Week 9 lec 1

Function overriding

Method Overriding in C++

- If we inherit a class into the derived class and provide a definition for one of the base class's function again inside the derived class, then that function is said to be **overridden**, and this mechanism is called **Function Overriding**

Requirements for Overriding a Function

1. Inheritance should be there. Function overriding cannot be done within a class. For this we require a derived class and a base class.

2. Function that is redefined must have exactly the same declaration in both base and derived class, that means same name, same return type and same parameter list.

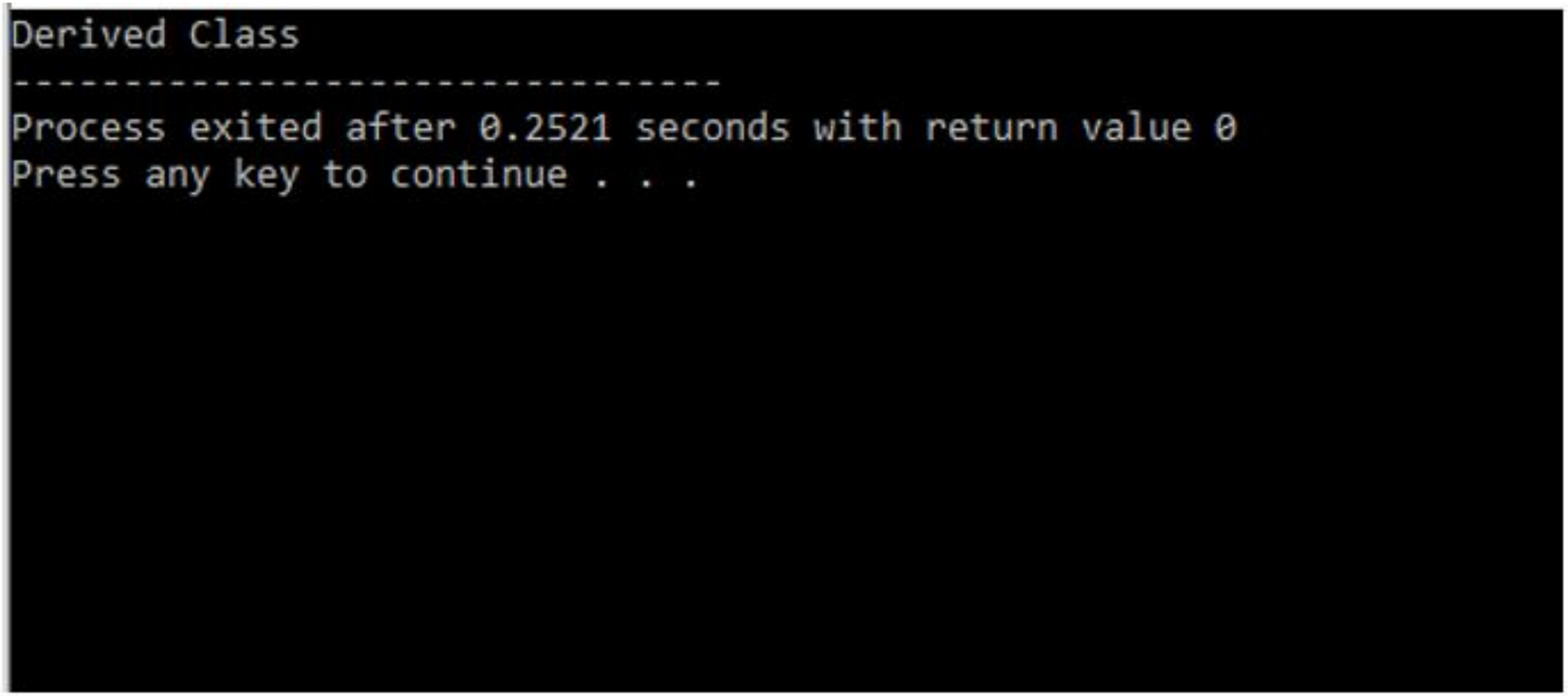
Early binding

```
class A {
   public:
   void display() {
      cout<<"Base class";  } };
class B:public A {
   public:
   int display() {
      cout<<"Derived Class";
           return 1; } };
int main() {
   B obj;
   obj.display();
   return 0;  }
```

Early binding

- In the above example, we are calling the overridden function using derived class object. Compiler gives preference to object type.
- Base class object will call base version of the function and derived class's object will call the derived version of the function. This is called early binding
- In early binding return type is ignored

Early binding



```
Derived Class
--------------------------------
Process exited after 0.2521 seconds with return value 0
Press any key to continue . . .
```

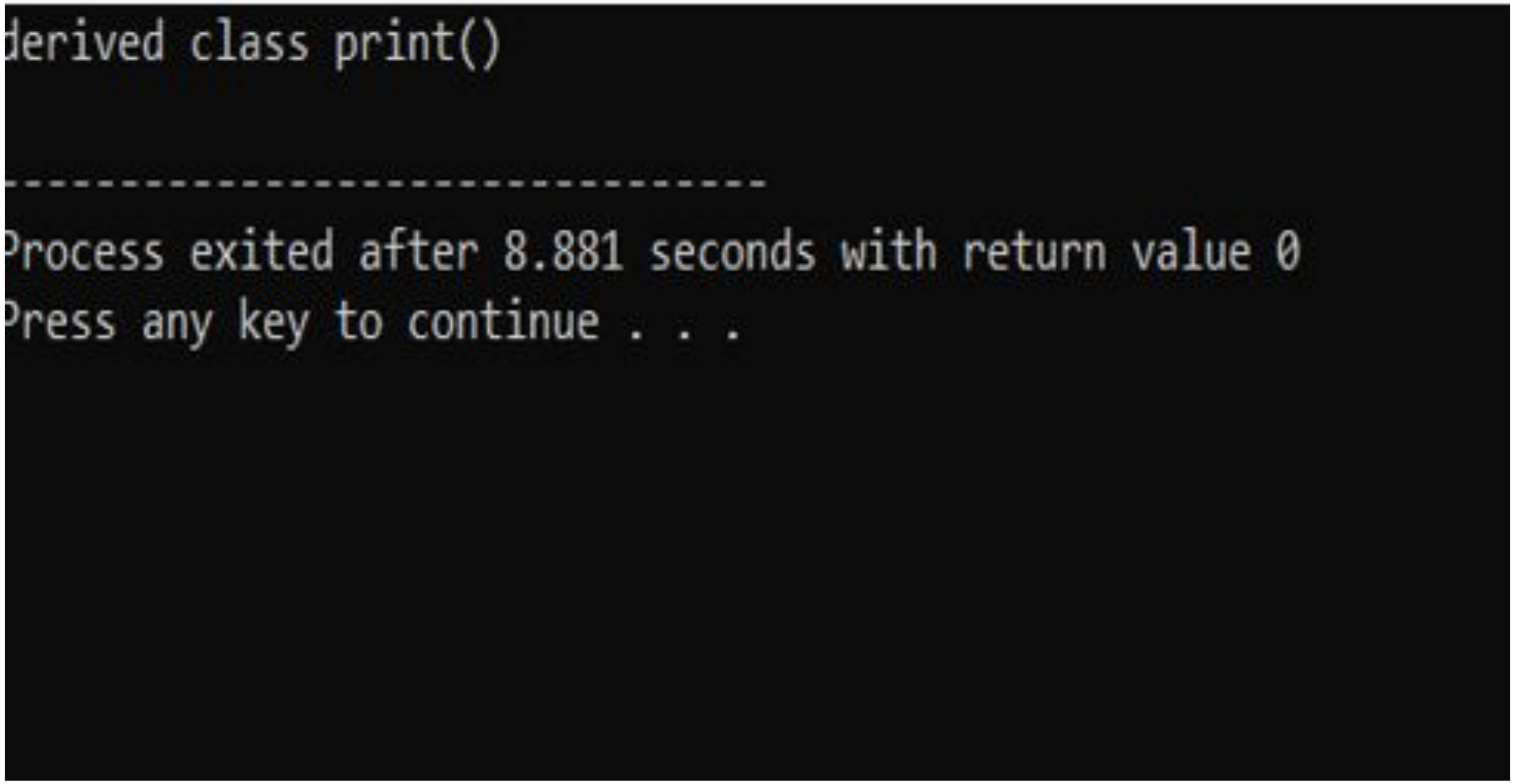
Function Call Binding with class Objects

- Connecting the function call to the function body is called **Binding**. When it is done before the program is run, its called **Early** Binding or **Static** Binding or **Compile-time** Binding.

