VELAMMAL INSTITUTE OF TECHNOLOGY

VELAMMAL KNOWLEDGE PARK, PANCHETTI,
CHENNAI-601204

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LAB MANUAL

SUBJECT CODE: CS2209
SUBJECT NAME: OBJECT ORIENTED PROGRAMMING LAB

SEMESTER: THIRD
YEAR: SECOND YEAR CSE
REGULATION: 2008

PREPARED BY,
L. MARIA MICHAEL VISUWASAM
ASST.PROF/CSE

VERIFIED BY
S. SOUNDARARAJAN
HOD/CSE

CS 2209 OBJECT ORIENTED PROGRAMMING LAB 0 0 3 2

(Common to CSE & IT)

1. Design C++ classes with static members, methods with default arguments, friend functions. (For example, design matrix and vector classes with static allocation, and a friend function to do matrix-vector multiplication)
2. Implement complex number class with necessary operator overloadings and type conversions such as integer to complex, double to complex, complex to double etc.
3. Implement Matrix class with dynamic memory allocation and necessary methods. Give proper constructor, destructor, copy constructor, and overloading of assignment operator.
4. Overload the new and delete operators to provide custom dynamic allocation of memory.
5. Develop a template of linked-list class and its methods.
6. Develop templates of standard sorting algorithms such as bubble sort, insertion sort, merge sort, and quick sort.
7. Design stack and queue classes with necessary exception handling.
8. Define Point class and an Arc class. Define a Graph class which represents graph as a collection of Point objects and Arc objects. Write a method to find a minimum cost spanning tree in a graph.
9. Develop with suitable hierarchy, classes for Point, Shape, Rectangle, Square, Circle, Ellipse, Triangle, Polygon, etc. Design a simple test application to demonstrate dynamic polymorphism and RTTI.
10. Write a C++ program that randomly generates complex numbers (use previously designed Complex class) and writes them two per line in a file along with an operator (+, -, *, or /). The numbers are written to file in the format (a + ib). Write another program to read one line at a time from this file, perform the corresponding operation on the two complex numbers read, and write the result to another file (one per line).

```
//program to using static members, default arguments and
friend function
#include<iostream.h>
#include<stdlib.h>
class Matrix;
class Vector
{
     private:
          static int size;
          int a[10];
     public:
          void getVector(int=2);
          void showVector();
          friend Vector mulVectorMatrix(Vector,Matrix);
};
int Vector::size;
void Vector::getVector(int num)
{
     size=num;
     cout<<"Enter "<<size<<" values"<<endl;
     for(int i=0;i<size;i++)
     {
          cin>>a[i];
     }
}
void Vector::showVector()
{
     cout<<"The vector elements are"<<endl;
     for(int i=0;i<size;i++)
     {
          cout<<a[i]<<"\t";
     }
     cout<<endl;
}
class Matrix
{
     private:
          static int rows;
          static int cols;
          int a[3][3];
     public:
          void getMatrix(int=2,int=2);
          void showMatrix();
          friend Vector mulVectorMatrix(Vector,Matrix);
};
int Matrix::rows;
```

```
int Matrix::cols;
void Matrix::getMatrix(int r,int c)
{
	rows=r;
	cols=c;
	cout<<"Enter "<<rows*cols<<" elements of a
matrix"<<endl;
	for(int i=0;i<r;i++)
	{
		for(int j=0;j<c;j++)
		{
			cin>>a[i][j];
		}
	}
}
void Matrix::showMatrix()
{
	cout<<"elements of a matrix"<<endl;
	for(int i=0;i<rows;i++)
	{
		for(int j=0;j<cols;j++)
		{
			cout<<a[i][j]<<"\t";
		}
		cout<<endl;
	}
}
Vector mulVectorMatrix(Vector v, Matrix m)
{
	Vector result;
	int sum=0;
	if(v.size==m.cols)
	{
		for(int i=0;i<v.size;i++)
		{
			for(int j=0;j<m.cols;j++)
			{
				sum=sum+v.a[j]*m.a[j][i];

			}
			result.a[i]=sum;
			sum=0;
		}
	}
	else
	{
```

```
          cout<<"Vector-Matrix multiplication is not
possible"<<endl;
          exit(1);
     }
     return result;
}
int main()
{
     Vector v1;
     v1.getVector();
     v1.showVector();
     Matrix m1;
     m1.getMatrix();
     m1.showMatrix();
     Vector v2;
     v2=mulVectorMatrix(v1,m1);
     cout<<"After Vector Matrix multiplication the ";
     v2.showVector();
     return 0;
}

                         /*output

Enter 2 values
2 2
The vector elements are
2       2
Enter 4 elements of a matrix
2
2
2
2
elements of a matrix
2       2
2       2
After Vector Matrix multiplication the elements of a vector
is
8       8


             */
```

```
// program to demonstrate operator overloading
#include<iostream.h>
class Complex
{
     private:
          int real;
          int imag;
     public:
          void getComplex();
          void showComplex();
          Complex operator +(Complex);
          Complex operator -(Complex);
};
void Complex::getComplex()
{
     cout<<endl<<"Enter a real and imaginary parts"<<endl;
     cin>>real>>imag;
}
void Complex::showComplex()
{
     cout<<endl<<"The complex number is: ";
     if(imag<0)
     {
          cout<<real<<imag<<"i";
     }
     else
     {
          cout<<real<<"+"<<imag<<"i";
     }
}
Complex Complex::operator +(Complex c1)
{
     Complex c;
     c.real=real+c1.real;
     c.imag=imag+c1.imag;
     return c;
}
Complex Complex::operator -(Complex c1)
{
     Complex c;
     c.real=real-c1.real;
     c.imag=imag-c1.imag;
   return c;
}
int main()
```

```
{
      Complex c1;
      c1.getComplex();
      c1.showComplex();
      Complex c2;
      c2.getComplex();
      c2.showComplex();
      Complex c3;
      c3=c1+c2;
      cout<<endl<<"After addition";
      c3.showComplex();
      c3=c1-c2;
      cout<<endl<<"After subtraction";
      c3.showComplex();
      return 0;
}
```

