no3; no1=x/113; no3=no1\*100; no2=x%no3; index=no1+no2; index=index%10; if(index==-1)

{

arr[index]=x;

}

while(arr[index]!=-1) index=(index+1)%10;

arr[index]=x;

}

}

void hashing::foldBoundHash()

{

for(int i=0; i<10; i++) arr[i]=-1;

for(int i=0; i<10; i++)

{

int x;

cout<<"\nEnter a number of 4 digits:"; cin>>x;

index=0;

long no,no1,no2,no3; no1=x/113; no3=no1\*113; no2=x%no3;

int tmp=0; while(no1>0)

{

int rem=no1%10; tmp=(tmp\*10)+rem; no1=no1/10;

}

int tmp1=0; while(no2>0)

{

int rem1=no2%10; tmp1=(tmp1\*10)+rem1; no2=no2/10;

}

index=tmp+tmp1; index=index%10; if(index==-1)

{

arr[index]=x;

}

while(arr[index]!=-1) index=(index+1)%10;

arr[index]=x;

}

}

int main()

{

cout<<"FYMCA Roll No:113 DIV :- B\n\n"; hashing h;

int op;

cout<<"Enter 1 for direct hashing\nEnter 2 for Subtraction hashing\nEnter 3 for Module Division Hashing"<<endl;

cout<<"Enter 4 for digit extraction hashing\nEnter 5 for Shift fold Hashing\nEnter 6 for shift boundary Hashing\n Enter 7 to exit"<<endl;

while(1)

{

cout<<"\nEnter choice:"; cin>>op;

switch(op)

{

case 1:

h.directHash(); h.display(); break;

case 2:

h.subHash();

h.display(); break;

case 3:

h.modDivision();

h.display1(); break;

case 4:

h.digitExHash(); h.display1(); break;

case 5:

h.foldShiftHash(); h.display1(); break;

case 6:

h.foldBoundHash(); h.display1(); break;

case 7:

return 0; default:

cout<<"Wrong choice";

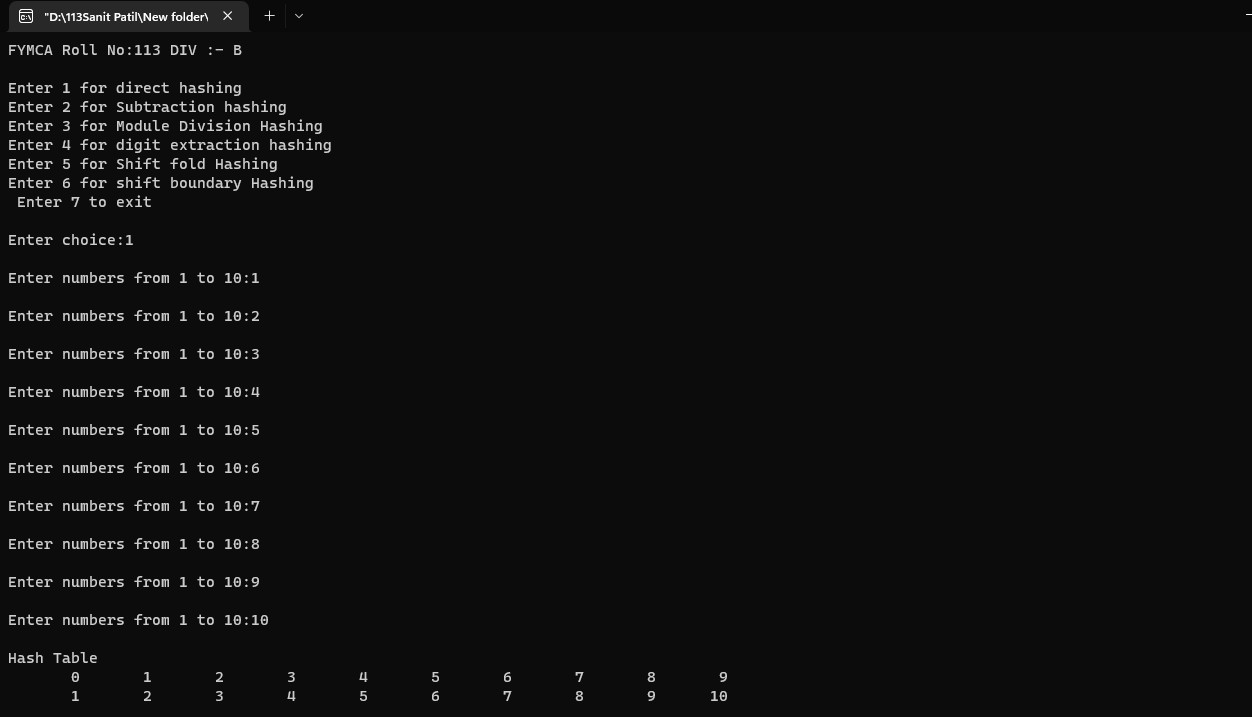
}

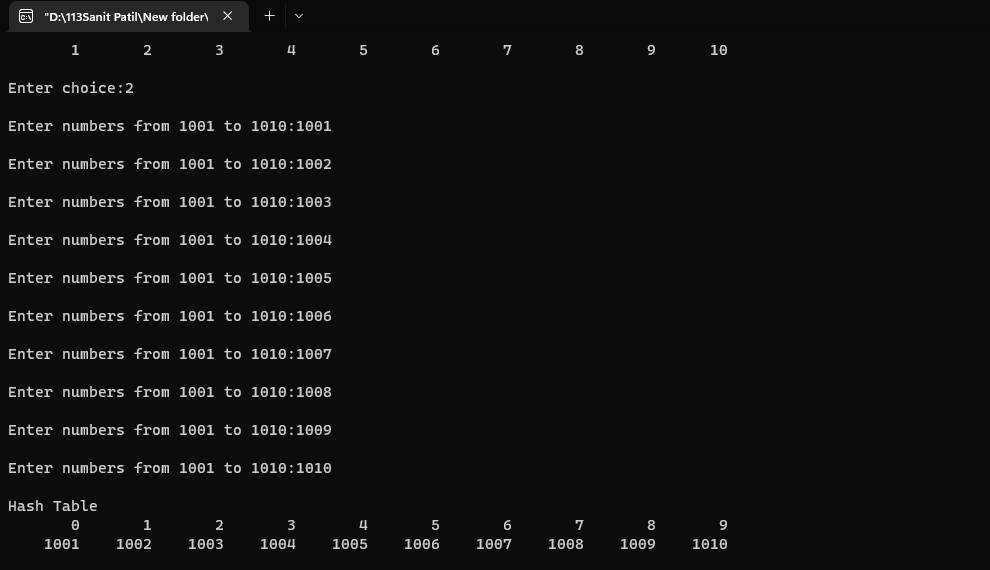
}

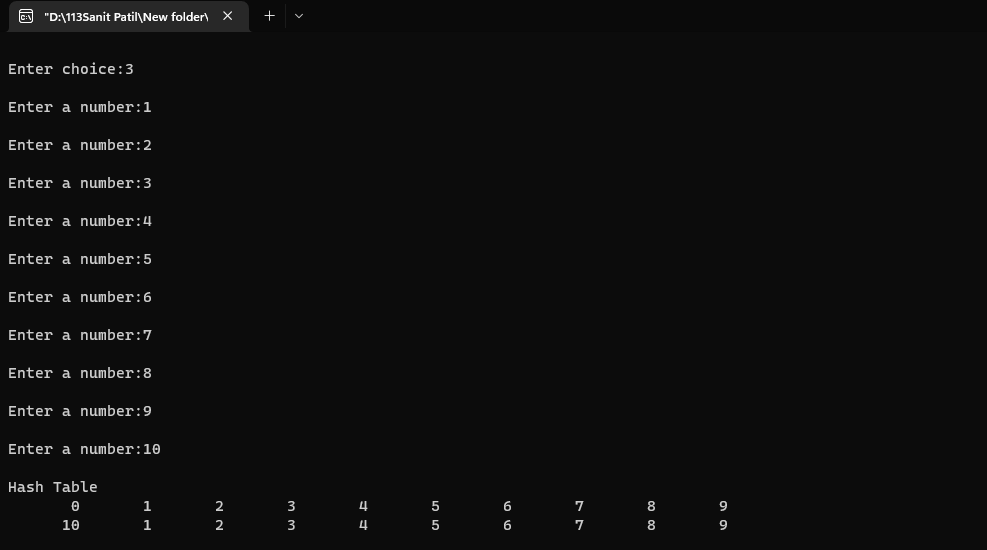
return 0;

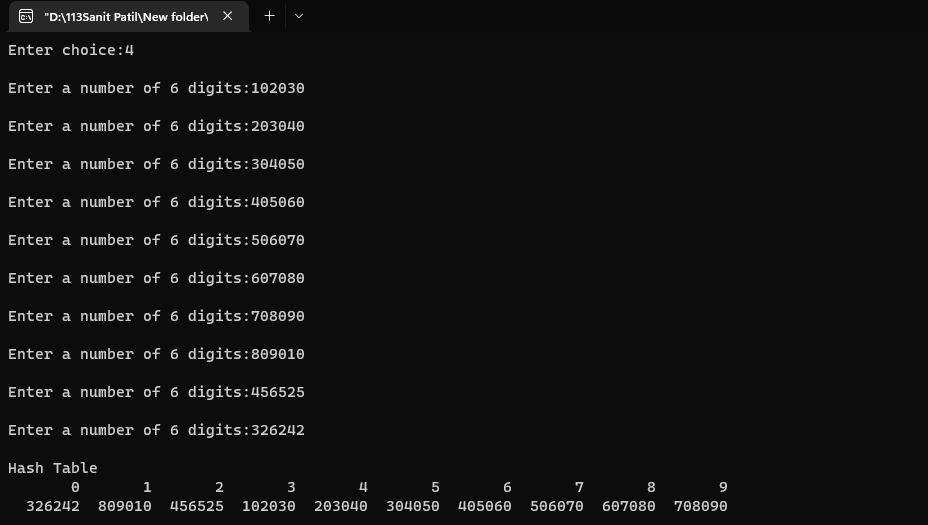
}

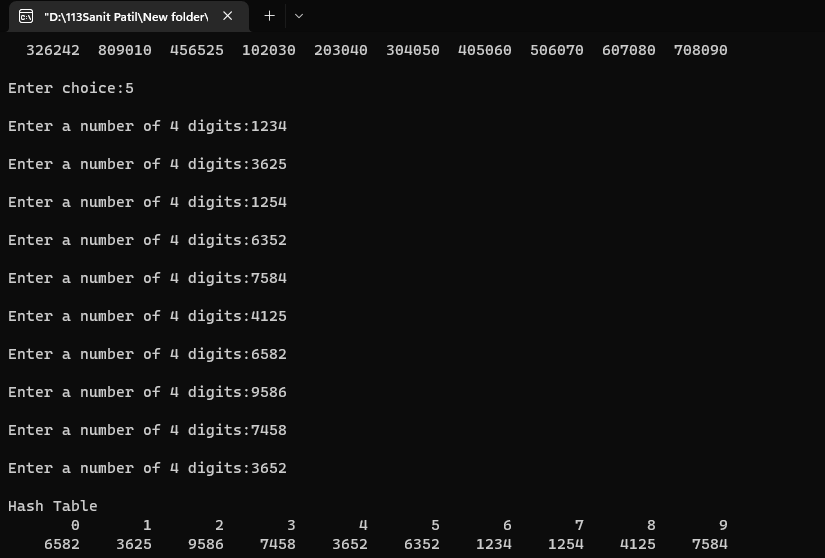
**Output:**

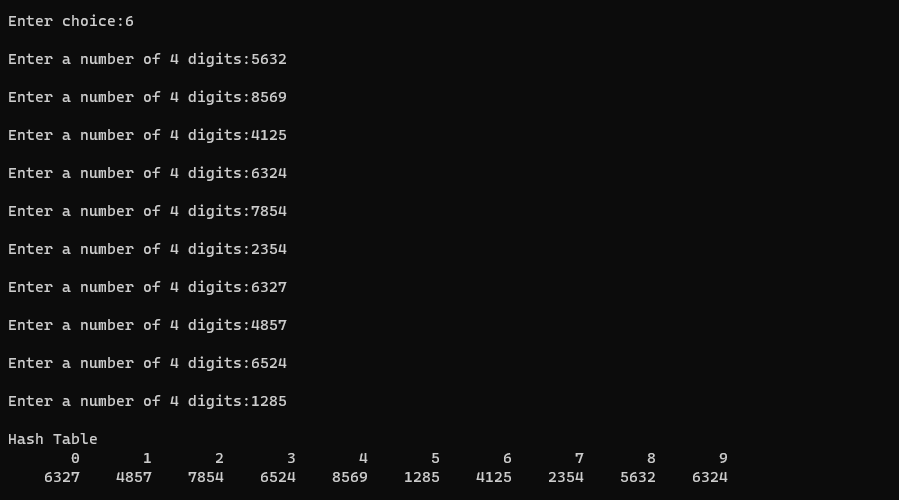


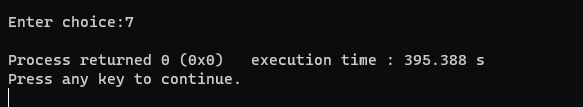












# Practical 3

### 1. Implementation of Stack using Array and Linked List

1. **Stack using Array Program**

#include<iostream> #define MAX 5

using namespace std;

class stack

{

public:

int top=-1;

int x, stk[5],i; int push(int x)

