R.V.R. & J.C.COLLEGE OF ENGINEERING (Autonomous)
Chandramoulipuram :: Chowdavaram :: Guntur-522019
(w.e.f. the academic year 2020-2021)
B.Tech., Computer Science and Business Systems

DEPARTMENT

OF

COMPUTER SCIENCE & BUSINESS SYSTEMS

CB – 253 (R-18)

OBJECT ORIENTED PROGRAMMING

II/IV B.Tech - CSBS (3rd – Semester)

RVR&JC COLLEGE OF ENGINEERING

(AUTONOMOUS)
(Sponsored by Nagarjuna Education Society)
(Affiliated to Acharya Nagarjuna University :: Approved by AICTE)
CHANDRAMOULIPURAM:: CHOWDAVARAM
GUNTUR – 522 019 :: ANDHRA PRADESH

R.V.R. & J.C.COLLEGE OF ENGINEERING (Autonomous)
Chandramoulipuram :: Chowdavaram :: Guntur-522019
(w.e.f. the academic year 2020-2021)
B.Tech., Computer Science and Business Systems

PROFORMA FOR LABORATORY – BASED COURSE DESCRIPTION

Course Code	**:**	**CB – 253**
Course Title	**:**	**Object Oriented Programming**
Year & Semester	**:**	**II/IV B – Tech (3rd – Semester)**
Periods/Week	**:**	**04**
Nature of the Course	**:**	**Engineering Core**
Name of the Instructors	**:**	**Smt. Y. Madhulika**
Designation	**:**	**Assistant. Professor**
E – Mail	**:**	**madhulika.yarlagadda@gmail.com**

R.V.R. & J.C.COLLEGE OF ENGINEERING (Autonomous)
Chandramoulipuram :: Chowdavaram :: Guntur-522019
(w.e.f. the academic year 2020-2021)
B.Tech., Computer Science and Business Systems

PROGRAM EDUCATIONAL OBJECTIVES (PEO):

- To provide fundamental knowledge in Basic Sciences, Computer Sciences and Management Sciences.
- To inculcate strong problem solving skills, to design, implement, test and maintain Software Systems.
- To impart good ethical practices, right professional conduct and responsible team Leadership.

PROGRAM OUTCOMES (PO):

Upon graduation, students of the program will:

1. Engineering Knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. Problem analysis: Identify formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. Conduct Investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including predication and modeling to complex engineering activities with an understanding of the limitations.
6. The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological changes.

4.PROGRAM SPECIFIC OUTCOMES (PSO's)

Upon graduation, students of the program will be able to

1. Apply Engineering knowledge and Analysis tools to solve problems in Computer Science & Engineering.
2. Design, implement and integrate Engineering Solutions with an understanding of professional, legal, managerial and financial issues.
3. Acquaint with the contemporary trends in industrial settings and there by innovate novel solutions to existing problems.

R.V.R. & J.C.COLLEGE OF ENGINEERING (Autonomous)
Chandramoulipuram :: Chowdavaram :: Guntur-522019
(w.e.f. the academic year 2020-2021)
B.Tech., Computer Science and Business Systems

LAB – COURSE CONTENT

.

1. Parameter passing: passing parameter by value vs by reference, passing array as constant pointer.
2. Function overloading: writing string operations like strcat and strncat, strcpy and strncpy as overloaded functions.
3. Dynamically allocating space for a pointer depending on input and doing this repeatedly, depending on different inputs and finally de-allocating the pointer.
4. Define class complex with all possible operations: constructor, destructor, copy constructor, assignment operator with the data members stored as pointer to integers.
5. Define class vector of integers with all possible operations like constructor, destructor, copy constructor and assignment operators.
6. Define class matrix of integers with all possible operations like constructor, destructor, copy constructor and assignment operators
7. Define class matrix of integers using vector, with all possible operations like constructor, destructor, copy constructor and assignment operators
8. Define class stack, queue, linked-list, array, set using some data-type (int) with data members kept as private and functions kept in both protected and public sections.
9. Define class complex with all possible operators: constructor, destructor, copy constructor, assignment operator and operators >, =, <=, ==, ++ (pre and post), +, +=, ( ), with the data members stored as pointer to integers.
10. Define class vector of integers with all possible operations like constructor, destructor, copy constructor and assignment operators>, =, <=, ==, ++ (pre and post), +, +=, ( ).
11. Define class matrix of integers with all possible operations like constructor, destructor, copy constructor and assignment operators>, =, <=, ==, ++ (pre and post), +, +=, ( ).
12. Define class matrix of integers using vector, with all possible operations like constructor, destructor, copy constructor and assignment operators>, =, <=, ==, ++ (pre and post), +, +=, ( ).
13. Define stack and queue inherited from array class, with standard functions and operators.
14. Define a class called ‘array’ with data type passed as template type with constructor, destructor, copy constructor and assignment operators and index operator.
15. Define template functions for compare and use it in the algorithms like bubble sort, insertion sort, merge sort.
16. Formatted input-output examples
17. Input manipulators
18. Overriding operators <>
19. Define class model for complex number, student class, book class and show it using UML diagram as well as concrete class.
20. Show behavioural modelling through sequence diagram and activity diagram for workflow in a typical log-in, log-out situation.

PRE – REQUISITES

CB 113-Fundamentals of Computer Science
CB 123 Data Structures & Algorithms

LAB COURSE – OBJECTIVES

At the end of the course, the student will understand:

- The Difference between object oriented programming and procedural programming
- The concepts of Constructors, inheritance, polymorphism and exception handling.

- The Application development using templates, files in C++
- The Different UML diagrams

LEARNING – OUTCOMES

After successful completion of the course, the students are able to:

CO1: Demonstrate object oriented programming concepts to solve real time problems
CO2: Experiment with the concepts of constructors, inheritance and polymorphism and exception handling
CO3: Create software applications using templates, and files in C++
CO4: Illustrate the different UML diagrams
.

CO-PO MAPPING

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	H	H	H	----	M	----	----	----	----	----	----	M
CO2	H	H	H	----	M	----	----	----	----	----	----	M
CO3	H	H	H	----	M	----	----	----	----	----	----	M
CO4	H	H	H	----	M	----	----	----	----	----	----	M

1. CO-PSO MAPPING

	PSO1	PSO2	PSO3
CO1	H	M	L
CO2	M	H	L
CO3	M	H	L
CO4	M	H	L

R.V.R. & J.C.COLLEGE OF ENGINEERING (Autonomous)
Chandramoulipuram :: Chowdavaram :: Guntur-522019
(w.e.f. the academic year 2020-2021)
B.Tech., Computer Science and Business Systems

