CHITKARA INHERITANCE (10 Marks)

Q1. The examination department prints the result of N students by implementing three classes:

1. **Student,**
2. **Marks**
3. **Result**.

Class **‘Student’** contains the **roll\_number**  and two member functions **set\_roll\_number()** and **get\_roll\_number()** to read and display the **Roll number**.

Class **‘Marks’** inherits class **Student** and contains the marks of two subjects as **maths, physics** as its data members and two member functions **set\_marks() and get\_marks()** to get and display the marks for two subjects.

Class **‘Result’** further inherits from **class Marks** and display Roll number, Marks of two subjects and Percentage of the student in the following form:

##### **Output Format**

#### Roll number = 678

#### Marks of Math's = 92

#### Marks of Physics = 98

#### Percentage = 95

**Test case 1**

**Output**

#### Roll number = 678

#### Marks of Math's = 92

#### Marks of Physics = 98

#### Percentage = 95

#### DRIVER CODE:

int main()

{

    Result R;

    R.set\_roll\_number(678);

    R.set\_marks(92,98);

    R.display\_result();

    return 0;

}

**Test case 2**

**Output**

#### Roll number = 987

#### Marks of Math's = 95

#### Marks of Physics = 98

#### Percentage = 96.5

#### DRIVER CODE:

int main()

{

    Result R;

    R.set\_roll\_number(987);

    R.set\_marks(95,98);

    R.display\_result();

    return 0;

}

**Test case 3**

**Output**

#### Roll number = 876

#### Marks of Math's = 92

#### Marks of Physics = 95

#### Percentage = 93.5

#### DRIVER CODE:

int main()

{

    Result R;

    R.set\_roll\_number(876);

    R.set\_marks(92,95);

    R.display\_result();

    return 0;

}

**Complete Code**

#include<iostream>

using namespace std;

class Student

{

    protected:

        int roll\_number;

    public:

        void set\_roll\_number(int);

        void get\_roll\_number(void);

};

void Student :: set\_roll\_number(int r)

{

    roll\_number = r;

}

void Student :: get\_roll\_number()

{

    cout<<"Roll number = "<< roll\_number <<endl;

}

class Marks : public Student

{

  protected:

      float maths;

      float physics;

   public:

      void set\_marks(float,float);

      void get\_marks(void);

};

void Marks :: set\_marks(float m, float p)

{

    maths = m;

    physics = p;

}

void Marks :: get\_marks()

{

     cout<< "Marks of Math's = "<< maths<<endl;

     cout<< "Marks of Physics = "<< physics<<endl;

}

class Result : public Marks

{

     float percentage;

     public:

void display\_result()

{

    get\_roll\_number();

    get\_marks();

    cout<< "Percentage = "<<(maths+physics)/2<<endl;

}

};

int main()

{

    Result R;

    R.set\_roll\_number(876);

    R.set\_marks(92,95);

    R.display\_result();

    return 0;

}

CHITKARA WALL (5 Marks)

2. Write a C++ program to calculate the area of a wall using class object and Parameterized constructor

**Test case 1**

**Output**

Area of Wall 1: 90.3

Area of Wall 2: 53.55

#### DRIVER CODE:

int main()

{

  Wall wall1(10.5, 8.6);

  Wall wall2(8.5, 6.3);

  cout << "Area of Wall 1: " << wall1.calculateArea() << endl;

  cout << "Area of Wall 2: " << wall2.calculateArea();

  return 0;

}

**Test case 2**

**Output**

Area of Wall 1: 143.64

Area of Wall 2: 142.35

#### DRIVER CODE:

int main()

{

  Wall wall1(18.9, 7.6);

  Wall wall2(19.5, 7.3);

  cout << "Area of Wall 1: " << wall1.calculateArea() << endl;

  cout << "Area of Wall 2: " << wall2.calculateArea();

  return 0;

}

**Test case 3**

**Output**

#### Area of Wall 1: 349.6

#### Area of Wall 2: 700.8

#### DRIVER CODE:

int main()

{

  Wall wall1(46, 7.6);

  Wall wall2(96, 7.3);

  cout << "Area of Wall 1: " << wall1.calculateArea() << endl;

  cout << "Area of Wall 2: " << wall2.calculateArea();

  return 0;