Return type is not same

```
class Base {
   public:
    void print() {
    cout << "base class print()" << endl; }};
class Derived : public Base {
   public:
    int print() {
        cout << "derived class print()" << endl;
                   return 1;}
                   };
int main() {
    Derived derived1;
    derived1.print();}
```

In early binding return type ignored



```
derived class print()

--------------------------------
Process exited after 8.881 seconds with return value 0
Press any key to continue . . .
```

Can we overload methods in sub class?

- Can we overload methods in sub class?

- Yes overloading can be done in different scopes but….

Can we overload methods in sub class?

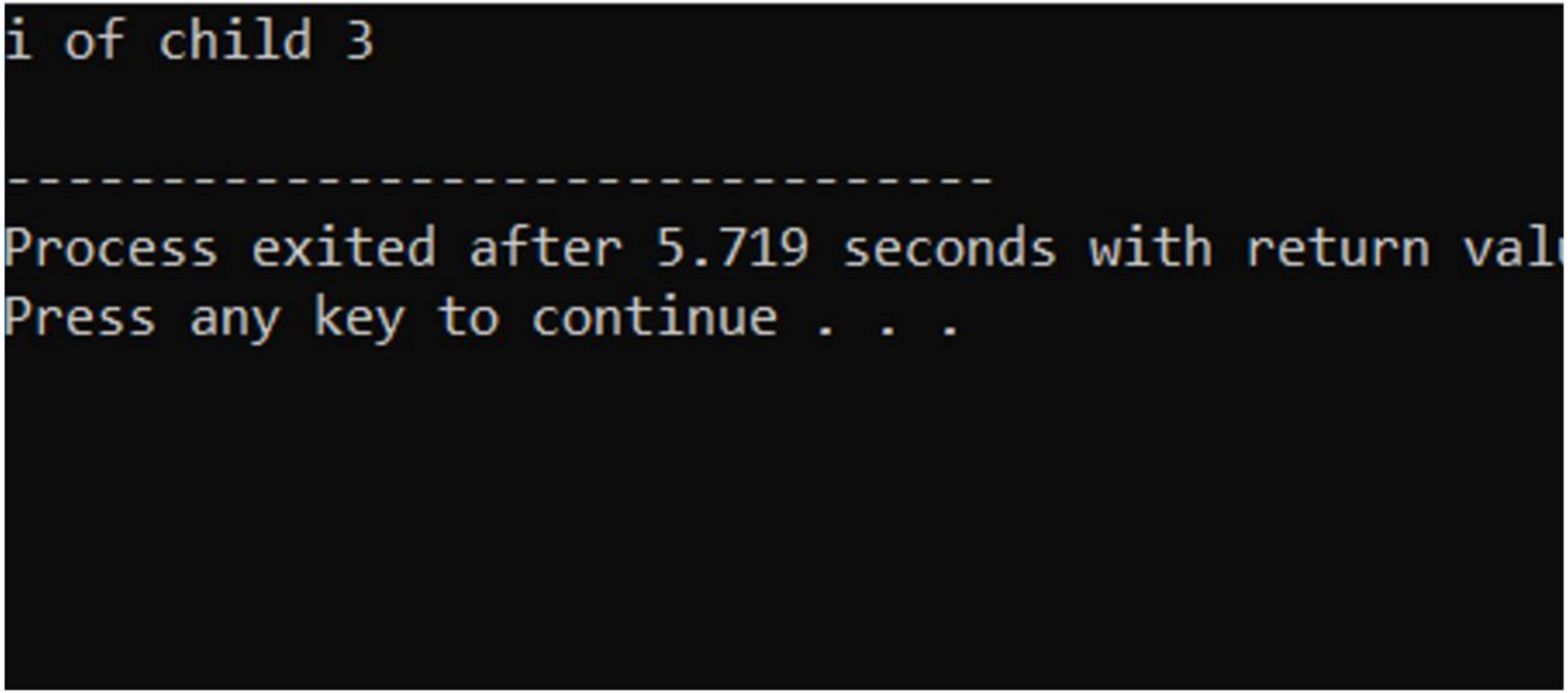
C++ handles overloading across subclasses in a much different way. A member function declaration in a derived class does not overload member functions with the same name declared in the base class.
Rather, a member function in a derived class hides the declaration of the member functions with the same name in base class, even though the function parameter lists are different.
Hence, to have member functions from a base class overload member functions from the derived class, the designer of the derived class can introduce the base class member functions into the scope of the derived class with using declarations.

Can we overload methods in sub class?

```
class Check{
    int i;
  public:
    Check(): i(0) {  }
void Display()
    { cout << "i of base  "<< i <<endl; }};
class Go : public Check{
    int i;
  public:
    Go(): i(1) {  }
 Display(int i)
    { cout << "i of child "<< i <<endl;  }};
```

Can we overload methods in sub class?

```
int main(){
   Go g;
   g.Display(3); // derived class will hide the Display() func of base class
  return 0;}
```

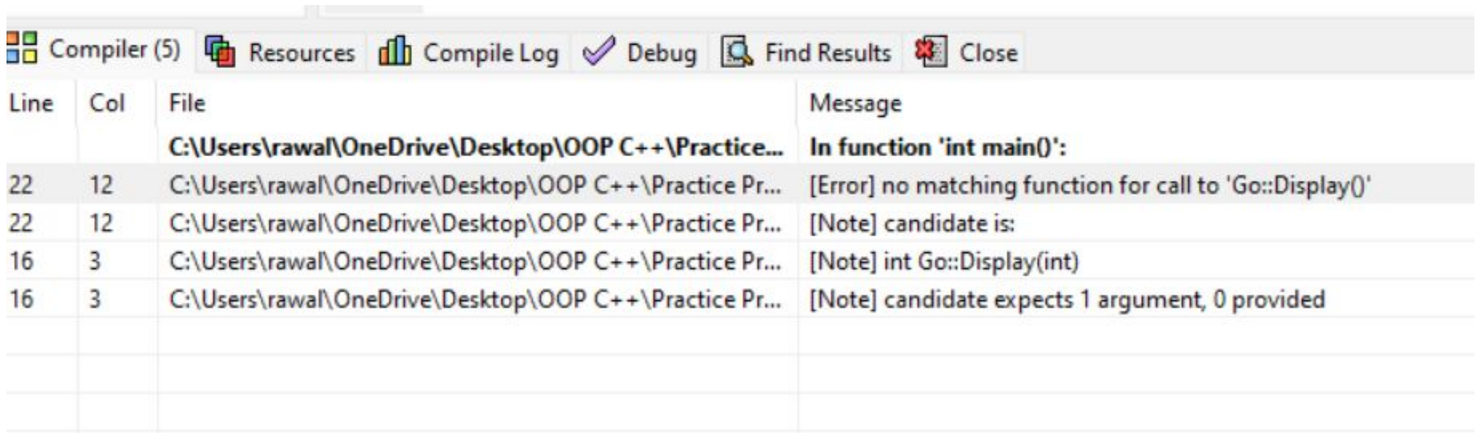


```
i of child 3


--------------------------------
Process exited after 5.719 seconds with return valu
Press any key to continue . . .
```

```
int main(){
      Go g;
      g.Display();
    return 0;}
```

Can we overload methods in sub class?


