**INDEX**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| S.No. | Name of the Experiment | Date | Page No. | Marks  Awarded | Remarks |
| 1 | Sequence Generator | 02-01-25 | 1 |  |  |
| 2 | Function to swap two numbers | 02-01-25 | 2 |  |  |
| 3 | Demonstrates default arguments | 02-01-25 | 3 |  |  |
| 4 | Class declaration definition, accessing | 09-01-25 | 4 |  |  |
|  | class members |  |  |  |  |
| 5 | Default Constructor, Parameterised an- | 09-01-25 | 5 |  |  |
|  | d destructor for a class |  |  |  |  |
| 6 | Inheritances | 23-01-25 | 6 |  |  |
| 7 | Constructor and destructor program | 23-01-25 | 8 |  |  |
| 8 | Demonstration of pointers | 30-01-25 | 9 |  |  |
| 9 | Demonstration of Overloading, Opera- | 30-01-25 | 10 |  |  |
|  | tor overloading and overriding |  |  |  |  |
| 10 | Demonstration of friend functions, | 05-02-25 | 11 |  |  |
|  | inline functions |  |  |  |  |
| 11 | Demonstration of virtual, static | 05-02-25 | 12 |  |  |
|  | function |  |  |  |  |
| 12 | Demonstrates friend functions, inline | 12-02-25 | 14 |  |  |
|  | function |  |  |  |  |
| 13 | Program to create an array of objects | 12-02-25 | 15 |  |  |
| 14 | Demonstrates Try and Catch blocks | 19-02-25 | 16 |  |  |
| 15 | Demonstrate the Catching of All | 19-02-25 | 17 |  |  |
|  | Exceptions |  |  |  |  |
| 16 | Demonstrates user defined exceptions | 06-03-25 | 18 |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**CPP RECORD**

**WEEK-1:**

1. **AIM:** Write a program to generate the following sequence

1

1 2

1 2 3

1 2 3 4

***PROGRAM:***

#include<iostream>

**using** **namespace** std;

**int** main(){

**for**(**int** i=1; i<=4; i++){

**for**(**int** j=1; j<=i; j++){

cout << j << " ";

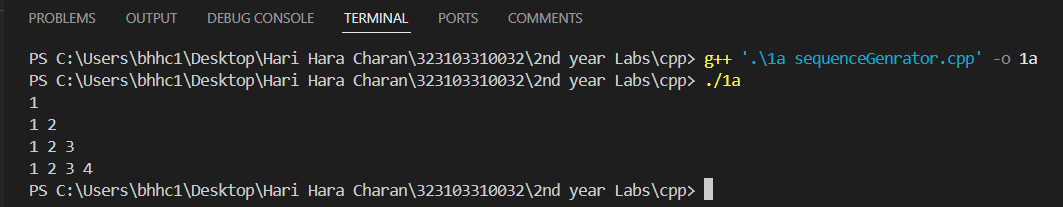
}

cout << endl;

}

}

***Output:***

****

1. **AIM:** Write a program which uses function to swap two integers and two float numbers by using reference variables.

***PROGRAM:***

#include<iostream>

**using** **namespace** std;

**void** swap(**int** &a, **int** &b){

**int** temp = a;

a = b;

b = temp;

}

**void** swap(**float** &a, **float** &b){

**float** temp = a;

a = b;

b = temp;

}

**int** main(){

**int** a,b;

**float** c,d;

cout << "Enter two integers: ";

cin >> a >> b;

cout << "Enter two float numbers: ";

cin >> c >> d;

cout<<"Integer numbers before swapping: "<<a<<" "<<b<<endl;

cout<<"Float numbers before swapping: "<<c<<" "<<d<<endl;

swap(a,b);

swap(c,d);

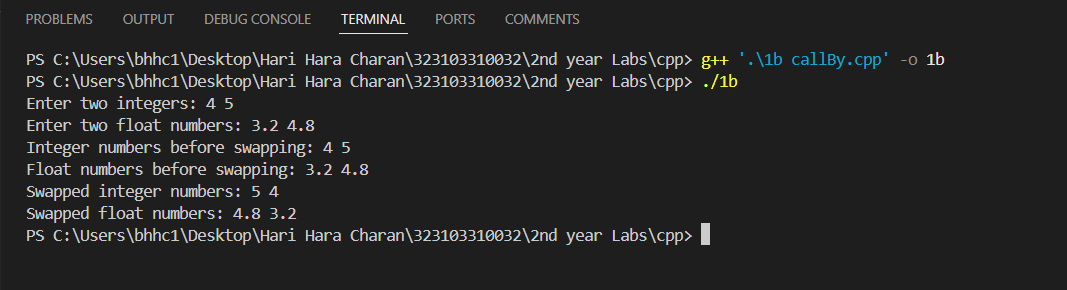
cout<<"Swapped integer numbers: "<<a<<" "<<b<<endl;

cout<<"Swapped float numbers: "<<c<<" "<<d<<endl;

**return** 0;

}

***Output:***

****

1. **AIM:** Write a program that demonstrates default arguments.

