# DSA PRACTICAL 1

## 1) Write a program to swap two numbers using function overloading.

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

void swap(int& a, int& b)
{
    int temp = a;
    a = b;
    b = temp;
}
void swap(float& a, float& b)
{
    float temp = a;
    a = b;
    b = temp;
}
void swap(char& a, char& b)
{
    char temp = a;
    a = b;
    b = temp;
}
int main()
{
    int num1, num2;
    cout<<"Enter 2 Numbers: ";
    cin>>num1>>num2;
    cout << "Before swapping:" << endl;
```

```
    cout << "num1: " << num1 << " num2: " << num2 <<
endl;
    swap(num1, num2);
    cout << "After swapping:" << endl;
    cout << "num1: " << num1 << " num2: " << num2 <<
endl;

    float float1 , float2 ;
    cout<<"Enter 2 Decimal Numbers: ";
    cin>>float1>>float2;
    cout << "Before swapping:" << endl;
    cout << "float1: " << float1 << " float2: " <<
float2 << endl;
    swap(float1, float2);
    cout << "After swapping:" << endl;
    cout << "float1: " << float1 << " float2: " <<
float2 << endl;

    char char1 , char2 ;
    cout<<"Enter 2 Characters: ";
    cin>>char1>>char2;
    cout << "Before swapping:" << endl;
    cout << "char1: " << char1 << " char2: " << char2
<< endl;
    swap(char1, char2);
    cout << "After swapping:" << endl;
    cout << "char1: " << char1 << " char2: " << char2
<< endl;

    return 0;
}
```

## OUTPUT:

Enter 2 Numbers: 2 3
Before swapping:

num1: 2 num2: 3
After swapping:
num1: 3 num2: 2
Enter 2 Decimal Numbers: 1.2 4.5
Before swapping:
float1: 1.2 float2: 4.5
After swapping:
float1: 4.5 float2: 1.2
Enter 2 Characters: a f
Before swapping:
char1: a char2: f
After swapping:
char1: f char2: a

## 2) Write a program to demonstrate default arguments.

```
#include<iostream>
using namespace std;
int mul(int x,int y=5,int z=8)
{
    return x*y*z;
}
int main()
{
    int a,b,c;
    cout<<"\nEnter 3 nos: ";
    cin>>a>>b>>c;
    cout<<"Result is: "<<mul(a,b,c)<<endl;
    cout<<"Result is: "<<mul(a,b)<<endl;
    cout<<"Result is: "<<mul(a)<<endl;
    cout<<"Result is: "<<mul(b)<<endl;
    cout<<"Result is: "<<mul(a,c)<<endl;
    return 0;
}
```

## OUTPUT:

Enter 3 nos: 1 2 5

Result is: 10
Result is: 16
Result is: 40
Result is: 80
Result is: 40

### 3) Write a program to illustrate dynamic memory allocation and deallocation.

```
#include<iostream>
using namespace std;
int main()
{
    int *a = new int;
    int *b = new int;
    *a = 5;
    *b = 6;
    *a = (*a)*(*a);
    cout<<"\na: "<<*a;
    delete a;
    *b = (*b)*(*b);
    cout<<"\nb: "<<*b;
    delete b;
}
```

## OUTPUT:

a: 25
b: 36

### 4) Write a program to demonstrate dynamic memory allocation and deallocation using arrays.

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

int main()
```

```
{
    int size;
    cout << "Enter the size of the array: ";
    cin >> size;
    int* arr = new int[size];
    cout << "Enter the elements of the array:" << endl;
    for (int i = 0; i < size; i++) {
        cin >> arr[i];
    }
    cout << "Elements of the array:" << endl;
    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
    delete[] arr;
    return 0;
}
```

**OUTPUT:**

Enter the size of the array: 3
Enter the elements of the array:
15 44 87
Elements of the array:
15 44 87

## 5) Write a program to illustrate the concept of class with member functions.

```
#include<iostream>
using namespace std;
class Student
{
    private:
        int rno;
        char name[50];
        float cgpa;
    public:
```

```
        void getdata()
        {
            cout<<"\nEnter student details
rno,name,cgpa: ";
            cin>>rno>>name>>cgpa;
        }
        void putdata()
        {
            cout<<"\nThe students details are: ";
            cout<<"\nRno: "<<rno<<"\nName:
"<<name<<"\nCGPA: "<<cgpa;
        }
};
int main()
{
    Student s;
    s.getdata();
    s.putdata();
}
```

## OUTPUT:

Enter student details rno,name,cgpa: 19 Sushma 9.54

The students details are:
Rno: 19
Name: Sushma
CGPA: 9.54

## 6) Write a program to demonstrate constructors.

```
#include<iostream>
using namespace std;
class Integer
{
    private:
        int a,b;
```

```
    public:
        Integer()
        {
            a=b=10;
        }
        Integer(int x,int y)
        {
            a=x;
            b=y;
        }
        Integer(Integer &i)
        {
            a=i.a;
            b=i.b;
        }
        void display()
        {
            cout<<"\n The values of a and b are:
"<<a<<" "<<b<<endl;
        }
};
int main()
{
    Integer i1;
    i1.display();
    Integer i2(4,5);
    i2.display();
    Integer i3(i2);
    i3.display();
    return 0;
}
```