Compiler (5) | Resources | Compile Log | Debug | Find Results | Close

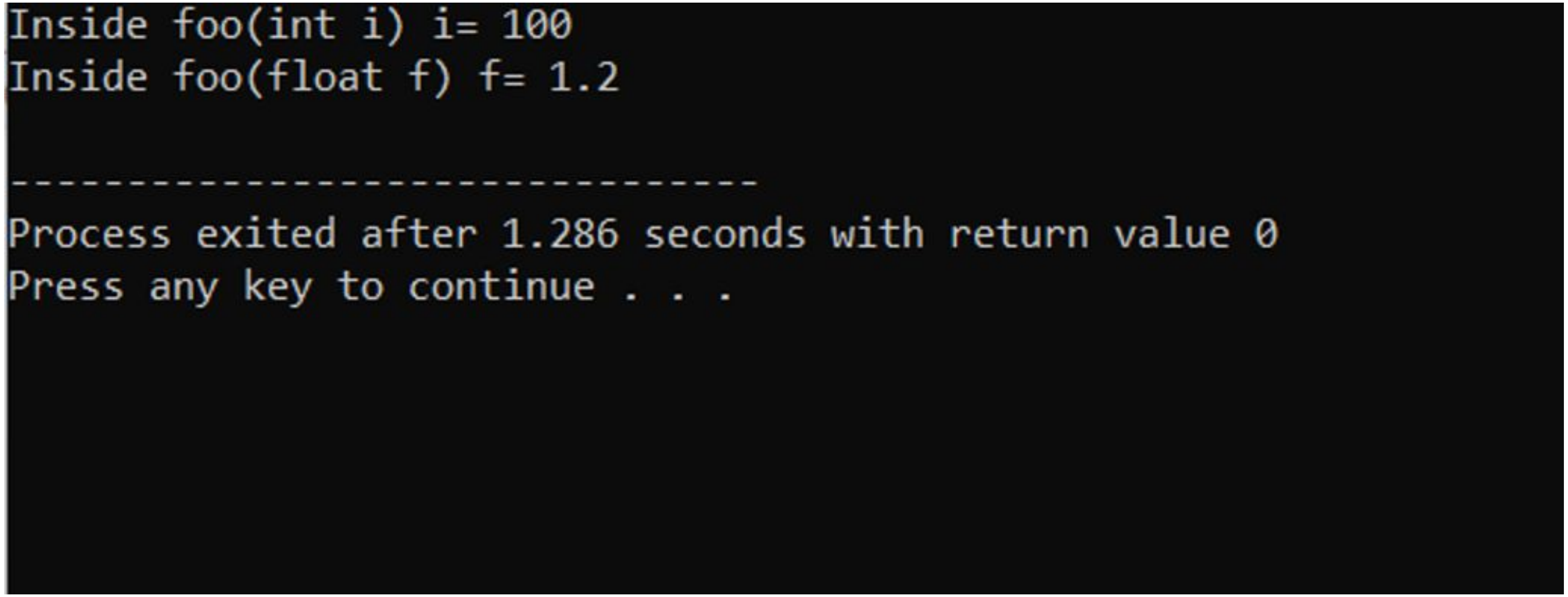
Line	Col	File	Message
		C:\Users\rawal\OneDrive\Desktop\OOP C++\Practice...	**In function 'int main()':**
22	12	C:\Users\rawal\OneDrive\Desktop\OOP C++\Practice Pr...	[Error] no matching function for call to 'Go::Display()'
22	12	C:\Users\rawal\OneDrive\Desktop\OOP C++\Practice Pr...	[Note] candidate is:
16	3	C:\Users\rawal\OneDrive\Desktop\OOP C++\Practice Pr...	[Note] int Go::Display(int)
16	3	C:\Users\rawal\OneDrive\Desktop\OOP C++\Practice Pr...	[Note] candidate expects 1 argument, 0 provided

overloading methods in sub class

```
#include <iostream>
using namespace std;
class Base{
  public:
        void foo(int i) {
      cout << "Inside foo(int i) i= " << i << "\n"; }};
class Derived : public Base{
  public:
          // using Base::foo; must be added
          // in absence of this simply foo(float f) is called
          // in its presence foo(int i) is called, no ambguity
 Base::foo;
void foo(float f) {
           cout << "Inside foo(float f) f= " << f << "\n"; }};
```

overloading methods in sub class

```
int main(){
        float  f = 1.2f;
        int    i = 100;
Derived obj;
obj.foo(i);  //no compiler error base foo
obj.foo(f);  //no compiler error derived foo
}
```



```
Inside foo(int i) i= 100
Inside foo(float f) f= 1.2

--------------------------------
Process exited after 1.286 seconds with return value 0
Press any key to continue . . .
```

overloading methods in sub class

overloading methods in sub class

The code using Base::foo; makes all the functions named foo (irrespective of their arguments) from base class directly accessible in the Derived class. Hence the functions named foo become overloaded in the scope of Derived class. In case of multiple foo functions in the base class, it is not possible to make only a few "foo" functions from the base class accessible in the derived class with "Base::foo; " declaration.
Since both versions of foo are available in the scope of Derived class, the compiler can choose specific function from them without any ambiguity here, and the call foo(i) is correctly resolved to foo(int t).

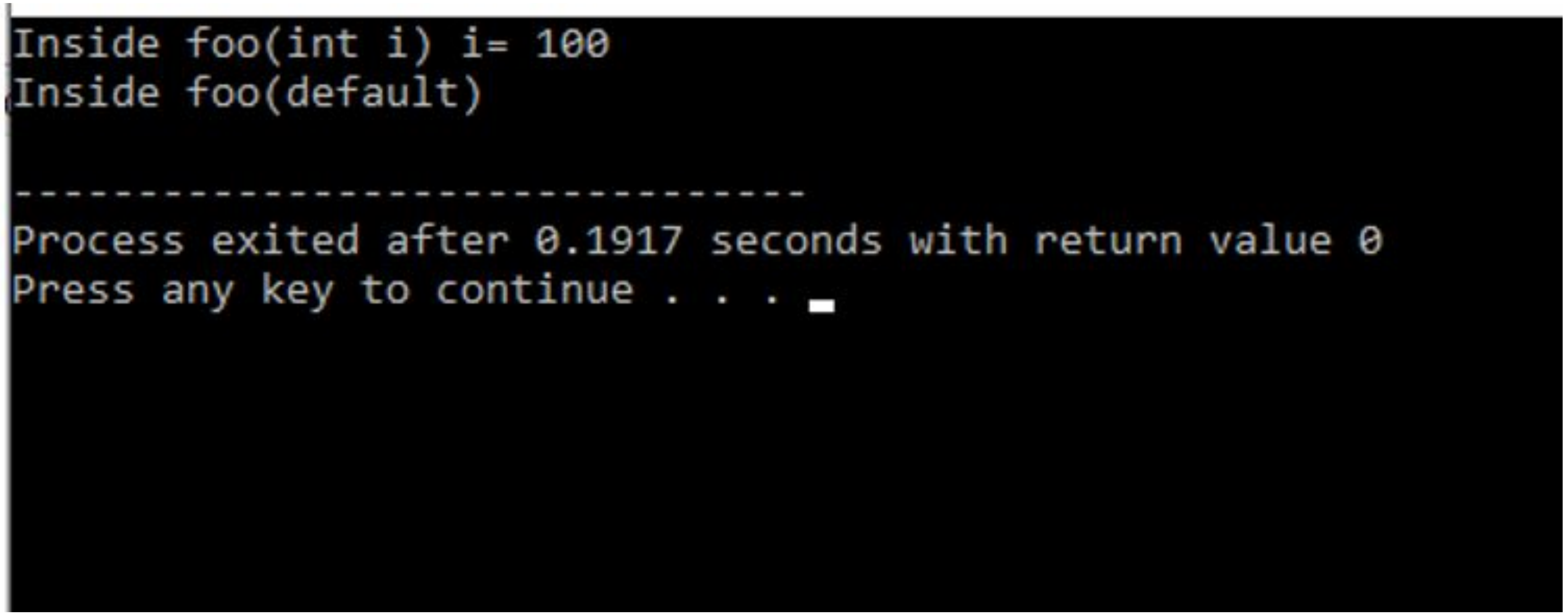
multiple foo functions in the base class

```
class Base{
   public:
           void foo(int i) {
        cout << "Inside foo(int i) i= " << i << "\n"; }
             void foo() {
        cout << "Inside foo(default)  "  << "\n"; }};
class Derived : public Base{
   public:
 Base::foo;
 void foo(float f) {
               cout << "Inside foo(float f) f= " << f << "\n"; }};
```

multiple foo functions in the base class

```
int main(){
        float  f = 1.2f;
        int    i = 100;
Derived obj;
obj.foo(i);
obj.foo();  //no compiler error
}
```

multiple foo functions in the base class



```
Inside foo(int i) i= 100
Inside foo(default)

--------------------------------
Process exited after 0.1917 seconds with return value 0
Press any key to continue . . .
```