{

if(top == MAX - 1)

{

cout<<"Stack Overflow\n\n";

}

else

{

cout<<"Enter the number to push to the stack :\n"; cin>>x;

stk[++top]=x; stk[top]=x;

}

}

int pop()

{

if(top==-1)

{

cout<<"Stack Underflow\n\n";

}

cout<<"Popped value -> "; x=stk[top];

top--; cout<<x<<"\n";

}

void display()

{

if(top==-1)

{

cout<<"Stack is empty.\n\n";

}

else

cout<<"Stack :\n"; for(i=top;i>=0;i--)

{

cout<<stk[i]<<"\n";

}

}

};

int main()

{

stack s; int ch,x;

while(ch!=4)

{

cout<<"1.Push\n2.Pop\n3.Display\n4.Exit\n"; cout<<"Enter the value for operation -> " ; cin>>ch;

switch(ch)

{

case 1:

s.push(x); break;

case 2:

s.pop(); break;

case 3:

s.display(); break;

case 4:

break; default:

cout<<"\nWrong choice.\n\n";

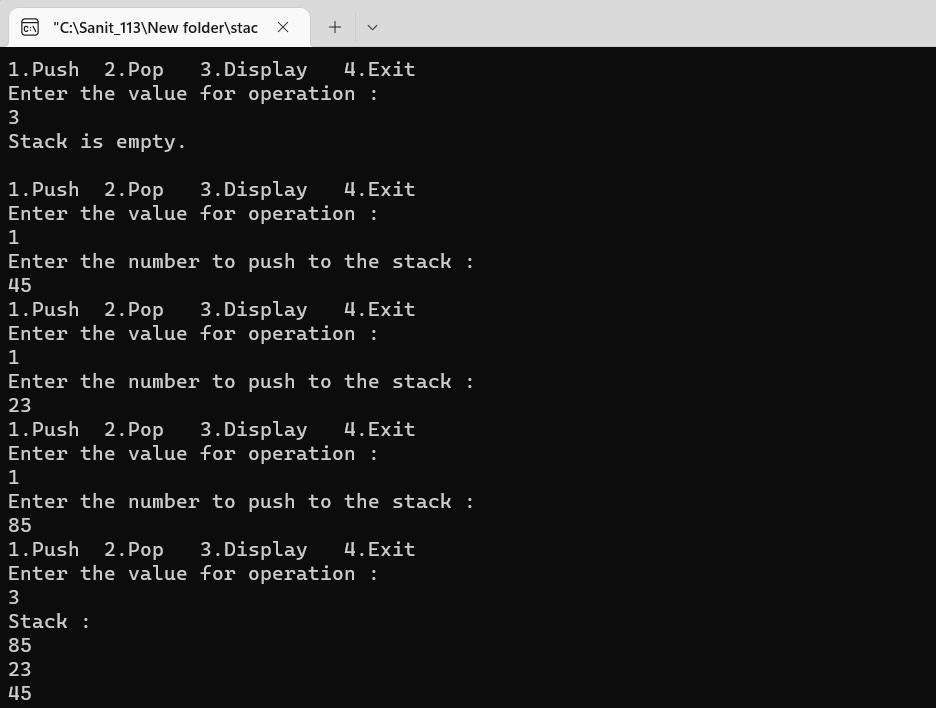
}

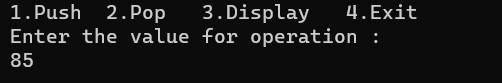
}

return 0;

}

**Output:**





### Stack using Linked List Program

#include<iostream> #include<stack>

using namespace std; class stk

{

public:

struct node

{

int data; node \* next;

};node \*top; stk()

{

top==NULL;

}

int push(int x)

{

node \* tmp = new node(); tmp -> data =x;

tmp ->next=top; top=tmp;

return tmp->data;

}

int pop()

{

if(top==NULL) cout<<"Stack is Empty";

else

{

node \*tmp= new node(); tmp = top;

top= tmp -> next; return tmp->data;

}

}

void display()

{

if(top==NULL)

{

}

else

{

cout<<"stack is empty";

node \*tmp = new node();

//tmp=NULL; tmp=top;

while(tmp->next!=NULL)

{

cout<<tmp->data<<"\n";

}

}

};

int main()

{

tmp=tmp->next;

}

cout<<"\n";

cout<<"Roll No 100\n\n"; stk sl;

int n,s; while(1)

{

cout<<"1]PUSH\n"<<"2]POP\n"<<"3]DISPLAY\n"<<"4]EXIT\n"<<"\nEnter your Choice:";

cin>>n; switch(n)

{

case 1:

cout<<"Enter Element to Insert: "; cin>>s;

sl.push(s); break; case 2:

sl.pop(); break; case 3:

sl.display(); break;

//case 4:

// break; default:

cout<<"\nInvalid Option\n";

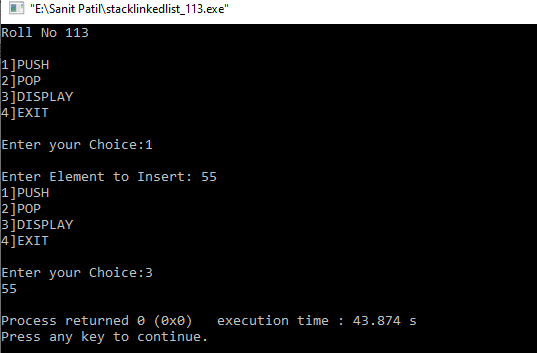
}

}

return 0;

}

## Output:



### Postfix Evaluation

#include<iostream> #include<conio.h> #include<string.h> #include<math.h> using namespace std; class postfix

{

public:

int top; char p[50];

long int A[50]; postfix()

{

top=-1;

}

void input();

void push(long int s); long int pop();

int full(); int empty();

long int eval\_post();

};

void postfix::input()

{

cout<<"enter a postfix expression\n"; cin>>p;

}

int postfix::full()

{

if(top==49)

return 1; else

return 0;

}

void postfix::push(long int s)

{

if(full())

cout<<"overflow\n"; else

{

top=top+1; A[top]=s;

}

}

int postfix::empty()

{

if(top==-1)

return 1; else

return 0;

}

long int postfix::pop()

{

if(empty())

cout<<"underflow\n"; else

return(A[top--]);

}

long int postfix::eval\_post()

{

long int a,b,temp,result,len; int i;

len=strlen(p); p[len]='#';

for(i=0; p[i]!='#'; i++)

{

if(p[i]<='9'&&p[i]>='0') push(p[i]-48);

else

{

a=pop();

b=pop();

switch(p[i])

{

case '+':

temp=b+a; break;

case '-':

temp=b-a; break;

case '\*':

temp=b\*a; break;

case '/':

temp=b/1; break;

case '%':

temp=b%a; break;

case '^':

temp=pow(b,a);

}

push(temp);

}

}

result=pop(); return result;

}

main()

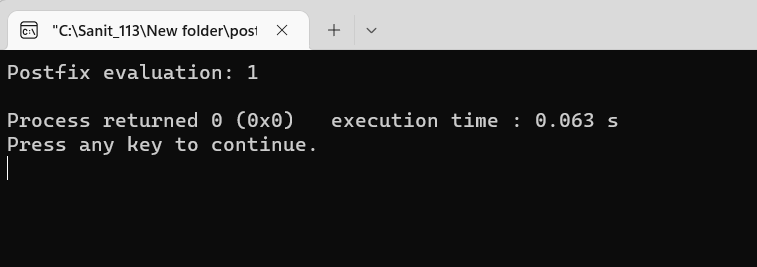
{

long int value; postfix f; f.input(); value=f.eval\_post();

cout<<"value of expression is "<<value; getch();

}

## Output:



### Balancing of parenthesis

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

struct node

{

int data;

struct node \*next;

};

struct node \*tmp=NULL; struct node \*tmp1=NULL; struct node \*top=NULL; struct node \*ptr=NULL; int push(char x)

{

tmp = new node; tmp->data=x; tmp->next=NULL; if(top == NULL)

{

top=tmp;

}

else

{

tmp1=top; top=tmp;

tmp->next=tmp1;

}

}

char pop()

{

if(top==NULL)

{

cout<<"Stack is empty.\n";

}

else

{

ptr=top; top=top->next;

return(ptr->data); delete(ptr);

}

}

int main()

{

int len,i; char c,d,e; char a[30];

cout<<"Enter expression :\n"; cin>>a;

len=strlen(a);

for(i=0; i<len; i++)

{

if(a[i]=='{' || a[i]=='[' || a[i]=='(')

{

push(a[i]);

}

else

{

switch(a[i])

{

case ')':

c=pop();

if(c=='{' || c=='[')

{

cout<<"Invalid"; getch();

}

break;

case ']':

d=pop();

if(d=='{' || d=='(')

{

cout<<"Invalid"; getch();

}

break;

case '}':

e=pop();

if(e=='(' || e=='[')

{

cout<<"Invalid"; getch();

}

break;

default:

cout<<"Enter the correct choice"; getch();

}

}

}

if(top==NULL)

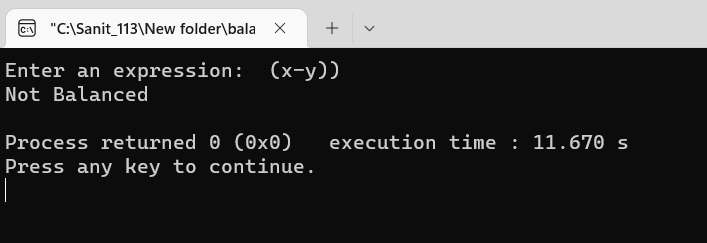
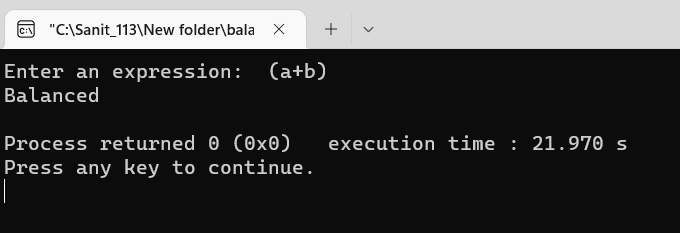
cout<<"Balanced\n"; else

cout<<"Unbalanced\n"; getch();

return 0;

}

**Output:**



# Practical 4

### 1. Write a program to implement different types of queues

1. **Write a program to implement Double Ended queue.**

#include <iostream> #include <cstdlib> using namespace std;

struct node

{

int info;

node \*next; node \*prev;

}\*head, \*tail; class dqueue

{

public:

int top1, top2; void insert(); void del(); void display(); dqueue()

{

top1 = 0;

top2 = 0; head = NULL; tail = NULL;

}

};

int main()

{

int choice; dqueue dl; while (1)

{

cout<<"1.Insert Element into the Deque"<<endl; cout<<"2.Delete Element from the Deque"<<endl; cout<<"3.Traverse the Deque"<<endl; cout<<"4.Quit"<<endl;

cout<<"Enter your Choice: "; cin>>choice;

cout<<endl; switch(choice)

{

case 1:

dl.insert(); break;

case 2:

dl.del(); break;

case 3:

dl.display(); break;

case 4:

exit(1); break;

default:

cout<<"Wrong Choice"<<endl;

}

}

return 0;

}

void dqueue::insert()

{

struct node \*temp; int ch, value;

if (top1 + top2 >= 50)

{

cout<<"Dequeue Overflow"<<endl; return;

}

if (top1 + top2 == 0)