***PROGRAM:***

#include<iostream>

**using** **namespace** std;

**void** defaultPara(**int** a, **int** b, **int** c=0, **int** d=0){

cout << "a: " << a <<", b: " << b << ", c: " << c << ", d: " << d << endl;

}

**int** main(){

**int** a,b,c,d;

cout<<"Enter 4 integers: ";

cin >> a >> b >> c >> d;

defaultPara(a,b);

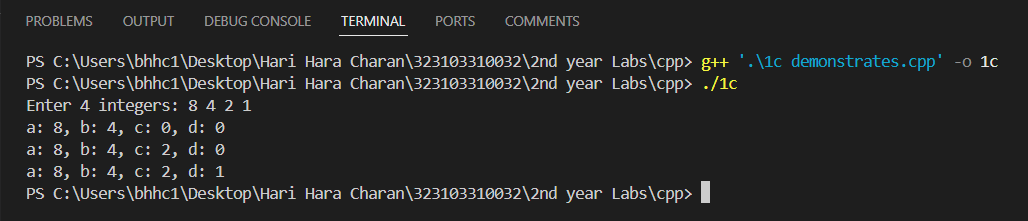
defaultPara(a,b,c);

defaultPara(a,b,c,d);

**return** 0;

}

***Output:***

****

**WEEK-1:**

1. **AIM:** Write a program Illustrating Class Declarations, Definition, and Accessing Class Members

***PROGRAM:***

#include <iostream>

**using** **namespace** std;

**class** Employee

{

public:

string name;

**double** salary;

**void** display()

{

cout << "Name: " << name << endl

<< "Salary: " << salary << endl;

}

};

**int** main()

{

Employee e1;

cout << "Enter name: ";

cin >> e1.name;

cout << "Enter salary: ";

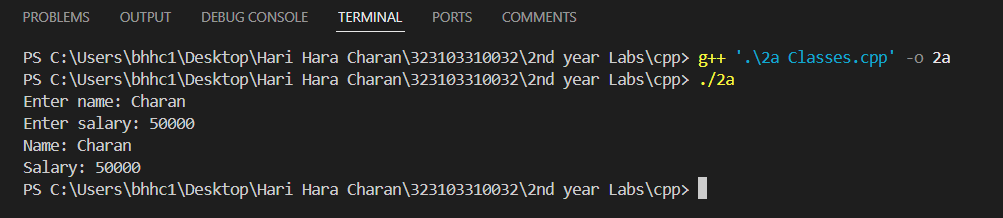
cin >> e1.salary;

e1.display();

**return** 0;

}

***Output:***

****

1. **AIM:** Write a program to illustrate default constructor, parameterized constructor and copy constructor, destructors for a class

***PROGRAM:***

#include <iostream>

**using** **namespace** std;

**class** Demo{

public:

**int** value;

Demo(){

value = 100;

cout << "Default constructor called" << endl;

}

Demo(**int** a){

value = a;

cout << "Parameterized constructor called"<< endl;

}

Demo(**const** Demo &obj){

value = obj.value;

cout << "Copy constructor called" << endl;

}

~Demo(){

cout << "Destructor called" << endl;

}

};

**int** main(){

Demo d1;

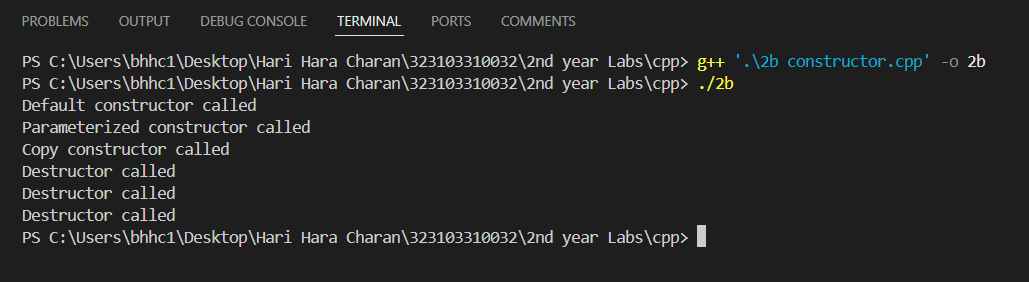
Demo d2(200);

Demo d3 = d1;

**return** 0;

}

***Output:***

****

**WEEK-3:**

1. **AIM:** Write a program that illustrates the following forms of inheritances Single Multiple Multilevel Hierarchical

***PROGRAM:***

#include<iostream>

**using** **namespace** std;

*// Parent class*

**class** Parent {

public:

**void** display() {

cout << "Parent class display function" << endl;

}

};

*// Single Inheritance*

**class** SingleChild : **public** Parent {

public:

**void** show() {

cout << "Single Child class show function" << endl;

}

};

*// Multiple Inheritance*

**class** AnotherParent {

public:

**void** print() {

cout << "Another Parent class print function" << endl;

}

};

**class** MultipleChild : **public** Parent, **public** AnotherParent {

public:

**void** reveal() {

cout << "Multiple Child class reveal function" << endl;

}

};

*// Multilevel Inheritance*

**class** IntermediateChild : **public** Parent {

public:

**void** interFunction() {

cout << "Intermediate Child class function" << endl;

}

};

**class** MultilevelChild : **public** IntermediateChild {

public:

**void** multilevelFunction() {

cout << "Multilevel Child class function" << endl;

}

};

*// Hierarchical Inheritance*

**class** HChild1 : **public** Parent {

public:

**void** hFunction1() {

cout << "Hierarchical Child class 1 function" << endl;

}

};

**class** HChild2 : **public** Parent {

public:

**void** hFunction2() {

cout << "Hierarchical Child class 2 function" << endl;

}

};

**int** main() {

*// Single Inheritance*

SingleChild sc;

sc.display();

sc.show();

*// Multiple Inheritance*

MultipleChild mc;

mc.display();

mc.print();

mc.reveal();

*// Multilevel Inheritance*

MultilevelChild mlc;

mlc.display();

mlc.interFunction();

mlc.multilevelFunction();

*// Hierarchical Inheritance*

HChild1 hc1;

hc1.display();

hc1.hFunction1();

HChild2 hc2;

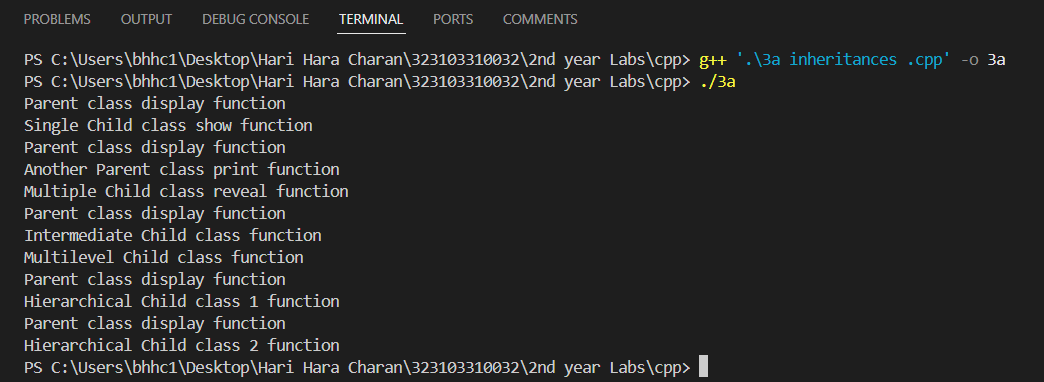
hc2.display();

hc2.hFunction2();

**return** 0;

}

***Output:***

****

1. **AIM:** Create multiple objects for the class and observe the order in which constructors and destructors are called.

***PROGRAM:***

#include<iostream>

**using** **namespace** std;

**class** Base{

public:

**int** a;

Base(**int** a){

**this**->a=a;

cout<<"constructor"<<a<<endl;

}

~Base(){

cout<<"destructor"<<a<<endl;

}

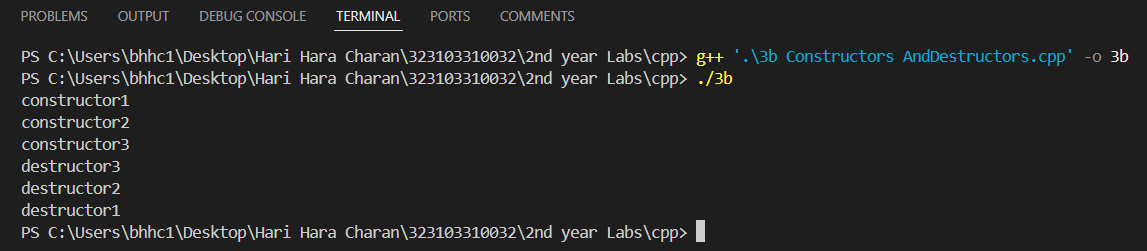
};

**int** main(){

Base b1(1),b2(2),b3(3);

}

***Output:***

****

**WEEK-4:**

1. **AIM:** Write a program to use pointers for both base and derived classes and call the member functions.