Output

```
Enter a real and imaginary parts
2 5

The complex number is: 2+5i
Enter a real and imaginary parts
1 3

The complex number is: 1+3i

After addition
The complex number is: 3+6i

After subtraction
The complex number is: 1+2i
```

```
//program to demonstrate new and delete operator
overloading
#include<iostream.h>
const int SIZE=5;
class Vector
{
     private:
          int *arr;
     public:
          void * operator new(size_t size)
          {
               Vector *v;
             v=::new Vector;
               v->arr=new int[SIZE];
               return v;
          }
          void getVector()
          {
               cout<<"Enter "<<SIZE<<" elements"<<endl;
               for(int i=0;i<SIZE;i++)
               {
                    cin>>arr[i];
               }
          }
          void showVector()
          {
               cout<<"The elements in the Vector is"<<endl;
               for(int i=0;i<SIZE;i++)
               {
                    cout<<arr[i]<<"\t";
               }
          }
          int sumVector()
          {
               int sum=0;
               for(int i=0;i<SIZE;i++)
               {
                    sum=sum+arr[i];
               }
               return sum;
          }
          void operator delete(void* vt)
          {
               Vector *v;
```

```
                v=(Vector *)vt;
                delete (int *)v->arr;
          ::delete v;
           }
};
int main()
{
     Vector *v1=new Vector;
     v1->getVector();
     v1->showVector();
     int sum;
     sum=v1->sumVector();
     cout<<endl<<"Sum="<<sum;
     delete v1;
     return 0;
}
```

Output

```
Enter 5 elements
1
2
3
4
5
The elements in the Vector is
1       2       3       4       5
Sum=15
```

```
//program to demonstrate new and delete operator
overloading
#include<iostream.h>
const int SIZE=5;
class Vector
{
     private:
          int *arr;
     public:
          void * operator new(size_t size)
          {
               Vector *v;
             v=::new Vector;
               v->arr=new int[SIZE];
               return v;
          }
          void getVector()
          {
               cout<<"Enter "<<SIZE<<" elements"<<endl;
               for(int i=0;i<SIZE;i++)
               {
                    cin>>arr[i];
               }
          }
          void showVector()
          {
               cout<<"The elements in the Vector is"<<endl;
               for(int i=0;i<SIZE;i++)
               {
                    cout<<arr[i]<<"\t";
               }
          }
          int sumVector()
          {
               int sum=0;
               for(int i=0;i<SIZE;i++)
               {
                    sum=sum+arr[i];
               }
               return sum;
          }
          void operator delete(void* vt)
          {
               Vector *v;
```

```
                v=(Vector *)vt;
                delete (int *)v->arr;
          ::delete v;
           }
};
int main()
{
     Vector *v1=new Vector;
     v1->getVector();
     v1->showVector();
     int sum;
     sum=v1->sumVector();
     cout<<endl<<"Sum="<<sum;
     delete v1;
     return 0;
}
```