SOLUTIONS

1./*Parameter passing : passing parameter by value vs by referene, passing array as constant pointer*/

```
#include<bits/stdc++.h>
using namespace std;
void passbyvalue(int x, int y)
{
 int sum=0,t=0;
 sum=x+y;
 cout<<"Addition through pass by value is "<<sum<<endl;
 t=x;
 x=y;
 y=t;
 cout<<"Swapping through pass by value  "<<x<<"  "<<y<<endl;
}
void passbyreference(int &x, int &y)
{
 int sum=0,t;
 sum=x+y;
 cout<<"Addition through pass by reference is "<<sum<<endl;
 t=x;
 x=y;
 y=t;
 cout<<"Swapping through pass by value  "<<x<<"  "<<y<<endl;
}
void passArray(int *ar,int n)
{
 int i, sum=0;
 for(i=0;i<n;i++)
 {
  cout<<"Enter "<<i+1<<" element  " ;
  cin>>ar[i];
  sum=sum+ar[i];
 }
 cout<<"Sum of given array is "<<sum<<endl;
}

int main()
{
 int a,b,n;
 cout<<"Enter two values for call by value & refrence  ";
 cin>>a>>b;
 int ar[n];
 passbyvalue(a,b);
 cout<<"Pass by values "<<a<<b<<endl;
 passbyreference(a,b);
 cout<<"Enter required no. of elements for passing array"<<endl;
 cin>>n;
 passArray(ar,n);
 return 0;
}
```

R.V.R. & J.C.COLLEGE OF ENGINEERING (Autonomous)
Chandramoulipuram :: Chowdavaram :: Guntur-522019

(w.e.f. the academic year 2020-2021)

B.Tech., Computer Science and Business Systems

Output:

```
Enter two values for call by value & refrence  1 2
Swapping through pass by value  2  1
Addition through pass by reference is 3
Enter required no. of elements for passing array
Enter 1 element  7
Enter 3 element  9
        Addition through pass by value is 3
        Pass by values 12
        Swapping through pass by value  2  1
          3
Enter 2 element  8
Sum of given array is 24
```

2. /*Function overloading: writing string operations like strcat and strncat, strcpy and strncpy as overloaded functions. */

```
#include<bits/stdc++.h>
using namespace std;
void strcpy(string s1)
{
 string s;
 s=s1;
 cout<<"The copied sentence is "<<s<<endl;
}
void strcpy(string s1,int n)
{
 string s;
 s=s1;
 int i;
 cout<<"String after n copy is "<<endl;
 for(i=0;i<n;i++)
 {
  cout<<s[i];
 }

}
void strcat(string s1, string s2)
{
 string s;
 s=s1+s2;
 cout<<"String after concatenation is "<<endl;
 cout<<s;
}
void strcat(string s1, string s2,int n)
{
 string s;
 int i;
 s=s1+s2;
 cout<<"String after n concatenation is "<<endl;
 for(i=0;i<n;i++)
 {
  cout<<s[i];
 }
}
void check(int ch)
{
  try
  {
 int n;
 string s1,s2;
 if(ch==1)
 {
  cout<<"Enter a word/sentence"<<endl;
```


```
 getline(cin, s1);
 getline(cin,s1);
 strcpy(s1);
 }
 else if (ch==2)
 {
 cout<<"Enter a word/sentence"<<endl;
 getline(cin,s1);
 getline(cin,s1);
 cout<<"Enter limit ";
 cin>>n;
 strcpy(s1,n);
 }
 else if(ch==3)
 {
 cout<<"Enter sentence 1 "<<endl;
 getline(cin,s1);
 getline(cin,s1);
 cout<<"Enter sentence 2  "<<endl;
 getline(cin,s2);
 strcat(s1,s2);
 }
 else if(ch==4)
 {
 cout<<"Enter sentence 1 "<<endl;
 getline(cin,s1);
 getline(cin,s1);
 cout<<"Enter sentence 2  "<<endl;
 getline(cin,s2);
 cout<<"Enter limit ";
 cin>>n;
 strcat(s1,s2,n);
 }
 if(ch<0 || ch>4)
 {
 throw ch;
 }
  }
  catch(int ch)
  {
     cout<<"Enter valid number"<<endl;
  cin>>ch;
  check(ch);
  }

}

int main()
{
 int ch;
```


```
cout<<"Enter 1 String Copy\n 2 String n copy \n 3 String Concatenation\n 4 String n concatenation"<<endl;
cin>>ch;
check(ch);
return 0;
}
```

Output:

```
Enter 1 String Copy
```

```
 2 String n copy

 3 String Concatenation

 4 String n concatenation

1

Enter a word/sentence

this is KRP

The copied sentence is this is KRP
```

```
Enter 1 String Copy

 2 String n copy

 3 String Concatenation

 4 String n

concatenation 2

Enter a

word/sentence KRP

Enter limit 1

String after n copy

is K
```


```
Enter 1 String Copy

 2 String n copy

 3 String Concatenation

 4 String n concatenation

3

Enter sentence 1

Ram

Enter sentence 2

Priyatham

String after concatenation is

RamPriyatham
Enter 1 String Copy
```

```
 2 String n copy

 3 String Concatenation

 4 String n concatenation

4

Enter sentence 1

Ram

Enter sentence 2

Priyatham

Enter limit 10

String after n concatenation is

Ram Priyat
```

3. /*Dynamically allocating space for a pointer depending on input and doing this repeatedly, depending on different inputs and finally de-allocating the pointer. */

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
 cout<<"Enter any of the following \n integer\ncharacter\nstring\ndouble\nfloat\nexit"<<endl;
 while(1)
 {
  string s;
  cout<<"Enter input type "<<endl;
  cin>>s;
  if(s=="character")
  {
   char *c = new char;
   cout<<"Enter character "<<endl;
   cin>>c;
   cout<<"Entered is  "<<*c<<endl;
   delete c;
  }
  else if(s == "exit")
  {
   exit(0);
  }
  else if (s=="string")
  {
   string *s1 = new string;
   cout<<"Enter a line "<<endl;
   getline(cin,*s1);
   getline(cin,*s1);
   cout<<"Entered is  "<<*s1<<endl;
   delete s1;
  }

else if(s=="integer")
  {
   int *i = new int;
   cout<<"Enter a number  "<<endl;
   cin>>*i;
   cout<<"Entered is  "<<*i<<endl;
   delete i;
  }

 else if(s=="double" || "float")
  {
   double *d = new double;
   cout<<"Enter a number  "<<endl;
   cin>>*d;
   cout<<"Entered is  "<<*d<<endl;
```


```
  delete d;
 }

 else
 {
  cout<<"Enter valid type "<<endl;
 }
 }
 return 0;
}
```

Output:

```
exit

Enter input type

string

Enter a line

Ram Priyatham

Entered is  Ram Priyatham

Enter input type

integer

Enter a number

51

Entered is  51

Enter input type

double

Enter a number

9.6

Entered is  9.6

Enter input type

exit
```

4./*Define class complex with all possible operations: constructor, destructor, copy constructor, assignment operator with the data members stored as pointer to integers. */

```
#include<bits/stdc++.h>
using namespace std;

class Complex
{
 int *real,*img;
 public:
  Complex()//Default COnstructor
  {
   real=0;
   img=0;
  }
  Complex(int x, int y) //Constructor
  {
   real =new int;
   img=new int;
   *real = x;
   *img=y;
   cout<<"Values inside Constructor is  "<<*real<<"+i"<<*img<<endl;
  }
  Complex(const Complex &c) //Copy Constructor
  {
   real= c.real;
   img=c.img;
   cout<<"Values inside Copy Constructor "<<*real<<"+i"<<*img<<endl;
  }
  Complex operator=(const Complex &ca)
  {
   real = ca.real;
   img = ca.img;
   cout<<"Values using Assignment Operators "<<*real<<"+i"<<*img<<endl;
  }
  ~Complex() //Destructor
  {
   cout<<"Destructor called "<<endl;
  }
};

int main()
{
 Complex c1(20,30);
 Complex c2(10,40);
 Complex c3;
 //Using Assignmnet operator
 c3=c1;
 //Using Copy Contructor
 Complex c4(c1);
```

}
Output:
}

Output:

```
Values inside Constructor is  20+i30

Values inside Constructor is  10+i40

Values using Assignment Operators 20+i30
```

```
Destructor called

Values inside Copy Constructor 20+i30

Destructor called

Destructor called

Destructor called

Destructor called
```

5./*Define class vector of integers with all possible operations like constructor, destructor, copy constructor and assignment operators */

```
#include<bits/stdc++.h>
#include<vector>
using namespace std;

class Vector
{
 vector<int> v;
 vector<int> cpy;
 public:
  Vector()
  {
  }
  Vector(int n)
  {
   int x;
   //vector<int> v;
   std::cout<<"Enter values ";
   for(int i=0;i<n;i++)
   {
    std::cin>>x;
    v.push_back(x);
   }
   std::cout<<"Values in vector are "<<std::endl;
   vector<int> :: iterator it;
   for(it=v.begin();it!=v.end();it++)
   {
    cout<<*it<<"  ";
   }
   std::cout<<std::endl;
   cout<<"Constructor ends here  "<<endl;
  }
  Vector( Vector &v1)
  {
   cpy = v1.v; // Means storing values of v vector datatype from v1 class
   vector<int> :: iterator it;
   for(it=cpy.begin();it!=cpy.end();it++)
   {
    std::cout<<*it<<"  ";
   }
   std::cout<<std::endl;
   std::cout<<"Copy Constructor ends here  "<<std::endl;
  }
  Vector& operator =(const Vector &asign)
  {
   cpy=asign.v;
   vector<int> :: iterator it1;
   for(it1=cpy.begin();it1!=cpy.end();it1++)
```


```
  {
   std::cout<<*it1<<"  ";
  }
  std::cout<<"Assignment operator ends here  "<<std::endl;
  return *this;
 }
 ~Vector()
 {
  std::cout<<"Destructor called "<<std::endl;
 }

};
 int main()
{
 int n;
 cout<<"Enter req no. of values for vector ";
 cin>>n;
 Vector v1(n);
 Vector v2(v1);
 Vector v3;
 v3=v2;
}
```

Output:

Output:

```
Enter req no. of values for vector 5

Enter values 1

2

3

4

5

Values in vector are
```


```
1  2  3  4  5

Constructor ends here

1  2  3  4  5

Copy Constructor ends here

Assignment operator ends here

Destructor called

Destructor called

Destructor called
```

6./*Define class matrix of integers with all possible operations like constructor, destructor, copy constructor and assignment operators */

```
#include<iostream>
using namespace std;
class matrix
{
 int row,col,i,j;
 int **m;//While declaring 2d array we should use **m if 1d array *m

public:
 matrix()//Default constructor
 {
 }
 matrix(int r,int c)
 {
  row=r;
  col=c;
  m= new int*[row];
  for(i=0;i<row;i++)
  {
   m[i]=new int [col];
  }// If declaring a 2d array dynamically we have to use above 5 lines
 }
 matrix(const matrix & cpy)//Copy Constructor
 {
  cout<<"Copy constructor invoked "<<endl;
  row=cpy.row;
  col=cpy.col;
  m=cpy.m;
  for(i=0;i<row;i++)
  {
   for(j=0;j<col;j++)
   {
    m[i][j]=cpy.m[i][j];
   }
  }
 }
 matrix operator =(const matrix &rhs)
 {
  row=rhs.row;
  col=rhs.col;
  m=new int *[row];
  for(i=0;i<row;i++)
  {
   m[i]=new int [col];
  }
  for(i=0;i<row;i++)
  {
   for(j=0;j<col;j++)
   {
```


```
   m[i][j]=rhs.m[i][j];
  }
 }
 //return *this;
}

void print()
{
 for(i=0;i<row;i++)
 {
  for(j=0;j<col;j++)
  {
   cout<<m[i][j]<<" "<<endl;
  }
 }
}
void enter()
{
 m= new int*[row];
 for(i=0;i<row;i++)
 {
  m[i]=new int [col];
 cout<<"Enter values of matrix"<<endl;
 for(i=0;i<row;i++)
 {
  for(j=0;j<col;j++)
  {
   cin>>m[i][j];
  }
 }
 }
}
~matrix()//Destructor
{
}
};

int main()
{
 matrix m1(1,2),m2(1,2);
 cout<<"Enter matrix 1 elements "<<endl;
 m1.enter();
 cout<<"Enter matrix 2 elements "<<endl;
 m2.enter();
 matrix m3(1,2);
 cout<<"Assignment operator "<<endl;
 m3=m1;
 m3.print();
 cout<<"Using copy constructor "<<endl;
 matrix m4(m2);
```


```
 m4.print();
}
```

Output:

```
Enter matrix 1 elements

Enter values of matrix

1

2

Enter matrix 2 elements

Enter values of matrix

3
```

```
4

Assignment operator

1

2

Using copy constructor

Copy constructor invoked

3

4
```

7./* Define class matrix of integers using vector, with all possible operations like constructor, destructor, copy constructor and assignment operators */

```
#include<bits/stdc++.h>
#include<vector>
#include<algorithm>
#include<iostream>
using namespace std;

class matrix
{
 int rows,col,i,j;
 vector<vector<int> >v;
 public:
  matrix()//Default constructor
  {
  }
  matrix(int r,int c)
  {
   rows=r;
   col=c;
  }
  matrix operator =(const matrix& rhs)
  {
   cout<<"Displaying Assignment operator "<<endl;
   rows=rhs.rows;
   col=rhs.col;
   v=rhs.v;
   return *this;
  }
  matrix(const matrix &cpy)
  {
   cout<<"Printing using copy constructor "<<endl;
   rows = cpy.rows;
   col=cpy.col;
   v=cpy.v;
  }
  void enter()
  {
   int num;
   cout<<"Enter elements "<<endl;
   for(i=0;i<rows;i++)
   {
    vector<int> v1;
    for(j=0;j<col;j++)
    {
     cin>>num;
     v1.push_back(num);
    }
    v.push_back(v1);
```


```
  }
 }
 void print()
 {
  for (int i = 0; i < v.size(); i++)
  {
   for (int j = 0; j < v[i].size(); j++)
   {
    cout << v[i][j] << " ";
   }
   cout<<endl;
     }
 }
 ~matrix()//Destructor
 {
 }
};
int main()
{
 matrix m1(2,2);
 m1.enter();
 matrix m2;
 m2=m1;
 m2.print();
 matrix m3(m1);
 m3.print();
}
```

Output:

```
Enter elements
1

2

3

4

Displaying Assignment operator

Printing using copy constructor 1 2

3 4

Printing using copy constructor 1 2

3 4
```

8./*Define class stack, queue, linked-list, array, set using some data-type (int) with data members kept as private and functions kept in both protected and public sections.*/