{

cout<<"Enter the value to be inserted: "; cin>>value;

head = new (struct node); head->info = value;

head->next = NULL; head->prev = NULL; tail = head; top1++;

cout<<"Element Inserted into empty deque"<<endl;

}

else

{

while (1)

{

cout<<endl;

cout<<"1.Insert Element at first"<<endl; cout<<"2.Insert Element at last"<<endl; cout<<"3.Exit"<<endl;

cout<<endl;

cout<<"Enter Your Choice: "; cin>>ch;

cout<<endl; switch(ch)

{

case 1:

cout<<"Enter the value to be inserted: "; cin>>value;

temp = new (struct node); temp->info = value;

temp->next = head; temp->prev = NULL; head->prev = temp; head = temp; top1++;

break; case 2:

cout<<"Enter the value to be inserted: "; cin>>value;

temp = new (struct node); temp->info = value;

temp->next = NULL; temp->prev = tail; tail->next = temp; tail = temp; top2++;

break; case 3:

return; break;

default:

cout<<"Wrong Choice"<<endl;

}

}

}

}

void dqueue::del()

{

if (top1 + top2 <= 0)

{

cout<<"Deque Underflow"<<endl;

return;

}

int ch; while (1)

{

cout<<endl;

cout<<"1.Delete Element at first"<<endl; cout<<"2.Delete Element at last"<<endl; cout<<"3.Exit"<<endl;

cout<<endl;

cout<<"Enter Your Choice: "; cin>>ch;

cout<<endl; switch(ch)

{

case 1:

head = head->next; head->prev = NULL; top1--;

break; case 2:

tail = tail->prev; tail->next = NULL; top2--;

break; case 3:

return; break;

default:

cout<<"Wrong Choice"<<endl;

}

}

}

void dqueue::display()

{

struct node \*temp; int ch;

if (top1 + top2 <= 0)

{

cout<<"Deque Underflow"<<endl; return;

}

while (1)

{

cout<<endl;

cout<<"1.Display Deque from Beginning"<<endl;

cout<<"2.Display Deque from End"<<endl; cout<<"3.Exit"<<endl;

cout<<endl;

cout<<"Enter Your Choice: "; cin>>ch;

cout<<endl; switch (ch)

{

case 1:

temp = head;

cout<<"Deque from Beginning:"<<endl; while (temp != NULL)

{

cout<<temp->info<<" "; temp = temp->next;

}

cout<<endl; break;

case 2:

cout<<"Deque from End:"<<endl; temp = tail;

while (temp != NULL)

{

cout<<temp->info<<" "; temp = temp->prev;

}

temp = tail; cout<<endl; break;

case 3:

return; break;

default:

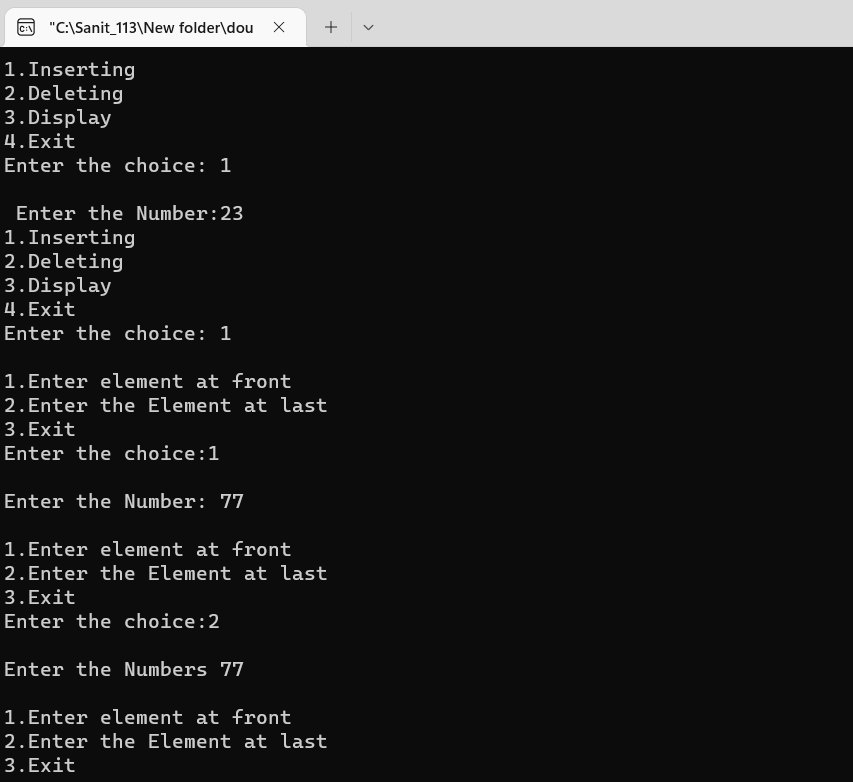
cout<<"Wrong Choice"<<endl;

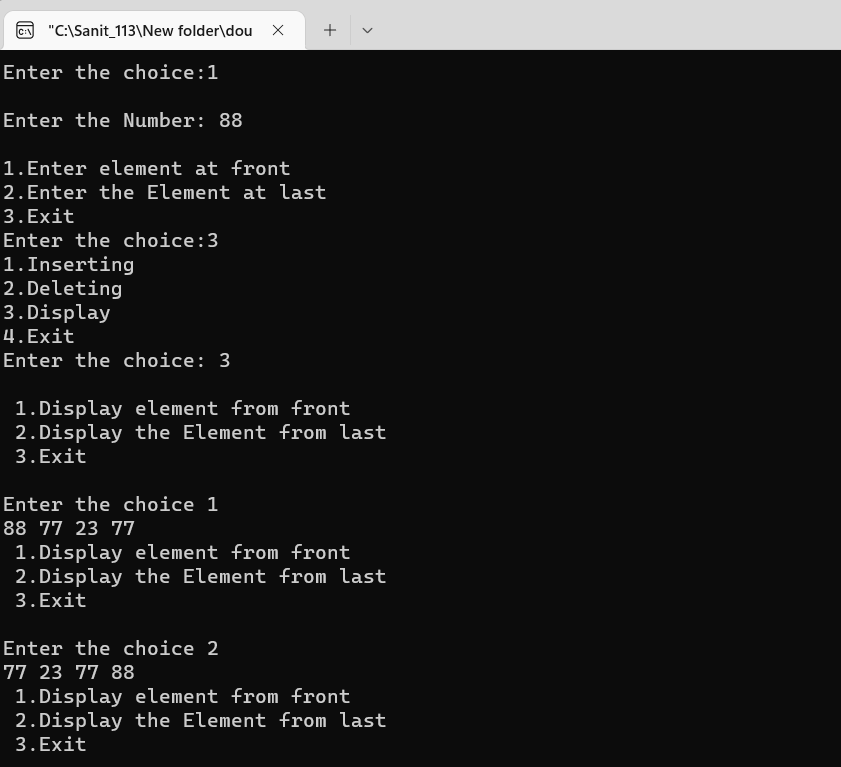
}

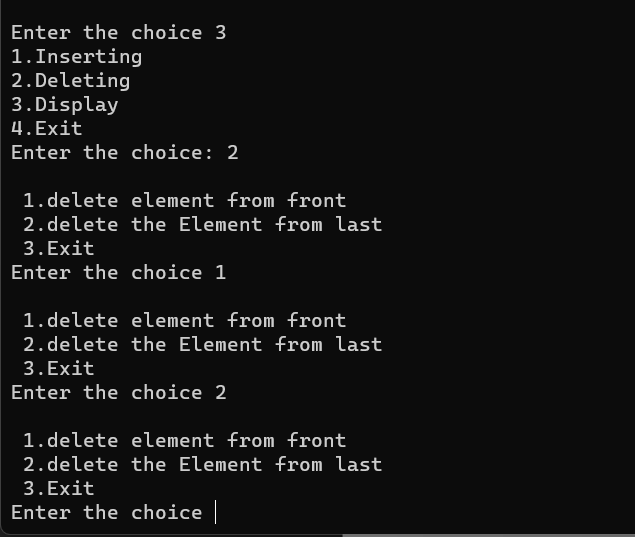
}

}

**Output:**







### Write a program to implement Priority queue.

#include<iostream> using namespace std; class priority

{

public: struct node

{

int pr; int data;

struct node \*next;

};

node \*front=NULL;

void insert(int item, int pr)

{

node \*tmp, \*q; tmp=new node; tmp->data=item; tmp->pr=pr;

if(front==NULL || pr < front->pr)

{

}

else

{

}

}

tmp->next=front; front=tmp;

q=front;

while(q->next!=NULL && q->next->pr<=pr) q=q->next;

tmp->next = q-> next; q->next=tmp;

void del()

{

node \*tmp,\*q; if(front==NULL)

cout<<"Queue is Empty";

else

{

}

}

tmp=front;

cout<<tmp->data<<" is deleted \n"; front=front->next;

void display()

{

if(front==NULL)

{

}

else

{

cout<<"Queue is Empty.\n\n";

node \*ptr; ptr=front;

cout<<"Item\t"<<"Priority\n"; while(ptr!=NULL)

{

cout<<ptr->data<<"\t"; cout<<ptr->pr<<"\t"; cout<<"\n";

ptr=ptr->next;

}

}

}

};

int main()

{

priority p; int ch,x,y;

while(ch!=0)

{

cout<<"1.Insert\n2.Delete\n3.Display\n"; cout<<"Enter the choice :\n";

cin>>ch; switch(ch)

{

case 1:

cout<<"Enter the Item :\n"; cin>>x;

cout<<"Enter the Priority :\n"; cin>>y;

p.insert(x,y); break;

case 2:

p.del(); break;

case 3:

p.display(); break;

default:

cout<<"Enter the correct choice"; break;

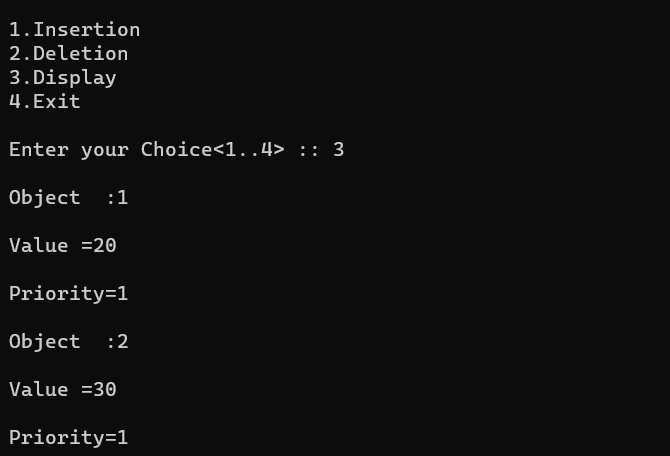
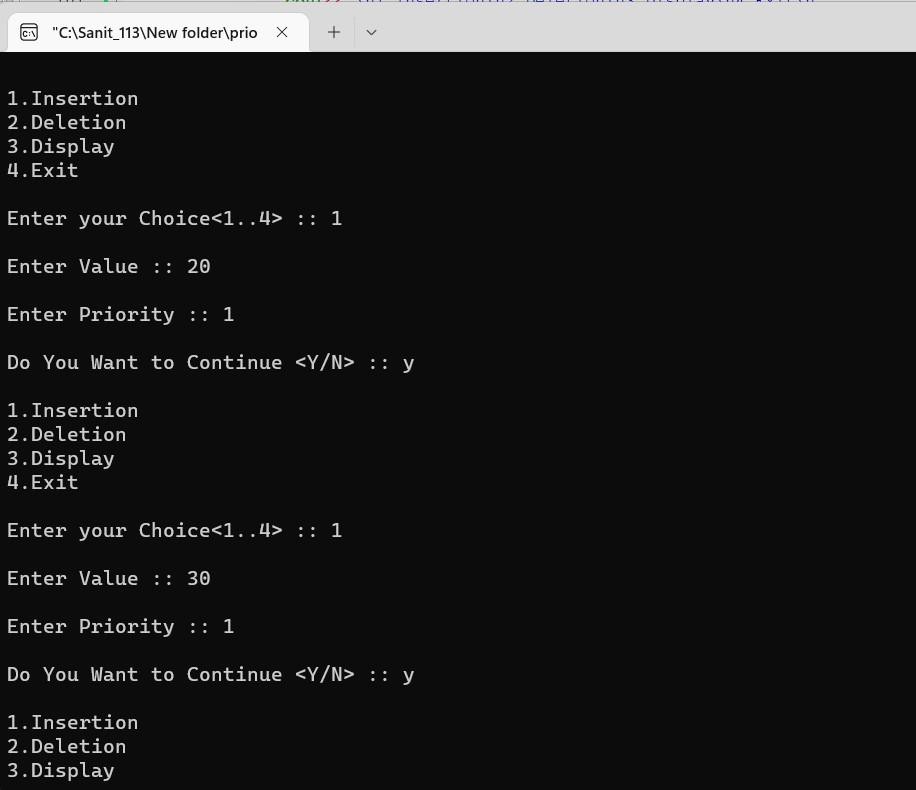
}

