**Problem No.**: 01

**Problem Name:** Access Modifier in C++, Public & Private.

**Code:**

#include <iostream>

using namespace std;

class Human{

private:

int age;

int height;

public:

Human(int ag, int h){

age=ag;

height=h;

}

void display(){

cout<<"Age: "<<age<<endl;

cout<<"Height: "<<height<<" feet"<<endl;

}

};

int main() {

Human h1(30, 6);

//h1.age;//not accessible

//h1.height;//not accessible

h1.display();

return 0;

}

**Output:**

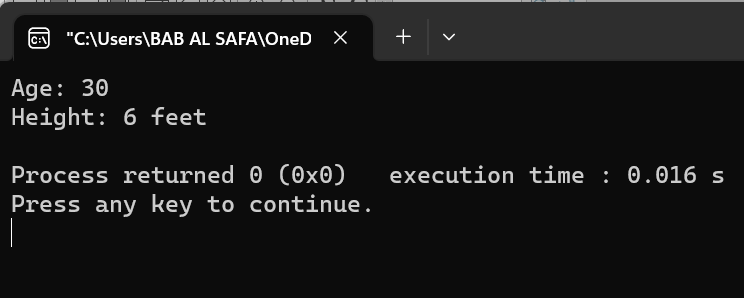


Fig 1.1: Output on console.

**Explanation:**

In this code, defined a class named Human with two private integer data members age and height. The class also has a public constructor and a public member function named display() to the age and height given to the constructor.

**Problem No.**: 02

**Problem Name**: Protected Access Modifier in C++.

**Code:**

#include <iostream>

using namespace std;

class Human{

protected:

int age;

int height;

Human(int ag, int h){

age=ag;

height=h;

}

void display(){

cout<<"Age: "<<age<<endl;

cout<<"Height: "<<height<<" feet"<<endl;

}

};

class Child:public Human{

public:

Child(int ag, int h):Human(ag, h){

}

void display(){

Human::display();

}

};

int main() {

Child c1(30, 6);

//c1.age;//not accessible

//c1.height;//not accessible

c1.display();

return 0;

}

**Output:**

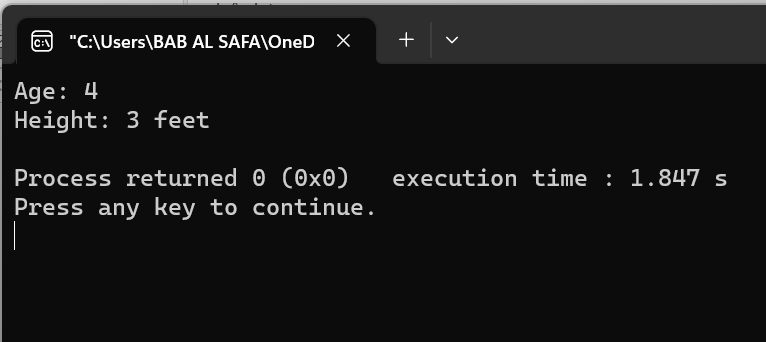


Fig 2.1: Output on console.

**Explanation:**

In this code, there defined a class Human with protected member variables and functions that are inherited by Child class in public mode which is then accessed by the main function.

**Problem No.**: 03

**Problem Name**: Function Overloading in C++.

**Code:**

#include <iostream>

using namespace std;

int add(int a, int b){

return a+b;

}

int add(char a, int b){

return a+b;

}

int add(int a, int b, int c){

return a+b+c;

}

int main() {

cout<<add(1,2)<<endl;

cout<<add('D',2)<<endl;

cout<<add(1,2,3)<<endl;

return 0;

}

**Output:**

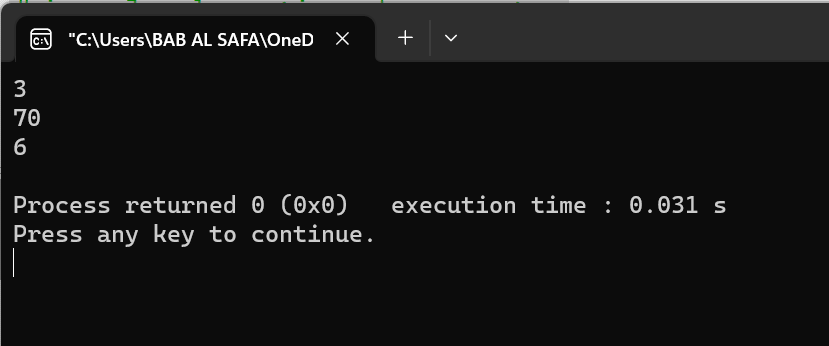


Fig 3.1: Output on console.