```
#include<bits/stdc++.h>
#include<iostream>
#include <cstdlib>
#include<set>*/
#define size 20
#define sizeq 20
using namespace std;
int top=-1;
int front =0;
int rear=0;
struct Node {
   int data;
   struct Node *next;
}*head=NULL,*temp,*ptr;
class stackss
{
 int data;
 //int top=-1;

 int stacks[size];
 public:
  void push(int data)
  {
   if(top>=size)
   {
    cout<<"Stack is full "<<endl;
   }
   else
   {
    top=top+1;
    stacks[top]=data;
   }
  }
  void pop()
  {
   int temp;
   if(top<=0)
   {
    cout<<"Stack is empty "<<endl;
   }
   else
   {
    temp=stacks[top];
    top=top-1;
    cout<<"Data popped is "<<temp<<endl;
   }
  }
  void stack_print()
```


```
  {
   int i=top;
   while(i>=0)
   {
    cout<<stacks[i]<<endl;
    i--;
   }
  }
};
class queues
{
 int queues[sizeq];
 int data;
public:
 void push(int data)
 {
  queues[rear]=data;
  rear++;
 }
 void pop()
 {
  int temp;
  temp=queues[front];
  front++;
  cout<<"Element popped is "<<temp<<endl;
 }
 void queue_print()
 {
  int i=front;
  while(i<rear)
  {
   cout<<queues[i]<<endl;
   i++;
  }
 }
};
class linked_list
{
public:
 void insert(int data)
 {
  temp=(struct Node *)malloc(sizeof(struct Node));
  temp->data=data;
  temp->next=NULL;
  if(head==NULL)
  {
   head=temp;
  }
  else
  {
```


```
  temp->next=head;
  head=temp;
 }
}
void delet()
{
 int t;
 ptr=head;
 t=ptr->data;
 head=ptr->next;
 ptr->next=NULL;
 free(ptr);
 cout<<"Element deleted is "<<t<<endl;
}
void linked_print()
{
 ptr=head;
 while(ptr!=NULL)
 {
  cout<<ptr->data<<endl;
  ptr=ptr->next;
 }
}
};
class arrays
{
 int a[10],n;
public:
 arrays()
 {
 }
 void insert(int n)
 {
  int i=0;
  for(i=0;i<n;i++)
  {
   cout<<"Enter data "<<endl;
   cin>>a[i];
  }
 }
 void arrays_print(int n)
 {
  for(int i=0;i<n;i++)
  {
   cout<<a[i]<<endl;
  }
 }
};
int main()
{
```


```
cout<<"Stack Operations "<<endl;
stackss s;
s.push(20);
s.push(40);
s.push(50);
s.stack_print();
s.pop();
s.pop();
s.stack_print();
cout<<"Queue operations "<<endl;
queues q;
q.push(10);
q.push(30);
q.push(60);
q.queue_print();
q.pop();
q.pop();
q.queue_print();
cout<<"Linked list operations "<<endl;
linked_list l;
l.insert(7);
l.insert(8);
l.insert(9);
l.linked_print();
l.delet();
l.delet();
cout<<"Array operations are "<<endl;
arrays a1;
a1.insert(4);
a1.arrays_print(4);
cout<<"Set operations are "<<endl;
set<int> s1;
s1.insert(11);
s1.insert(2);
s1.insert(3);
cout<<"Set values are "<<endl;
set<int>::iterator it1;
for(it1=s1.begin();it1!=s1.end();it1++)
{
 cout<<*it1<<"  ";
}
s1.erase(3);
cout<<endl<<"After deleting element set values are "<<endl;
set<int>::iterator it2;
for(it2=s1.begin();it2!=s1.end();it2++)
{
 cout<<*it2<<"  ";
}
}
```

Output:

```
20

Data popped is 50

Data popped is 40

20

Queue operations

10

30

60

Element popped is 10

Element popped is 30

60

Linked list operations

9

8

7

Element deleted is 9

Element deleted is 8

Array operations are

Enter data

1

Enter data
```


```
2

Enter data

3

Enter data

4

1

2

3

4

Set operations are

Set values are

2  3  11

After deleting element set values are

2  11
```

9./*Define class complex with all possible operators: constructor, destructor, copy constructor, assignment operator and operators >, <, >=, <=, ==, ++ (pre and post), +, +=, ( ), with the data members stored as pointer to integers*/

```
#include<iostream>
//#include<bits/stdc++.h>
using namespace std;
class Complex
{
 int real,imag,mag;
 public:
  Complex()
  {
  }
  Complex(int r,int i)
  {
   real=r;
   imag=i;
  }
  Complex operator () (int r,int i,int m)
  {
   Complex d;
    d.real=r;
    d.imag=i;
    d.mag=m;
   return d;
  }
  Complex operator > (const Complex &rhs)
  {
   if(real>rhs.real && imag>rhs.imag)
   {
    cout<<"1st object is larger "<<endl;
   }
   else
   {
    cout<<"2nd object is larger "<<endl;
   }
  }
  Complex operator < (const Complex&rhs)
  {
   if(real<rhs.real&& imag<rhs.imag)
   {
    cout<<"1st object is smaller "<<endl;
   }
   else
   {
    cout<<"2nd object is smaller "<<endl;
   }
  }
```


```
Complex operator >= (const Complex&rhs)
{
 if(real>rhs.real && imag>rhs.imag)
 {
  cout<<"1st object is larger "<<endl;
 }
 else if(real==rhs.real && imag==rhs.imag)
 {
  cout<<"Both are equal "<<endl;
 }
 else
 {
  cout<<"2nd object is larger "<<endl;
 }
}
Complex operator <= (const Complex&rhs)
{
 if(real<rhs.real&& imag<rhs.imag)
 {
  cout<<"1st object  is smaller "<<endl;
 }
 else if(real == rhs.real && imag == rhs.imag)
 {
  cout<<"Both are equal "<<endl;
 }
 else
 {
  cout<<"2nd object is smaller "<<endl;
 }
}
Complex operator ++() //Pre Increment
{
 Complex c;
 c.real=++real;
 c.imag=++imag;
 return c;
}
Complex operator ++(int) //Post Increment
{
 Complex c;
 c.real=real++;
 c.imag=imag++;
 return c;
}
Complex operator + (const Complex &rhs)
{
 Complex c;
 c.real=real+rhs.real;
 c.imag=imag+rhs.imag;
```


```
 return c;
 }
 Complex operator += (const Complex &rhs)
 {
  Complex c;
  c.real=real+rhs.real;
  c.imag=imag+rhs.imag;
  return c;
 }
 Complex operator = (const Complex &rhs)
 {
  real=rhs.real;
  imag=rhs.imag;
 }
 void print()
 {
  Complex c;
  cout<<"Real value is "<<real<<endl;
  cout<<"Imaginary value is "<<imag<<endl;
 }
 void printm()
 {
  Complex c;
  cout<<"Real value is "<<real<<endl;
  cout<<"Imaginary value is "<<imag<<endl;
  cout<<"Magnitude part is "<<mag<<endl;
 }
 ~Complex() //Destructor
 {
 }
};

int main()
{
 Complex c1(1,2),c2(3,4);
 c1>c2;
 c1<c2;
 cout<<"Post Incrementing "<<endl;
 c1++;
 c1.print();
 cout<<"Pre Incrementing "<<endl;
 ++c2;
 c2.print();
 cout<<"Adding two classes "<<endl;
 Complex c3=c1+c2;
 c3.print();
 cout<<"Adding using short hand operators "<<endl;
 c3.operator +=(c2);
 c3.print();
```


```
cout<<"Assignment operator overloading "<<endl;
Complex c4;
c4=c1;
c4.print();
cout<<"Paranthesis operator overloading "<<endl;
Complex c5=c2(5,4,6);
c5.printm();
}
```

Output:

```
2nd object is larger

1st object is smaller

Post Incrementing

Real value is 2

Imaginary value is 3
```

```
Pre Incrementing

Real value is 4

Imaginary value is 5

Adding two classes

Real value is 6

Imaginary value is 8

Adding using short hand operators

Real value is 6

Imaginary value is 8

Assignment operator overloading

Real value is 2

Imaginary value is 3

Paranthesis operator overloading

Real value is 5

Imaginary value is 4

Magnitude part is 6
```

10./*Define class vector of integers with all possible operations like constructor, destructor, copy constructor and assignment operators>, <, >=, <=, ==, ++ (pre and post), +, +=, ( ) */

```
#include<bits/stdc++.h>
using namespace std;
class vectors
{
 int n;
 vector<int> v;
 public:
  vectors()
  {
  }
  vectors(int x)
  {
   n=x;
  }
  vectors& operator <(const vectors &rhs)
  {
   int c1=0,c2=0;
   for(int i=0;i<n;i++)
   {
    if(v[i]<rhs.v[i])
    {
     c1++;
    }
    else if(v[i]>rhs.v[i])
    {
     c2++;
    }
   }
   if(c1<c2)
   {
    cout<<"2nd vector is smaller"<<endl;
   }
   else
   {
    cout<<"1st vector is smaller"<<endl;
   }
  }
  vectors& operator >(const vectors &rhs)
  {
   int c1=0,c2=0;
   for(int i=0;i<n;i++)
   {
    if(v[i]>rhs.v[i])
    {
     c1++;
```


```
  }
  else if(v[i]<rhs.v[i])
  {
   c2++;
  }
 }
 if(c1>c2)
 {
  cout<<"1st vector is larger"<<endl;
 }
 else if(c1<c2)
 {
  cout<<"2nd vector is larger"<<endl;
 }
}
vectors& operator <=(const vectors &rhs)
{
 int c1=0,c2=0;
 for(int i=0;i<n;i++)
 {
  if(v[i]<rhs.v[i])
  {
   c1++;
  }
  else if(v[i]>rhs.v[i])
  {
   c2++;
  }
 }
 if(c1<c2)
 {
  cout<<"2nd vector is smaller"<<endl;
 }
 else if(c1==c2)
 {
  cout<<"Both are same "<<endl;
 }
 else
 {
  cout<<"1st vector is smaller"<<endl;
 }
}
vectors& operator >=(const vectors &rhs)
{
 int c1=0,c2=0;
 for(int i=0;i<n;i++)
 {
  if(v[i]>rhs.v[i])
  {
```

```
  c1++;
 }
 else if(v[i]<rhs.v[i])
 {
  c2++;
 }
}
if(c1>c2)
{
 cout<<"1st vector is larger"<<endl;
}
else if(c1==c2)
{
 cout<<"Both are equal"<<endl;
}
else
{
 cout<<"2nd vector is larger"<<endl;
}
}
vectors& operator ==(const vectors &rhs)
{
 int cnt=0;
 for(int i=0;i<n;i++)
 {
  if(v[i]==rhs.v[i])
  {
   cnt++;
  }
 }
 if(cnt==n)
 {
  cout<<"Both vectors are equal"<<endl;
 }
 else
  cout<<"Both vectors are not equal"<<endl;
}
vectors& operator +=(const vectors &rhs)
{
 for(int i=0;i<n;i++)
 {
  v[i]=v[i]+rhs.v[i];
 }
}
vectors& operator +(const vectors &rhs)
{

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


```
      }
 }
 vectors& operator =(const vectors &rhs)
 {
  v=rhs.v;
  cout<<"Assignment operator printing "<<endl;
  return *this;
 }
 void print()
 {    for(int i=0;i<v.size();i++)    {     cout<<v[i]<<endl;    }   }
 void enter()
 {
  int t;
  cout<<"Enter elements "<<endl;
  for(int i=0;i<n;i++)
  {
   cin>>t;
   v.push_back(t);
  }
 }
};
 int main()
{
 vectors v1(3);
 v1.enter();
 vectors v2;
 /*v2=v1;
 v2.print();*/
 vectors v3(3);
 v3.enter();
 v1<v3;
 v1<=v3;
 v1>v3;
 v1>=v3;
 v1==v3;
 v1+=v3;
 v1.print();
}
```

Output:

```
Enter elements

1

2

3

Enter elements

4

5

6

1st vector is smaller

1st vector is smaller

2nd vector is larger

2nd vector is larger

Both vectors are not equal

5

7

9
```

