OOPS LAB ASSIGNMENT – 2

NAME : Swarnendu Banerjee

ROLL : 002311001016

DEPARTMENT : IT – A1

Q16. Write a simple class that represents a class of geometrical points each of which has three coordinates. The class should have appropriate constructor(s). Also add a member function distance() that calculates Euclidian distance between two points. Now create two points, find the distance between them and print it.

Code:

```
#include<iostream>

using namespace std;

#include <cmath>


class point{

  double x,y,z;

  public:

    point(double x=0,double y=0,double z=0):x(x),y(y),z(z){}


    double distance(point &p2){

     double d= sqrt(pow(x-p2.x,2)+pow(y-p2.y,2)+pow(z-p2.z,2));

     return d;

    }


};


int main() {

  point p1(1,1,1);

  point p2(2,2,2);

   cout << "The distance between the points is = " << p1.distance(p2);
  return 0;

}
```

Q17. Write a class for the geometrical shape rectangle. Write suitable constructors and member functions Add a member function area() that calculates the area of a rectangle. Create 4 rectangles and print their respective area.

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

class Rectangle{
int h;
 int b;
 public :
Rectangle(int h,int b)
{ this->h=h;
this->b=b;
}
int area() {
return h*b;
}
};
int main()
{
```

```
for(int i=0;i<4;i++)
{ int h,b;
 cout<<"enter height and breadth :\n"<<endl;

cin>>h>>b;

Rectangle ob(h,b);
cout<< " Area = "<<ob.area()<<endl;
 }
}
```

18. Write a class that represents a class of wireless device. A device has a location (point object may be used), a fixed unique id, and a fixed circular transmission range. Write suitable constructors and member functions for this class. Instantiates 10 such devices. Choose location (coordinates) and transmission range of the devices randomly. Now, for each of these devices, find the neighbor devices (i.e. devices that belong to the transmission range). Suppose, all of these devices have moved to a new location (randomly chosen). Find out the new set of neighbors for each of these devices.

CODE:

```
#include<iostream>
#include<math.h>
using namespace std;

class Devices
{
```

```
int x,y,uid,r;
public:
Devices()
{
   this->x = 0;
   this->y = 0;
   this->uid = 0;
   this-> r = 0;
}

void get(int a, int b, int c, int d)
{
   this->x = a;
   this->y = b;
   this->uid = c;
   this-> r = d;
}

void Neighbour(Devices a, Devices b)
{
   double dist = sqrt((a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y));
   if(dist<r)
   cout << b.uid << " ";
}
```

```
  int getUID()
  {
     return this->uid;
  }

};
int main()
{
   Devices arr[10];

   for(int i = 0; i < 10; i++)
   {
      int a,b,d;
      cout << "Enter co-ordinates of the " << i+1 << " device : ";
      cin >> a >> b;
      cout << "Enter transmission range of the " << i+1 << " device :
";
      cin >> d;
      arr[i].get(a,b,i+1,d);
   }

   for(int i = 0; i < 10; i++)
   {
      cout << "Neighbours of " << arr[i].getUID() << " are : ";
      for(int j = 0; j < 10; j++)
```

```
        {
           if(i!=j)
           {
              arr[i].Neighbour(arr[i],arr[j]);
           }

        }
        cout << "\n";
    }
  return 0;
}
```

19. Write a class Vector for one dimensional array. Write suitable constructor/copy constructor. Also add member functions for perform basic operations (such as addition, subtraction, equality, less, greater etc.). Create vectors and check if those operations are working correctly.

CODE:

```
#include <iostream>
#include <vector>

using namespace std;

class Vector {
   int size;
   int* arr;
```

```
public:
  Vector(int size = 0) : size(size), arr(new int[size]) {}

  void input() {
    cout << "Enter the elements of the vector: ";
    for (int i = 0; i < size; i++) {
      cin >> arr[i];
    }
  }

  void add( Vector& b) {
    cout << "Summation Values of two vectors is: ";
    for (int i = 0; i < size; i++) {
      cout << arr[i] + b.arr[i] << " ";
    }
    cout << "\n";
  }

  void subtract(Vector& b) {
    cout << "Subtracted Values of two vectors are: ";
    for (int i = 0; i < size; i++) {
      cout << arr[i] - b.arr[i] << " ";
    }
    cout << "\n";
  }
```

```
    int compare( Vector& b) {
        for (int i = 0; i < size; i++) {
            if (arr[i] < b.arr[i]) {
                return -1;
            } else if (arr[i] > b.arr[i]) {
                return 1;
            }
        }
        return 0;
    }

    ~Vector() {
        delete[] arr;
    }



};

int main() {
    int n;
    cout << "Enter the size of the arrays: ";
    cin >> n;

    Vector A(n), B(n);
    A.input();
    B.input();