}

return 0;

}

## Output:



### Write a program to implement Circular queue.

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

{

public:

int q[4],x,result; int front=-1;

int rear=-1; int maxsize=4;

void enqueue(int x)

{

if(rear==maxsize-1)

{

}

else

{

}

}

cout<<"Queue full.\n\n";

rear++; q[rear]=x;

void dequeue()

{

if(rear==-1)

{

}

else

{

cout<<"Queue empty.\n\n";

cout<<"Deleted.\n"; if(front==rear)

{

}

else

{

front=-1; rear=-1;

x=q[front]; front++;

}

}

}

void display()

{

if(rear==-1)

{

cout<<"Queue empty.\n\n";

}

else

{

cout<<"Queue :\n";

for(int i=front+1;i<=rear;i++)

{

cout<<q[i]<<"\n";

}

}

}

};

int main()

{

int ch,x; queue q; do

{

cout<<"1.Enqueue\n2.Dequeue\n3.Display\n4.Exit\n"; cout<<"Enter your choice :\n";

cin>>ch; switch(ch)

{

case 1:

cout<<"Enter the value :\n"; cin>>x;

q.enqueue(x);

break; case 2:

q.dequeue(); break;

case 3:

q.display(); break;

case 4:

break; default:

cout<<"\nInvalid choice!!\n";

}

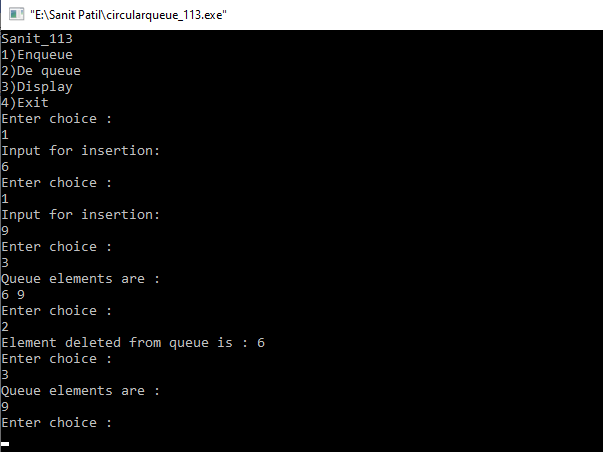
}

while(ch<4); return 0;

}

## Output:

**Output:**



# Practical 4

### 1. Write a program to implement different types of Linked List.

1. **Write a program to implement Queue using Linked List.**

#include<iostream> using namespace std; class linkqueue

{

public: struct node

{

int data;

struct node \*next;

};

node \*front = NULL; node \*rear = NULL; void enqueue(int x)

{

node \*tmp, \*q; tmp=new node; tmp->data=x; tmp->next=NULL;

if(front==NULL && rear==NULL)

{

front=rear=tmp;

}

else

{

rear->next=tmp; rear=tmp;

}

}

void dequeue()

{

struct node \*tmp=front; if(front==NULL)

{

cout<<"Queue is empty\n\n";

}

cout<<"Deleted\n"; if(front==rear)

{

}

else

{

front=rear=NULL;

front=front->next;

}

}

void display()

{

if(front==NULL)

{

}

else

{

cout<<"Queue is empty.\n\n";

node \*ptr; ptr=front; cout<<"Queue :\n"; while(ptr!=NULL)

{

cout<<ptr->data<<"\t"; cout<<"\n";

ptr=ptr->next;

}

}

}

};

int main()

{

linkqueue q; int ch,x,n; do

{

cout<<"1.Enqueue\n2.Dequeue\n3.Display\n4.Exit\n"; cout<<"Enter the value for operation :\n"; cin>>ch;

switch(ch)

{

case 1:

cout<<"Enter the value :\n"; cin>>x;

q.enqueue(x); break;

case 2:

q.dequeue(); break;

case 3:

q.display(); break;

case 4:

break; default:

cout<<"\nWrong choice.\n\n";

}

}

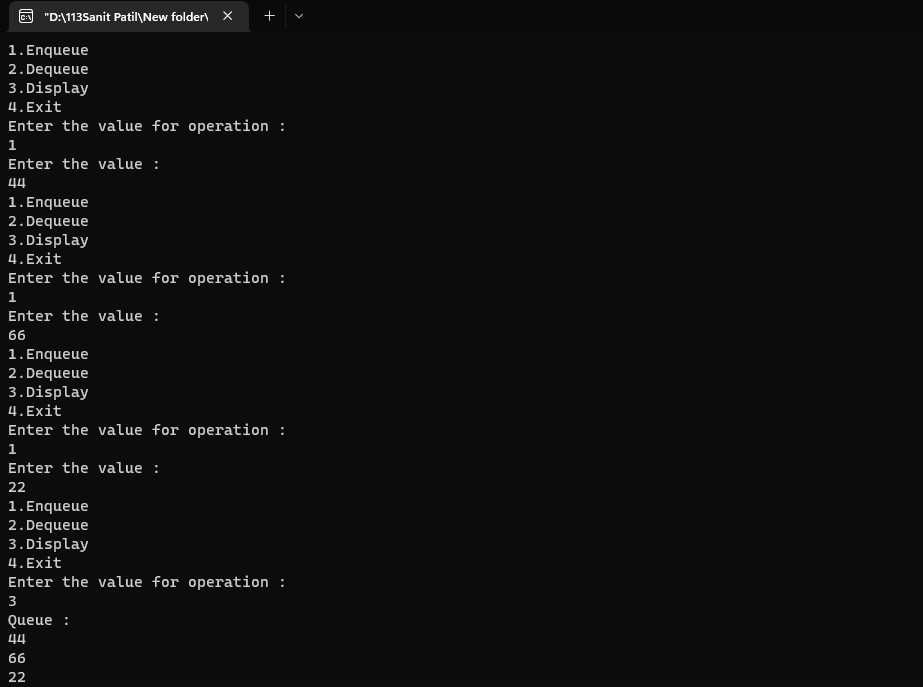
while(n!=4);

return 0;

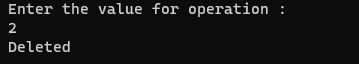
}

**Output:**

#### Insert



* 1. **Delete**



1. **Write a program to implement Singly Linked List.**

#include<iostream> using namespace std; class linklist

{

public:

int flag=true;

int pos, i, value, count=0; struct node

{

int data;

struct node \*next;

};

struct node \*tmp=NULL; struct node \*tmp1=NULL; struct node \*start=NULL; struct node \*p=NULL; struct node \*ptr=NULL; struct node \*ptr1=NULL; struct node \*ptr2=NULL; struct node \*ptr3=NULL;

int insert\_at\_beg(int x)

{

tmp = new node; tmp->data=x; tmp->next=NULL;

if(start == NULL)

{

}

else

{

}

}

start=tmp;

start->next=NULL;

tmp1=start; start=tmp;

start->next=tmp1;

int insert\_at\_end(int x)

{

tmp = new node; tmp->data=x; tmp->next=NULL; p=start;

while(p->next!=NULL)

{

p=p->next;

}

p->next=tmp; tmp->next=NULL;

}

int insert\_at\_pos(int x)

{

cout<<"Insert the position :\n"; cin>>pos;

tmp = new node; tmp->data=x; tmp->next=NULL; p=start; while(p!=NULL)

{

p=p->next; count++;

}

if(pos==1)

{

start=tmp;

start->next=NULL;

}

else if(pos > 1 && pos<count)

{

p=start; for(i=1;i<pos;i++)

{

ptr=p;

p=p->next;

}

}

else

{

}

}

ptr->next=tmp; tmp->next=p;

cout<<"Invalid position.\n";

int search\_pos()

{

cout<<"Insert the value:\n"; cin>>value;

count=0; flag=false; if(start==NULL)

{

cout<<"List is empty\n";

}

else

{

p=start; while(p!=NULL)

{

count++;

if(p->data == value)

{

position.\n";

flag==true;

cout<<"Value found at "<<count<<"

}

p=p->next;

}

}

}

void del()

{

cout<<"Delete the position:\n"; cin>>pos;

if(start==NULL)

{

cout<<"List is empty.\n";

}

else

{

if(pos==1)

{

}

else

{

tmp=start; start=start->next; delete (tmp);

while(p!=NULL)

{

count++; p=p->next;

}

}

if(pos>1 && pos<count)

{

p=start; for(i=1;i<pos;i++)

{

ptr=p;

p=p->next;

}

ptr->next=p->next;

}

}

}

int sort()

{

int x; if(start==NULL)

{

cout<<"List is empty.\n";

}

ptr=start; while(ptr!=NULL)

{

for(p=ptr->next;p!=NULL;p=p->next)

{

if(ptr->data>p->data)

{

x=ptr->data;

ptr->data=p->data; p->data=x;

}

}

ptr=ptr->next;

}

}

int rev()

{

if(start==NULL)

{

cout<<"List is empty.\n";

}

if(start->next==NULL)

{

cout<<"only one.\n";

}

ptr1=start; ptr2=ptr1->next; ptr3=ptr2->next; ptr1->next=NULL; ptr2->next=ptr1; while(ptr3!=NULL)

{

ptr1=ptr2; ptr2=ptr3; ptr3=ptr3->next; ptr2->next=ptr1;

}

start=ptr2;

}

void display()

{

if(start==NULL)

{

cout<<"List is empty.\n";

}

else

{

p=start;

cout<<"Singly Linked List :\n"; while(p!=NULL)

{

cout<< p->data<<" -> "; p=p->next;

}

cout<<"\n";

}

}

};

int main()

{

linklist l; int ch,x; while(ch!=0)

{

cout<<"\n1.Insert at beginning\n2.Insert at end\n3.Insert at position\n4.Delete\n5.Search\n6.Display\n7.Sort\n8.Reverse\n9.Exit";

cout<<"\nEnter the choice:\n"; cin>>ch;

switch(ch)

{

case 1:

cout<<"Enter the value\n"; cin>>x; l.insert\_at\_beg(x); l.display();

break;

case 2:

cout<<"Enter the value\n"; cin>>x; l.insert\_at\_end(x); l.display();

break;

case 3:

cout<<"Enter the value\n"; cin>>x; l.insert\_at\_pos(x); l.display();

break;

case 4:

l.del();

l.display(); break;

case 5:

l.search\_pos ();

l.display(); break;

case 6:

l.display ();break;

case 7:

l.sort(); l.display ();break;

case 8:

l.rev(); l.display ();break;

case 9:

break;

default:

cout<<"Wrong choice\n";

}

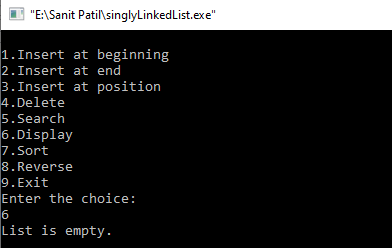
}

return 0;

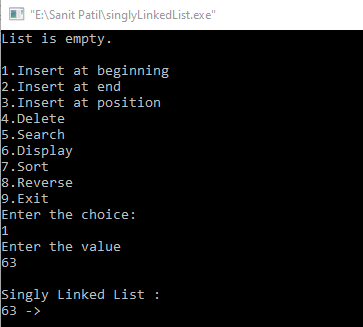
}

## Output:

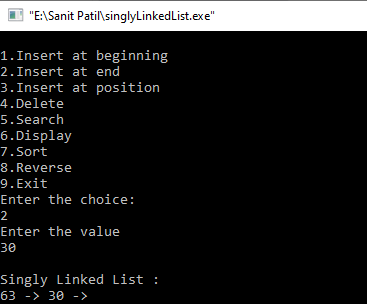
List Empty:



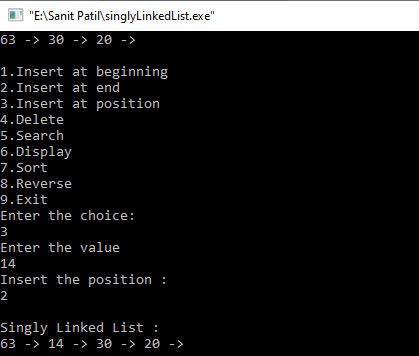
Insert at Beginning:



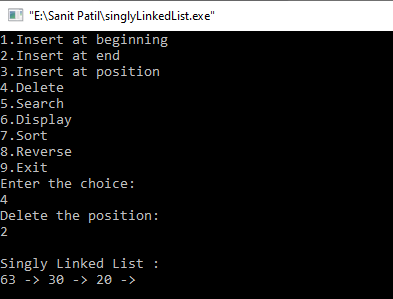
Insert at End:



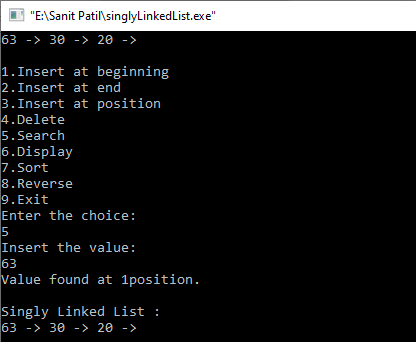
Insert at Position:



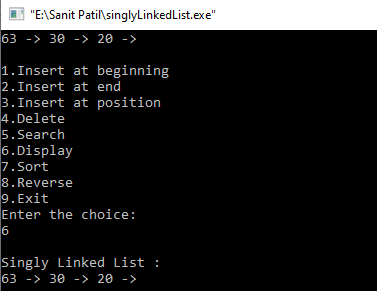
Delete:

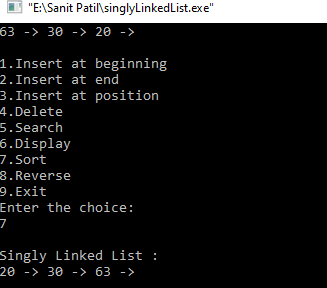


Search:

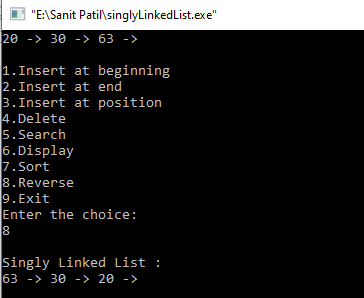


Display:



Sort:

Reverse:



1. **Write a program to implement Doubly Linked List.**

#include<iostream> using namespace std; class doubly

{

public:

struct node

{

int data;

struct node \*next; struct node \*prev;

};

struct node \*tmp=NULL; struct node \*ptr=NULL; struct node \*start=NULL; struct node \*p=NULL; struct node \*p1=NULL; struct node \*p2=NULL; void create(int x)

{

tmp=new node; tmp->data=x; tmp->next=NULL; if(start==NULL)

{

}

else

{

tmp->prev=NULL; start=tmp;

p=start;

while(p->next!=NULL)

{

p=p->next;

}

p->next=tmp; tmp->prev=p;

}

}

void add\_atbegin(int x)

{

if(start==NULL)

{

cout<<"List is empty.\n\n";

}

tmp=new node; tmp->data=x; tmp->next=start; start->prev=tmp; start=tmp;

}

void add\_after(int x,int pos)

{

if(start==NULL)

{

cout<<"List is empty.\n\n";

}

p=start;

for(int i=0;i<pos-1;i++)

{

p=p->next; if(p==NULL)

{

cout<<"Position does not exist.\n\n";

}

}

tmp=new node; tmp->data=x;

if(p->next==NULL)

{

}

else

{

}

}

1. >next=tmp; tmp->next=NULL; tmp->prev=p;

tmp->next=p->next; tmp->next->prev=tmp; p->next=tmp;

tmp->prev=p;

void del(int x)

{

if(start->data==x) //deleting first element

{

tmp=start; start=start->next;

start->prev=NULL; delete(tmp);

}

p=start;

while(p->next->next!=NULL) //deleting element in between

{

if(p->next->data==x)

{

tmp=p->next;

p->next=tmp->next; tmp->next->prev=p; delete(tmp);

}

p=p->next;

}

if(p->next->data==x) //last element deleted

{

tmp=p->next; delete(tmp); p->next=NULL;

}

}

void reverse()

{

p1=start; p2=p1->next; p1->next=NULL; p1->prev=p2;

while(p2!=NULL)

{

p2->prev=p2->next; p2->next=p1; p1=p2;

p2=p2->prev;

}

start=p1;

cout<<"List reversed.\n";

}

void count()

{

p=start; int cnt=0;

while(p!=NULL)

{

p=p->next;

cnt++;

}

cout<<"Number of element are "<<cnt<<".\n";

}

void search()

{

int count=0,value; int flag=0;

cout<<"Enter the element to be searched :\n"; cin>>value;

if(start==NULL)

{

}

else

{

cout<<"List is empty.\n\n";

p=start; while(p!=NULL)

{

count++;

if(p->data==value)

{

"<<count<<".\n";

flag=1;

cout<<"Element found at position

}

p=p->next;

}

}

}

void sort()

{

if(start==NULL)

{

cout<<"list is empty.\n\n";

}

ptr=start; while(ptr!=NULL)

{

for(p=ptr->next;p!=NULL;p=p->next)

{

if(ptr->data > p->data)

{

int x=ptr->data; ptr->data = p->data;

p->data = x;

}

}

ptr=ptr->next;

}

}

void display()

{

if(start==NULL)

{

}

else

{

cout<<"List is empty.\n\n"; return;

p=start;

cout<<"\nDoubly Linked List :\n"; while(p!=NULL)

{

cout<<p->data<<" -> "; p=p->next;

}

cout<<"\n\n";

}

}

};

int main()

{

doubly d; int x,ch; int pos;

cout<<"\nFYMCA B ROLL NO :\_86\n"; while(ch!=0)

{

cout<<"1.Create a list\n2.Add at begin\n3.Add after\n4.Search\n5.Reverse\n6.Count\n7.Sort\n";

cout<<"8.Display\n9.Delete\n10.Exit"; cout<<"\nEnter the choice:\n"; cin>>ch;

switch(ch)

{

case 1:

cout<<"Enter the value :\n"; cin>>x;

d.create(x);

d.display(); break;

case 2:

cout<<"Enter the value :\n"; cin>>x;

d.add\_atbegin(x); d.display(); break;

case 3:

cout<<"Enter the position :\n"; cin>>pos;

cout<<"Enter the value :\n"; cin>>x;

d.add\_after(x,pos); d.display(); break;

case 4:

d.search();

d.display(); break;

case 5:

d.reverse();

d.display(); break;

case 6:

d.count();

d.display(); break;

case 7:

cout<<"Before sorting -"; d.display();

d.sort();

cout<<"After sorting -"; d.display();

break;

case 8 :

d.display();

break;

case 9:

cout<<"Enter the element to be delete :\n"; cin>>x;

d.del(x);

d.display(); break;

case 10:

break;

default:

cout<<"Wrong choice.\n";

}

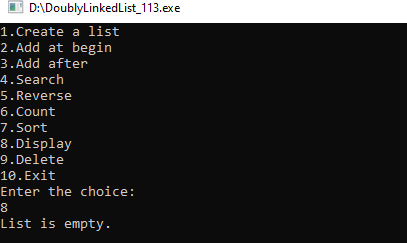
}

return 0;

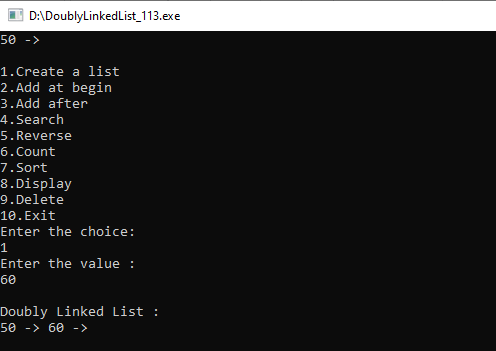
}

## Output:

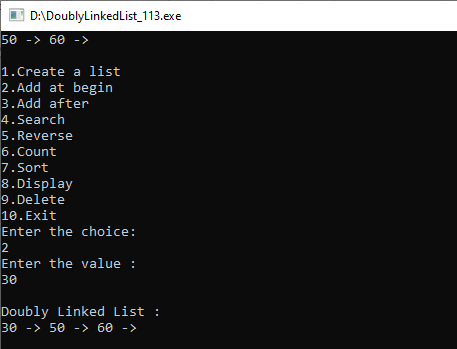
List is empty



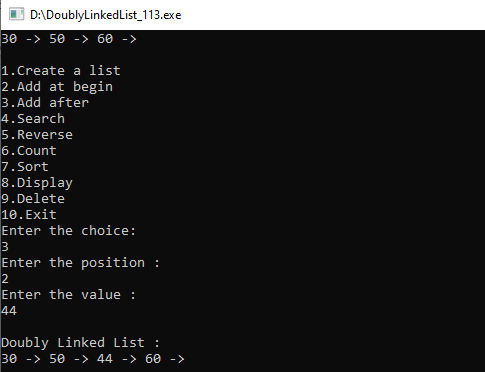
Create List



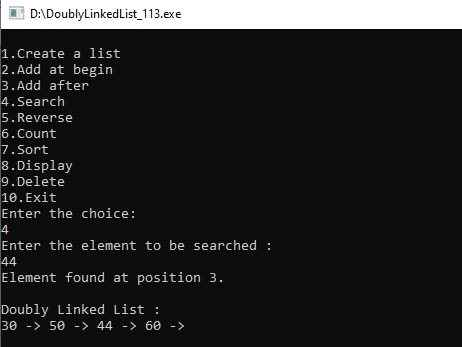
Add at beginning



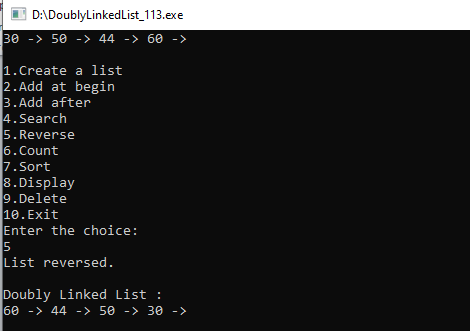
Add after



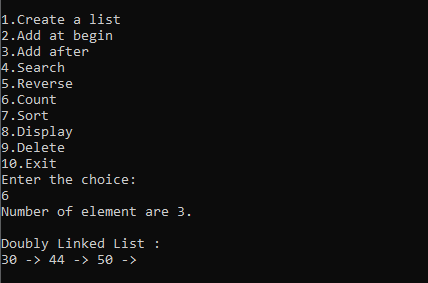
Search



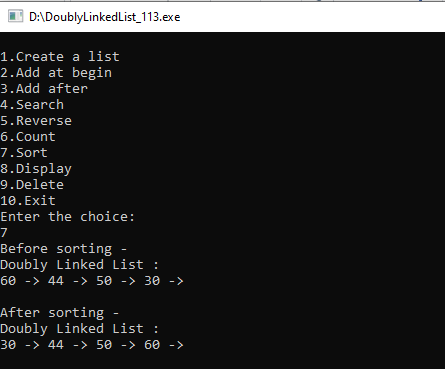
Reverse



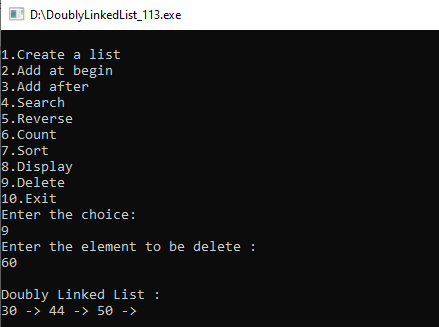
Count



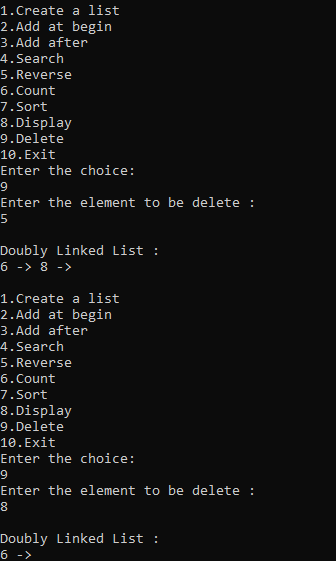
Sort



Display



Delete



### Write a program to implement Circular Linked List.

#include<iostream>

using namespace std; class singly\_circular

{

public:

int flag=true;

int pos, i, value, count=0; struct node

{

int data;

struct node \*next; struct node \*prev;