Function Overloading VS Function Overriding

1. **Inheritance:** Overriding of functions occurs when one class is inherited from another class. Overloading can occur without inheritance.
2. **Function Signature:** Overloaded functions must differ in function signature ie either number of parameters or type of parameters should differ. In overriding, function signatures must be same.
3. **Scope of functions:** Overridden functions are in different scopes; whereas overloaded functions are in same scope.
4. **Behavior of functions:** Overriding is needed when derived class function has to do some added or different job than the base class function. Overloading is used to have same name functions which behave differently depending upon parameters passed to them.

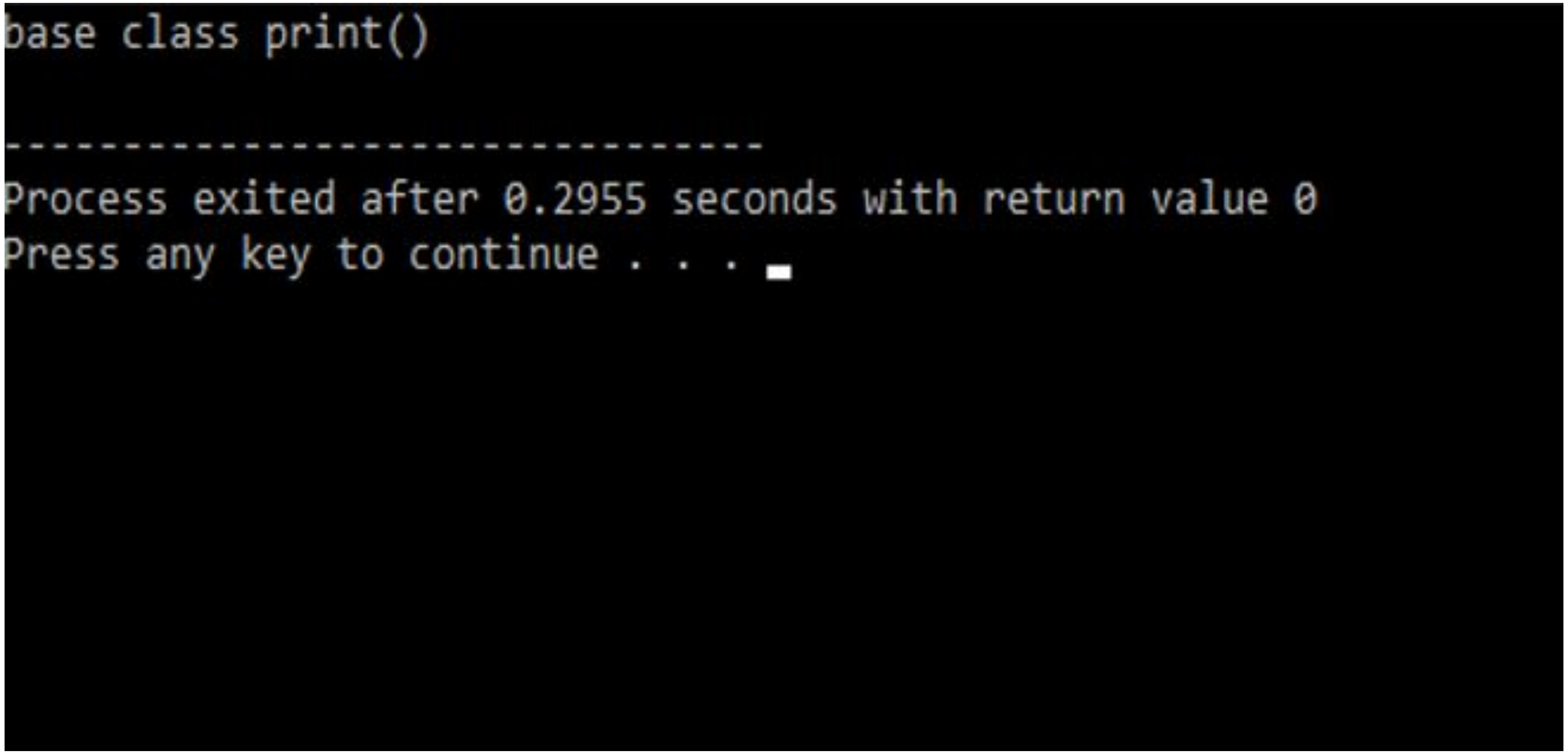
Week 10 Lecture 1

- **Runtime Polymorphism**

Compile-time(early binding)

```
class Base {
  public:
   void print() {
   cout << "base class print()" << endl; }};
class Derived : public Base {
  public:
   void print() {
       cout << "derived class print()" << endl;}};
int main() {
   Derived derived1;
   Base* base1 ;
  base1= &derived1;
   base1->print();}
```

Compile-time(early binding)



```
base class print()

--------------------------------
Process exited after 0.2955 seconds with return value 0
Press any key to continue . . .
```

Problem

In the above example, although, the object is of Derived class, still Base class's method is called. This happens due to Early Binding.
Compiler on seeing **Base class's pointer**, set call to Base class’s `print()` function, without knowing the actual object type.

Solution
Virtual Function is a function in base class, which is override in the derived class, and which tells the compiler to perform **Late Binding** on this function.
`Virtual` Keyword is used to make a member function of the base class Virtual.

virtual function

- A virtual function is a member function that you expect to be redefined in derived classes. When you refer to a derived class object using a pointer or a reference to the base class, you can call a virtual function for that object and execute the derived class's version of the function.
- Virtual functions ensure that the correct function is called for an object, regardless of the expression used to make the function call.

virtual function

- They are mainly used to achieve Runtime polymorphism
- Functions are declared with a **virtual** keyword in base class.
- The resolving of function call is done at Run-time.