    A.add(B);
    B.subtract(A);

    int comparisonResult = A.compare(B);
    if (comparisonResult == -1) {
        cout << "Second vector is greater than first vector\n";
    } else if (comparisonResult == 1) {
        cout << "First vector is greater than second vector\n";
    } else {
        cout << "Both the vectors are equal\n";
    }

    return 0;
}
```

20. Write a class IntArray for one dimensional integer array. Implement the necessary constructor, copy constructor, and destructor (if necessary) in this class. Implement other member functions to perform operations, such adding two arrays, reversing an array, sorting an array etc. Create an IntArray object having elements 1, 2 and 3 in it. Print its elements. Now, create another IntArray object which is an

exact copy of the previous object. Print its elements. Now, reverse the elements of the last object. Finally print elements of both the objects.

CODE:

```
#include <iostream>
#include <algorithm>

using namespace std;

class IntArray {
   int size;
   int* arr;

public:

   IntArray(int size) : size(size), arr(new int[size]) {}



   IntArray(const IntArray& other) : size(other.size), arr(new
int[other.size]) {
       for (int i = 0; i < size; i++) {
           arr[i] = other.arr[i];
       }
   }
```

```
}



void input() {
   cout << "Enter the elements of the array: ";
   for (int i = 0; i < size; i++) {
      cin >> arr[i];
   }
}



void add(const IntArray& b) {
   if (size != b.size) {
      cout << "Arrays must have the same size for addition";
      return;
   }
   IntArray result(size);
   cout << "Summation values of two arrays: ";
   for (int i = 0; i < size; i++) {
      result.arr[i] = arr[i] + b.arr[i];
      cout << result.arr[i] << " ";
   }
   cout << endl;
}

void reverse() {
    for (int i = 0; i < size / 2; i++) {
        int temp = arr[i];
        arr[i] = arr[size - i - 1];
        arr[size - i - 1] = temp;
    }
}


void Sort() {
    bool swapped = true;
    for (int i = 0; swapped; i++) {
        swapped = false;
        for (int j = 0; j < size - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                swap(arr[j], arr[j + 1]);
                swapped = true;
            }
        }
    }
}

void print() const {
    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }
```

```
        cout << endl;
    }
    ~IntArray() {
        delete[] arr;
    }

};

int main() {
    int n;
    cout << "Enter the size of the array: ";
    cin >> n;

    IntArray array1(n);
    array1.input();

    cout << "Array 1:" << endl;
    array1.print();

    IntArray array2(array1);
    cout << "Array 2 (copied from Array 1):" << endl;
    array2.print();



    array2.reverse();
    cout << "Reversed Array 2 :" << endl;
    array2.print();



    array2.Sort();
    cout << "Array 2 (sorted using bubble sort):" << endl;
    array2.print();

    cout << "Array 1 (unchanged):" << endl;
    array1.print();



    array1.add(array2);

    return 0;
}
```

21. Create a simple class SavingsAccount for savings account used in banks as follows: Each SavingsAccount object should have three data members to store the account holder's name, unique account number and balance of the account. Assume account numbers are integers and generated sequentially. Note that once an account number is allocated to an account, it does not change during the entire operational period of the account. The bank also specifies a rate of interest for all savings accounts created. Write relevant methods (such as withdraw, deposit etc.) in the class. The bank restricts that each account must have a minimum balance of Rs. 1000. Now create 100 SavingsAccount objects specifying balance at random ranging from Rs. 1,000 to 1,00,000. Now, calculate the interest for one year to be paid to each

account and deposit the interest to the corresponding balance. Also find out total amount of interest to be paid to all accounts in one year.

CODE:

```
#include <iostream>

#include <vector>

#include <cstdlib>

#include <ctime>

#include <numeric>


using namespace std; class

SavingsAccount { private:

   static int accountCounter;

string name;     int

accountNumber;     double

balance;     static const double

interestRate;


public:

   SavingsAccount( string name, double balance) : name(name),
accountNumber(++accountCounter), balance(balance) {}

   void deposit(double amount) {

balance += amount;

   }
   void withdraw(double amount) {
if (balance - amount >= 1000) {
balance -= amount;
       } else {
            cout << "Cannot withdraw. Minimum balance required is
1000." <<  endl;
       }
   }

   void addInterest() {         double interest =
balance * interestRate;
deposit(interest);
   }

   double getBalance() const {
return balance;
   }

   static double totalInterestPaid(const  vector<SavingsAccount>&
accounts)   {                     return     accumulate(accounts.begin(),
accounts.end(), 0.0,
[](double sum, const SavingsAccount& acc) {
           return sum + acc.balance * interestRate;
       });
   }
};
```

```
int SavingsAccount::accountCounter = 0; const
double SavingsAccount::interestRate = 0.04;

int main() {     srand(static_cast<unsigned
int>(time(0)));

     vector<SavingsAccount> accounts;     for (int i = 0;
i < 100; ++i) {
accounts.push_back(SavingsAccount("Account" +
to_string(i+1), 1000 + rand() % 100000));
    }

    for (auto& account : accounts) {
account.addInterest();
    }

    double totalInterest = SavingsAccount::totalInterestPaid(accounts);
cout << "Total interest paid to all accounts: " << totalInterest <<  endl;

    return 0;
}
```

22. Write some programs to understand the notion of constant member functions, mutable data members etc.

CODE:

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

class Demo {
private:     int a;
mutable int b;

public:
   Demo(int x, int y) : a(x), b(y) {}

   int getA() const {
return a;
   }

   int getB() const {
return b;
   }

   void modifyB(int newValue) const {
      b = newValue;
   }
};

int main() {
```

```
Demo obj(10, 20);


    cout << "a: " << obj.getA() <<  endl;
cout << "b: " << obj.getB() <<  endl;


   obj.modifyB(30);
 cout << "b after modification: " << obj.getB() <<  endl;


   return 0;
}
```

23. Write the definition for a class called Complex that has private floating point data members for storing real and imaginary parts. The class has the following public member functions:

setReal() and setImg() to set the real and imaginary part respectively.

getReal() and getImg() to get the real and imaginary part respectively.

disp() to display complex number object sum() to sum two complex numbers & return a complex number Write main function to create three complex number objects. Set the value in two objects and call sum() to calculate sum and assign it in third object. Display all complex numbers.

CODE:

```
#include <iostream>
```

```
using namespace std;

class Complex {
private:
    float real;
    float img;

public:

    Complex(): real(0), img(0) {}

    void setReal(float r) {
        real = r;
    }

    void setImg(float i) {
        img = i;
    }



    float getReal()  {
        return real;
    }

    float getImg()  {
        return img;
    }


    void disp()  {
        if (img >= 0) {
            cout << real << " + " << img << "i" << endl;
        } else {
            cout << real << " - " << -img << "i" << endl;
        }
    }


    Complex sum( Complex& c) {
        Complex result;
        result.real = real + c.real;
        result.img = img + c.img;
        return result;
    }
};

int main() {
    Complex c1, c2, c3;
```

```
float r1, i1, r2, i2;

cout << "Enter real and imaginary part for complex number 1: ";
cin >> r1 >> i1;
c1.setReal(r1);
c1.setImg(i1);

cout << "Enter real and imaginary part for complex number 2: ";
cin >> r2 >> i2;
c2.setReal(r2);
c2.setImg(i2);

c3 = c1.sum(c2);



cout << "Complex number 1: ";
c1.disp();

cout << "Complex number 2: ";
c2.disp();

cout << "Sum of complex numbers (c1 + c2): ";
c3.disp();

return 0;
```

}

24. Complete the class with all function definitions for a stack

class Stack { int *buffer, top; public :

Stack(int); //create a stack with specified size void push(int);

//push the specified item int pop(); //return the top element void

disp(); //displays elements in the stack in top to bottom order

};

Now, create a stack with size 10, push 2, 3, 4 and 5 in that order and finally pop one element. Display elements present in the stack.

CODE:

```
#include<iostream>

using namespace std;


class Stack {

int *buffer, top,size;

public :

Stack(int size):size(size),buffer(new int[size]),top(-1){};

void push(int x){

  if(top>=size){

  cout << "Stack overflow!" <<  endl;

  }

  else{

  top++;

  buffer[top]=x;}
}
int pop(){
 if(top==-1){
   return -1;
 }
 else{
 return buffer[top--];
}}
void disp(){
  for (int i = top; i >= 0; --i) {
             cout << buffer[i] << " ";
        }
         cout <<  endl;
    }};

int main() {
 Stack s(10);
s.push(2);
s.push(3);
s.push(4);
s.push(5);
s.disp();
s.pop();
s.disp();
return 0;
```

}

25. Complete the class with all function definitions for a circular queue class Queue { int *data; int front, rear; public :

Queue(int); //create queue with specified size void

add(int);//add specified element to the queue int

remove();//delete element from the queue

void disp(); //displays all elements in the queue(front to rear order)

};

Now, create a queue with size 10 add 2, 3, 4 and 5 in that order and finally delete two elements. Display elements present in the stack.