};

struct node \*tmp=NULL; struct node \*start=NULL; struct node \*last=NULL; struct node \*p=NULL; struct node \*ptr=NULL;

void create(int x)

{

tmp=new node; tmp->data=x; if(last==NULL)

{

}

else

{

}

}

last=tmp;

tmp->next=last;

tmp->next=last->next; last->next=tmp; last=tmp;

void add\_atbegin(int x)

{

if(last==NULL)

{

cout<<"List is empty.\n";

}

tmp=new node; tmp->data=x;

tmp->next=last->next; last->next=tmp;

}

void add\_after(int x,int pos)

{

if(last==NULL)

{

cout<<"List is empty.\n";

}

p=last->next;

for(int i=0;i<pos-1;i++)

{

p=p->next; if(p==last->next)

{

cout<<"Position does not exist.\n";

//break;

}

}

tmp=new node;

tmp->next=p->next; tmp->data=x;

p->next=tmp; if(p==last)

{

last=tmp;

}

}

void del(int x)

{

//p=last->next;

if(last->next==last && last->data==x) // for only one node

{

tmp=last; last=NULL; delete(tmp); return;

}

p=last->next;

if(p->data==x) //first element deleted

{

tmp=p;

last->next=p->next; delete(tmp); return;

}

while(p->next!=last)

{

if(p->next->data==x)

{

tmp=p->next;

p->next=tmp->next; delete(tmp);

//cout<<"Deleted item "<<x; return;

}//delete element in between

p=p->next;

}

if(p->next->data==x)

{

tmp=p->next;

p->next=last->next; delete(tmp); last=p;

return;

}//last element deleted cout<<"Element not found.\n";

}

void search1(int x)

{

int pos=1;

while(p->next!=last)

{

if(p->data==x)

{

cout<<"Element found at position "<<pos-1<<".\n";

}

p=p->next; pos++;

}

if(p==NULL)

cout<<"Item not found.\n";

}

void sort()

{

int x; if(last==NULL)

{

cout<<"List is empty.\n\n";

}

p=last->next;

while(p!=last)

{

ptr=p->next; while(ptr!=last->next)

{

if(ptr!=last->next)

{

if(p->data>ptr->data)

{

x=p->data;

p->data=ptr->data; ptr->data=x;

}

}

ptr=ptr->next;

}

p=p->next;

}

}

int count1()

{

if(last==NULL)

{

}

else

{

cout<<"List is empty.\n\n";

p=last->next; while(p!=last)

{

count++; p=p->next;

}

count++;

cout<<"Number of element are "<<count<<"\n";

}

}

void display()

{

if(last==NULL)

{

cout<<"List is empty.\n\n"; return;

}

p=last->next;

cout<<"Circular Linked List :\n"; while(p!=last)

{

cout<<p->data<<" -> "; p=p->next;

}

cout<<last->data<<"\n\n";

}

};

int main()

{

singly\_circular d; int x,ch;

int pos;

while(ch!=9)

{

cout<<"1.Create a list\n2.Add at begin\n3.Add after\n4.Search\n";

cout<<"5.Sort\n6.Count\n7.Display\n8.Delete\n9.Exit\n"; cout<<"Enter the choice:\n";

cin>>ch;

switch(ch)

{

case 1:

cout<<"Enter the value :\n"; cin>>x;

d.create(x);

d.display(); break;

case 2:

cout<<"Enter the value :\n"; cin>>x;

d.add\_atbegin(x); d.display(); break;

case 3:

cout<<"Enter the position :\n"; cin>>pos;

cout<<"Enter the value :\n"; cin>>x;

d.add\_after(x,pos); d.display(); break;

case 4:

cout<<"Enter element to be searched:\n"; cin>>x;

d.search1(x);

d.display(); break;

case 5:

cout<<"Before sorting -"; d.display();

d.sort();

cout<<"After sorting -"; d.display();

break;

case 6:

d.count1();

d.display(); break;

case 7:

d.display(); break;

case 8:

cout<<"Enter the element to be delete

:\n";cin>>x;

d.del(x); d.display ();break;

case 9:

break;

default:

cout<<"Wrong choice.\n";

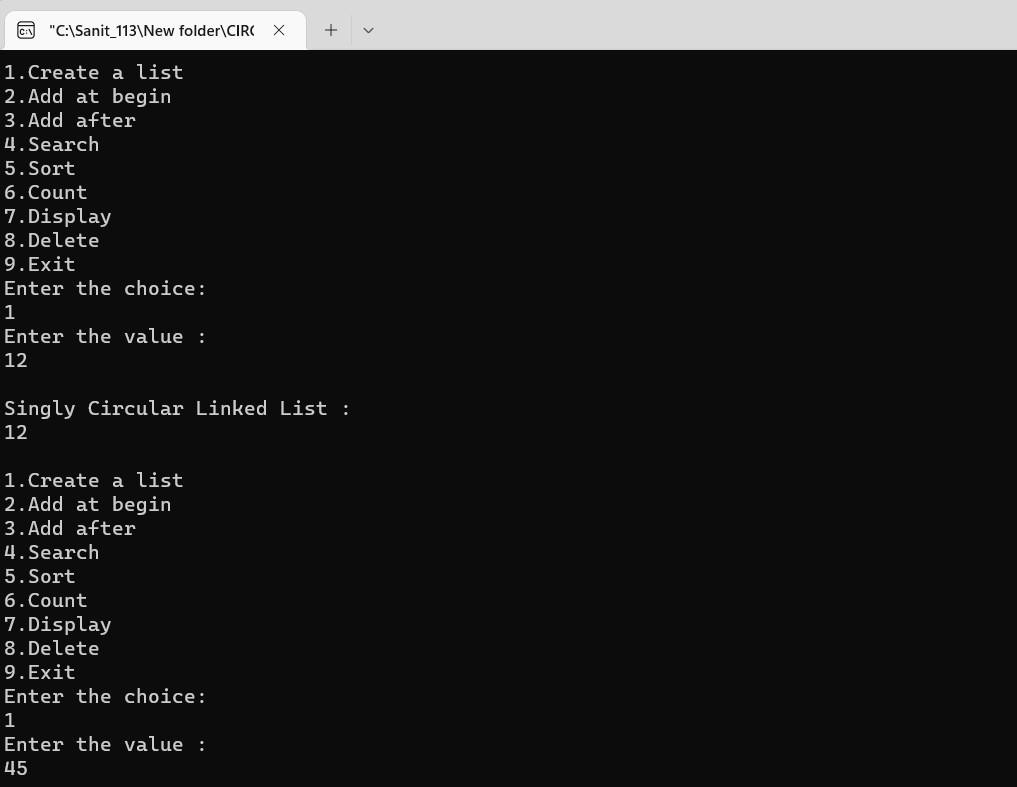
}

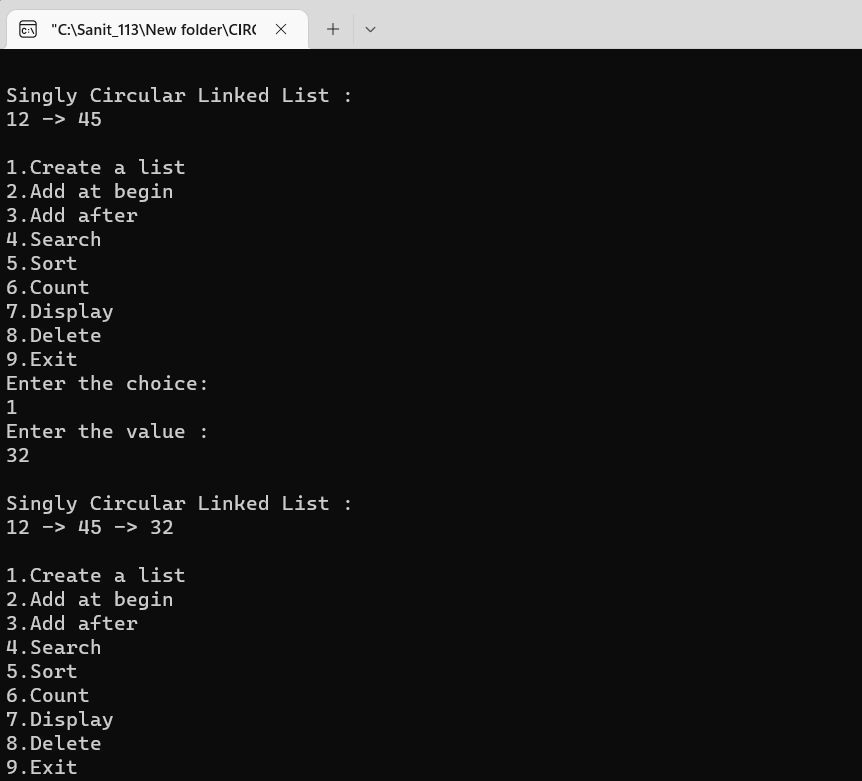
}

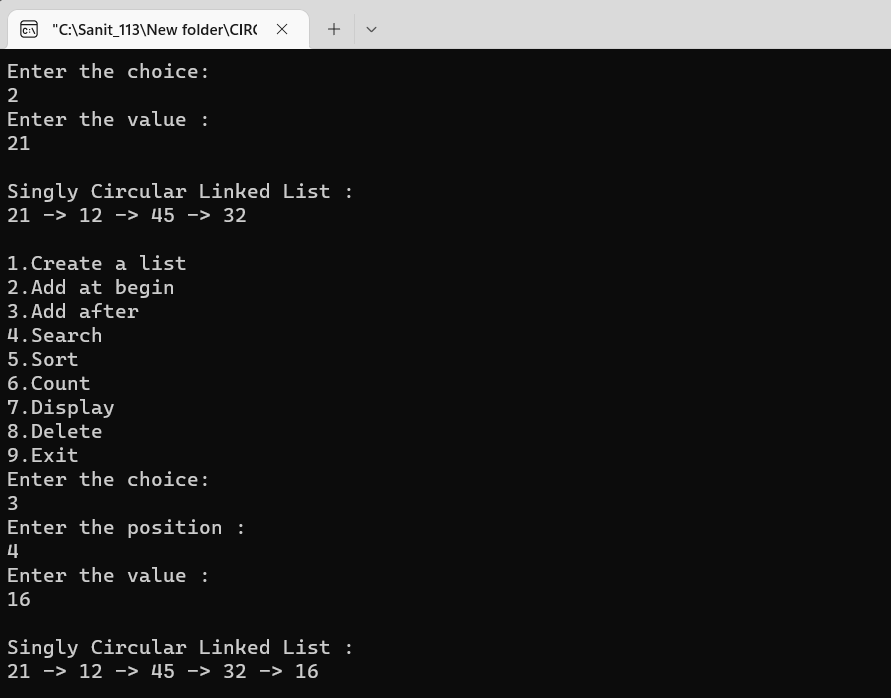
return 0;