}

Complete Code

#include <iostream>

using namespace std;

class Wall

{

  private:

    double length;

    double height;

  public:

    // parameterized constructor to initialize variables

    Wall(double len, double hgt)

    {

      length = len;

      height = hgt;

    }

    double calculateArea()

    {

      return length \* height;

    }

};

int main()

{

  // create object and initialize data members

  Wall wall1(46, 7.6);

  Wall wall2(96, 7.3);

  cout << "Area of Wall 1: " << wall1.calculateArea() << endl;

  cout << "Area of Wall 2: " << wall2.calculateArea();

  return 0;

}

Addition and Subtraction using Pointers (5 Marks)

#### Q3.You have given two integer variables, say a and b and two pointer type variables pa and pb. pa and pb contains the memory address of a and b respectively. Call the function update() which modify the value of a and b in memory. The updated values of a and b are the sum (a+b) and absolute difference |(a-b) | respectively. Complete the update () function in the drive source code.

Function Description :

The function update() accepts the following parameters:

1 int \*a: an integer

2 int \*b: an integer

and is declared with a void return type, so there is no value to return. Modify the values in memory only so that these integer variables ‘a’ and ‘b’ must be updated with sum and absolute difference respectively.

a=a+b

b=|a-b|

#### **Input Format :**

#### Input will contain two integers, and , separated by a newline.

#### **Output Format :**

#### The output will appear in two lines. 1st line contains their sum 2nd line contains the difference

#### **Sample Input**

40 // first integer value

50 // second integer value

#### 

#### **Sample Output :**

90 // Their sum i.e., 40+50=90

10 // Their absolute difference i.e.,|40-50| = 10

**Test case 1**

**Output**

#### 99

#### 31

#### DRIVER CODE:

int main()

{

int a = 34;

int b = 65;

int \*pa = &a, \*pb = &b;

update(pa, pb);

cout<<a<<endl;

cout<<b<<endl;

return 0;

}

**Test case 2**

**Output**

#### 108

#### 22

#### DRIVER CODE:

int main()

{

int a = 43;

int b = 65;

int \*pa = &a, \*pb = &b;

update(pa, pb);

cout<<a<<endl;

cout<<b<<endl;

return 0;

}

**Test case 3**

**Output**

#### 97

#### 33

#### DRIVER CODE:

int main()

{

int a = 65;

int b = 32;

int \*pa = &a, \*pb = &b;

update(pa, pb);

cout<<a<<endl;

cout<<b<<endl;

return 0;

}

COMPLETE CODE

#include <iostream>

using namespace std;

void update(int \*a,int \*b)

{

    int p=\*a;

    \*a=(\*a)+(\*b);

    int k=p-(\*b);

    if(k<0)

    {

     k=k\*(-1);

    }

    \*b=k;

}

int main()

{

int a = 65;

int b = 32;

int \*pa = &a, \*pb = &b;

update(pa, pb);

cout<<a<<endl;

cout<<b<<endl;

return 0;

}

Q4.Write a c++ program to calculate interest rate using function overloading

**Test case 1**

**Output**

1.2

4.2

#### DRIVER CODE:

int main(){

int PA=3000;

int T=2;

float R=0.02;

int PA1=6000;

int T1=1;

float R1=0.07;

Interestrate(PA,T,R);

Interestrate(PA1,R1,T1);

}

**Test case 2**

**Output**

2.1

4

#### DRIVER CODE:

int main(){

int PA=7000;

int T=3;

float R=0.01;

int PA1=2000;

int T1=4;

float R1=0.05;

Interestrate(PA,T,R);

Interestrate(PA1,R1,T1);

}

**Test case 3**

**Output**

4.5

#### DRIVER CODE:

int main(){

int PA=1000;

int T=5;

float R=0.09;

int PA1=5000;

int T1=2;

float R1=0.01;

Interestrate(PA,T,R);

Interestrate(PA1,R1,T1);

}

COMPLETE CODE

#include<iostream>

using namespace std;

void Interestrate(int x,int y, float z)

{

cout<<((x\*y\*z)/100)<<endl;

}

void Interestrate(int x,float y, int z){

cout<<((x\*y\*z)/100)<<endl;

}

int main(){

int PA=1000;

int T=5;

float R=0.09;

int PA1=5000;

int T1=2;

float R1=0.01;

Interestrate(PA,T,R);

Interestrate(PA1,R1,T1);

}

Q5. Write a c++ program that uses an area() function for the calculation of area of a traingle or a rectangle or a square. Number of sides 3 for triangle ,2 for rectangle and 1 for square suggest about the shape for which area is to be calculated.

**Test case 1**

**Output**

Area of triangle: 2.90474

Area of rectangle: 30

Area of square: 16

#### DRIVER CODE:

int main(){

area(2,3,4);

area(5,6);

area(4); }

**Test case 2**

**Output**

Area of triangle: 14.6969

Area of rectangle: 10

Area of square: 36

#### DRIVER CODE:

int main(){

area(5,6,7);

area(2,5);

area(6);

}

**Test case 3**

**Output**

Area of triangle: 6

Area of rectangle: 120

Area of square: 36

#### DRIVER CODE:

int main(){

area(4,3,5);

area(10,12);

area(6);

}

COMPLETE CODE

#include<bits/stdc++.h>

using namespace std;

void area(float a,float b,float c){

float s=(a+b+c)/2;

float ans=sqrt(s\*(s-a)\*(s-b)\*(s-c));

cout<<"Area of triangle: "<<ans<<endl;

}

void area(float a,float b){

cout<<"Area of rectangle: "<<a\*b<<endl;

}

void area(float b){

cout<<"Area of square: "<<b\*b<<endl;

}

int main(){

area(2,3,4);

area(5,6);

area(4);

}

Q6. Calculate the sum of the diagonal of 4\*4 matrix where N=4.Print the total sum of both the diagonal.

**Test case 1**

**Output**

Total sum of diagonal: 68

#### DRIVER CODE:

int main(){

int arr[4][4]={{1,2,3,4},

{5,6,7,8},

{9,10,11,12},

{13,14,15,16}};

diagonalsum(arr,4);

}

**Test case 2**

**Output**

Total sum of diagonal: 4

#### DRIVER CODE:

int main(){

int arr[4][4]={{1,0,0,1},

{1,0,0,1,},

{1,0,0,1},

{1,0,0,1}};

diagonalsum(arr,4);

}

**Test case 3**

**Output**

Total sum of diagonal: 14

#### DRIVER CODE:

int main(){

int arr[4][4]={{1,0,0,1},

{1,2,3,1,},

{1,3,2,1},

{1,0,0,1}};

diagonalsum(arr,4);

}

COMPLETE CODE

#include<bits/stdc++.h>

using namespace std;

void diagonalsum(int arr[4][4],int N){

int sum=0;

for(int i=0;i<N;i++){

sum+=arr[i][i];

}

for(int i=0;i<N;i++ ){

sum+=arr[i][N-i-1];

}

cout<<"Total sum of diagonal: "<<sum<<endl;

}

int main(){

int arr[4][4]={{1,0,0,1},

{1,2,3,1,},

{1,3,2,1},

{1,0,0,1}};

diagonalsum(arr,4);

}

Q7. Find the element which is present once in an array and all other element have their duplicate present in an array.

**Test case 1**

**Output**

3

#### DRIVER CODE:

int main(){

int arr[5]={1,1,2,2,3};

cout<<uniqueOne(arr,5);

}

**Test case 2**

**Output**

7

#### DRIVER CODE:

int main(){

int arr[6]={7,2,2,6,6,6};

cout<<uniqueOne(arr,6); }

**Test case 3**

**Output**

8

#### DRIVER CODE:

int main(){

int arr[8]={1,0,8,0,2,2,0,1};

cout<<uniqueOne(arr,8);

}

COMPLETE CODE

#include<bits/stdc++.h>

using namespace std;

int uniqueOne(int \*arr,int n){

sort(arr,arr+n);

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

if(i==0){

if(arr[i]!=arr[i+1]){

return arr[i];

}

}

if(i>0){

if(arr[i]!=arr[i+1] and arr[i]!=arr[i-1]){

return arr[i];

}

}

}

return arr[n-1];

}

int main(){

int arr[5]={1,1,2,2,3};

cout<<uniqueOne(arr,5);

}

Lucky Numbers (Recursion)(5 marks)

Q7.Lucky numbers are subset of integers. Rather than going into much theory, let us see the process of arriving at lucky numbers,

Take the set of integers

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,……

First, delete every second number, we get following reduced set.

1, 3, 5, 7, 9, 11, 13, 15, 17, 19,…………

Now, delete every third number, we get

1, 3, 7, 9, 13, 15, 19,….….

Continue this process indefinitely……

Any number that does NOT get deleted due to above process is called “lucky”.

Note: Gap start from 2,3,4……

Input:

N = 5

Output:

0

Explanation: 5 is not a lucky number

as it gets deleted in the second

iteration.

Test Case 1:

Output:

1

DRIVER CODE:

int main(){

int n = 19;

if(isLucky(n,2)){

cout<<"1";

}

else{

cout<<"0";

}

}

Test Case 2:

Output:

0

DRIVER CODE:

int main(){

int n = 23;

if(isLucky(n)){

cout<<"1";

}

else{

cout<<"0";

}

}

Test Case 3:

Output:

1

DRIVER CODE:

int main(){

int n = 25;

if(isLucky(n)){

cout<<"1";

}

else{

cout<<"0";

}

}

Complete Code:-

#include <bits/stdc++.h>

using namespace std;

bool isLucky(int n){

static int gap = 2;

if(n % gap == 0){

return 0;

}

if(n < gap){

return 1;

}

n -= (n/gap);

gap++;

return isLucky(n);

}

int main(){

int n = 5;

cout << isLucky(n) << endl;

return 0;

}

Matrix Multiplication(10 Marks)(2D Array):-

Q8. You are given 2 matrices of size N \* M and O \* P. Check If they can be multiplied. If, they can be multiplied, then, multiply them and print a single matrix of size N \* O. Else print “Not Possible”

Input:

mat1 = { mat2 = {

{1,2,3}, {3,4,5,6},

{4,5,6}, {7,8,9,10},

{7,8,9} {11,12,13,14}

} }

Output:

50 56 62 68

113 128 143 158

176 200 224 248

Test Case 1:

Output:

10 10 10 10

20 20 20 20

30 30 30 30

40 40 40 40

DRIVER CODE:

const int n = 3, m = 3, o = 3, p = 4;

int main()

{

int mat1[n][m] = {

{1, 2, 3},

{4, 5, 6},

{7, 8, 9}};

int mat2[o][p] = {

{3, 4, 5, 6},

{7, 8, 9, 10},

{11, 12, 13, 14}};

matrixMultiplication(mat1, mat2);

return 0;

}

Test Case 2:

Output:

1024 1174 1234

2132 2466 2580

2925 3128 3381

2312 2746 2860

2369 2746 2873

DRIVER CODE:

const int n = 5, m = 4, o = 4, p = 3;

int main()

{

int mat1[n][m] = {

{11, 12, 13, 14},

{24, 25, 26, 29},

{17, 28, 39, 59},

{34, 35, 16, 19},

{27, 28, 32, 30},

};

int mat2[o][p] = {

{13, 34, 25},

{37, 28, 39},

{11, 12, 13},

{21, 22, 23},

};

matrixMultiplication(mat1, mat2);

return 0;

}

Test Case 3:

Output:

“Not Possible”

DRIVER CODE:

const int n = 3, m = 2, o = 3, p = 5;

int main()

{

int mat1[n][m] = {

{10, 2},

{9, 5},

{7, 15}

};

int mat2[o][p] = {

{1, 14, 5},

{7, 8, 19},

{11, 2, 9},

};

matrixMultiplication(mat1, mat2);

return 0;

}

Complete Code

#include <bits/stdc++.h>

using namespace std;

const int n = 5, m = 4, o = 4, p = 3;

void matrixMultiplication(int mat1[n][m], int mat2[o][p])

{

if (m != o)

{

cout << "Not Possible" << endl;

return;

}

int ans[n][p];

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

{

for (int j = 0; j < p; j++)

{

ans[i][j] = 0;

for (int k = 0; k < m; k++)

{

ans[i][j] += mat1[i][k] \* mat2[k][j];

}

}

}

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

{

for (int j = 0; j < p; j++)

{

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

}

cout << endl;

}

}

int main()

{

int mat1[n][m] = {

{11, 12, 13, 14},

{24, 25, 26, 29},

{17, 28, 39, 59},

{34, 35, 16, 19},

{27, 28, 32, 30},

};

int mat2[o][p] = {

{13, 34, 25},

{37, 28, 39},

{11, 12, 13},

{21, 22, 23},

};

matrixMultiplication(mat1, mat2);

return 0;

}

Matrix Transpose(5 Marks)(2D Array):-

Q9. You are given a matrix of size N \* N. Complete the findTranspose() function so that it can find the transpose of the given matrix.

NOTE: You have to do the swapping in the given matrix only

Input:

mat = {

{1,2},

{3,4}

}

Output:

1 3

2 4

Test Case 1:

Output:

1 4 7

2 5 8

3 6 9

DRIVER CODE:

const int n = 3;

int main()

{

int mat[n][n] = {

{1, 2, 3},

{4, 5, 6},

{7, 8, 9},

};

findTranspose(mat);

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

{

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

{

cout << mat[i][j] << " ";

}

cout << endl;

}

return 0;

}

Test Case 2:

Output:

10 14 37 37

12 15 28 28

13 16 19 19

12 19 9 9

DRIVER CODE:

const int n = 4;

int main()

{

int mat[n][n] = {

{10, 12, 13, 12},

{14, 15, 16, 19},

{37, 28, 19, 9},

{37, 28, 19, 9},

};

findTranspose(mat);

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

{

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

{

cout << mat[i][j] << " ";

}

cout << endl;

}

return 0;

}

Test Case 3:

Output:

1 1 3 3 5

2 5 2 8 6

4 6 9 7 9

5 9 8 6 3

7 2 1 9 1

DRIVER CODE:

const int n = 5;

int main()

{

int mat[n][n] = {

{1, 2, 4, 5, 7},

{1, 5, 6, 9, 2},

{3, 2, 9, 8, 1},

{3, 8, 7, 6, 9},

{5, 6, 9, 3, 1},

};

findTranspose(mat);

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

{

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

{

cout << mat[i][j] << " ";

}

cout << endl;

}

return 0;

Complete Code

#include <bits/stdc++.h>

using namespace std;

const int n = 5;

void findTranspose(int mat[n][n])

{

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

{

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

{

swap(mat[i][j], mat[j][i]);

}

}

}

int main()

{

int mat[n][n] = {

{1, 2, 4, 5, 7},

{1, 5, 6, 9, 2},

{3, 2, 9, 8, 1},

{3, 8, 7, 6, 9},

{5, 6, 9, 3, 1},

};

findTranspose(mat);

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

{

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

{

cout << mat[i][j] << " ";

}

cout << endl;

}

return 0;

}

VOLUME OF CUBOID (OOPS)(5 marks)

Q10. An abstract class ThreeDshape is available to the programmer containing two member variables dimension1 and dimension2. Dimension 1 and dimension 2 are initialized in the default constructor of ThreeDshape. The class Cuboid is inherited from the class ThreeDshape. The default constructor of class Cuboid initializes the dimension3. Cuboid class has to implement the volume function given in ThreeDshape. Compute and display the volume of the cuboid in main function.

NOTE: No Dimension should not be negative or zero. If so, output should be “Invalid Cuboid”

Sample Input:

-3 // Dimension 1

2 // Dimension 2

5 // Dimension 3

Output

Invalid Cuboid

Test Case 1:

Output:

40

DRIVER CODE:

int main()

{

Cuboid c(2, 4, 5);

c.volume();

return 0;

}

Test Case 2:

Output:

Invalid Cuboid

DRIVER CODE:

int main()

{

Cuboid c(2, -4, 5);

c.volume();

return 0;

}

Test Case 3:

Output:

Invalid Cuboid

DRIVER CODE:

int main()

{

Cuboid c(2, -4, -5);

c.volume();

return 0;

}

Complete Code

#include <iostream>

using namespace std;

class ThreeDshape

{

public:

int dim1, dim2;

ThreeDshape()

{

dim1 = 0;

dim2 = 0;

}

};

class Cuboid : ThreeDshape

{

public:

int dim3;

Cuboid(int dim1, int dim2, int dim3)

{

this->dim1 = dim1;

this->dim2 = dim2;

this->dim3 = dim3;

}

void volume()

{

if (dim1 > 0 && dim2 > 0 && dim3 > 0)

{

cout << dim1 \* dim2 \* dim3;

}

else

{

cout << "Invalid Cuboid";

}

}

};

int main()

{

Cuboid c(2, 4, 5);

c.volume();

return 0;

}