Output

```
Enter 5 elements
1
2
3
4
5
The elements in the Vector is
1       2       3       4       5
Sum=15
```

```
//program to demonstrate template for bubble sort
#include<iostream.h>
template<class T>
class Bubble
{
     private:
          T a[10];
          int size;
     public:
          Bubble();
          void getData();
          void showData();
          void sortData();
};
template<class T>
Bubble<T>::Bubble()
{
}
template<class T>
void Bubble<T>::getData()
{
     cout<<"Enter the size of the array";
     cin>>size;
     cout<<"Enter "<<size<<" elements";
     for(int i=0;i<size;i++)
     {
          cin>>a[i];
     }
}
template<class T>
void Bubble<T>::showData()
{
     cout<<"The array elements are"<<endl;
     for(int i=0;i<size;i++)
     {
          cout<<a[i]<<"\t";
     }
     cout<<endl;
}
template<class T>
void Bubble<T>::sortData()
{
     T temp;
     for(int i=0;i<size-1;i++)
```

```
	{
		for(int j=i+1;j<size;j++)
		{
			if(a[i]>a[j])
			{
				temp=a[i];
				a[i]=a[j];
				a[j]=temp;
			}
		}
	}
}
int main()
{
	Bubble<int> b1;
	b1.getData();
	b1.showData();
	b1.sortData();
	cout<<"After sorting ";
	b1.showData();
	Bubble<float> b2;
	b2.getData();
	b2.showData();
	b2.sortData();
	cout<<"After sorting ";
	b2.showData();
	return 0;
}
```

Output

```
Enter the size of the array 4
Enter 4 elements 12 34 25 47
The array elements are
12      34      25      47
After sorting The array elements are
12      25      34      47
Enter the size of the array 4
Enter 4 elements 2.5 3.6 1.8 2.7
The array elements are
2.5     3.6     1.8     2.7
After sorting The array elements are
1.8     2.5     2.7     3.6
```

```
//Queue with exception handling
#include<iostream.h>
#define SIZE 10
class Queue
{
     private:
          int rear;
          int front;
          int s[SIZE];
     public:
          Queue()
          {
               front=0;
               rear=-1;
          }
          void insert(int);
          void del();
          int isEmpty();
          int isFull();
          void display();
};
int Queue::isEmpty()
{
     return(front>rear?1:0);
}
int Queue::isFull()
{
     return(rear==SIZE?1:0);
}
void Queue::insert(int item)
{
     try
     {
          if(isFull())
          {
               throw "Full";
          }
          else
          {
               rear=rear+1;
               s[rear]=item;
          }
     }
     catch(char *msg)
     {
```

```
	cout<<msg;
	}
}
void Queue::del()
{
	int item;
	try
	{
		if(isEmpty())
		{
			throw "Empty";
		}
		else
		{
		item=s[front];
			front=front+1;
			cout<<"\n DELETED ELEMENT IS %d\n\n"<<item;
		}
	}
	catch(char *msg)
	{
		cout<<msg;
	}
}
void Queue::display()
{
	cout<<"\n";
	for(int i=front;i<=rear;i++)
	{
		cout<<s[i]<<"\t";
	}
}
int main()
{
	int ch;
	Queue q;
	int item;
	do
	{
		cout<<"\n\n1.INSERTION \n";
		cout<<"2.DELETION \n";
		cout<<"3.EXIT \n";
		cout<<"\nENTER YOUR CHOICE : ";
		cin>>ch;
		switch(ch)
		{
			case 1:
```

```
                    cout<<"\n\t INSERTION \n";
                    cout<<"\nENTER AN ELEMENT : ";
                    cin>>item;
                    q.insert(item);
                    q.display();
                    break;
               case 2:
                    cout<<"\n\t DELETION \n";
                    q.del();
                    q.display();
                    break;
          }
     }while(ch!=3);
     return 0;
}
```