11./*. Define class matrix of integers with all possible operations like constructor, destructor, copy constructor and assignment operators>, <, >=, <=, ==, ++ (pre and post), +, +=, ( )*/

```
#include<bits/stdc++.h>
using namespace std;

class matrix
{
 int rows,col,i,j;
 int **m;
 public:
  matrix()//Default Constructor
  {
  }
  matrix(int r,int c)// Parameterized constructor
  {
   rows = r;
   col=c;
   m=new int*[rows];
   for(i=0;i<rows;i++)
   {

    m[i]= new int [col];
   }
  }
  matrix(const matrix&cpy)//Copy Constructor
  {
   rows=cpy.rows;
   col=cpy.col;
   m=cpy.m;
  }
  matrix operator =(const matrix&rhs)//Assignment operator
  {
   rows=rhs.rows;
   col=rhs.col;
   m=rhs.m;
  }
  matrix operator >(const matrix &rhs)
  {
   int c1=0,c2=0;
   for(i=0;i<rows;i++)
   {
    for(j=0;j<col;j++)
    {
     if(m[i][j]>rhs.m[i][j])
     {
      c1++;
     }
     else if(m[i][j]<rhs.m[i][j])
      c2++;
```


```
  }
 }
 if(c1>c2)
  cout<<"First matrix is larger"<<endl;
 else
  cout<<"Second matrix is larger"<<endl;
}
matrix operator >=(const matrix &rhs)
{
 int c1=0,c2=0;
 for(i=0;i<rows;i++)
 {
  for(j=0;j<col;j++)
  {
   if(m[i][j]>rhs.m[i][j])
   {
    c1++;
   }
   else if(m[i][j]<rhs.m[i][j])
   {
    c2++;
   }
  }
 }
 if(c1>c2)
  cout<<"First matrix is larger"<<endl;
 else if(c1==c2)
  cout<<"Both matrices are equal"<<endl;
 else
  cout<<"Second matrix is larger"<<endl;
}
matrix operator <(const matrix &rhs)
{
 int c1=0,c2=0;
 for(i=0;i<rows;i++)
 {
  for(j=0;j<col;j++)
  {
   if(m[i][j]<rhs.m[i][j])
   {
    c1++;
   }
   else if(m[i][j]>rhs.m[i][j])
    c2++;
  }
 }
 if(c1>c2)
  cout<<"First matrix is smaller"<<endl;
 else
```


```
  cout<<"Second matrix is smaller"<<endl;
}
matrix operator <=(const matrix &rhs)
{
 int c1=0,c2=0;
 for(i=0;i<rows;i++)
 {
  for(j=0;j<col;j++)
  {
   if(m[i][j]<rhs.m[i][j])
   {
    c1++;
   }
   else if(m[i][j]>rhs.m[i][j])
    c2++;
  }
 }
 if(c1>c2)
  cout<<"First matrix is smaller"<<endl;
 else if(c1=c2)
  cout<<"Both matrices are equal"<<endl;
 else
  cout<<"Second matrix is smaller"<<endl;
}
matrix operator ==(const matrix &rhs)
{
 int cnt=0;
 for(i=0;i<rows;i++)
 {
  for(j=0;j<col;j++)
  {
   if(m[i][j]==rhs.m[i][j])
    cnt++;
  }
 }
 if(cnt==rows*col)
  cout<<"Both matrices are equal"<<endl;
 else
  cout<<"Both matrices are not equal"<<endl;
}
matrix operator +=(const matrix &rhs)
{
 for(i=0;i<rows;i++)
 {
  for(j=0;j<col;j++)
  {
   m[i][j]=m[i][j]+rhs.m[i][j];
  }
 }
```


```
}
matrix operator +(const matrix &rhs)
{
 for(i=0;i<rows;i++)
 {
  for(j=0;j<col;j++)
  {
   m[i][j]=m[i][j]+rhs.m[i][j];

  }
 }
}
matrix operator ++()//Pre increment
{
 for(i=0;i<rows;i++)
 {
  for(j=0;j<col;j++)
  {
   ++m[i][j];
  }
 }
}
matrix operator ++(int)//Post increment
{

 for(i=0;i<rows;i++)
 {
  for(j=0;j<col;j++)
  {
   m[i][j]++;
  }
 }
}

void enter()
{
 for(i=0;i<rows;i++)
 {
  for(j=0;j<col;j++)
  {
   cin>>m[i][j];
  }
 }
}
void print()
{
 for(i=0;i<rows;i++)
 {
  for(j=0;j<col;j++)
```


```
   {
    cout<<m[i][j]<<" ";
   }
   cout<<endl;
  }
 }
 ~matrix()//Destructor called
 {
 }
};

int main()
{
 matrix m1(2,2),m2(2,2);
 cout<<"Enter matrix 1 elements "<<endl;
 m1.enter();
 cout<<"Printing using assignment operator"<<endl;
 m2=m1;
 m2.print();
 matrix m3(m1);
 cout<<"Printing using copy constructor "<<endl;
 m3.print();
 cout<<"Enter matrix 2 elements "<<endl;
 matrix m4(2,2);
 m4.enter();
 m1>m4;
 m1<m4;
 m1<=m4;
 m1>=m4;
 m1==m4;
 m1+=m4;
 cout<<"Adding using shorthand operator"<<endl;
 m1.print();
 m1+m4;
 cout<<"Adding using binary operator"<<endl;
 m1.print();
 cout<<"Pre incrementing operators "<<endl;
 ++m3;
 m3.print();
 cout<<"Post incrementing operators "<<endl;
 m3++;
 m3.print();
}
```

Output:

```
3 4

Printing using copy constructor

1 2

3 4

Enter matrix 2 elements

5

6

7

8

Second matrix is larger

First matrix is smaller

First matrix is smaller

Second matrix is larger

Both matrices are not equal

Adding using shorthand operator

6 8

10 12

Adding using binary operator

11 14

17 20
```

```
Pre incrementing

operators 12 15

18 21

Post incrementing

operators 13 16

19 22
```

12./*Define class matrix of integers using vector, with all possible operations like constructor, destructor, copy constructor and assignment operators>, <, >=, <=, ==, ++ (pre and post), +, +=, ( ). */

```
#include<bits/stdc++.h>
using namespace std;

class matrix
{
 int rows,col,i,j;
 vector<vector<int> > m;
 public:
  matrix()//Default constructor
  {
  }
  matrix(int r,int c)
  {
   rows=r;
   col=c;
  }
  matrix(const matrix&cpy)//Copy Constructor
  {
   cout<<"Displaying using copy constructor"<<endl;
   rows=cpy.rows;
   col=cpy.col;
   m=cpy.m;
  }
  matrix& operator =(const matrix &rhs)
  {
   cout<<"Displaying using assignment operator"<<endl;
   rows=rhs.rows;
   col=rhs.col;
   m=rhs.m;
   return *this;
  }
  matrix& operator >(const matrix &rhs)
  {
   int c1=0,c2=0;
   for(i=0;i<rows;i++)
   {
    for(j=0;j<col;j++)
    {
     if(m[i][j]>rhs.m[i][j])
     {
      c1++;
     }
     else if(m[i][j]<rhs.m[i][j])
      c2++;
    }
```


```
 }
 if(c1>c2)
  cout<<"First matrix is larger"<<endl;
 else
  cout<<"Second matrix is larger"<<endl;
 return *this;
}
matrix& operator >=(const matrix &rhs)
{
 int c1=0,c2=0;
 for(i=0;i<rows;i++)
 {
  for(j=0;j<col;j++)
  {
   if(m[i][j]>rhs.m[i][j])
   {
    c1++;
   }
   else if(m[i][j]<rhs.m[i][j])
   {
    c2++;
   }
  }
 }
 if(c1>c2)
  cout<<"First matrix is larger"<<endl;
 else if(c1==c2)
  cout<<"Both matrices are equal"<<endl;
 else
  cout<<"Second matrix is larger"<<endl;
 return *this;
}
matrix& operator <(const matrix &rhs)
{
 int c1=0,c2=0;
 for(i=0;i<rows;i++)
 {
  for(j=0;j<col;j++)
  {
   if(m[i][j]<rhs.m[i][j])
   {
    c1++;
   }
   else if(m[i][j]>rhs.m[i][j])
    c2++;
  }
 }
 if(c1>c2)
  cout<<"First matrix is smaller"<<endl;
```


```
 else
  cout<<"Second matrix is smaller"<<endl;
 return *this;
}
matrix& operator <=(const matrix &rhs)
{
 int c1=0,c2=0;
 for(i=0;i<rows;i++)
 {
  for(j=0;j<col;j++)
  {
   if(m[i][j]<rhs.m[i][j])
   {
    c1++;
   }
   else if(m[i][j]>rhs.m[i][j])
    c2++;
  }
 }
 if(c1>c2)
  cout<<"First matrix is smaller"<<endl;
 else if(c1=c2)
  cout<<"Both matrices are equal"<<endl;
 else
  cout<<"Second matrix is smaller"<<endl;
 return *this;
}
matrix& operator ==(const matrix &rhs)
{
 int cnt=0;
 for(i=0;i<rows;i++)
 {
  for(j=0;j<col;j++)
  {
   if(m[i][j]==rhs.m[i][j])
    cnt++;
  }
 }
 if(cnt==rows*col)
  cout<<"Both matrices are equal"<<endl;
 else
  cout<<"Both matrices are not equal"<<endl;
 return *this;
}
matrix& operator +=(const matrix &rhs)
{
 for(i=0;i<rows;i++)
 {
  for(j=0;j<col;j++)
```


```
  {
   m[i][j]=m[i][j]+rhs.m[i][j];
  }
 }
 return *this;
}
matrix& operator +(const matrix &rhs)
{
 for(i=0;i<rows;i++)
 {
  for(j=0;j<col;j++)
  {
   m[i][j]=m[i][j]+rhs.m[i][j];
   }
 }
 return *this;
}
matrix& operator ++()//Pre increment
{
 for(i=0;i<rows;i++)
 {
  for(j=0;j<col;j++)
  {
   ++m[i][j];
  }
 }
 return *this;
}
matrix& operator ++(int)//Post increment
{

 for(i=0;i<rows;i++)
 {
  for(j=0;j<col;j++)
  {
   m[i][j]++;
  }
 }
 return *this;
}
void enter()
{
 cout<<"Enter elements "<<endl;
 for(i=0;i<rows;i++)
 {
  int t;
  vector<int> m1;
  for(j=0;j<col;j++)
  {
```


```
    cin>>t;
    m1.push_back(t);
   }
   m.push_back(m1);
  }
 }
 void print()
 {
  for(i=0;i<rows;i++)
  {
   for(j=0;j<col;j++)
   {
    cout<<m[i][j]<<"  ";
   }
   cout<<endl;
  }  }
 ~matrix()//Destructor
 {  }
};
 int main()
{
 matrix m1(2,2);
 m1.enter();
 m1.print();
 matrix m2(m1);
 m2.print();
 matrix m3;
 m3=m1;
 m3.print();
 matrix m4(2,2);
 m4.enter();
 m4>m1;
 m1<m4;
 m1<=m4;
 m1>=m4;
 m1==m4;
 m1+=m4;
 cout<<"Adding using shorthand operator"<<endl;
 m1.print();
 m1+m4;
 cout<<"Adding using binary operator"<<endl;
 m1.print();
 cout<<"Pre incrementing operators "<<endl;
 ++m3;
 m3.print();
 cout<<"Post incrementing operators "<<endl;
 m3++;
 m3.print();
}
```