**Explanation:**

In this code, defined a function named add() and then overloaded it with different number of parameters and different types of parameters.

**Problem No.**: 04

**Problem Name**: Constructor Overloading In C++.

**Code:**

#include <iostream>

using namespace std;

class Complex

{

int a, b;

public:

Complex(){

a = 0;

b =0;

}

Complex(int x, int y)

{

a = x;

b = y;

}

Complex(int x){

a = x;

b = 0;

}

void printNumber()

{

cout << a << " + " << b << "i" << endl;

}

};

int main(){

Complex c1(4,6);

c1.printNumber();

Complex c2(5);

c2.printNumber();

return 0;

}

**Output:**

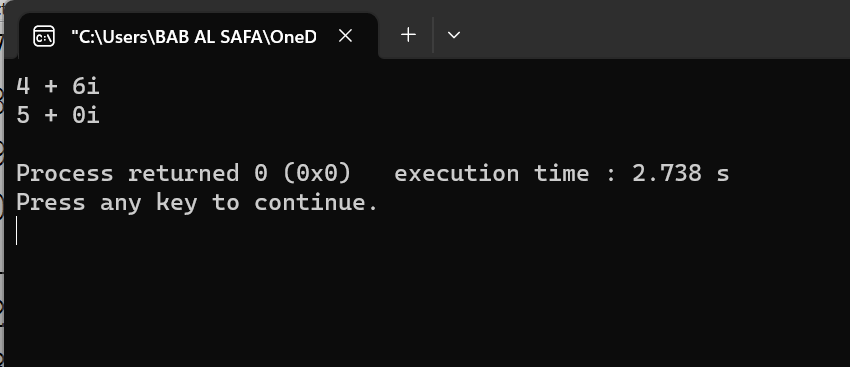


Fig 4.1: Output on console.

**Explanation:**

In this code, it is defined a class called Complex with a constructor, which is later overloaded by changing the parameters.

**Problem No.**: 05

**Problem Name**: Virtual Functions in C++ .

**Code:**

#include<iostream>

using namespace std;

class BaseClass{

    public:

        int var\_base=1;

         virtual void display(){

            cout<<"1 Dispalying Base class variable var\_base "<<var\_base<<endl;

        }

};

class DerivedClass : public BaseClass{

    public:

            int var\_derived=2;

            void display(){

                cout<<"2 Dispalying Base class variable var\_base "<<var\_base<<endl;

                cout<<"2 Dispalying Derived class variable var\_derived "<<var\_derived<<endl;

            }

};

int main(){

    BaseClass \*bptr;

    BaseClass b;

    DerivedClass d;

    bptr = &d;

    bptr->display();

    return 0;

}

**Output:**

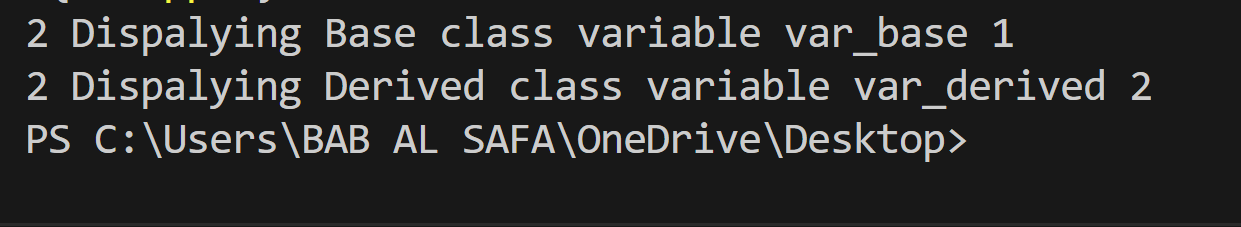


Fig 5.1: Output on console

**Explanation:**

In this code, by creating a pointer of the base class pointing to the derived class, and by creating a virtual function in base class which was overridden n derived class, when that display() function is called via the base class, the display function of the derived class is being executed.