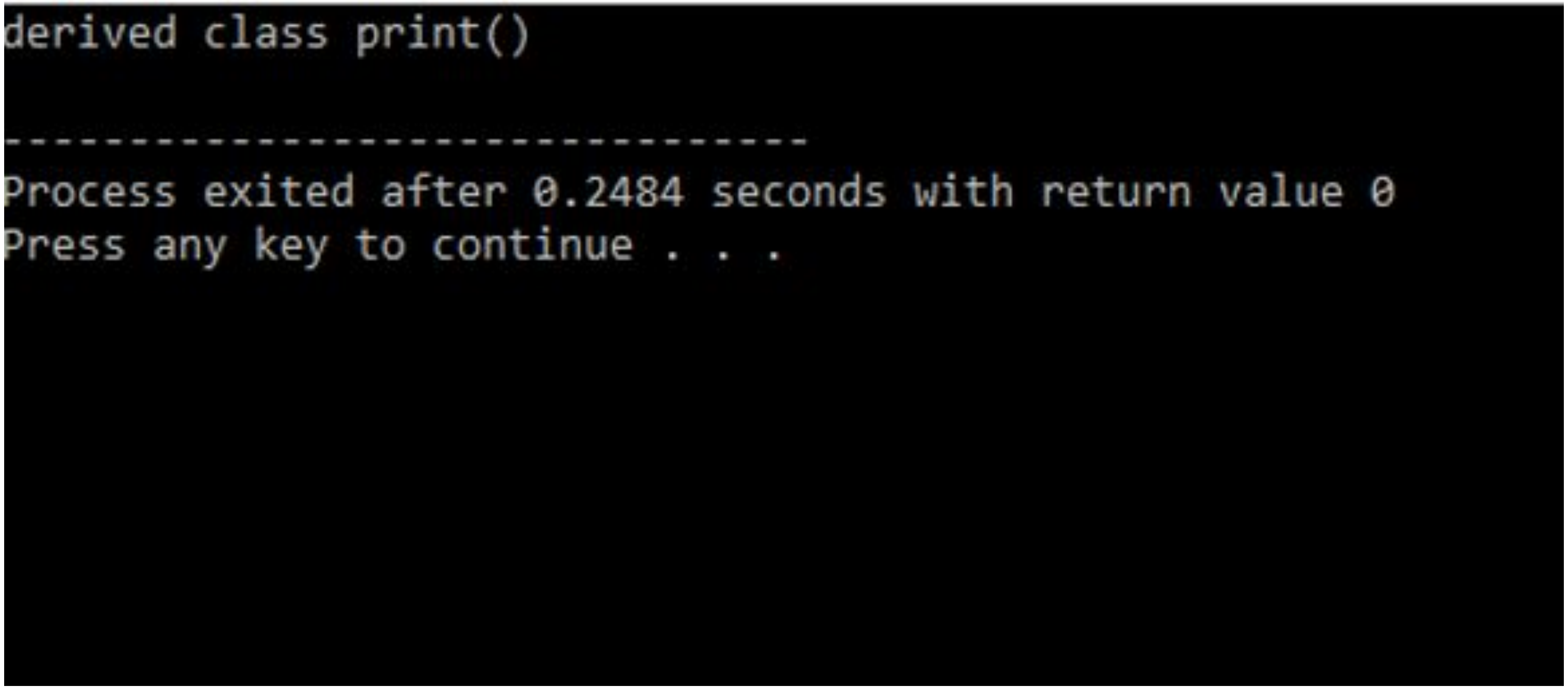
Late Binding in C++

- In Late Binding function call is resolved at runtime. Hence, now compiler determines the type of object at runtime, and then binds the function call. Late Binding is also called **Dynamic** Binding or **Runtime** Binding.

virtual function

```
class Base {
   public:
    virtual void print() {
    cout << "base class print()" << endl; }};
class Derived : public Base {
   public:
    void print() {
        cout << "derived class print()" << endl;}};
int main() {
    Derived derived1;
    Base* base1 ;
base1= &derived1;
    base1->print();}
```

virtual function



```
derived class print()

--------------------------------
Process exited after 0.2484 seconds with return value 0
Press any key to continue . . .
```

Runtime Polymorphism.

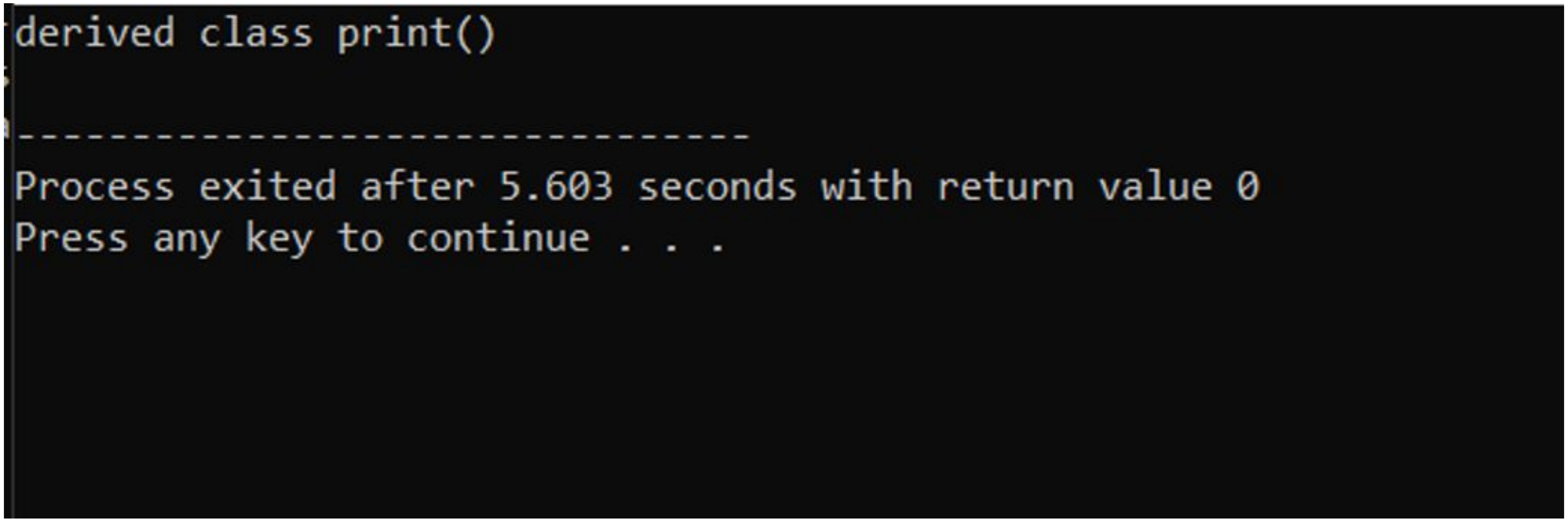
- Runtime polymorphism is achieved only through a pointer (or reference) of base class type
- Late binding(Runtime) is done in accordance with the content of pointer (i.e. location pointed to by pointer) and Early binding(Compile time) is done according to the type of pointer

When calling a function using pointers or references, the following rules apply:

1. Functions in derived classes override virtual functions in base classes only if their type is the same. A function in a derived class cannot differ from a virtual function in a base class in its return type only; the argument list must differ as well.
2. A call to a virtual function is resolved according to the underlying type of object for which it is called.
3. The virtual keyword can be used when declaring overriding functions in a derived class, but it is unnecessary; overrides of virtual functions are always virtual.

Rule 3

```
class Base {
   public:
    virtual void print() {
    cout << "base class print()" << endl; }};
class Derived : public Base {
   public:
   virtual void print() { // virtual keyword with derived version can be used
        cout << "derived class print()" << endl;}};
int main() {
    Derived derived1;
    Base* base1 = &derived1;
    base1->print();}
```



```
derived class print()

--------------------------------
Process exited after 5.603 seconds with return value 0
Press any key to continue . . .
```