CODE:

```
#include <iostream>

using namespace std;

class Queue {
private:
   int* data;
   int front, rear, size, count;

public:
   Queue(int size) : size(size), front(0), rear(0), count(0),  data(new
int[size]){
   }

   void add(int element) {
      if (count < size) {
         data[rear] = element;
         rear = (rear + 1) % size;
```

```
            ++count;
        } else {
             cout << "Queue overflow!" <<  endl;
        }
    }

    int remove() {
        if (count > 0) {
            int element = data[front];
            front = (front + 1) % size;
            --count;
            return element;
        } else {
             cout << "Queue underflow!" <<  endl;
            return -1;
        }
    }

    void disp() const {
        int i = front;
        int items = count;
        while (items > 0) {
             cout << data[i] << " ";
            i = (i + 1) % size;
            --items;
        }
         cout <<  endl;
    }
};

int main() {
    Queue q(10);

    q.add(2);
    q.add(3);
    q.add(4);
    q.add(5);

    q.disp();

    q.remove();
    q.remove();

    q.disp();

    return 0;
}
```

26. Write a class for your Grade card. The grade card is given to each student of a department per semester. The grade card typically contains the name of the department, name of the student, roll number, semester, a list of subjects with their marks and a calculated CGPA. Create 60 such grade cards in a 3rd semester with relevant data and find the name and roll number of student having highest CGPA.

CODE:

```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>

using namespace std;
class Subject {
```

```
public:
    string name;
   int marks;

   Subject( string name, int marks) : name(name), marks(marks) {}
};

class GradeCard {
private:
    string department;
    string studentName;
    string rollNumber;
   int semester;
    vector<Subject> subjects;
   double cgpa;

   void calculateCGPA() {
      int totalMarks = 0;
      for (const auto& subject : subjects) {
         totalMarks += subject.marks;
      }
      cgpa = static_cast<double>(totalMarks) / subjects.size();
   }

public:
   GradeCard( string department,  string studentName,  string
rollNumber, int semester, const  vector<Subject>& subjects)
       : department(department), studentName(studentName),
rollNumber(rollNumber), semester(semester), subjects(subjects) {
       calculateCGPA();
   }

   double getCGPA() const {
       return cgpa;
   }

    string getStudentName() const {
       return studentName;
   }

    string getRollNumber() const {
       return rollNumber;
   }
};

int main() {
    vector<GradeCard> gradeCards;

   for (int i = 1; i <= 60; ++i) {
        vector<Subject> subjects = {
           {"Math", 85 + rand() % 15},
           {"Physics", 80 + rand() % 20},
           {"Chemistry", 75 + rand() % 25},
           {"Computer Science", 90 + rand() % 10},
           {"Biology", 70 + rand() % 30},
       };
       gradeCards.push_back(GradeCard("CSE", "Student" +
to_string(i), "Roll" +  to_string(i), 3, subjects));
   }

   auto maxCGPA =  max_element(gradeCards.begin(),
gradeCards.end(), [](const GradeCard& a, const GradeCard& b) {
       return a.getCGPA() < b.getCGPA();
   });

    cout << "Highest CGPA: " << maxCGPA->getCGPA() <<  endl;
    cout << "Student Name: " << maxCGPA->getStudentName() <<
endl;
    cout << "Roll Number: " << maxCGPA->getRollNumber() <<
endl;

   return 0;
}
```

27. Create a class for Book. A book has unique isbn (string), title, a list of authors, and a price. Write relevant functions. Now write a class BookStore which has a list of books. There may be multiple copies of a book in the book store. Write relevant member functions. Write a function books() that returns list of unique isbn numbers of the books, a function noOfCopies() that returns the number of copies available for a given isbn number and a function totalPrice() that returns the total price of all the books. Create a book store having a number of books (multiple copies). Now, for each book, print number of copies of that book along with its title.

CODE:

```
#include <iostream>
#include <vector>
#include <map>
using namespace std;
class Book {
private:
   string isbn;
   string title;
   vector<string> authors;
   double price;

public:
   Book(string isbn, string title, const vector<string>& authors,
double price)
      : isbn(isbn), title(title), authors(authors), price(price) {}

   string getISBN() const {
      return isbn;
   }

   string getTitle() const {
      return title;
   }
```

```
    double getPrice() const {
        return price;
    }
};

class BookStore {
private:
    map<string, int> inventory;

public:
    void addBook(const Book& book, int quantity) {
        inventory[book.getISBN()] += quantity;
    }

    void sellBook(const string& isbn) {
        if (inventory[isbn] > 0) {
            --inventory[isbn];
        } else {
            cout << "Book out of stock!" << endl;
        }
    }

    void disp() const {
        for (const auto& pair : inventory) {
            cout << "ISBN: " << pair.first << " - Quantity: " <<
pair.second << endl;
        }
    }
};

int main() {
    BookStore store;
    Book book1("123456789", "C++ Primer", {"Stanley Lippman",
"JoséLajoie"}, 49.99);
    Book book2("987654321", "Effective C++", {"Scott Meyers"},
39.99);

    store.addBook(book1, 10);
    store.addBook(book2, 5);

    store.disp();

    store.sellBook("123456789");

    store.disp();

    return 0;
}
```