Output

```
1.INSERTION
2.DELETION
3.EXIT

ENTER YOUR CHOICE : 1

               INSERTION

ENTER AN ELEMENT : 12

12

1.INSERTION
2.DELETION
3.EXIT

ENTER YOUR CHOICE : 1

               INSERTION

ENTER AN ELEMENT : 35

12      35

1.INSERTION
2.DELETION
3.EXIT
```

```
ENTER YOUR CHOICE : 1

               INSERTION

ENTER AN ELEMENT : 27

12      35      27

1.INSERTION
2.DELETION
3.EXIT

ENTER YOUR CHOICE : 2

               DELETION

 DELETED ELEMENT IS %d

12
35      27

1.INSERTION
2.DELETION
3.EXIT

ENTER YOUR CHOICE : 3
```

```
// Stack using Exception handling
#include<iostream.h>
#include<process.h>
#define SIZE 10
class Stack
{
     private:
          int a[SIZE];
          int top;
     public:
          Stack();
          void push(int);
          int pop();
          int isEmpty();
          int isFull();
          void display();
};
Stack::Stack()
{
     top=0;
}
int Stack::isEmpty()
{
     return (top==0?1:0);
}
int Stack::isFull()
{
     return(top==SIZE?1:0);
}
void Stack::push(int i)
{
     try
     {
          if(isFull())
          {
               throw "Full";
          }
          else
          {
               a[top]=i;
               top++;
          }
     }
     catch(char *msg)
     {
```

```
            cout<<msg;
      }
}
int Stack::pop()
{
      try
      {
            if(isEmpty())
            {
                  throw "Empty";
            }
            else
            {
                  return(a[--top]);
            }
      }
      catch(char *msg)
      {
            cout<<msg;
      }
      return 0;
}
void Stack::display()
{
      if(!isEmpty())
      {
            for(int i=top-1;i>=0;i--)
                  cout<<a[i]<<endl;
      }
}
int main()
{
      Stack s;
      int ch=1;
      int num;
      while(ch!=0)
      {
            cout<<"1. push"<<endl
                  <<"2. pop"<<endl
                  <<"3. display"<<endl
                  <<"0. Exit"<<endl;
            cout<<"Enter ur choice :";
            cin>>ch;
            switch(ch)
            {
                  case 0:   exit(1);
                  case 1:   cout<<"Enter the number to push";
```

```
                              cin>>num;
                              s.push(num);
                              break;
               case 2:   cout<<"a number was popped from
the stack"<<endl;
                              s.pop();
                              break;
               case 3:   cout<<"The numbers are"<<endl;
                              s.display();
                              break;
               default:
                    cout<<"try again";
          }
     }
     return 0;
}
```

Output

```
Enter ur choice :1
Enter the number to push4
1. push
2. pop
3. display
0. Exit
Enter ur choice :1
Enter the number to push8
1. push
2. pop
3. display
0. Exit
Enter ur choice :3
The numbers are
8
4
1. push
2. pop
3. display
0. Exit
Enter ur choice :1
Enter the number to push6
1. push
2. pop
3. display
0. Exit
Enter ur choice :3
```

```
The numbers are
6
8
4
1. push
2. pop
3. display
0. Exit
Enter ur choice :2
a number was popped from the stack
1. push
2. pop
3. display
0. Exit
Enter ur choice :3
The numbers are
8
4
1. push
2. pop
3. display
0. Exit
Enter ur choice :0
```

```
// Minimum cost spanning tree
#include<iostream.h>
class Graph
{
     private:
          int g[20][20];
          int visited[20];
          int d[20];
          int p[20];
          int v,e;
     public:
          void creategraph()
          {
               int i,j,a,b,w;
               cout<<"Enter number of vertices : ";
               cin>>v;
               cout<<"Enter number of edges : ";
               cin>>e;
               for(i=1;i<=v;i++)
               {
                    for(j=1;j<=v;j++)
                    {
                         g[i][j]=0;
                    }
               }
             for(i=1;i<=v;i++)
             {
                    p[i]=visited[i]=0;
                    d[i]=32767;
               }
               for(i=1;i<=e;i++)
               {
                    cout<<"Enter edge a,b and weight w : ";
                    cin>>a>>b>>w;
                    g[a][b]=g[b][a]=w;
               }
          }
          void prim()
          {
               int current,totalvisited,mincost,i,min;
               current=1;
               d[current]=0;
               totalvisited=1;
               visited[current]=1;
               while(totalvisited!=v)
```

```
                {
                     for(i=1;i<=v;i++)
                     {
                          if(g[current][i]!=0)
                          {
                               if(visited[i]==0)
                               {
                                    if(d[i]>g[current][i])
                                    {
                                         d[i]=g[current][i];
                                         p[i]=current;
                                    }
                               }
                          }
                     }
                }
                min=32767;
                for(i=1;i<=v;i++)
                {
                     if(visited[i]==0)
                     {
                          if(d[i]<min)
                          {
                             min=d[i];
                             current=i;
                          }
                     }
                     visited[current]=1;
                     totalvisited++;
                }
                mincost=0;
                for(i=1;i<=v;i++)
                mincost+=d[i];
                cout<<"Minimum Cost = "<<mincost<<endl;
                cout<<"Minimum Span Tree is \n";
                for(i=2;i<=v;i++)
                cout<<"Vertex "<<i<<" is connected to
"<<p[i];
          }
};
int main()
{
     int i;
     Graph g1;
     g1.creategraph();
     g1.prim();
     return 0;
```

```
}
```