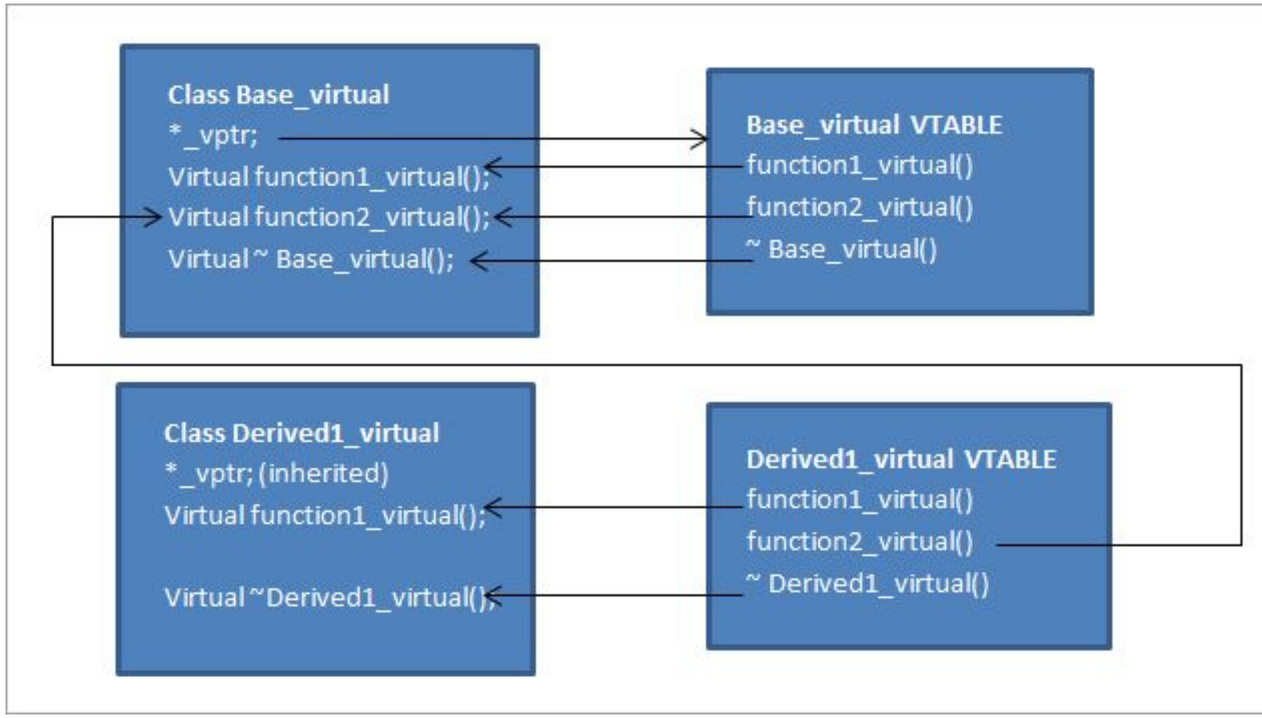
Working of virtual functions(concept of VTABLE and VPTR)

- If a class contains a virtual function, then compiler itself does two things:

1. ***vtable:*** A table of function pointers, maintained per class.

It is a table that contains the memory addresses of all virtual functions(only VFs) of a class in the order in which they are declared in a class. This table is used to resolve function calls in dynamic/late binding manner. Every class that has virtual function will get its own Vtable.

The vtable for the base class is straightforward. In the case of the derived class, only function1_virtual is overridden.
Hence we see that in the derived class vtable, function pointer for function1_virtual points to the overridden function in the derived class. On the other hand function pointer for function2_virtual points to a function in the base class.
Thus in the above program when the base pointer is assigned a derived class object, the base pointer points to _vptr of the derived class.
So when the call b->function1_virtual() is made, the function1_virtual from the derived class is called and when the function call b->function2_virtual() is made, as this function pointer points to the base class function, the base class function is called.



Working of virtual functions(concept of VTABLE and VPTR)

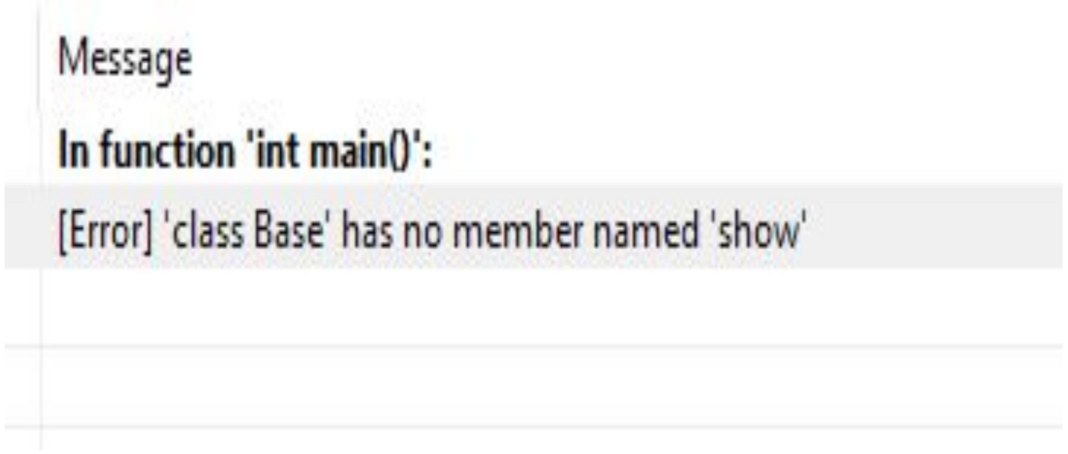
2. vptr: A pointer to vtable, maintained per object instance.

- After creating Vtable address of that table gets stored inside a pointer i.e. VPTR (Virtual Pointer). When you create an object of a class which contains virtual function then a hidden pointer gets created automatically in that object by the compiler. That pointer points to a virtual table of that particular class.

Derived-Class Member-Function Calls via Base-Class Pointers

```
class Base {
   public:
    virtual void print() {
    cout << "base class print()" << endl; }};
class Derived : public Base {
   public:
    void print() {
        cout << "derived class print()" << endl;}
    void show() {
        cout << "only derived class member function" << endl;}};
int main() {
    Derived derived1;
    Base* base1 = &derived1;
    base1->print();
    base1->show();}
```

Derived-Class Member-Function Calls via Base-Class Pointers



```
Message
In function 'int main()':
[Error] 'class Base' has no member named 'show'
```

Question ??

- Can I override static function C++?

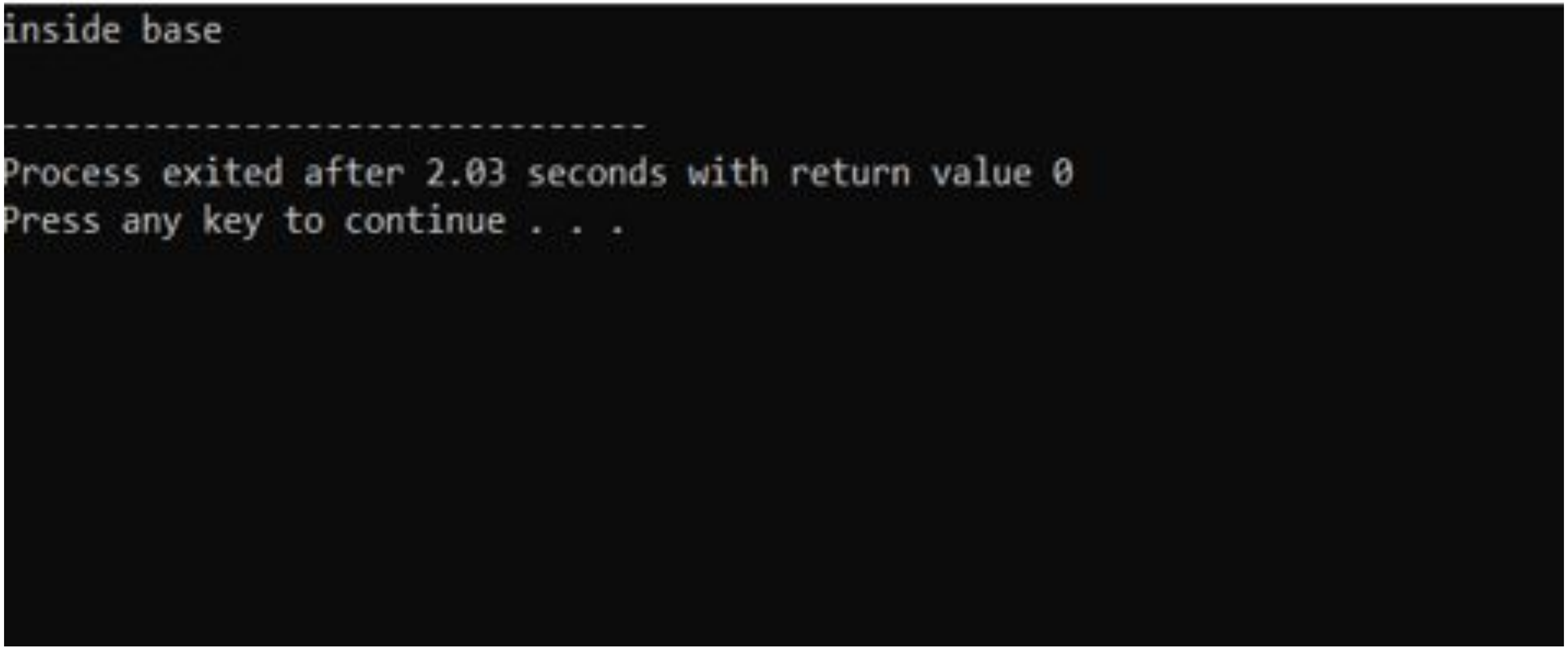
Can I override static function C++?