## OUTPUT:

The values of a and b are: 10 10
The values of a and b are: 4 5

The values of a and b are: 4 5

## 7) Write a program to demonstrate destructors.

```
#include<iostream>
using namespace std;
int c=0;
class test
{
    public:
        test()
        {
            c++;
            cout<<"\nobject created : "<<c;
        }
        ~test()
        {
            cout<<"\nobject destroyed: "<<c;
            c--;
        }
};
int main()
{
    cout<<"\nEntered into main()";
    test t1,t2,t3;
    {
        cout<<"\nEntered into block 1";
        test t4;
    }
    {
    cout<<"\nEntered into block 2: ";
    test t5;
    }
    cout<<"\nReentered into main";
    return 0;
```

```
}
```

## OUTPUT:

Entered into main()
object created : 1
object created : 2
object created : 3
Entered into block 1
object created : 4
object destroyed: 4
Entered into block 2:
object created : 4
object destroyed: 4
Reentered into main
object destroyed: 3
object destroyed: 2
object destroyed: 1

## 8) Write a program to illustrate Simple Inheritance.

```
#include<iostream>
using namespace std;
class student
{
    private:
        int rno;
        char name[20];
    public:
        void get_details()
        {
            cout<<"\nEnter rno and name: ";
            cin>>rno>>name;
        }
        void put_details()
        {
            cout<<"\nName: "<<name<<"\nrno: "<<rno;
        }
```

```
};
class marks:public student{
    private:
        float marks;
    public:
        void get_marks()
        {
            cout<<"\nEnter marks: ";
            cin>>marks;
        }
        void put_marks()
        {
            cout<<"\nMarks: "<<marks;
        }
};
int main()
{
    marks obj;
    obj.get_details();
    obj.get_marks();
    obj.put_details();
    obj.put_marks();
    return 0;
}
```

## OUTPUT:

Enter rno and name: 19 Sushma

Enter marks: 43

Name: Sushma
rno: 19
Marks: 43

## 9) Write a program to illustrate Multilevel Inheritance.

```
#include<iostream>
using namespace std;
class student
{
    private:
        int rno;
        char name[20];
    public:
        void get_details()
        {
            cout<<"\nEnter rno and name: ";
            cin>>rno>>name;
        }
        void put_details()
        {
            cout<<"\nName: "<<name<<"\nrno: "<<rno;
        }
};
class marks:public student
{
    protected:
        float m1,m2;
    public:
        void get_marks()
        {
            cout<<"\nEnter marks: ";
            cin>>m1>>m2;
        }
        void put_marks()
        {
            cout<<"\nMarks: "<<m1<<" "<<m2;
        }
};
```

```
class test:public marks
{
    private:
    float total;
    public:
        void display()
        {
            total=m1+m2;
            cout<<"\nTotal: "<<total;
        }
};
int main()
{
    test obj;
    obj.get_details();
    obj.get_marks();
    obj.put_details();
    obj.put_marks();
    obj.display();
    return 0;
}
```

## OUTPUT:

Enter rno and name: 19 Sushma

Enter marks: 23 33

Name: Sushma
rno: 19
Marks: 23 33
Total: 56

## 10) Write a program to illustrate Multiple Inheritance.

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

```
class student
{
    protected:
        int rno;
        char name[20];
    public:
        void get_details()
        {
            cout<<"\nEnter  rno and name: ";
            cin>>rno>>name;
        }
};
class marks {
   protected:
        float m;
    public:
        void get_marks()
        {
            cout<<"\nEnter marks: ";
            cin>>m;
        }

};
class displaying:public student,public marks
{
    public:
        void display()
        {
            cout<<"\Student details are: ";
            cout<<"\nName: "<<name<<"\nRno:
"<<rno<<"\nMarks: "<<m;

        }
} ;
int main()
```

```
{
    displaying obj;
    obj.get_details();
    obj.get_marks();
    obj.display();
    return 0;
}
```

## OUTPUT:

Enter  rno and name: 19 Sushma

Enter marks: 13

Student details are:
Name: Sushma
Rno: 19
Marks: 13

## 11) Write a program to illustrate Hierarchical Inheritance.

```
#include<iostream>
using namespace std;
class student
{
    protected:
        int rno;
        char name[30];
    public:
        void get_details()
        {
            cout<<"\nEnter rno and name: ";
            cin>>rno>>name;
        }
        void put_details()
        {
            cout<<"\nThe student details are: ";
```

```
            cout<<"\nName: "<<name<<"\nrno : "<<rno;
        }
};
class science: public student
{
    protected:
        float s1;
    public:
        void get_scmarks()
        {
            cout<<"\nEnter science marks: ";
            cin>>s1;
        }
        void put_scmarks()
        {
            cout<<"\nScience marks are: "<<s1;
        }
};
class arts:public student{
    protected:
        float a1;
    public:
        void get_artsmarks()
        {
            cout<<"\nEnter arts marks: ";
            cin>>a1;
        }
        void put_artsmarks()
        {
            cout<<"\nArts marks are: "<<a1;
        }
} ;
int main()
{
    science obj1;
```

```
    obj1.get_details();
    obj1.get_scmarks();
    obj1.put_details();
    obj1.put_scmarks();
    arts obj2;
    obj2.get_details();
    obj2.get_artsmarks();
    obj2.put_details();
    obj2.put_artsmarks();
    return 0;
}
```

## OUTPUT:

Enter rno and name: 19 Sushma

Enter science marks: 54

The student details are:
Name: Sushma
rno : 19
Science marks are: 54

Enter rno and name: 16 Laxmi

Enter arts marks: 60

The student details are:
Name: Laxmi
rno : 16
Arts marks are: 60

## 12) Write a program to illustrate Hybrid Inheritance.

```
#include<iostream>
using namespace std;
class student
{
```

```
    protected:
        int rno;
        char name[20];
    public:
        void getdata()
        {
            cout<<"\nEnter rno and name: ";
            cin>>rno>>name;
        }
        void putdata()
        {
            cout<<"\nThe student details are: ";
            cout<<"\nName: "<<name<<"\n rno: "<<rno;
        }
};
class marks:public student
{
    protected:
        float m1,m2;
    public:
        void get_marks()
        {
            cout<<"\nEnter two subject marks: ";
            cin>>m1>>m2;
        }
        void put_marks()
        {
            cout<<"\nMarks are: "<<m1<<"\t"<<m2;
        }
};
class sports
{
    protected:
        float score;
    public:
```

```
        void get_score()
        {
            cout<<"\nEnter score: ";
            cin>>score;
        }
        void put_score()
        {
            cout<<"\nScore is: "<<score;
        }
};
class result:public marks,public sports
{
    protected:
        float res;
    public:
        void display()
        {
            res=m1+m2+score;
            cout<<"\nThe result is: "<<res;
        }
};
int main()
{
    result obj;
    obj.getdata();
    obj.get_marks();
    obj.get_score();
    obj.putdata();
    obj.put_marks();
    obj.put_score();
    obj.display();
}
```

## OUTPUT:

Enter rno and name: 19 Sushma

Enter two subject marks: 34 33

Enter score: 9.8

The student details are:
Name: Sushma
 rno: 19
Marks are: 34  33
Score is: 9.8
The result is: 76.8

## 13) Write a program to demonstrate Stack operations.

```
#include <iostream>
using namespace std;
#define max 10
class Stack
{
    private:
        int a[max];
        int top;
    public:
        Stack()
        {
            top = -1;
        }
        void push(int x)
        {
            if (top < max-1)
            {
                a[++top] = x;
                cout << "Pushed element " << x << "
into the stack" << endl;
            }
            else
                cout<<"\nStack is full";
```

```
        }
        void pop()
        {
            if (top==-1)
                cout << "Stack is empty";
            else
            {
                int temp = a[top];
                a[top]= NULL;
                top--;
                cout << "Popped element " << temp << "
from the stack" << endl;
            }
        }
        void display()
        {
            if (top==-1)
                cout << "Stack is empty";
            else
            {
                cout<<"\nStack Elements are: "<<endl;
                for(int i = top; i>=0;--i)
                        cout<<a[i]<<" ";
            }
        }
};
int main()
{
    Stack S;
    int choice,ele;
    while(true)
    {
        cout<<"\n\nStack Operations are:
\n(1)Push\n(2)Pop\n(3)Display\n(4)Exit\n";
        cout<<"Enter your choice: ";
```

```
        cin>>choice;
        if(choice==1)
        {
            cout<<"\nEnter an element to push into the
stack: ";
            cin>>ele;
            S.push(ele);
        }
        if(choice==2)
           S.pop();
        if(choice==3)
            S.display();
        if(choice==4)
        {
            cout<<"\nOut of the Stack";
            exit(0);
        }
        if(choice<1 || choice>4)
            cout<<"Invalid Input";
    }
    return 0;
}
```

## OUTPUT:

Stack Operations are:
(1)Push
(2)Pop
(3)Display
(4)Exit
Enter your choice: 1

Enter an element to push into the stack: 584
Pushed element 584 into the stack

Stack Operations are:
(1)Push

(2)Pop
(3)Display
(4)Exit
Enter your choice: 1

Enter an element to push into the stack: 652
Pushed element 652 into the stack

Stack Operations are:
(1)Push
(2)Pop
(3)Display
(4)Exit
Enter your choice: 1

Enter an element to push into the stack: 952
Pushed element 952 into the stack

Stack Operations are:
(1)Push
(2)Pop
(3)Display
(4)Exit
Enter your choice: 3

Stack Elements are:
952 652 584

Stack Operations are:
(1)Push
(2)Pop
(3)Display
(4)Exit
Enter your choice: 2
Popped element 952 from the stack

Stack Operations are:

(1)Push
(2)Pop
(3)Display
(4)Exit
Enter your choice: 4

Out of the Stack

## 14) Write a program to demonstrate Queue operations.

```
#include <iostream>
using namespace std;
#define max 10
class Queue
{
    private:
        int q[max];
        int front;
        int rear;
    public:
        Queue()
        {
            front = rear = -1;
        }
        void insertion(int x)
        {
            if (rear<max-1)
            {
                rear++;
                q[rear]=x;
                cout<<"\nThe inserted element is:
"<<q[rear];
                if(front==-1)
                    front = 0;
                else
                    cout<<"\nQueue is full";
```

```
            }
        }
        void deletion()
        {
            if (front==-1)
                cout << "Queue is empty";
            else
            {
                cout<<"\nThe deleted element is:
"<<q[front];
                q[front] = NULL;
                front++;
                if(front>rear)
                    front = rear = -1;
            }
        }
        void display()
        {
            if (front==-1)
                cout << "Queue is empty";
            else
            {
                cout << "Elements in the queue: ";
                for (int i = front; i <= rear; ++i)
                {
                    cout << q[i] << " ";
                }
                cout << endl;
            }
        }
};
int main()
{
    Queue q;
    int choice,ele;
```

```
    while(true)
    {
        cout<<"\nQueue Operations are:
\n(1)Insertion\n(2)Deletion\n(3)Display\n(4)Exit\n";
        cout<<"Enter your choice: ";
        cin>>choice;
        if(choice==1)
        {
            cout<<"\nEnter an element: ";
            cin>>ele;
            q.insertion(ele);
        }
        if(choice==2)
           q.deletion();
        if(choice==3)
            q.display();
        if(choice==4)
        {
            cout<<"\nOut of the Queue";
            exit(0);
        }
        if(choice<1 || choice>4)
            cout<<"Invalid Input";
    }
    return 0;
}
```

## OUTPUT:

Queue Operations are:
(1)Insertion
(2)Deletion
(3)Display
(4)Exit
Enter your choice: 1

Enter an element: 10

```
The inserted element is: 10
Queue Operations are:
(1)Insertion
(2)Deletion
(3)Display
(4)Exit
Enter your choice: 1

Enter an element: 20

The inserted element is: 20
Queue is full
Queue Operations are:
(1)Insertion
(2)Deletion
(3)Display
(4)Exit
Enter your choice: 1

Enter an element: 30

The inserted element is: 30
Queue is full
Queue Operations are:
(1)Insertion
(2)Deletion
(3)Display
(4)Exit
Enter your choice: 2

The deleted element is: 10
Queue Operations are:
(1)Insertion
(2)Deletion
(3)Display
(4)Exit
Enter your choice: 3
Elements in the queue: 20 30
```

Queue Operations are:
(1)Insertion
(2)Deletion
(3)Display
(4)Exit
Enter your choice: 4

Out of the Queue

## 15) Write a program to demonstrate a Singly linked list.

```
#include<iostream>
using namespace std;
class linkedList
{
     struct node
      {
               int data;
              node *ptr;
     } *p;
     public:
          linkedList()
          {
               p = NULL;
          }
          void InsBeg(int x)
          {
               node *q;
               q = new node;
               q->data = x;
               q->ptr = p;
               p = q;
          }
          void InsEnd(int x)
          {
               node *q, *r;
```

```
    if (p == NULL)
    {
        p = new node;
        p->data = x;
        p->ptr = NULL;
    }
    else
    {
        q = p;
        while (q->ptr != NULL)
        {
            q = q->ptr;
        }
        r = new node;
        r->data = x;
        r->ptr = NULL;
        q->ptr = r;
    }
}
void insertAfter(int x, int z)
{
    node *q, *r;
    q = p;
    while (q != NULL)
    {
        if (q->data == z)
        {
                r = new node;
                r->data = x;
                r->ptr = q->ptr;
                q->ptr = r;
                q = NULL;
        }
        else
        {
```

```
                    q = q->ptr;
                }
            }
        }
        void insByPos(int x, int c)
        {
            node *q, *r;
            q = p;
            for (int i = 1; i < c; i++)
            {
                q = q->ptr;
                if (q == NULL)
                {
                    cout <<"\nLess Position";
                    exit(0);
                }
            }
            r = new node;
            r->data = x;
            r->ptr = q->ptr;
            q->ptr = r;
        }
        void deleteBegin()
        {
            node *q;
            if (p == NULL)
            {
                cout <<"\nList is empty";
            }
            else
            {
                q = p;
                p = p->ptr;
                cout <<"\nThe deleted element is: "<<
q->data;
```

```
                delete q;
            }
        }
        void deleteEnd()
        {
            node *q, *r;
            q = p;
            if (p == NULL)
            {
                cout <<"\nList is empty";
            }
            else
            {
                while (q != NULL)
                {
                    if ((q->ptr)->ptr == NULL)
                    {
                        r = q->ptr;
                        cout <<"\nThe deleted element
is: "<< r->data;
                        delete r;
                        q->ptr = NULL;
                        q = NULL;
                    }
                    else
                    {
                        q = q->ptr;
                    }
                }
            }
        }
        void delMid(int z)
        {
            node *q, *r;
            q = p;
```

```
            if (p == NULL)
            {
                cout <<"\nList is empty";
            }
            else
            {
                if (q->data == z)
                {
                    p = p->ptr;
                    cout <<"\nThe deleted element is:
"<< q->data;
                    delete q;
                }
                else
                {
                    while (q != NULL)
                    {
                        if (q->data == z)
                        {
                            r->ptr = q->ptr;
                            cout <<"\nThe deleted
element is: "<< q->data;
                            delete q;
                            q = NULL;
                        }
                        else
                        {
                            r = q;
                            q = q->ptr;
                        }
                    }
                }
            }
        }
        void display()
```

```
        {
            node *q;
            if (p == NULL)
            {
                cout <<"\nList is empty";
            }
            else
            {
                cout <<"\nThe element of the list are:
";
                q = p;
                while (q != NULL)
                {
                    cout <<"\nThe element is: "<<
q->data;
                    q = q->ptr;
                }
            }
        }
    };
    main()
    {
        linkedList l;
        int x, choice, z;
        char ch = 'Y';
        while (ch == 'Y' || ch == 'y')
        {
            cout << "1.Insert Begin"<<endl;
            cout << "2.Insert End"<<endl;
            cout << "3.Insert After"<<endl;
            cout << "4.Insert By Pos"<<endl;
            cout << "5.Delete Begin"<<endl;
            cout << "6.Delete End"<<endl;
            cout << "7.Delete Middle"<<endl;
            cout << "8.Display"<<endl;
```

```
            cout << "9.Exit"<<endl;
            cout << "Enter your choice: ";
            cin >> choice;
            switch (choice)
            {
                case 1:
                    cout <<"\nEnter element to insert:
";
                    cin >> x;
                    l.InsBeg(x);
                    break;
                case 2:
                    cout <<"\nEnter element to insert:
";
                    cin >> x;
                    l.InsEnd(x);
                    break;
                case 3:
                    cout <<"\nEnter element to insert:
";
                    cin >> x;
                    cout <<"\nEnter after which
element to make the insertion: ";
                    cin >> z;
                    l.insertAfter(x, z);
                    break;
                case 4:
                    cout <<"\nEnter element to insert:
";
                    cin >> x;
                    cout <<"\nEnter position: ";
                    cin >> z;
                    l.insByPos(x, z);
                    break;
                case 5:
```

```
                    cout <<"\nDelete at begin: ";
                    l.deleteBegin();
                    break;
                case 6:
                    cout <<"\nDelete at end: ";
                    l.deleteEnd();
                    break;
                case 7:
                    cout <<"\nDelete middle";
                    cout <<"\nEnter value to delete:
";
                    cin >> z;
                    l.delMid(z);
                    break;
                case 8:
                    cout <<"\nLinkedList is: ";
                    l.display();
                    break;
                case 9:
                    exit(0);
                default:
                    cout <<"\nInvalid choice";
                    break;
            }
            cout <<"\nDo you want to continue? (Y/N):
";
            cin >> ch;
        }
}
```

## OUTPUT:

1.Insert Begin
2.Insert End
3.Insert After
4.Insert By Pos
5.Delete Begin

6.Delete End
7.Delete Middle
8.Display
9.Exit
Enter your choice: 1

Enter element to insert: 10

Do you want to continue? (Y/N): y
1.Insert Begin
2.Insert End
3.Insert After
4.Insert By Pos
5.Delete Begin
6.Delete End
7.Delete Middle
8.Display
9.Exit
Enter your choice: 1

Enter element to insert: 20

Do you want to continue? (Y/N): y
1.Insert Begin
2.Insert End
3.Insert After
4.Insert By Pos
5.Delete Begin
6.Delete End
7.Delete Middle
8.Display
9.Exit
Enter your choice: 2

Enter element to insert: 30

Do you want to continue? (Y/N): y
1.Insert Begin
2.Insert End
3.Insert After

4.Insert By Pos
5.Delete Begin
6.Delete End
7.Delete Middle
8.Display
9.Exit
Enter your choice: 2

Enter element to insert: 40

Do you want to continue? (Y/N): y
1.Insert Begin
2.Insert End
3.Insert After
4.Insert By Pos
5.Delete Begin
6.Delete End
7.Delete Middle
8.Display
9.Exit
Enter your choice: 3

Enter element to insert: 50

Enter after which element to make the insertion: 10

Do you want to continue? (Y/N): y
1.Insert Begin
2.Insert End
3.Insert After
4.Insert By Pos
5.Delete Begin
6.Delete End
7.Delete Middle
8.Display
9.Exit
Enter your choice: 4

Enter element to insert: 60

```
Enter position: 3

Do you want to continue? (Y/N): y
1.Insert Begin
2.Insert End
3.Insert After
4.Insert By Pos
5.Delete Begin
6.Delete End
7.Delete Middle
8.Display
9.Exit
Enter your choice: 8

LinkedList is:
The element of the list are:
The element is: 20
The element is: 10
The element is: 50
The element is: 60
The element is: 30
The element is: 40
Do you want to continue? (Y/N): y
1.Insert Begin
2.Insert End
3.Insert After
4.Insert By Pos
5.Delete Begin
6.Delete End
7.Delete Middle
8.Display
9.Exit
Enter your choice: 5

Delete at begin:
The deleted element is: 20
Do you want to continue? (Y/N): y
1.Insert Begin
2.Insert End
3.Insert After
```

4.Insert By Pos
5.Delete Begin
6.Delete End
7.Delete Middle
8.Display
9.Exit
Enter your choice: 6

Delete at end:
The deleted element is: 40
Do you want to continue? (Y/N): y
1.Insert Begin
2.Insert End
3.Insert After
4.Insert By Pos
5.Delete Begin
6.Delete End
7.Delete Middle
8.Display
9.Exit
Enter your choice: 7

Delete middle
Enter value to delete: 60

The deleted element is: 60
Do you want to continue? (Y/N): y
1.Insert Begin
2.Insert End
3.Insert After
4.Insert By Pos
5.Delete Begin
6.Delete End
7.Delete Middle
8.Display
9.Exit
Enter your choice: 8

LinkedList is:
The element of the list are:

The element is: 10
The element is: 50
The element is: 30
Do you want to continue? (Y/N): n

## 16) Write a program to demonstrate a Doubly linked list.

```
#include<iostream>
using namespace std;
class DoubleLinkedList
{
    struct Node
    {
        int data;
        Node *lptr;
        Node *rptr;
    }*p;
    public:
        DoubleLinkedList()
        {
            p = NULL;
        }
        void insertBeginning(int x)
        {
            Node *q;
            if (p == NULL)
            {
                p = new Node;
                p->lptr = NULL;
                p->data = x;
                p->rptr = NULL;
            }
            else
            {
                q = new Node;
                q->lptr = NULL;
```

```
            q->data = x;
            q->rptr = p;
            p->lptr = q;
            p = q;
        }
}
void insertEnd(int x)
{
    Node *q, *r;
    if (p == NULL)
    {
        p = new Node;
        p->data = x;
        p->lptr = NULL;
        p->rptr = NULL;
    }
    else
    {
        q = p;
        while (q->rptr != NULL)
        {
            q = q->rptr;
        }
        r = new Node;
        r->data = x;
        r->lptr = q;
        r->rptr = NULL;
        q->rptr = r;
    }
}
void insertAfter(int x, int element)
{
    Node *q, *r;
    q = p;
    if (p == NULL)
```

```
            {
                cout << "List is empty." << endl;
                return;
            }
            while (q != NULL)
            {
                if (q->data == element)
                {
                    r = new Node;
                    r->data = x;
                    r->lptr = q;
                    r->rptr = q->rptr;
                    if (q->rptr != NULL)
                    {
                        (q->rptr)->lptr = r;
                    }
                    q->rptr = r;
                    return;
                }
                q = q->rptr;
            }
            cout << "Element " << element << " not
found in the list." << endl;
        }
        void insertByPosition(int x, int position)
        {
            Node *q, *r;
            q = p;
            for (int i = 1; i < position; i++)
            {
                q = q->rptr;
                if (q == NULL)
                {
                    cout << "\nLess positions." <<
endl;
```

```
                    return;
                }
            }
            r = new Node;
            r->data = x;
            r->lptr = q;
            r->rptr = q->rptr;
            if (q->rptr != NULL)
            {
                (q->rptr)->lptr = r;
            }
            q->rptr = r;
        }
        void deleteBeginning()
        {
            Node *q;
            if (p == NULL)
                cout << "List is empty." << endl;
            q = p;
            p = p->rptr;
            if (p != NULL)
            {
                p->lptr = NULL;
            }
            cout << "Deleted element is: " << q->data
<< endl;
            delete q;
        }
        void deleteEnd()
        {
            Node *q, *r;
            q = p;
            if (p == NULL)
                cout << "List is empty." << endl;
            while (q->rptr != NULL)
```

```
            {
                q = q->rptr;
            }
            r = q;
            q = q->lptr;
            if (q != NULL)
            {
                q->rptr = NULL;
            }
            cout << "Deleted element is: " << r->data
<< endl;
            delete r;
        }
        void deleteMiddle(int x)
        {
            Node *q, *r;
            q = p;
            if (p == NULL)
                cout << "List is empty." << endl;
            while (q != NULL)
            {
                if (q->data == x)
                {
                    if (q == p)
                    {
                        p = p->rptr;
                        if (p != NULL)
                        {
                            p->lptr = NULL;
                        }
                        cout << "Deleted element is: "
<< q->data << endl;
                        delete q;
                    }
                    else
```

```
                    {
                        r = q;
                        q = q->rptr;
                        r->lptr->rptr = q;
                        if (q != NULL)
                        {
                            q->lptr = r->lptr;
                        }
                        cout << "Deleted element is: "
<< r->data << endl;
                        delete r;
                    }
                }
                q = q->rptr;
            }
            cout << "Element " << x << " not found in
the list." << endl;
        }
        void display()
        {
            Node *q;
            if (p == NULL)
                cout << "List is empty." << endl;
            q = p;
            cout << "Elements of the list are: ";
            while (q != NULL)
            {
                cout << q->data << " ";
                q = q->rptr;
            }
            cout << endl;
        }
    };
int main()
{
```

```
    DoubleLinkedList list;
    int element, elementAfter, position, choice;
    while (true)
    {
        cout << "Linked List Operations:\n";
        cout << "1. Insert at Beginning\n";
        cout << "2. Insert at End\n";
        cout << "3. Insert after an Element\n";
        cout << "4. Insert by Position\n";
        cout << "5. Delete Beginning Element\n";
        cout << "6. Delete End Element\n";
        cout << "7. Delete an Element From the List\n";
        cout << "8. Display the List\n";
        cout << "9. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;
        switch (choice)
        {
           case 1:
               cout << "Enter an element to insert: ";
               cin >> element;
               list.insertBeginning(element);
               break;
           case 2:
               cout << "Enter an element to insert: ";
               cin >> element;
               list.insertEnd(element);
               break;
           case 3:
               cout << "Enter the element to insert:
";
               cin >> element;
               cout << "Enter the element after which
to insert: ";
               cin >> elementAfter;
```

```
                list.insertAfter(element,
elementAfter);
                break;
            case 4:
                cout << "Enter the position and element
to insert: ";
                cin >> position >> element;
                list.insertByPosition(element,
position);
                break;
            case 5:
                list.deleteBeginning();
                break;
            case 6:
                list.deleteEnd();
                break;
            case 7:
                cout << "Enter the element to delete
from the List: ";
                cin >> element;
                list.deleteMiddle(element);
                break;
            case 8:
                list.display();
                break;
            case 9:
                cout << "Linked list is closed." <<
endl;
                exit(0);
            default:
                cout << "Invalid choice." << endl;
                break;
        }
    }
    return 0;
```

```
}
```

## OUTPUT:

```
Linked List Operations:
1. Insert at Beginning
2. Insert at End
3. Insert after an Element
4. Insert by Position
5. Delete Beginning Element
6. Delete End Element
7. Delete an Element From the List
8. Display the List
9. Exit
Enter your choice: 1
Enter an element to insert: 10
Linked List Operations:
1. Insert at Beginning
2. Insert at End
3. Insert after an Element
4. Insert by Position
5. Delete Beginning Element
6. Delete End Element
7. Delete an Element From the List
8. Display the List
9. Exit
Enter your choice: 1
Enter an element to insert: 20
Linked List Operations:
1. Insert at Beginning
2. Insert at End
3. Insert after an Element
4. Insert by Position
5. Delete Beginning Element
6. Delete End Element
7. Delete an Element From the List
8. Display the List
9. Exit
Enter your choice: 2
Enter an element to insert: 30
Linked List Operations:
```

1. Insert at Beginning
2. Insert at End
3. Insert after an Element
4. Insert by Position
5. Delete Beginning Element
6. Delete End Element
7. Delete an Element From the List
8. Display the List
9. Exit
Enter your choice: 2
Enter an element to insert: 40
Linked List Operations:
1. Insert at Beginning
2. Insert at End
3. Insert after an Element
4. Insert by Position
5. Delete Beginning Element
6. Delete End Element
7. Delete an Element From the List
8. Display the List
9. Exit
Enter your choice: 3
Enter the element to insert: 50
Enter the element after which to insert: 10
Element 10 not found in the list.
Linked List Operations:
1. Insert at Beginning
2. Insert at End
3. Insert after an Element
4. Insert by Position
5. Delete Beginning Element
6. Delete End Element
7. Delete an Element From the List
8. Display the List
9. Exit
Enter your choice: 4
Enter the position and element to insert: 3 60
Linked List Operations:
1. Insert at Beginning
2. Insert at End