***PROGRAM:***

#include <iostream>

**using** **namespace** std;

**class** Base {

public:

**int** base\_var;

**void** show() {

cout << "Base class called" << endl;

cout << "Base class variable: " << base\_var << endl;

}

};

**class** Derived : **public** Base {

public:

**int** derived\_var;

**void** show() {

cout << "Derived class called" << endl;

cout << "Base class variable: " << base\_var << endl;

cout << "Derived class variable: " << derived\_var << endl;

}

};

**int** main() {

Base b;

Derived d;

Base \*bptr;

Derived \*dptr;

bptr = &b;

dptr = &d;

bptr->base\_var = 100;

dptr->base\_var = 200;

dptr->derived\_var = 300;

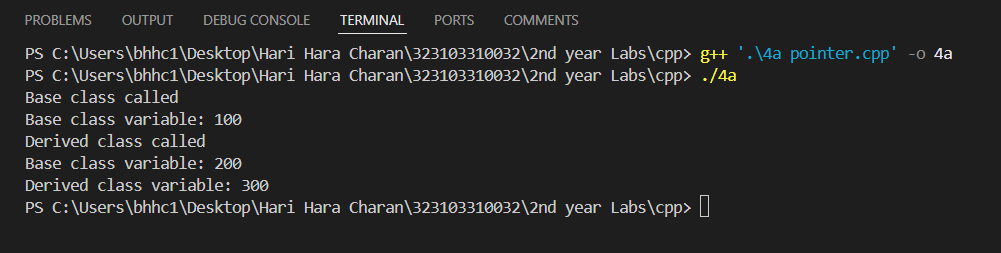
bptr->show();

dptr->show();

**return** 0;

}

***Output:***

******

**(b)AIM:** Write a program that demonstrates function overloading, operator overloading, overriding

***PROGRAM:***

#include <iostream>

**using** **namespace** std;

**class** Base {

public:

**int** base\_var;

**void** show() {

cout << "Base class called" << endl;

cout << "Base class variable: " << base\_var << endl;

}

**void** show(**int** x) {

cout << "Base class called with parameters" << endl;

cout << "Base class variable: " << x << endl;

}

**void** **operator**+(**int** x) {

base\_var += x;

}

};

**class** Derived : **public** Base {

public:

**int** derived\_var;

**void** show() {

cout << "Derived class called" << endl;

cout << "Base class variable: " << base\_var << endl;

cout << "Derived class variable: " << derived\_var << endl;

}

**void** show(**int** x) {

cout << "Derived class called" << endl;

cout << "Base class variable: " << x << endl;

cout << "Derived class variable: " << derived\_var << endl;

}

**void** **operator**+(**int** x) {

base\_var += x;

derived\_var += x;

}

};

**int** main() {

Base b;

Derived d;

Base \*bptr;

Derived \*dptr;

bptr = &b;

dptr = &d;

bptr->base\_var = 100;

dptr->base\_var = 200;

dptr->derived\_var = 300;

bptr->show();

dptr->show();

bptr->show(500);

dptr->show(600);

bptr->**operator**+(100);

dptr->**operator**+(200);

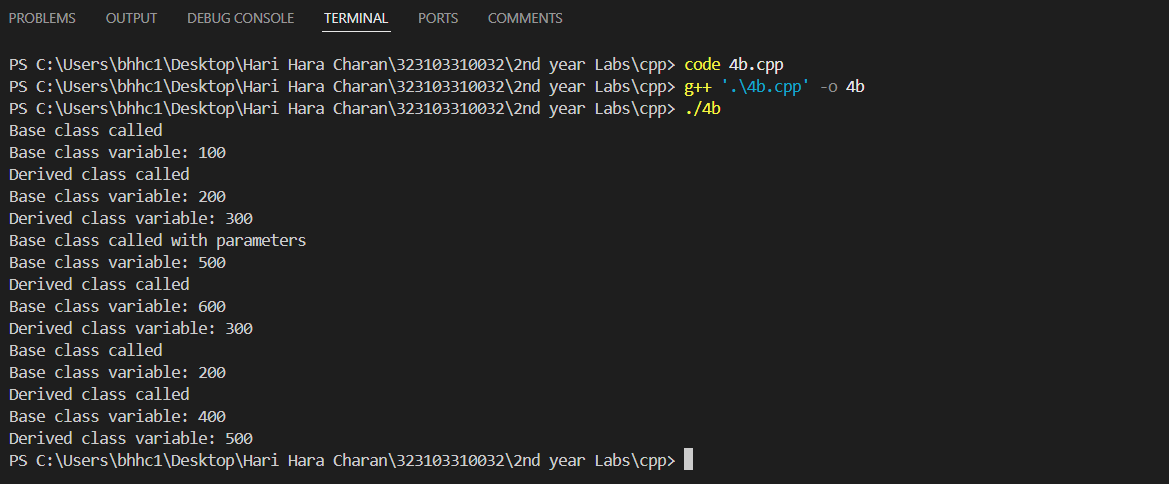
bptr->show();

dptr->show();

**return** 0;

}

***Output:***



**WEEK-5:**

1. **AIM:** Write a program that demonstrates friend functions, inline functions,

***PROGRAM:***

#include <iostream>

**using** **namespace** std;

**class** Rectangle {

private:

**int** length, width;

public:

Rectangle(**int** l, **int** w) : length(l), width(w) {}

**inline** **int** area() {

**return** length \* width;

}

**friend** **void** printDimensions(Rectangle rect);

};

**void** printDimensions(Rectangle rect) {

cout << "Length: " << rect.length << ", Width: " << rect.width << endl;