- **Static methods cannot be overridden** because method overriding only occurs in the context of dynamic (i.e. runtime) lookup of methods. Static methods (by their name) are looked up statically (i.e. at compile-time).

Early binding

```
#include <iostream>
using namespace std;
class base{
            public:
             static void func(){
                                    cout << "inside base"<< endl;                   }};
class child:public base{
            public:
                        void func(){
                                    cout << "inside child"<< endl;       }};
int main(){
            base* b ;
            child c;
            b = &c;
            b -> func();
}
```

Early binding

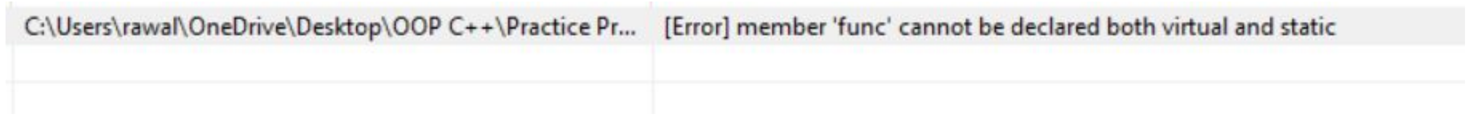


```
inside base


--------------------------------
Process exited after 2.03 seconds with return value 0
Press any key to continue . . .
```

Static in run time polymorphism

```
#include <iostream>
using namespace std;
class base{
        public:
         virtual static void func(){
                        cout << "inside base"<< endl;           }};
class child:public base{
        public:
                void func(){
                        cout << "inside child"<< endl;      }};
int main(){
        base* b ;
        child c;
        b = &c;
        b -> func();
}
```



```
C:\Users\rawal\OneDrive\Desktop\OOP C++\Practice Pr...    [Error] member 'func' cannot be declared both virtual and static
```

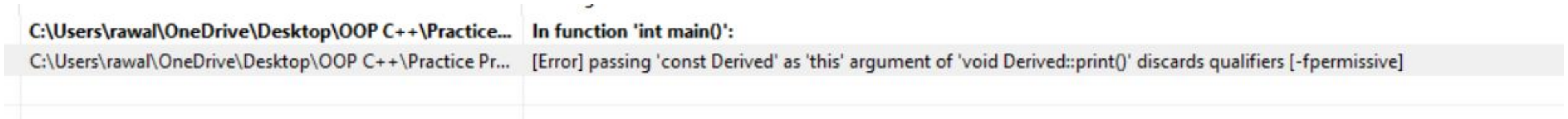
Important point

- A derived class virtual function is only considered overridden if its signature and return type must match exactly

A derived class virtual function is only considered overridden if its signature and return type must match exactly

```
class Base {
  public:
   virtual void print() {
   cout << "base class print()" << endl; }};
class Derived : public Base {
  public:
   int print() {
       cout << "derived class print()" << endl;
                   return 1;}
                   };
int main() {
   Derived derived1;
   Base* base1 = &derived1 ; //up casting
   base1->print();
         }
```

```
#include <iostream>
using namespace std;
class Base {
   public:
     virtual void print() const{
    cout << "base class print()" << endl; }};
class Derived : public Base {
   public:
    void print() {
        cout << "derived class print()" << endl;
              }
              };
int main() {
const Derived derived1;
derived1.print();
       }
```



```
C:\Users\rawal\OneDrive\Desktop\OOP C++\Practice...   In function 'int main()':
C:\Users\rawal\OneDrive\Desktop\OOP C++\Practice Pr...   [Error] passing 'const Derived' as 'this' argument of 'void Derived::print()' discards qualifiers [-fpermissive]
```

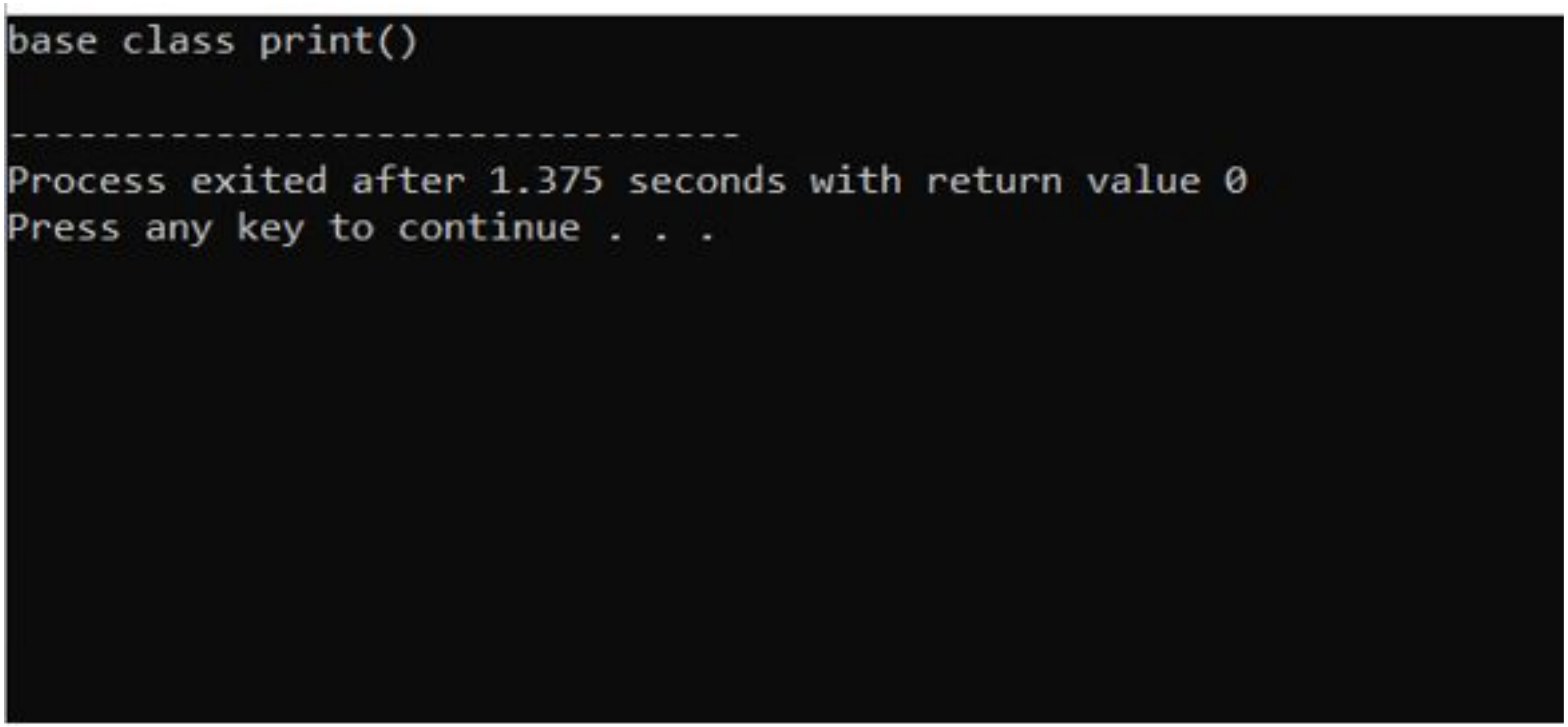
Question ???

can we override const method with non const in c++

It is neither overloading nor overriding. Rather, it is *hiding*.