Output

```
Enter number of vertices : 4
Enter number of edges : 6
Enter edge a,b and weight w : 1 2 2
Enter edge a,b and weight w : 1 3 6
Enter edge a,b and weight w : 1 4 2
Enter edge a,b and weight w : 2 3 4
Enter edge a,b and weight w : 2 4 5
Enter edge a,b and weight w : 3 4 3
Minimum Cost = 7
Minimum Span Tree is
Vertex 2 is connected to 1
Vertex 3 is connected to 4
Vertex 4 is connected to 1
```

```
// program to demonstrate dymanic polymorphism
#include<iostream.h>
class Shape
{
     public:
          void virtual getData()
          {
      }
          void virtual area()
          {
          }
};
class Square:public Shape
{
     private:
          int side;
     public:
          void getData()
          {
               cout<<"Enter the side value of a square ";
               cin>>side;
          }
          void area()
          {
                cout<<"Area of square is:
"<<side*side<<endl;
          }
};
class Rectangle:public Shape
{
     private:
          int len;
          int hgt;
     public:
          void getData()
          {
               cout<<"Enter the length and breadth of a
rectangle ";
               cin>>len>>hgt;
          }
          void area()
          {
               cout<<"Area of rectangle is:
"<<len*hgt<<endl;
          }
```

```
};
class Circle:public Shape
{
     private:
          float rad;
     public:
          void getData()
          {
               cout<<"Enter the radius of a circle ";
               cin>>rad;
          }
          void area()
          {
               cout<<"Area of circle is:
"<<3.14*rad*rad<<endl;
          }
};
int main()
{
     Shape *sp;
     Square sq;
     Rectangle rt;
     Circle cl;
     sp=&sq;
     sp->getData();
     sp->area();
     sp=&rt;
   sp->getData();
     sp->area();
     sp=&cl;
     sp->getData();
     sp->area();
     return 0;
}
```