}

**int** main() {

**int** l, w;

cout << "Enter length and width: ";

cin >> l >> w;

Rectangle rect(l, w);

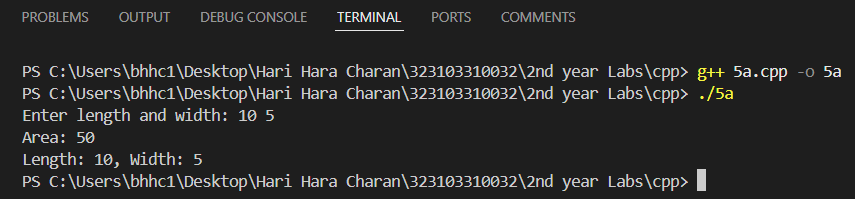
cout << "Area: " << rect.area() << endl;

printDimensions(rect);

**return** 0;

}

***Output:***



**(b) AIM:** Write a program that demonstrates virtual, static functions.

***PROGRAM:***

#include <iostream>

**using** **namespace** std;

**class** Base {

public:

**virtual** **void** show() {

cout << "Base class show()" << endl;

}

**static** **void** display() {

cout << "Base class static display()" << endl;

}

};

**class** Derived : **public** Base {

public:

**void** show() override {

cout << "Derived class show()" << endl;

}

**static** **void** display() {

cout << "Derived class static display()" << endl;

}

};

**int** main() {

Base\* basePtr;

Derived dObj;

basePtr = &dObj;

basePtr->show();

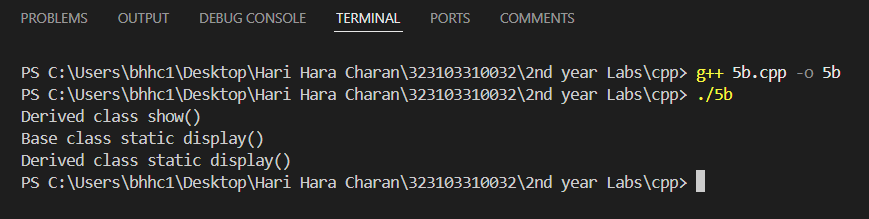
Base::display();

Derived::display();

**return** 0;

}

***Output:***



**WEEK-6:**

1. **AIM:** Write a program that demonstrates friend functions, inline functions.

***PROGRAM:***

#include <iostream>

**using** **namespace** std;

**class** Rectangle {

public:

**int** length, width;

Rectangle(**int** l, **int** w) : length(l), width(w) {}

**void** display() {

cout << "Length: " << length << ", Width: " << width << endl;

}

};

Rectangle modifyRectangle(Rectangle r) {

r.length += 10;

r.width += 10;

**return** r;

}

**int** main() {

Rectangle rect1(10, 20);

cout << "Original: ";

rect1.display();

Rectangle rect2 = modifyRectangle(rect1);

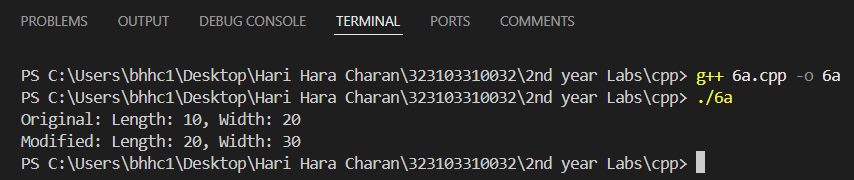
cout << "Modified: ";

rect2.display();

**return** 0;

}

***Output:***



**(b) AIM:** Write a program to create an array of objects.

***PROGRAM:***

#include <iostream>

**using** **namespace** std;

**class** Student {

public:

string name;

**int** age;

Student() {}

Student(string n, **int** a) : name(n), age(a) {}

**void** display() {

cout << "Name: " << name << ", Age: " << age << endl;

}

};

**void** displayStudents(Student students[], **int** size) {

**for** (**int** i = 0; i < size; i++) {

students[i].display();

}

}

**int** main() {

**const** **int** SIZE = 8;

Student students[SIZE] = {

{"Charan", 19}, {"Bharat", 20}, {"Tarun", 18},

{"Praveen", 20}, {"Akhil", 19}, {"Chandu", 18},

{"Babasree", 20}, {"Ramaraju", 18}

};

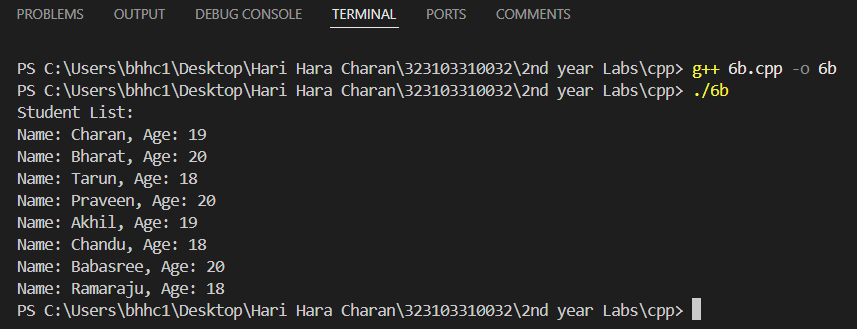
cout << "Student List:\n";

displayStudents(students, SIZE);