**Problem No.**: 06

**Problem Name**: Function overriding in C++.

**Code:**

#include<iostream>

using namespace std;

class BaseClass{

    public:

        int var\_base=1;

         void display(){

            cout<<"1 Dispalying Base class variable var\_base "<<var\_base<<endl;

        }

};

class DerivedClass : public BaseClass{

    public:

            int var\_derived=2;

            void display(){

                cout<<"2 Dispalying Base class variable var\_base "<<var\_base<<endl;

                cout<<"2 Dispalying Derived class variable var\_derived "<<var\_derived<<endl;

            }

};

int main(){

    BaseClass b;

    DerivedClass d;

    b.display();

    d.display();

    return 0;

}

**Output:**

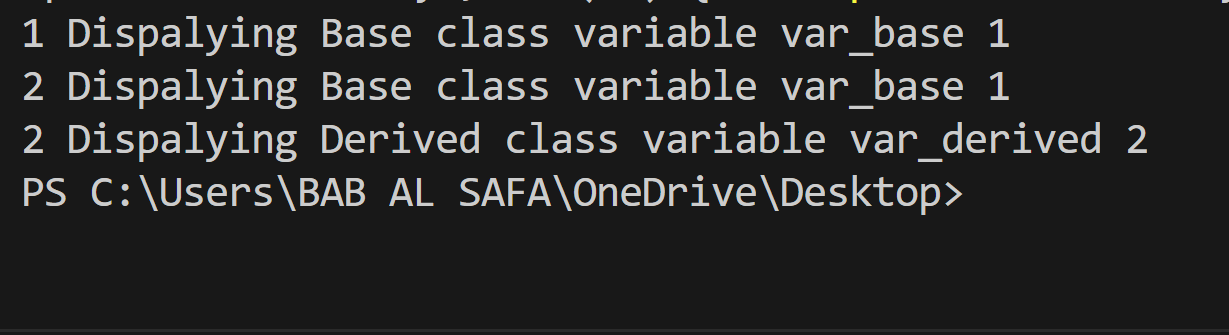


Fig 6.1: Output on console

**Explanation:**

In this code, derived function display() has been overridden. And based on which object is calling it, the function changes.

**Problem No.**: 07

**Problem Name**: Friend Functions in C++ .

**Code:**

#include<iostream>

using namespace std;

void f();

class BaseClass{

private:

int a;

public:

void disp(){

cout<<"Value of a = "<<a<<endl;

}

friend void f(BaseClass& b);

friend int main();

};

void f(BaseClass& b){

cout<<"Enter value of a = ";

cin>>b.a;

}

int main(){

BaseClass b;

f(b);

b.disp();

cout<<"Inside main() function, Enter value of a = ";

cin>>b.a;

b.disp();

return 0;

}

**Output:**

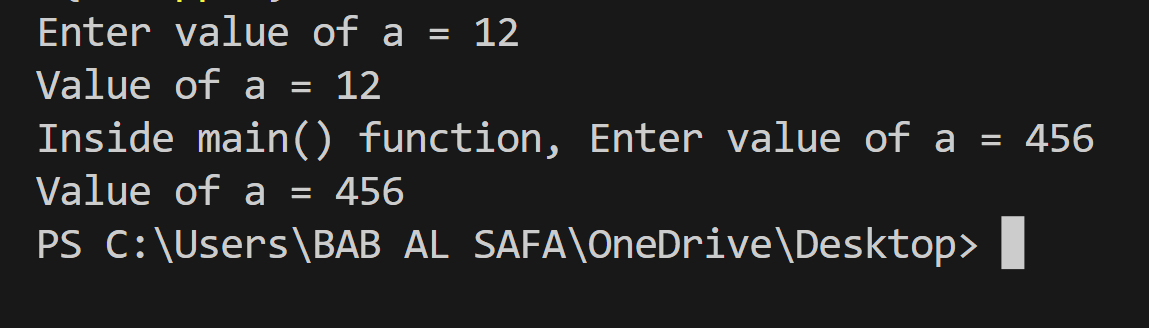


Fig 7.1: Output on console

**Explanation:**

In this code, the main function and a non-member function of class BaseClass got access to it’s private member a by declaring them as friend.