Output

```
Enter the side value of a square 2
Area of square is: 4
Enter the length and breadth of a rectangle 2 3
Area of rectangle is: 6
Enter the radius of a circle 5
Area of circle is: 78.5
```

```
// program to demonstrate file operation

#include<iostream.h>
#include<fstream.h>
#include<ctype.h>
class Complex
{
     private:
          int real;
          int imag;
     public:
          void getComplex();
          void showComplex();
          friend istream& operator >>(istream&, Complex&);
          friend ostream& operator <<(ostream&, Complex&);
          friend Complex operator +(Complex, Complex);
          friend Complex operator -(Complex, Complex);
};
void Complex::getComplex()
{
     cout<<"Enter real and imaginary part:";
     cin>>real>>imag;
}
void Complex::showComplex()
{
     cout<<real;
     if(imag<0)
     {
          cout<<imag<<"i"<<endl;
     }
     else
     {
          cout<<"+"<<imag<<"i"<<endl;
     }
}
istream& operator >>(istream &fin, Complex &c)
{
     fin>>c.real;
     fin>>c.imag;
     return fin;
}
ostream& operator <<(ostream &fout, Complex &c)
{
     fout<<c.real<<" ";
     fout<<c.imag<<" ";
```

```
     return fout;
}
Complex operator +(Complex c1, Complex c2)
{
     Complex c;
     c.real=c1.real+c2.real;
     c.imag=c1.imag+c2.imag;
     return c;
}
Complex operator -(Complex c1, Complex c2)
{
     Complex c;
     c.real=c1.real-c2.real;
     c.imag=c1.imag-c2.imag;
     return c;
}
int main()
{
     Complex c1,c2,c3;
     char oper;
     char ch;
     fstream file;
     fstream rsul;
     rsul.open("result.dat",ios::out);
     file.open("z:\complex.dat",ios::in|ios::out|ios::binar
y);

     if(rsul.fail())
     {
          cout<<"unable to open"<<endl;
     }
     do
     {
          cout<<"Enter real and imaginary part of two
complex numbers"<<endl;
          cin>>c1;
          file<<c1;
          cin>>c2;
          file<<c2;
          cout<<"Enter a operator";
          cin>>oper;
          file<<oper<<endl;
          cout<<"Another? ";
          cin>>ch;
     }while(toupper(ch)=='Y');
     file.seekg(0);
     while(1)
```

```
	{
		file>>c1;
		file>>c2;
		file>>oper;
		if(file.fail())
			break;
		cout<<c1;
		cout<<c2;
		cout<<oper;
		switch(oper)
		{
			case '+':
				c3=c1+c2;
				cout<<endl;
				c3.showComplex();
				rsul<<c1;
				rsul<<c2;
				rsul<<" "<<c3<<endl;
				break;
			case '-':
				c3=c1-c2;
				cout<<endl;
				c3.showComplex();
				rsul<<c1<<c2<<" "<<c3<<endl;
				break;
		}
	}
	file.close();
	rsul.close();
	return 0;
}
```

```
Enter real and imaginary part of two complex numbers
3 4 2 5
Enter a operator +
Another? y
Enter real and imaginary part of two complex numbers
4 6 1 3
Enter a operator -
Another? n
```

```
3 4 2 5 +
5+9i
4 6 1 3 -
```

```
3+3i

complex.dat
3  4  2  5  +
4  6  1  3  -

result.dat
3  4  2  5   5  9
4  6  1  3   3  3
```

```
//Operator overloading – Matrix with  dynamic memory
allocation
#include<iostream.h>
#include<conio.h>
class Matrix
{
      private:
         int rowSize;
         int colSize;
         int **data;
      public:
         Matrix()
         {
         }
         Matrix(int r,int c)
         {
             rowSize=r;
             colSize=c;
             data=new int*[rowSize];
                       for(int i=0;i<rowSize;i++)
                              data[i]=new int[colSize];
             }
             Matrix(const Matrix& obj)
             {

rowSize=obj.rowSize;

colSize=obj.colSize;
                                                 data=new
int*[rowSize];
                                                 for(int
i=0;i<rowSize;i++)

data[i]=new int[colSize];


for(i=0;i<rowSize;i++)
                                                 {

     for(int j=0;j<colSize;j++)
                                                              {

data[i][j]=obj.data[i][j];
                                                              }
                                             }
```

```
                    }
                   ~Matrix()
                    {
                                        for(int
i=0;i<rowSize;i++)
                                                            delete
data[i];
                                        delete[] data;
                    }
                    Matrix& operator=(Matrix &obj);
                    Matrix& operator+(Matrix m1);
                    Matrix& operator-(Matrix m1);
                    Matrix& operator*(Matrix m1);
                    void getMatrix();
                    void displayMatrix();

};
Matrix& Matrix::operator=(Matrix& obj)
{
            rowSize=obj.rowSize;