**return** 0;

}

***Output:***



**WEEK-7:**

1. **AIM:** Write a program that handles Exceptions. Use a Try Block to Throw it and a Catch Block to Handle it Properly

***PROGRAM:***

#include <iostream>

**int** main() {

try {

**int** a = 10, b = 0;

**if** (b == 0) {

throw std::runtime\_error("Division by zero error");

}

std::cout << "Result: " << (a / b) << std::endl;

} catch (**const** std::exception &e) {

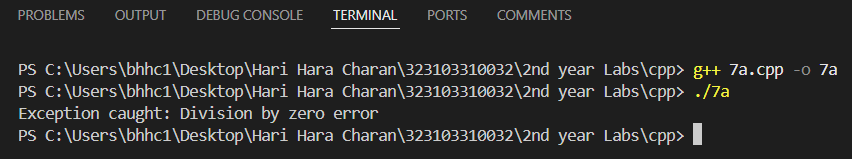
std::cout << "Exception caught: " << e.what() << std::endl;

}

**return** 0;

}

***Output:***

******

**(b)Aim:** Write a Program to Demonstrate the Catching of All Exceptions

***PROGRAM:***

#include <iostream>

#include <stdexcept>

**int** main() {

try {

throw "An unknown error occurred";

} catch (**const** std::runtime\_error& e) {

std::cout << "Runtime error caught: " << e.what() << std::endl;

} catch (**const** std::out\_of\_range& e) {

std::cout << "Out of range error caught: " << e.what() << std::endl;

} catch (**const** **char**\* msg) {

std::cout << "Exception caught: " << msg << std::endl;

} catch (...) {

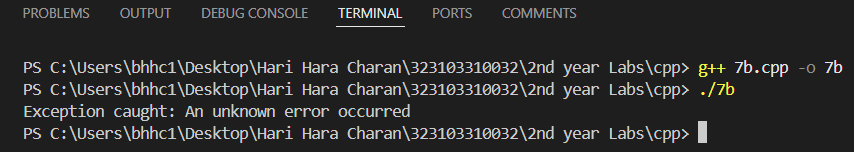
std::cout << "Exception caught: An unknown error occurred" << std::endl;

}

**return** 0;

}

***Output:***

****

**WEEK-8:**

1. **Aim:** Write a Program to demonstrates user defined exceptions

***PROGRAM:***

#include <iostream>

#include <exception>

class MyException : public std::exception {

public:

**const** **char**\* what() **const** noexcept override {

**return** "User-defined exception occurred!";

}

};

**int** main() {

try {

throw MyException();

} catch (**const** MyException &e) {

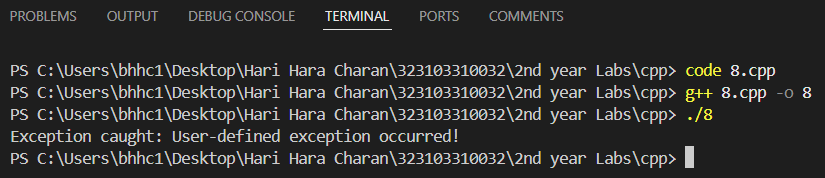
std::cout << "Exception caught: " << e.what() << std::endl;

}

**return** 0;

}

***Output:***



**WEEK-9:**

1. **Aim:** Write a program to create a generic template for adding two integers and two float values and make use of the template to perform addition.

***PROGRAM:***

#include <iostream>

#include <conio.h>

**using** **namespace** std;

**template** <**class** t>

t Sum(t a, t b){

**return** a + b;

}

**int** main() {

system("cls");

cout << "Sum of integers: " << Sum(5, 9) << endl;

cout << "Sum of floats: " << Sum(1.1, 2.2) << endl;

cout << "Sum of boolean: " << Sum(true, false) << endl;

cout << "Sum of char: " << Sum('!', '/') << endl;

**return** 0;

}

***Output:***

******

**(b)Aim:** Write a program to create a generic template pass two different types of parameters to the same function and using classes

***PROGRAM:***

#include <iostream>

#include <conio.h>

**using** **namespace** std;

**template** <**class** T1, **class** T2>

**class** A {

T1 a;

T2 b;

public:

A(T1 x, T2 y) {

a = x;

b = y;

}

**void** sum() {

cout << "sumition of the variables: " << a + b << endl;

}

};

**int** main() {

system("cls");

A<**int**, **double**> obj1(5, 9.5);

obj1.sum();

A<**float**, **int**> obj2(1.1, 2);

obj2.sum();

A<**bool**, **int**> obj3(true, 2);

obj3.sum();

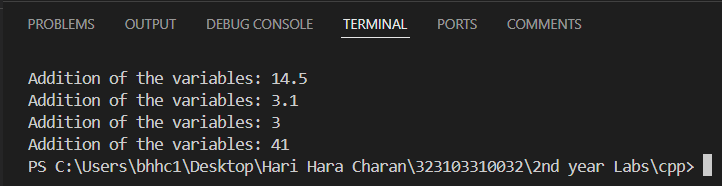
A<**char**, **int**> obj4('!', 8);

obj4.sum();

**return** 0;

}

***Output:***

****

**WEEK-10:**

1. **Aim:** Write a program to implement the matrix ADT using a class. The operations supported by this ADT are:

a) Addition of two matrices

b) subtraction of two matrices

c) Multiplication of two matrices

***PROGRAM:***

#include <iostream>

using namespace std;

class Matrix {

private:

**int** rows, cols;

**int** mat[10][10];

public:

Matrix(**int** r = 0, **int** c = 0) {

rows = r;

cols = c;

}

**void** inputMatrix() {

cout << "Enter elements for " << rows << "x" << cols << " matrix:\n";

**for** (**int** i = 0; i < rows; i++) {

**for** (**int** j = 0; j < cols; j++) {

cin >> mat[i][j];

}

}

}

**void** displayMatrix() **const** {

cout << "Matrix:\n";

**for** (**int** i = 0; i < rows; i++) {

**for** (**int** j = 0; j < cols; j++) {

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

}

cout << endl;

}

}

Matrix add(**const** Matrix& m) {

Matrix result(rows, cols);

**for** (**int** i = 0; i < rows; i++)

**for** (**int** j = 0; j < cols; j++)

result.mat[i][j] = mat[i][j] + m.mat[i][j];

**return** result;

}

Matrix subtract(**const** Matrix& m) {

Matrix result(rows, cols);

**for** (**int** i = 0; i < rows; i++)

**for** (**int** j = 0; j < cols; j++)

result.mat[i][j] = mat[i][j] - m.mat[i][j];

**return** result;

}

Matrix multiply(**const** Matrix& m) {

Matrix result(rows, m.cols);

**for** (**int** i = 0; i < rows; i++)

**for** (**int** j = 0; j < m.cols; j++) {

result.mat[i][j] = 0;

**for** (**int** k = 0; k < cols; k++)

result.mat[i][j] += mat[i][k] \* m.mat[k][j];

}

**return** result;

}

};

**int** main() {

**int** r1, c1, r2, c2;

cout << "Enter rows and columns for Matrix-1: ";

cin >> r1 >> c1;

Matrix mat1(r1, c1);

mat1.inputMatrix();

cout << "Enter rows and columns for Matrix-2: ";

cin >> r2 >> c2;

Matrix mat2(r2, c2);

mat2.inputMatrix();

**if** (r1 == r2 && c1 == c2) {

Matrix sum = mat1.add(mat2);

cout << "\nMatrix Addition Result:\n";

sum.displayMatrix();

} **else** {

cout << "\nAddition not possible (dimension mismatch)\n";

}

**if** (r1 == r2 && c1 == c2) {

Matrix diff = mat1.subtract(mat2);

cout << "\nMatrix Subtraction Result:\n";

diff.displayMatrix();

} **else** {

cout << "\nSubtraction not possible (dimension mismatch)\n";

}

**if** (c1 == r2) {

Matrix prod = mat1.multiply(mat2);

cout << "\nMatrix Multiplication Result:\n";

prod.displayMatrix();

} **else** {

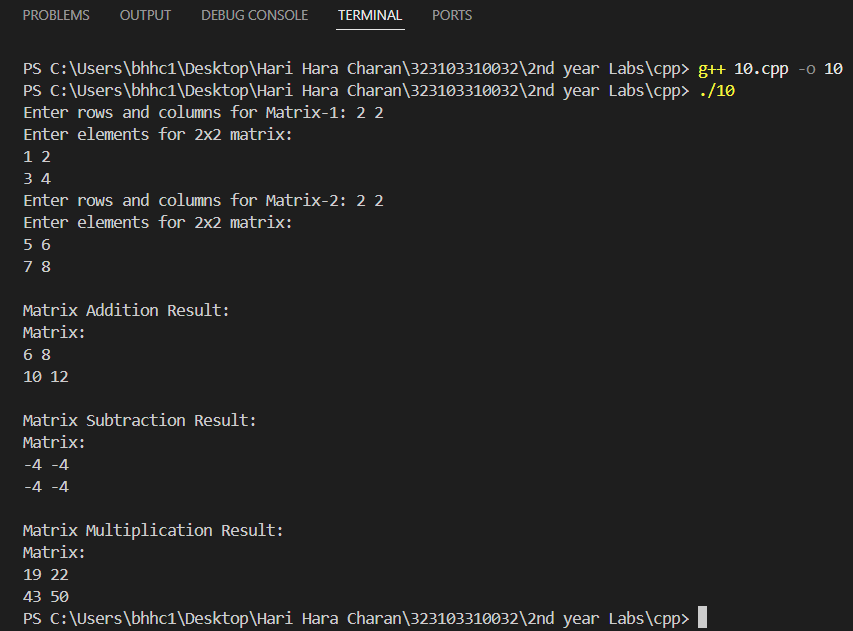
cout << "\nMultiplication not possible (column-row mismatch)\n";

}

**return** 0;

}

***Output:***

****

**WEEK-11:**

1. **Aim:** Accept two stacks as input from the user and perform operations on it using stack class available in Standard Template Library (STL).