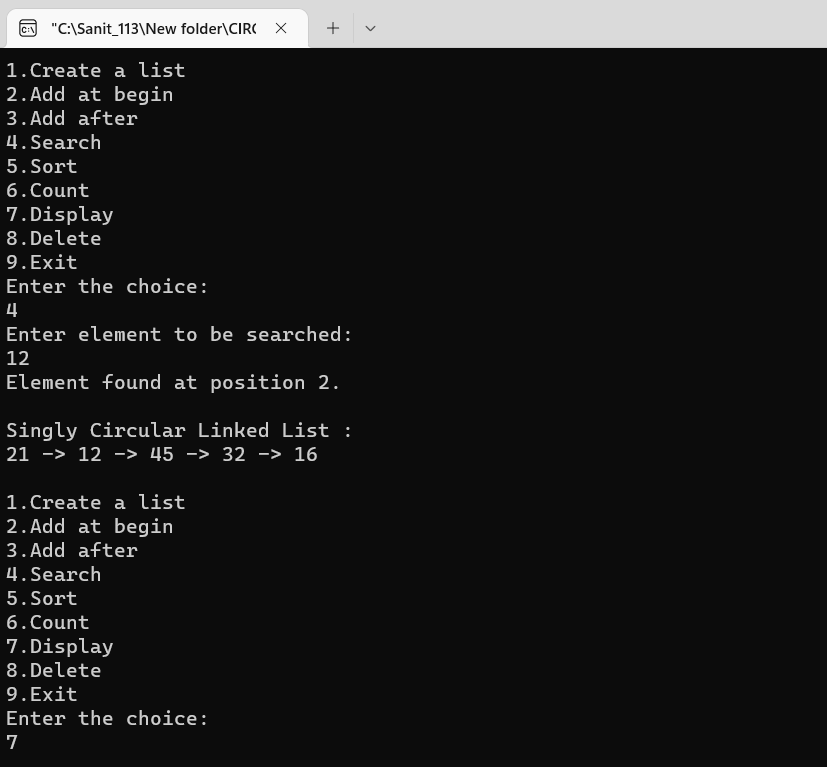
}

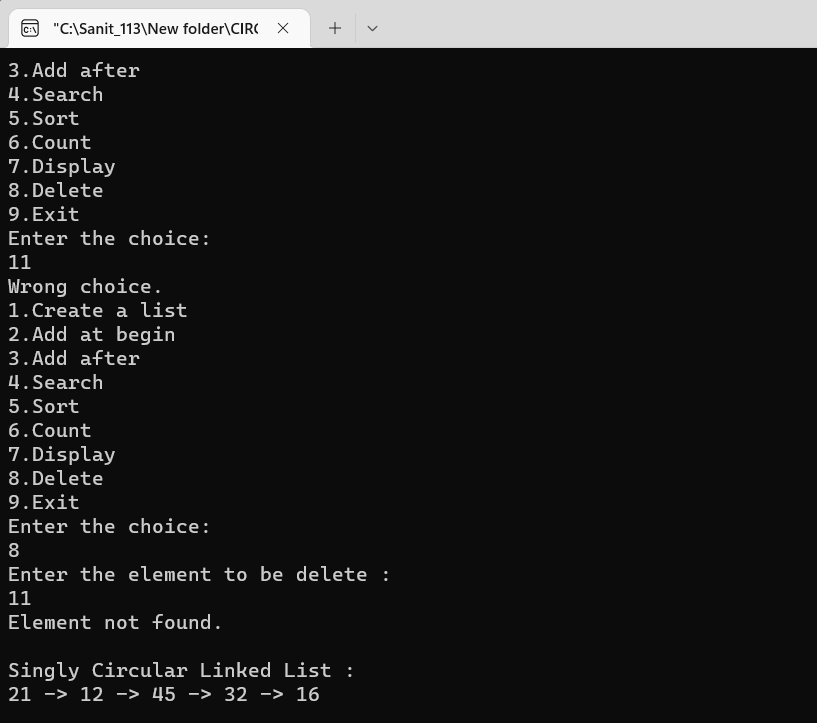
**Output:**











# Practical N0: 06

## Write A C++ Program to Implement a Binary Search Trees

#include<iostream> #include<process.h> #include<conio.h> using namespace std; struct node

{

int data;

struct node \*left; struct node \*right;

};

class BST

{

public:

node \*tree;

BST()

{

tree=NULL;

}

void createTree(node\*\*,int item); void preorder(node \*);

void inorder(node \*); void postorder(node \*); int totalNodes(node \*); void removeTree(node \*\*);

void findsmallestNode(node \*); void findLargestNode(node \*); void deleteNode(int);

};

void BST::createTree(node \*\*tree,int item)

{

if (\*tree==NULL)

{

}

else

{

\*tree=new node; (\*tree)->data=item; (\*tree)->left=NULL; (\*tree)->right=NULL;

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;

}

}

int BST::totalNodes(node \*tree)

{

if(tree==NULL) return 0;

else

}

return(totalNodes(tree->left)+totalNodes(tree->right)+1);

void BST::removeTree(node \*\*tree)

{

if((\*tree)!=NULL)

{

removeTree(&(\*tree)->left); removeTree(&(\*tree)->right); delete(\*tree);

}

}

void BST::findsmallestNode(node \*tree)

{

if(tree==NULL||tree->left==NULL) cout<<tree->data;

else

}

findsmallestNode(tree->left);

node \*find\_Insucc(node \*curr)

{

node \*succ=curr->right; if(succ!=NULL)

{

while(succ->left!=NULL) succ=succ->left;

}

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; while(curr!=NULL && flag!=1)

{

if(item<curr->data)

{

pred=curr; curr=curr->left;

}

else if(item>curr->data)

{

}

else

{

}

}

pred=curr; curr=curr->right;

flag=1;

if(flag==0)

{

cout<<"\n item does not exist:no deletion\n"; getch();

}

if(curr->left==NULL && curr->right==NULL) delcase=1;

else if(curr->left!=NULL && curr->right!=NULL) delcase=3;

else

delcase=2; if(delcase==1)

{

if(pred->left==curr) pred->left=NULL;

else

pred->right=NULL; delete(curr);

pred->right;

}

if(delcase==2)

{

if(pred->left==curr)

{

if(curr->left==NULL)

pred->left=curr->right;

}

else

{

else

pred->left=curr->left;

if(curr->left==NULL)

pred->right=curr->right; else

pred->right=curr->left; delete(curr);

}

if(delcase==3)

{

succ=find\_Insucc(curr); int item1=succ->data; deleteNode(item1); curr->data=item1;

}

}

}

int main()

{

BST obj; int choice;

int height=0,total=0,n,item; node \*\*tmp;

while(1)

{

cout<<"\n Binary search tree common operation\n"; cout<<"1)Create Tree \n";

cout<<"2)Traversal \n"; cout<<"3)Total Nodes\n"; cout<<"4)Remove Tree\n"; cout<<"5)Insert Nodes\n"; cout<<"6)Find Smallest Nodes \n"; cout<<"7)Find Largest Node \n"; cout<<"8)Delete Node\n"; cout<<"9)Exit\n";

cout<<"Enter your choice :"; cin>>choice;

switch(choice)

{

case 1:

cout<<"\n Creating Tree--- ";

cout<<"How many nodes u want to enter :"; cin>>n;

for(int i=0; i<n; i++)

{

cout<<"Enter Values :"; cin>>item; obj.createTree(&obj.tree,item);

}

break; case 2:

cout<<"\n Inorder Traversal :"; obj.inorder(obj.tree);

cout<<"\n preorder Traversal :"; obj.preorder(obj.tree); cout<<"\n Postorder Traversal :"; obj.postorder(obj.tree);

getch(); break;

case 3:

total=obj.totalNodes(obj.tree); cout<<"Total nodes :"<<total; getch();

break; case 4:

obj.removeTree(&obj.tree);

cout<<"\n Tree is removed from memory"; getch();

break; case 5:

cout<<"\n Insert node in a tree \n"; cout<<"Enter value :";

cin>>item; obj.createTree(&obj.tree,item); cout<<"\nItem is inserted\n"; getch();

break; case 6:

cout<<"\n\nSmallest node is:\n"; obj.findsmallestNode(obj.tree); getch();

break; case 7:

cout<<"\n\nLargest node is:\n"; obj.findLargestNode(obj.tree); getch();

break; case 8:

cout<<"\n\n Deleting a node from a tree--\n"; cout<<"Enter value=";

cin>>item; obj.deleteNode(item); break;

case 9:

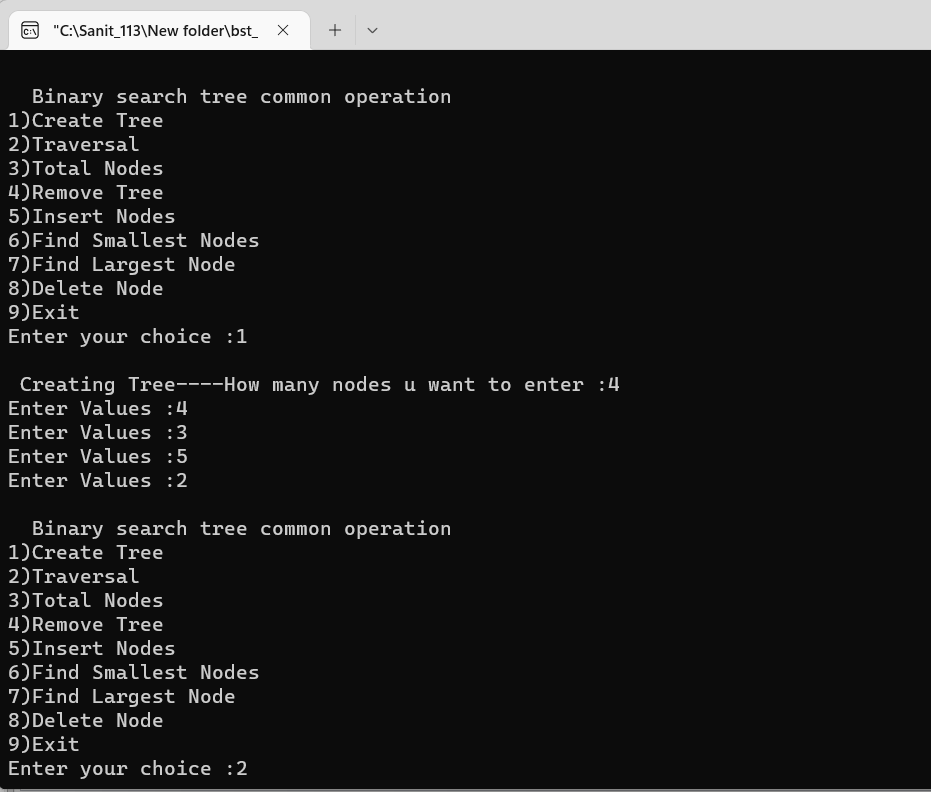
exit(1); break;

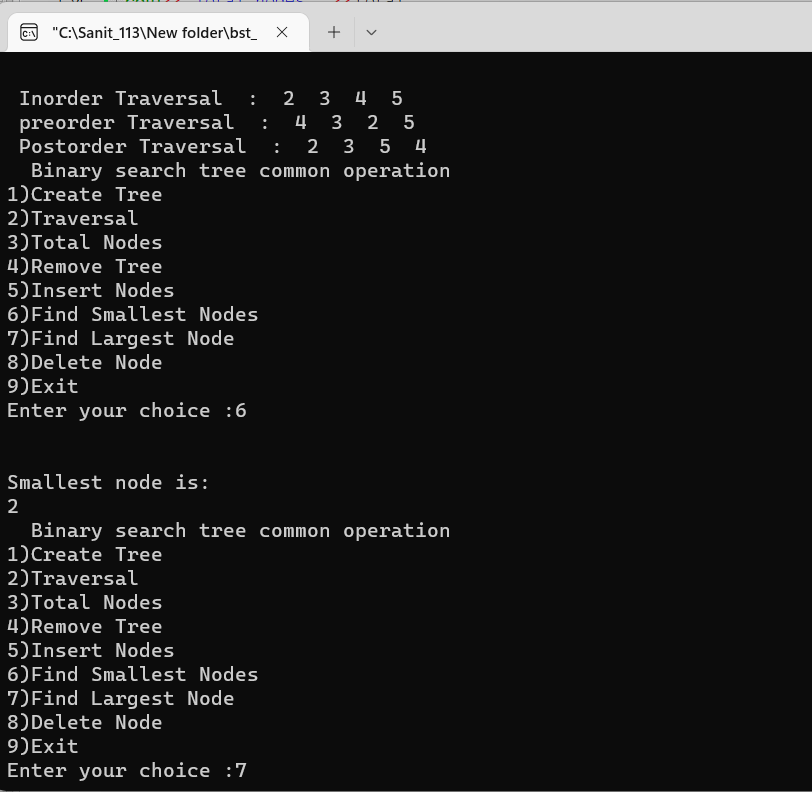
}

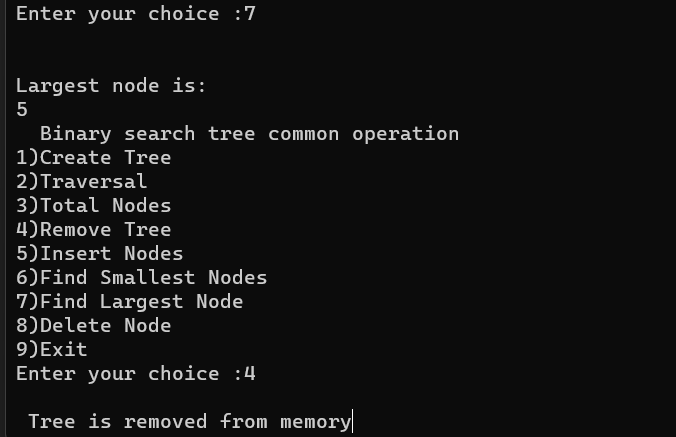
}

}

## Output:







* 1. **write a program to Implement Heap with different operation performed.**

## Min Heap program

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

void min\_heapify(int\* m, int i, int n)