Output:


```
Displaying using copy constructor

7  8

9  10

Displaying using assignment operator

7  8

9  10
Enter elements

11

12

13

14

First matrix is larger

First matrix is smaller

First matrix is smaller

Second matrix is larger

Both matrices are not equal

Adding using shorthand operator

18  20

22  24

Adding using binary operator

29  32

35  38

Pre incrementing operators

8  9

10  11

Post incrementing operators

9  10

11  12
```

13./* Define stack and queue inherited from array class, with standard functions and operators*/

```
#include<bits/stdc++.h>
using namespace std;
#define size 20
int top=-1,front=0,rear=0;
class arrays
{
 public:
  int a[size];
  void insert(int t)
  {
   a[++top]=t;
  }
};
class stacks:public arrays
{
 public:
  void push(int ele)
  {
   if(top>=size)
    cout<<"Stack is full"<<endl;
   else
    a[++top]=ele;
  }
  void pop()
  {
   int t;
   if(top<=-1)
    cout<<"Stack is empty "<<endl;
   else
   {
    cout<<"Element popped is "<<a[top]<<endl;
    top--;
   }
  }
  void display()
  {
   cout<<"Stack elements are "<<endl;
   for(int i=top;i!=-1;i--)
    cout<<a[i]<<endl;
  }
};
class queues:public arrays
{
 public:
  void enqueue(int t)
  {
```


```
  if(top>=size)
   cout<<"Queue is full"<<endl;
  else
   a[rear++]=t;
 }
 void dequeue()
 {
  int t;
  if(top==front)
   cout<<"Queue is empty "<<endl;
  else
  {
   cout<<"Element dequeue is "<<a[front]<<endl;
   front++;
  }
 }
 void displayq()
 {
  cout<<"Elements in queue are "<<endl;
  for(int i=front;i<rear;i++)
   cout<<a[i]<<endl;
 }
};
int main()
{
 arrays a;
 stacks s;
 s.push(10);
 s.push(20);
 s.push(30);
 s.pop();
 s.display();
 queues q;
 q.enqueue(50);
 q.enqueue(60);
 q.enqueue(70);
 q.displayq();
 q.dequeue();
 q.displayq();
}
```

Output:

```
Element popped is 30

Stack elements are

20

10

Elements in queue are

50

60

70

Element dequeue is 50

Elements in queue are

60

70
```

14./*Define a class called 'array' with data type passed as template type with constructor,destructor, copy constructor and assignment operators and index operator. */

```
#include<bits/stdc++.h>
using namespace std;

template<typename T> //General declaration of class or function template
class arrays
{
  T *a;
  int n;
  public:
   arrays()
   {
   }
   arrays(int size)//Constructor
   {
    n=size;
    a=new T[n];
   }
   arrays(const arrays&cpy)
   {
    cout<<"Copy constructor"<<endl;
    n=cpy.n;
    a=cpy.a;
   }
   arrays& operator =(const arrays&rhs)
   {
    cout<<"Assignment operator"<<endl;
    n=rhs.n;
    a=rhs.a;
   }
   void enter()
   {
    cout<<"Enter elements "<<endl;
    for(int i=0;i<n;i++)
     cin>>a[i];
   }
   void print()
   {
    for(int i=0;i<n;i++)
     cout<<a[i]<<endl;
   }
   ~arrays()//Destructor
   {    }
};
int main()
{
 //arrays <int>a1; ---General declaration of class template in main
 arrays <int>a1(5);
```


```
a1.enter();
arrays <int>a2(a1);
a2.print();
arrays <int>a3;
a3=a1;
a3.print(); }
```

Output:

```
Enter elements

1

2

3

4

5

Copy constructor

1

2

3

4

5

Assignment operator

1

2

3

4

5
```

15./*Define template functions for compare and use it in the algorithms like bubble sort, insertion sort, merge sort. */

```
#include<bits/stdc++.h>
using namespace std;

template<typename T>

T compare(T a[],int n)
{
 int i,j,t;
 for(i=0;i<n;i++)
 {
  for(j=0;j<n;j++)
  {
   if(a[i]>a[j])
   {
    swap(a[i],a[j]);
   }
  }
 }
}
template<typename T>
void BubbleSort(T a[],int n)
{
 int i,j,t;
 for(i=0;i<n;i++)
 {
  for(j=0;j<n;j++)
  {
   if(a[i]<a[j])
   {
    swap(a[i],a[j]);
   }
  }
 }
}
template<typename T>
void InsertionSort(T a1[],int n1)
{
 int i, j;
 T t;
 for (int i = 1; i < n1; ++i)
 {
  t = a1[i];
  j = i - 1;
  while (j >= 0 && a1[j] > t)
  {
   a1[j + 1] = a1[j];
   j = j - 1;
```


```
  }
  a1[j + 1] = t;
 }
}
template<typename T>
void Merge(T a2[], int l, int h)
{
   int z, x, y, m;
   vector<T> t(h -l + 1);
   m = (l + h) / 2;
   z = 0;
   x = l;
   y = m + 1;
   while (x <= m && y <= h)
   {
      if (a2[x]<a2[y])
      {
         t[z] = a2[x];
         ++x, ++z;
      }
      else
      {
         t[z] = a2[y];
         ++y, ++z;
      }
   }
   while(x<=m)
   {
      t[z] = a2[x];
      ++x, ++z;
   }
   while(y<=h)
   {
      t[z] = a2[y];
      ++y, ++z;
   }
   for(int i = l; i <= h; ++i)
   {
      a2[i] = t[i-l];
   }
}
template<typename T>
void MergeSort(T a2[],int l,int h)
{
 int m;
  if (l < h)
    {
       m= (l+ h) / 2;
       MergeSort(a2,l,m);
```

```
        MergeSort(a2, m+ 1, h);
        Merge(a2,l,h);
    }
}
template<typename T>
void print(T a[],int n)
{
 for(int i=0;i<n;i++)
  cout<<a[i]<<" ";
}
int main()
{
 int a[5]={5,6,3,8,2};
 cout<<"Printing using Bubble Sort"<<endl;
 BubbleSort<int>(a,5);
 print<int>(a,5);
 cout<<endl<<"Printing using Insertion Sort "<<endl;
 double a1[5]={9.0,10.7,6.5,6.3,4.5};
 InsertionSort<double>(a1,5);
 print<double>(a1,5);
 cout<<endl<<"Printing using Merge Sort "<<endl;
 int a2[5]={9,5,3,6,7};
 MergeSort<int>(a2,0,4);
 print<int>(a2,5);
}
```