***PROGRAM:***

#include <iostream>

#include <stack>

using namespace std;

**int** main() {

stack<**int**> stack1, stack2;

**int** n1, n2, element;

cout << "Enter the number of elements in the first stack: ";

cin >> n1;

cout << "Enter the elements of the first stack: ";

**for** (**int** i = 0; i < n1; i++) {

cin >> element;

stack1.push(element);

}

cout << "Enter the number of elements in the second stack: ";

cin >> n2;

cout << "Enter the elements of the second stack: ";

**for** (**int** i = 0; i < n2; i++) {

cin >> element;

stack2.push(element);

}

cout << "First Stack: ";

**while** (!stack1.empty()) {

cout << stack1.top() << " ";

stack1.pop();

}

cout << endl;

cout << "Second Stack: ";

**while** (!stack2.empty()) {

cout << stack2.top() << " ";

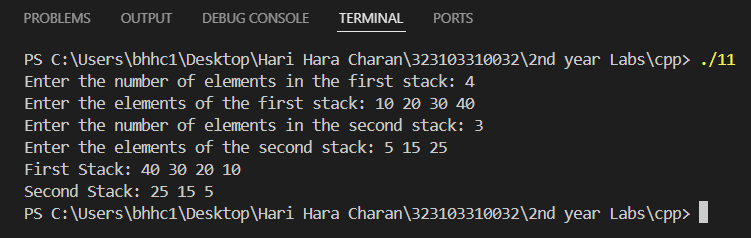
stack2.pop();

}

cout << endl;

**return** 0;

}

***Output:***

**WEEK-12:**

1. **Aim:** Write a program implementing a queue class with required operations using STL.

***PROGRAM:***

#include <iostream>

#include <queue>

using namespace std;

class Queue

{

queue<**int**> q;

public:

**void** push(**int** x){

q.push(x);

}

**void** pop(){

**if** (q.empty()){

cout << "Queue is empty" << endl;

**return**;

}

q.pop();

}

**int** front(){

**if** (q.empty()){

cout << "Queue is empty" << endl;

**return** -1;

}

**return** q.front();

}

**int** back(){

**if** (q.empty()){

cout << "Queue is empty" << endl;

**return** -1;

}

**return** q.back();

}

**int** size(){

**return** q.size();

}

**bool** empty(){

**return** q.empty();

}

};

**int** main(){

Queue q;

**int** x, n;

cout << "1. Push\n2. Pop\n3. Front\n4. Back\n5. Size\n6. Exit" << endl;

**while**(1){

cout << "Enter your choice: ";

cin >> x;

**switch** (x)

{

**case** 1:

cout << "Enter the element to push: ";

cin >> n;

q.push(n);

**break**;

**case** 2:

q.pop();

**break**;

**case** 3:

cout << "Front element: " << q.front() << endl;

**break**;

**case** 4:

cout << "Back element: " << q.back() << endl;

**break**;

**case** 5:

cout << "Size of queue: " << q.size() << endl;

**break**;

**case** 6:

exit(0);

default:

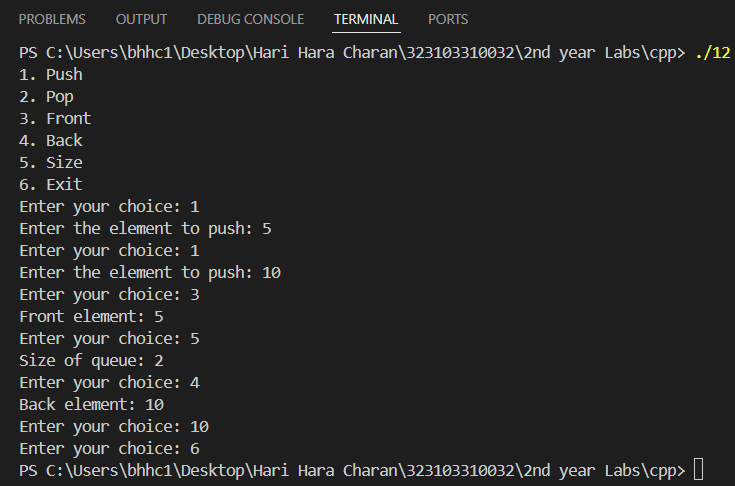
**break**;

}

}

**return** 0;

}

****

***Output:***