{

int j, temp; temp = m[i]; j = 2 \* i; while (j <= n)

{

if (j < n &&m[j + 1] < m[j]) j = j + 1;

if (temp < m[j]) break;

else if (temp >= m[j])

{

m[j / 2] = m[j]; j = 2 \* j;

}

}

m[j / 2] = temp; return;

}

void build\_minheap(int\* a, int n)

{

int i;

for (i = n / 2; i>= 1; i--)

{

min\_heapify(a, i, n);

}

}

int main()

{

cout<< " FYMCA Roll No:113 DIV :- B \n"; 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\_minheap(a, n); cout<< "Min Heap\n"; for (i = 1; i<= n; i++)

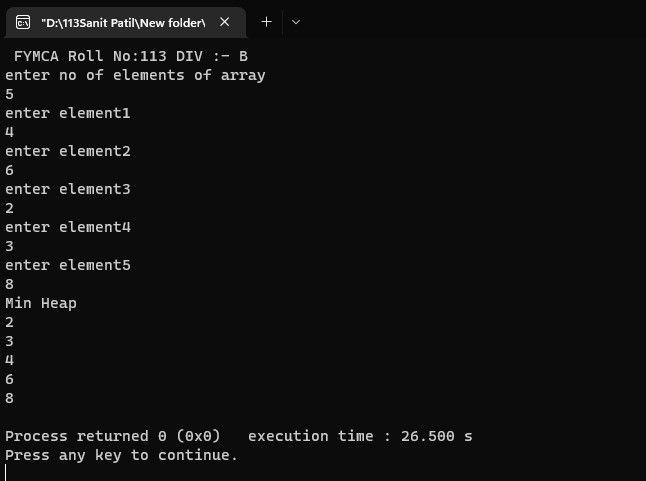
{

cout<< a[i] <<endl;

}

}

## Output:



* + 1. **Max Heap program**

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

void max\_heapify(int\* m, int i, int n)

{

int l, temp; temp = m[i]; l = 2 \* i; while (l <= n)

{

if (l < n &&m[l + 1] > m[l]) l = l + 1;

if (temp > m[l]) break;

else if (temp <= m[l])

{

m[l / 2] = m[l]; l = 2 \* l;

}

}

m[l / 2] = temp; return;

}

void build\_maxheap(int\* m, int n)

{

int i;

for (i = n / 2; i>= 1; i--)

{

max\_heapify(m, i, n);

}

}

int main()

{

cout<< " FYMCA Roll No:113 DIV :- B \n"; int n, i, y;

cout<< "enter no of elements of array\n"; cin>> n;

int m[20];

for (i = 1; i<= n; i++)

{

cout<< "enter element" << (i) <<endl; cin>> m[i];

}

build\_maxheap(m, n); cout<< "Max Heap\n";

for (i = 1; i<= n; i++)

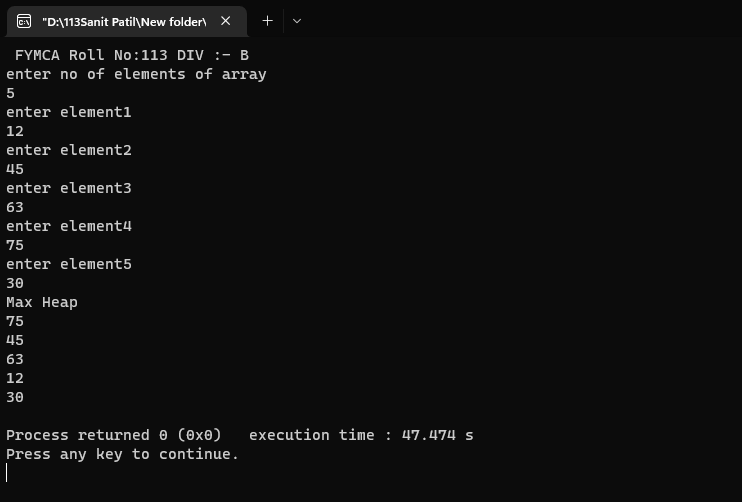
{

cout<< m[i] <<endl;

}

}

**Output:**



# Practical N0: 07

## write a program to implement a graph Storage Structure.

1. **Adjacency matrix program**

#include<iostream> using namespace std; #define MAX 20

class AdjacencyMatrix

{

private:

int n;

int \*\*adj; bool \*visited;

public:

AdjacencyMatrix(int n)

{

this->n=n; visited=new bool[n]; adj=new int\*[n]; for(int i=0; i<n; i++)

{

adj[i]=new int[n]; for(int j=0; j<n; j++)

{

adj[i][j]=0;

}

}

}

void add\_edge(int origin, int destin)

{

if(origin>n||destin>n||origin<0||destin<0)

{

cout<<"Invalid edge!\n";

}

else

{

adj[origin-1][destin-1]=1;

}

}

void display()

{

int i,j;

for(i=0; i<n; i++)

{

for(j=0; j<n; j++)

cout<<adj[i][j]<<" "; cout<<endl;

}

}

};

int main()

{

cout<<"FYMCA B Roll No:-113"<<endl; cout<<"Adjacency Matrix\n";

int nodes,max\_edges,origin,destin; cout<<"Enter number of nodes : "; cin>>nodes;

AdjacencyMatrix am(nodes); max\_edges=nodes\*(nodes-1); for(int i=0; i<max\_edges; i++)

{

cout<<"Enter edges(-1-1 to exit):"; cin>>origin>>destin;

if((origin==-1)&&(destin==-1)) break;

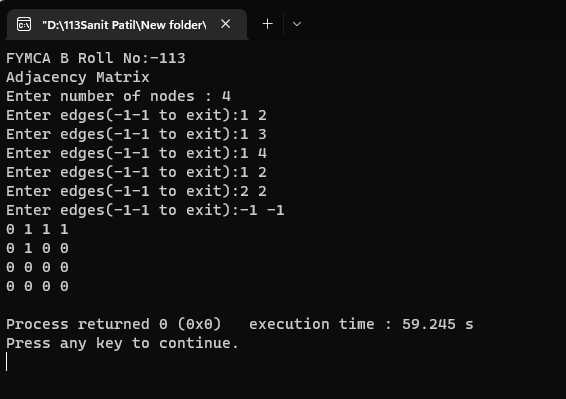
am.add\_edge(origin,destin);

}

am.display(); return 0;

}

## Output:



### write a program to Create a minimum spanning tree using any method Kruskal’s Algorithm

#include<iostream> #include<stdlib.h> #define max 30

using namespace std; struct edge

{

int weight; int u;

int v;

struct edge \*link;

};

struct edge \*frnt=NULL; struct edge \*tmp;

int i,j,wt;

int father[max]; struct edge tree[max]; int wt\_tree;

int cnt=0;

void make\_tree();

void insert\_tree(int i, int j, int wt); void insert\_pque(int i, int j, int wt); struct edge \*del\_pque();

void create\_graph()

{

int i, n, max\_edges, origin, destin; cout<<"Enter the no. of nodes : "; cin>>n;

max\_edges=n\*(n-1)/2; for(i=1; i<max\_edges; i++)

{

cout<<"Enter edges (0 0 to quit) weight : "; cin>>origin;

cin>>destin; if((origin==0)&&(destin==0))

break;

cout<<"Enter weight for this edge : "; cin>>wt;

if(origin>n || destin>n || origin<=0 || destin<=0)

{

cout<<"Invalid edge \n"; i--;

}

else

insert\_pque(origin,destin,wt);

}

if(i<n-1)

{

cout<<"Spanning tree is not possible \n"; exit(1);

}

}

void insert\_pque(int i,int j,int wt)

{

struct edge \*tmp, \*q;

tmp = (struct edge \*) malloc(sizeof(struct edge)); tmp->u=i;

tmp->v=j;

tmp->weight=wt;

if(frnt==NULL || tmp->weight<frnt->weight)

{

tmp->link=frnt; frnt=tmp;

}

else

{

q=frnt;

while(q->link!=NULL&&q->link->weight<=tmp->weight) q=q->link;

tmp->link=q->link; q->link=tmp;

if(q->link==NULL) tmp->link=NULL;

}

}

void make\_tree()

{

edge \*tmp;

int node1, node2, root\_n1, root\_n2, wt\_root=0,n,cnt=0; while(cnt<n-1)

{

tmp=del\_pque(); node1=tmp->u; node2=tmp->v; cout<<"N1 ="<<node1; cout<<"N2 ="<<node2; while(node1>0)

{

root\_n1=node1; node1=father[node1];

}

while(node2>0)

{

root\_n2=node2; node2=father[node2];

}

cout<<"root N1= "<<root\_n1; cout<<"root N2= "<<root\_n2; if(root\_n1!=root\_n2)

{

insert\_tree(tmp->u,tmp->v,tmp->weight); wt\_tree=wt\_tree+tmp->weight; father[root\_n2]=root\_n1;

}

}

}

void insert\_tree(int i, int j, int wt)

{

cout<<"\nThis edge inserted in the spanning tree \n"; cnt++;

tree[cnt].u=i; tree[cnt].v=j; tree[cnt].weight=wt;

}

struct edge \*del\_pque()

{

struct edge \* tmp; tmp=frnt;

cout<<"\nEdge processed \n"<<tmp->u; cout<<"\nEdge processed \n"<<tmp->v; cout<<"\nEdge processed \n"<<tmp->weight; frnt=frnt->link;

return tmp;

}

int main()

{

cout<<"FYMCA B Roll no :- 113\n"; int i,j,wt\_tree,cnt=0;

struct edge tree[max]; create\_graph(); make\_tree();

cout<<"\nEdges to be included in spanning tree \n"; for(i=1; i<=cnt; i++)

{

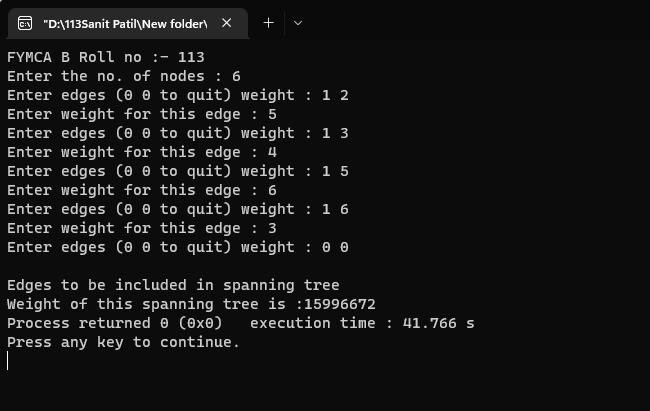
cout<<tree[i].u; cout<<tree[j].v;

}

cout<<"Weight of this spanning tree is :"<<wt\_tree; return 0;

}

## Output:



### Breath First Search Program

#include<iostream> #include<stdio.h> #define max 20

using namespace std; int adj[max][max]; bool visited[max]; int n,frnt;

void create\_graph()

{

int i,max\_edges,origin,destin; cout<<"Enter number of Nodes: "; cin>>n;

max\_edges=n\*(n-1); for(i=1; i<=max\_edges; i++)

{

cout<<"Enter edge(0 0 to quit) :"<<i<<"\n"; cin>>origin>>destin; if(origin==0||destin==0)

break; if(origin>n||destin>n||origin<=0||destin<=0)

{

cout<<"Invalid Edge \n"; i--;

}

else

{

adj[origin][destin]=1;

}

}

}

void display()

{

int i,j;

for(i=1; i<=n; i++)

{

for(j=1; j<=n; j++)

{

cout<<adj[i][j]<<"\t";

}

cout<<"\n";

}

}

void bfs(int v)

{

int i,frnt,rear;

int que[20]; frnt=rear=-1; cout<<v; visited[v]=true; rear++;

frnt++; que[rear]=v; while(frnt<=rear)

{

v=que[frnt]; frnt++;

for(i=1; i<n; i++)

{

if(adj[v][i]==1 && visited[i]==false)

{

cout<<i<<" "<<"\t"; visited[i]=true; rear++; que[rear]=i;

}

}

}

}

int main()

{

int i,v,ch; create\_graph(); while(1)

{

cout<<"\n1.Breadth first Search Using Stack\n2.exit\nEnter your choice\n";

cin>>ch; switch(ch)

{

case 1:

cout<<"Enter Starting node for Breadth First

Search:\n";

cin>>v;

for(i=1; i<=n; i++) visited[i]=false;

bfs(v); break;

case 2:

exit(0); break;

default:

cout<<"Wrong Choice";

break;

}

}

return 0;

}

## Output:

