**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 << " ";

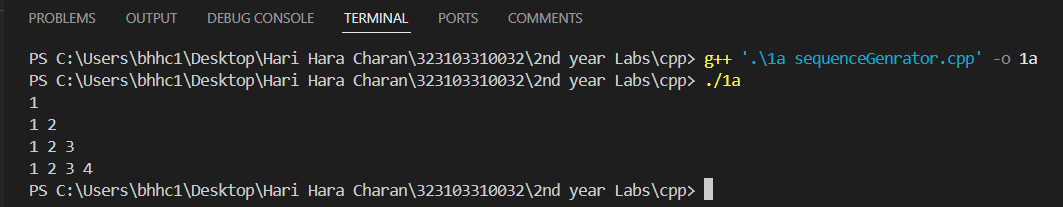
}

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);

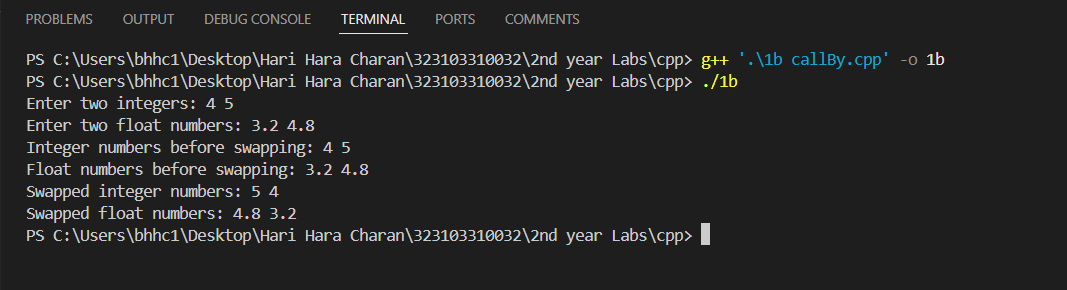
cout<<"Swapped integer numbers: "<<a<<" "<<b<<endl;

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);

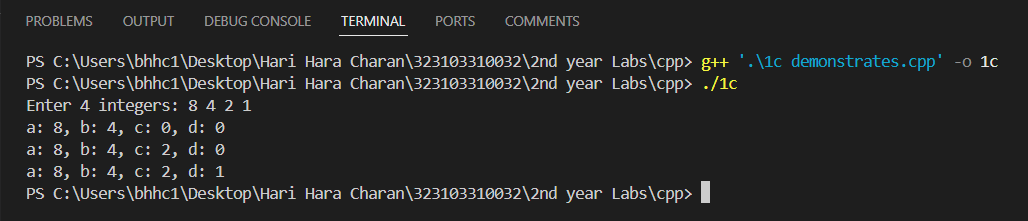
defaultPara(a,b,c);

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: ";

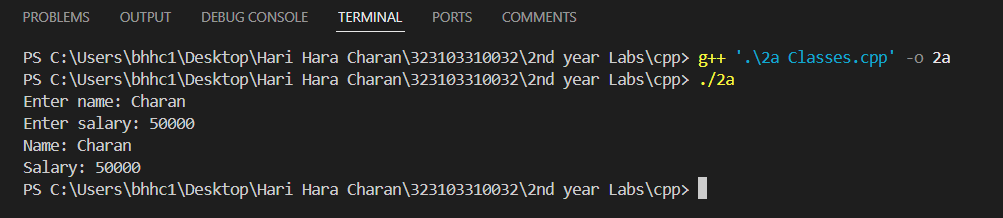
cin >> e1.salary;

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;

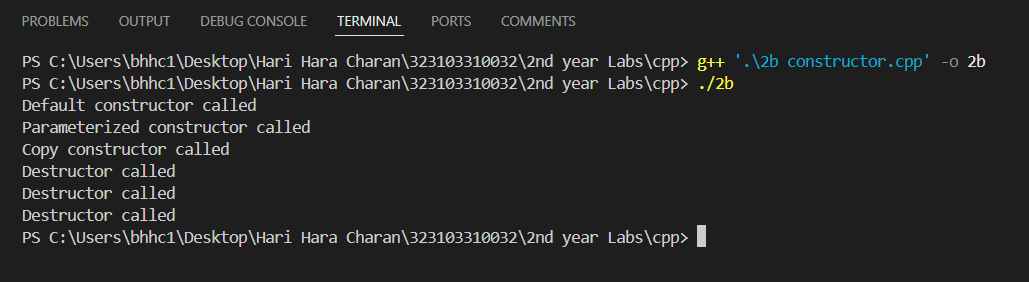
Demo d2(200);

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;

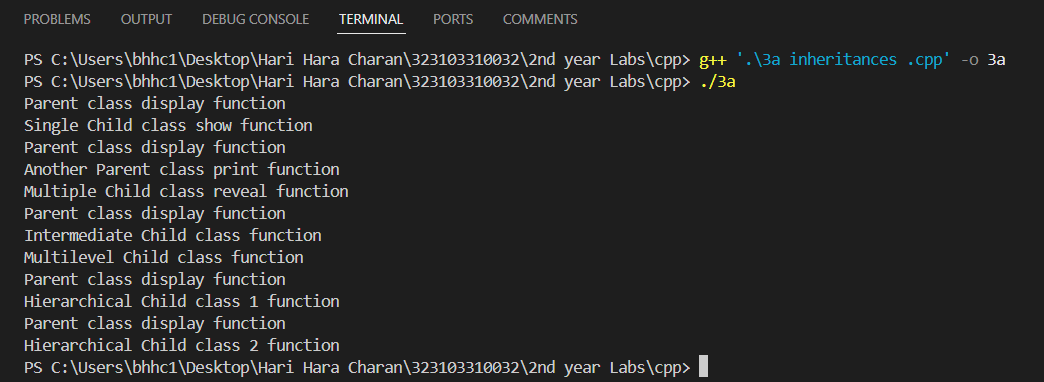
hc2.display();

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;

}

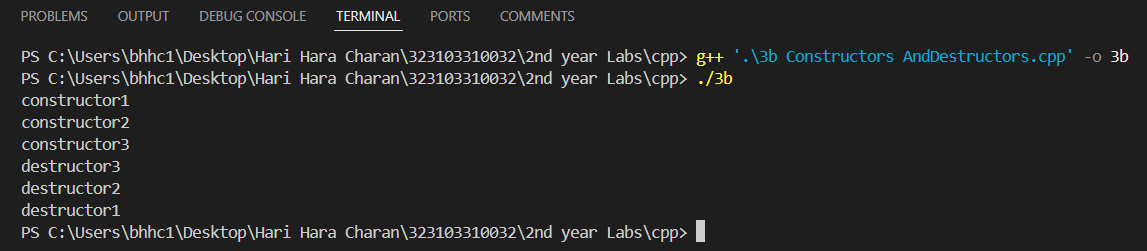
};

**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;

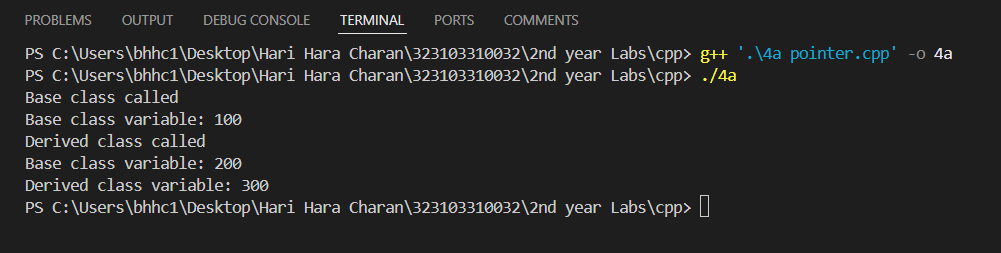
bptr->show();

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:***