3. Insert after an Element
4. Insert by Position
5. Delete Beginning Element
6. Delete End Element
7. Delete an Element From the List
8. Display the List
9. Exit
Enter your choice: 8
Elements of the list are: 20 10 50 60 30 40
Linked List Operations:
1. Insert at Beginning
2. Insert at End
3. Insert after an Element
4. Insert by Position
5. Delete Beginning Element
6. Delete End Element
7. Delete an Element From the List
8. Display the List
9. Exit
Enter your choice: 5
Deleted element is: 20
Linked List Operations:
1. Insert at Beginning
2. Insert at End
3. Insert after an Element
4. Insert by Position
5. Delete Beginning Element
6. Delete End Element
7. Delete an Element From the List
8. Display the List
9. Exit
Enter your choice: 6
Deleted element is: 40
Linked List Operations:
1. Insert at Beginning
2. Insert at End
3. Insert after an Element
4. Insert by Position
5. Delete Beginning Element
6. Delete End Element

7. Delete an Element From the List
8. Display the List
9. Exit
Enter your choice: 7
Enter the element to delete from the List: 60
Deleted element is: 60
Element 60 not found in the list.
Linked List Operations:
1. Insert at Beginning
2. Insert at End
3. Insert after an Element
4. Insert by Position
5. Delete Beginning Element
6. Delete End Element
7. Delete an Element From the List
8. Display the List
9. Exit
Enter your choice: 7
Enter the element to delete from the List: 100
Element 100 not found in the list.
Linked List Operations:
1. Insert at Beginning
2. Insert at End
3. Insert after an Element
4. Insert by Position
5. Delete Beginning Element
6. Delete End Element
7. Delete an Element From the List
8. Display the List
9. Exit
Enter your choice: 8
Elements of the list are: 10 50 30
Linked List Operations:
1. Insert at Beginning
2. Insert at End
3. Insert after an Element
4. Insert by Position
5. Delete Beginning Element
6. Delete End Element
7. Delete an Element From the List

8. Display the List
9. Exit
Enter your choice: 9
Linked list is closed.

## 17) Write a program to demonstrate a Circular Singly linked list.

```
#include <iostream>
using namespace std;
class CircularLinkedList
{
    struct node
    {
        int data;
        node *ptr;
    }*p;
    public:
        CircularLinkedList()
        {
            p = NULL;
        }
        void insertion(int x)
        {
            node *q, *r;
            if (p == NULL)
            {
                p = new node;
                p->data = x;
                p->ptr = p;
            }
            else
            {
                q = p;
                while (q->ptr != p)
                {
```

```
                    q = q->ptr;
                }
                r = new node;
                r->data = x;
                r->ptr = p;
                q->ptr = r;
            }
        }
        void deletion(int z)
        {
            node *q, *r, *k;
            q = p;
            if (q->data == z)
            {
                if (p->ptr == p)
                {
                    cout << "\nThe deleted element is
" << p->data;
                    delete p;
                    p = NULL;
                }
                else
                {
                    while (q->ptr != p)
                    {
                        q = q->ptr;
                    }
                    r = q->ptr;
                    q->ptr = p->ptr;
                    p = p->ptr;
                    cout << "\nThe deleted element is
" << r->data;
                    delete r;
                }
            }
```

```
            else
            {
                r = q;
                q = q->ptr;
                while (q != p)
                {
                    if (q->data == z)
                    {
                        cout << "\nThe deleted element
is " << q->data;
                        r->ptr = q->ptr;
                        delete q;
                        q = p;
                    }
                    else
                    {
                        r = q;
                        q = q->ptr;
                    }
                }
            }
        }
        void display()
        {
            node *q;
            if (p == NULL)
            {
                cout << "List is empty." << endl;
            }
            else
            {
                q = p;
                cout << "Elements of the list are:" <<
endl;
                do
```

```
                {
                    cout << q->data << " ";
                    q = q->ptr;
                } while (q != p);
                cout << endl;
            }
        }
};
int main()
{
    CircularLinkedList l;
    int choice, x, z;
    do
    {
        cout << "Linked List Operations:\n";
        cout << "1. Insertion\n";
        cout << "2. Deletion\n";
        cout << "3. Display\n";
        cout << "4. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;
        switch (choice)
        {
        case 1:
            cout << "Enter an element to insert: ";
            cin >> x;
            l.insertion(x);
            break;
        case 2:
            cout << "Enter the element you want to
delete: ";
            cin >> z;
            l.deletion(z);
            break;
        case 3:
```

```
            l.display();
            break;
        case 4:
            cout << "linked list closed" << endl;
            exit(0);
            break;
        }
    } while (choice <= 4);
    return 0;
}
```

## OUTPUT:

Linked List Operations:
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 1
Enter an element to insert: 10
Linked List Operations:
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 1
Enter an element to insert: 20
Linked List Operations:
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 1
Enter an element to insert: 30
Linked List Operations:
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 2

Enter the element you want to delete: 80
Linked List Operations:
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 3
Elements of the list are:
10 20 30
Linked List Operations:
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 2
Enter the element you want to delete: 30
The deleted element is 30
Linked List Operations:
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 3
Elements of the list are:
10 20
Linked List Operations:
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 4
linked list closed