Output:

```
Printing using Bubble Sort

2 3 5 6 8

Printing using Insertion Sort

4.5 6.3 6.5 9 10.7

Printing using Merge Sort

3 5 6 7 9
```

16./*Formatted input-output examples */

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
 int a=5;
 cout<<setw(5)<<a<<endl;//5 spaces width and then printing
 cout<<"Hii"<<endl;//For this 5 spaces width will not get printed
 double d=40.598687;
 cout<<setprecision(4)<<d<<endl;//Precision restricted to 4 characters
 char ch='K';
 cout.width(10);
 cout.fill('*');//By calling this it fills the blankspaces to symbol specied here in example *
 cout<<ch<<endl;
 cout<<"This is "<<setw(5)<<setfill(' ')<<"KRP"<<endl; //prints "This is" and 5 spaces and "KRP"
}
```

Output:

```
    5

Hii

40.6

*********K

This is   KRP
```

17./*Input manipulators*/

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
 string s;
 getline(cin,s);
 cout<<s<<endl;//endl gives cursor to next line
 cout<<"This is KRP"<<flush; // using flush makes the cursor remain same position
 cout<<"KRP"<<ends; // ends gives a  single blank character
}
```

Output:

```
Ram Priyatham

Ram Priyatham

This is KRPKRP
```

18./*Overriding operators <<, >> */

```
#include<bits/stdc++.h>
using namespace std;

class Complex
{
 int real,img;
 public:
  Complex(int r=0, int i=0)
  {
   real = r;img = i;
  }
 friend ostream & operator << (ostream &out, const Complex &c);
 friend istream & operator >> (istream &in,  Complex &c);
};
ostream & operator << (ostream &out, const Complex &c) //using const bcoz other objects cant modify
{
 out<< c.real;
 out<< "+i" <<c.img<<endl;
 return out;
}
istream & operator >> (istream &in, Complex &c)
{
 cout<<" Enter real part ";
 in>>c.real;
 cout<<" Enter Imaginary part ";
 in>>c.img;
}

int main()
{
 Complex c1;
 cin>>c1;

 cout<<"The Complex Object is ";
 cout<<c1;
 return 0;
}
```

Output:

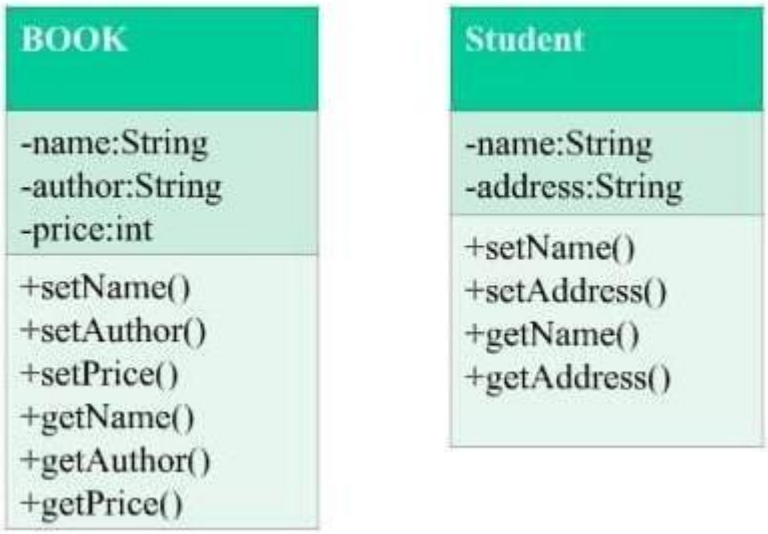
```
 Enter real part 10

 Enter Imaginary part 20

The Complex Object is 10+i20
```

R.V.R. & J.C.COLLEGE OF ENGINEERING (Autonomous)
Chandramoulipuram :: Chowdavaram :: Guntur-522019
(w.e.f. the academic year 2020-2021)
B.Tech., Computer Science and Business Systems

19./*Define class model for complex number, student class, book class and show it using UML diagram as well as concrete class.*/

```
#include<bits/stdc++.h>
using namespace std;

class BOOK
{
 public:
  string name,author;
  int price;

  void setName();
  void setAuthor();
  void setPrice();
  void getName();
  void getAuthor();
  void getPrice();
};

class Student()
{
 public:
  string name,address;

  void setName();
  void setAddress();
  void getName();
  void getAddress();
}
```

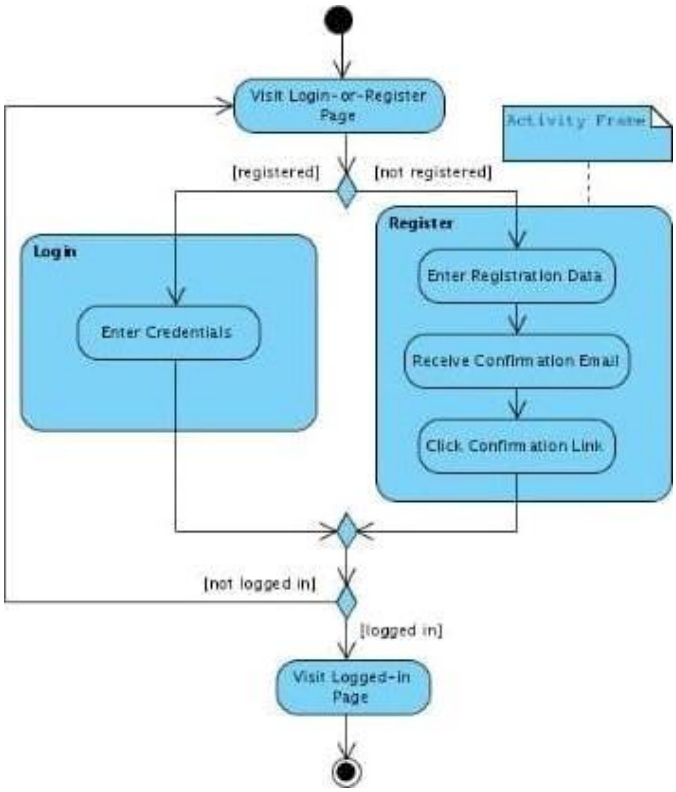
UML Diagram



R.V.R. & J.C.COLLEGE OF ENGINEERING (Autonomous)
Chandramoulipuram :: Chowdavaram :: Guntur-522019
(w.e.f. the academic year 2020-2021)
B.Tech., Computer Science and Business Systems

20. /*Show behavioural modelling through sequence diagram and activity diagram for workflow in a typical log-in, log-out situation.*/

- Activity diagram



- Sequence diagram