            colSize=obj.colSize;
            data=new int*[rowSize];
            for(int i=0;i<rowSize;i++)
                 data[i]=new int[colSize];

            for(i=0;i<rowSize;i++)
            {
                              for(int j=0;j<colSize;j++)
                              {

     data[i][j]=obj.data[i][j];
                              }
            }
            return *this;
}
Matrix& Matrix::operator+(Matrix m1)
{
            static Matrix temp(rowSize,colSize);
            for(int i=0;i<rowSize;i++)
            {
                              for(int j=0;j<colSize;j++)
                              {

     temp.data[i][j]=this->data[i][j]+m1.data[i][j];
                              }
            }
            return temp;
```

```
}

Matrix& Matrix::operator-(Matrix m1)
{
            static Matrix temp(rowSize,colSize);
            for(int i=0;i<rowSize;i++)
            {
                        for(int j=0;j<colSize;j++)
                        {

     temp.data[i][j]=this->data[i][j]-m1.data[i][j];
                        }
            }
            return temp;
}
Matrix& Matrix::operator*(Matrix m1)
{
            static Matrix temp(rowSize,m1.colSize);
            for(int i=0;i<rowSize;i++)
            {
                        for(int j=0;j<m1.colSize;j++)
                        {
                                    temp.data[i][j]=0;
                                    for(int
k=0;k<colSize;k++)
                                    {

temp.data[i][j]+=data[i][k]*m1.data[k][j];
                                    }

                        }
            }
            return temp;
}

void Matrix::getMatrix()
{
            for(int i=0;i<rowSize;i++)
            {
                        for(int j=0;j<colSize;j++)
                        {
                                    cin>>data[i][j];
                        }
            }
}

void Matrix::displayMatrix()
```

```
{
            for(int i=0;i<rowSize;i++)
            {
                        cout<<"\n";
                        for(int j=0;j<colSize;j++)
                        {

cout<<data[i][j]<<"\t";
                        }
            }
        cout<<endl;
}


int main()
{
     cout<<"Enter 2 - 2x2 matix elements";
    Matrix m1(2,2);
    Matrix m2(2,2);
    m1.getMatrix();
    m2.getMatrix();
   Matrix m3;

      cout<<endl<<"Matrix 1\n";
    m1.displayMatrix();

    cout<<"Matrix 2\n";
   m2.displayMatrix();
     m3=m1+m2;
     cout<<"Result +\n";
     m3.displayMatrix();
     m3=m1-m2;
     cout<<"Result -\n";
     m3.displayMatrix();
     m3=m1*m2;
   cout<<"Result *\n";
   m3.displayMatrix();

    getch();
    return 0;
}
```

```
// program to demonstrate template using linked list
#include<iostream.h>
#include<stdlib.h>
template <class T>
class List
{
     private:
          T data;
          List *next;
     public:
          List();
          List(T dat);
          int getData()
          {
               return data;
          }
          void insertData(List<T> *);
          friend void displayData(List<T> *);
};
template<class T>
List<T>::List()
{
     data=0;
     next=NULL;
}
template<class T>
List<T>::List(T dat)
{
     data=dat;
     next=NULL;
}
template<class T>
void List<T>::insertData(List<T> *node)
{
    List<T> *last=this;
     while(last->next)
     {
          last=last->next;
     }
     last->next=node;
}
template<class T>
void displayData(List<T> *first)
{
```

```
	List<T> *start;
	cout<<"List elements are: ";
	for(start=first;start;start=start->next)
	{
		cout<<start->data<<"\t";
	}
	cout<<endl;
}
int main()
{
	int choice;
	int data;
	List<int> *first=NULL;
	List<int> *node;
	while(1)
	{
		cout<<"Linked list"<<endl;
		cout<<"1. Insert"<<endl;
		cout<<"2. Display"<<endl;
		cout<<"3. Exit"<<endl;
		cout<<"Enter your choice :";
		cin>>choice;
		switch(choice)
		{
			case 1:
				cout<<"Enter a data :";
				cin>>data;
				node=new List<int>(data);
				if(first==NULL)
				{
					first=node;
				}
				else
				{
					first->insertData(node);
				}
				break;
			case 2:
				displayData(first);
				break;
			case 3:
				exit(1);
			default:
				cout<<"Incorrect option"<<endl;
				continue;
		}
	}
```

```
}

/*

                    Output
Linked list
1. Insert
2. Display
3. Exit
Enter your choice :1
Enter a data :23
Linked list
1. Insert
2. Display
3. Exit
Enter your choice :1
Enter a data :34
Linked list
1. Insert
2. Display
3. Exit
Enter your choice :1
Enter a data :45
Linked list
1. Insert
2. Display
3. Exit
Enter your choice :2
List elements are: 23    34      45
Linked list
1. Insert
2. Display
3. Exit
Enter your choice :3

*/
```