- const is part of the signature, and **leaving it off changes the signature, and thus hides the method rather than overrides it**. "

```
class Base {
   public:
     virtual void print() const{
    cout << "base class print()" << endl; }};
class Derived : public Base {
   public:
    void print() {
        cout << "derived class print()" << endl;
              }
              };
int main() {
    Derived derived1;
    Base* base1 = &derived1 ;
    base1->print();
       }
```



```
base class print()

--------------------------------
Process exited after 1.375 seconds with return value 0
Press any key to continue . . .
```

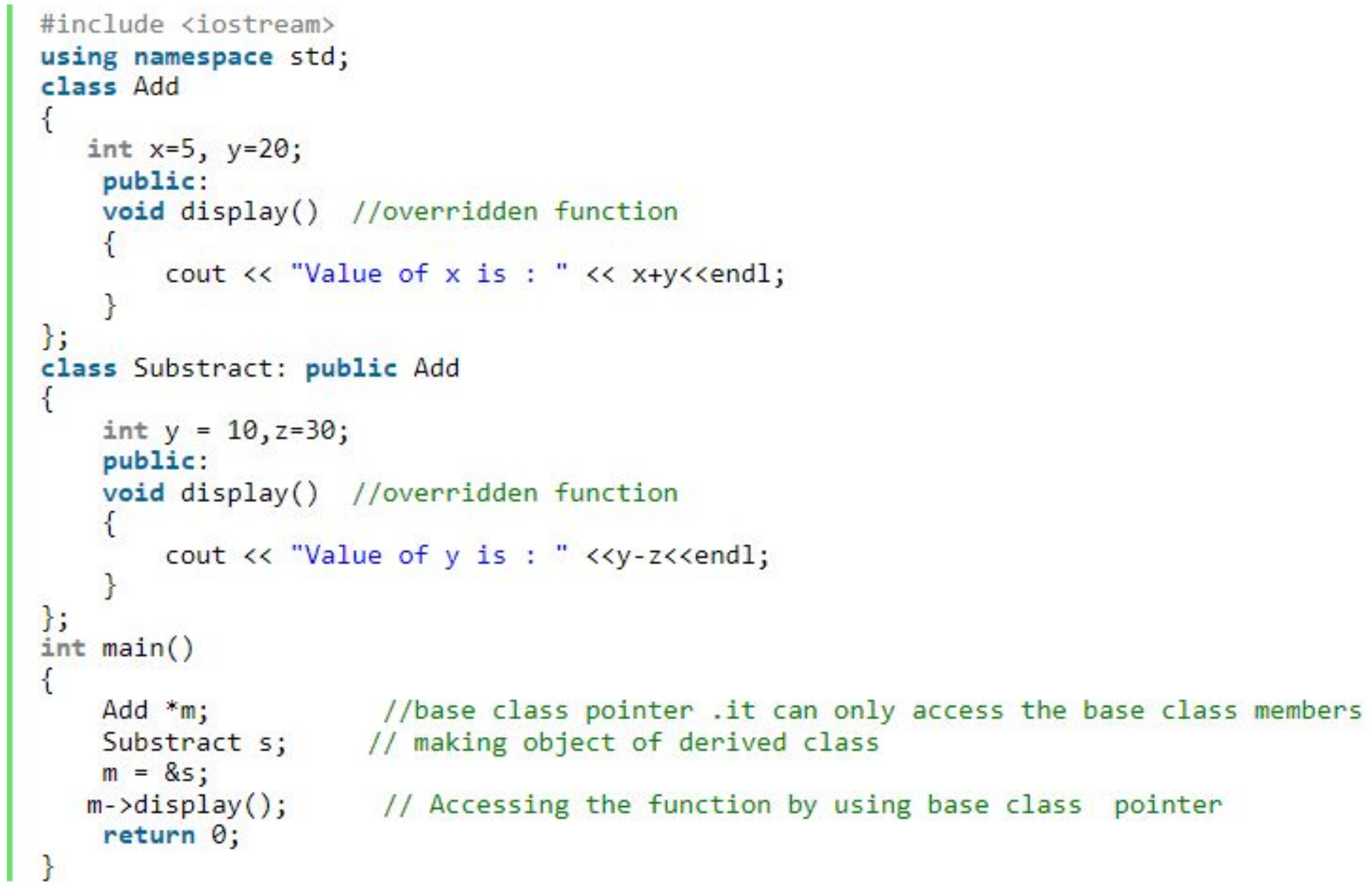
How to prevent class inheritance and method overriding in C++

- **Use of final specifier in C++ 11:**

uses final specifier after the function parameter list to prevent method overriding

```
class Base {
   public:
     virtual void print() final{
    cout << "base class print()" << endl; }};
class Derived : public Base {
   public:
    void print() {
        cout << "derived class print()" << endl;    }};
int main() {
    Derived derived1;
    Base* base1 = &derived1 ;
    base1->print();}
```

Example



```
#include <iostream>
using namespace std;
class Add
{
   int x=5, y=20;
    public:
    void display()  //overridden function
    {
        cout << "Value of x is : " << x+y<<endl;
    }
};
class Substract: public Add
{
    int y = 10,z=30;
    public:
    void display()  //overridden function
    {
        cout << "Value of y is : " <<y-z<<endl;
    }
};
int main()
{
    Add *m;            //base class pointer .it can only access the base class members
    Substract s;      // making object of derived class
    m = &s;
   m->display();       // Accessing the function by using base class  pointer
    return 0;
}
```

Exercise

- Create two classes named Mammals and MarineAnimals. Create another class named BlueWhale which inherits both the above classes. Now, create a function in each of these classes which prints "I am mammal", "I am a marine animal" and "I belong to both the categories: Mammals as well as Marine Animals" respectively. Now, create an object for each of the above class and try calling
  1 - function of Mammals by the object of Mammal
  2 - function of MarineAnimal by the object of MarineAnimal
  3 - function of BlueWhale by the object of BlueWhale
  4 - function of each of its parent by the object of BlueWhale
- Types of Inheritance = ??
- Write a Basic Sekeletone

Exercise

- Create a base class named Person with name, age, and gender as its data members. Define a constructor to initialize the data members and a function display() to display the details of the person.
- Create a derived class named Employee which inherits Person class. Define empid and salary as its data members. Define a constructor to initialize the data members and override the display() function to display the details of the employee.
- Create another derived class named Manager which inherits Employee class. Define department as its data member. Define a constructor to initialize the data members and override the display() function to display the details of the manager.
- Create an object for Person, Employee, and Manager classes and call their display() functions.

Types of Inheritance = ??
Write a Basic Sekeletone

Task

You are part of a team, developing a smart home automation system using C++. The system is responsible for controlling various appliances in a house, including lights, thermostats. The goal is managing these devices&#39; states and settings. For this we have base class Device and appliances lights,
thermostats are inherited from Device class.

1. Device class, it has data member isOn and deviceType(lightController or ThermostatController), constructor which initialize isOn and deviceType(lightController or ThermostatController) variable (values can be either true or false), method "isDeviceOn" to check which device is on or off.
2. LightController class (It manages the state of lights in different rooms). It has data member brightness and zone (Different area of home e.g., Living Room, Kitchen, Bed Room.). The constructor which initializes attributes brightness, zone and isOn , device type attributes via base class constructor.

Cont..

Implement methods to control the lights in various ways:

a. Turn on or off a light in a specific zone (Living Room, Kitchen, Bed Room.) in the house (if isOn is true then brightness should be 100 otherwise 0).

b. Adjust the brightness of a light in specified area of home.

c. Set a timer for a light with specified brightness level to turn it on or off for specified duration in specific area of home.

3. ThermostatController class (It regulates the temperature of different zones (Living Room, Kitchen, Bed Room.) in the house. It has data member temperature and zone. Constructor which initializes temperature variable. Implement methods to control the thermostats in different ways:

a. Set the desired temperature for a specific zone (Increase or decrease the temperature for a specific zone).

b. Schedule a temperature change at a specific time for specified duration in specific area of home.

Create objects based on given scenario.