Polymorphism

Topics:

What is Polymorphism?

- The process of representing one form in multiple form in known as polymorphism . Typically, polymorphism occurs when there is a hierarchy of classes and they are related by inheritance.

- It means that a call to a member function will cause a different function to be executed depending on the type of object that invokes the function.

What is Polymorphism in OOP?

- After the inheritance it is another most important feature of OOP.
- In polymorphism, the member functions with same name are define in base class and also in each derived class.
- Polymorphism is used to keep the interface of base class to its derived classes.
- Polymorphism can be achieved by mean of virtual functions.
- In polymorphism one pointer to a base class object may also point any object of its derived class.

How Polymorphism can be achieved:

- Polymorphism can be achieved in Three ways:
  1. Function Overloading
  2. Function Overriding
  3. Dynamic Binding.

Types of Polymorphism:

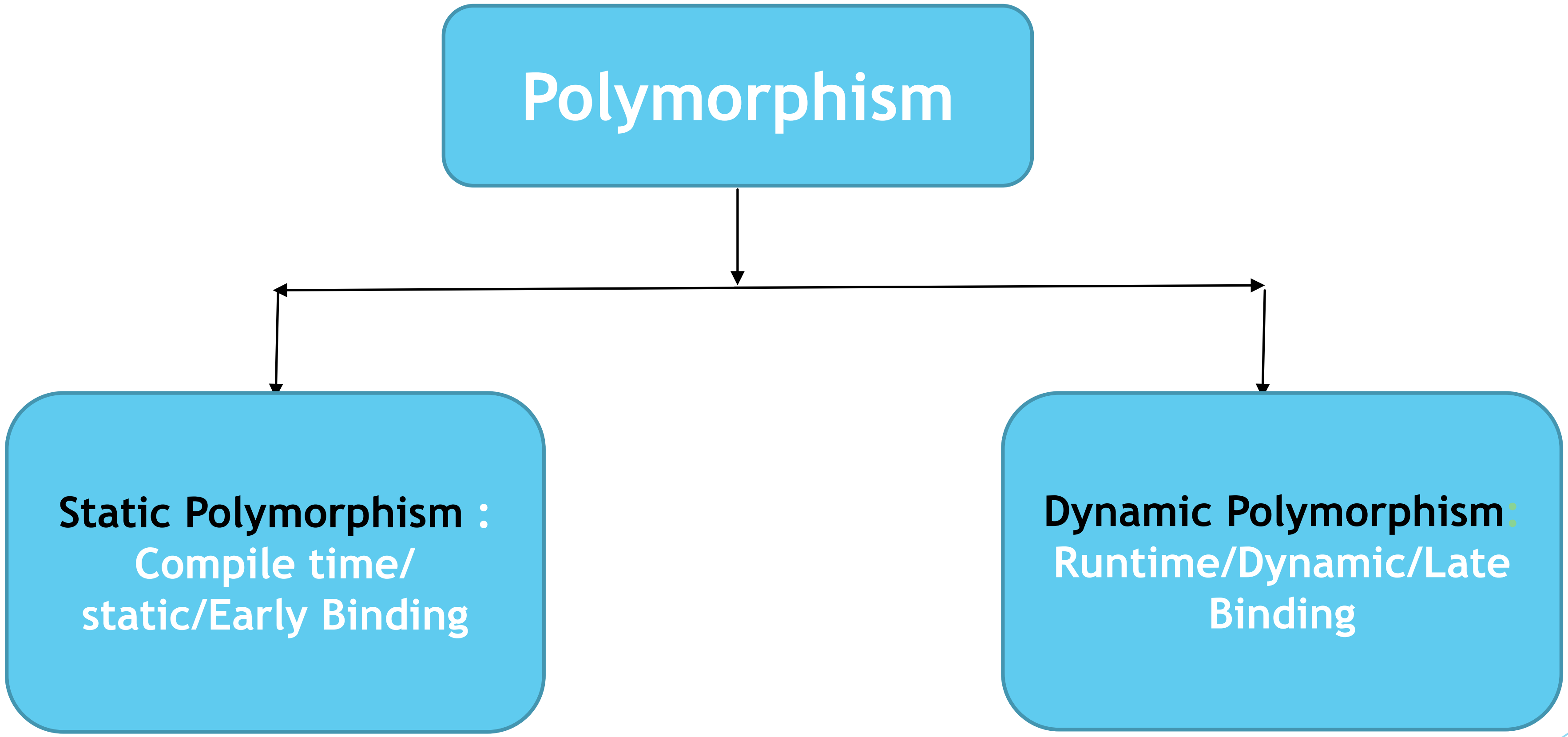


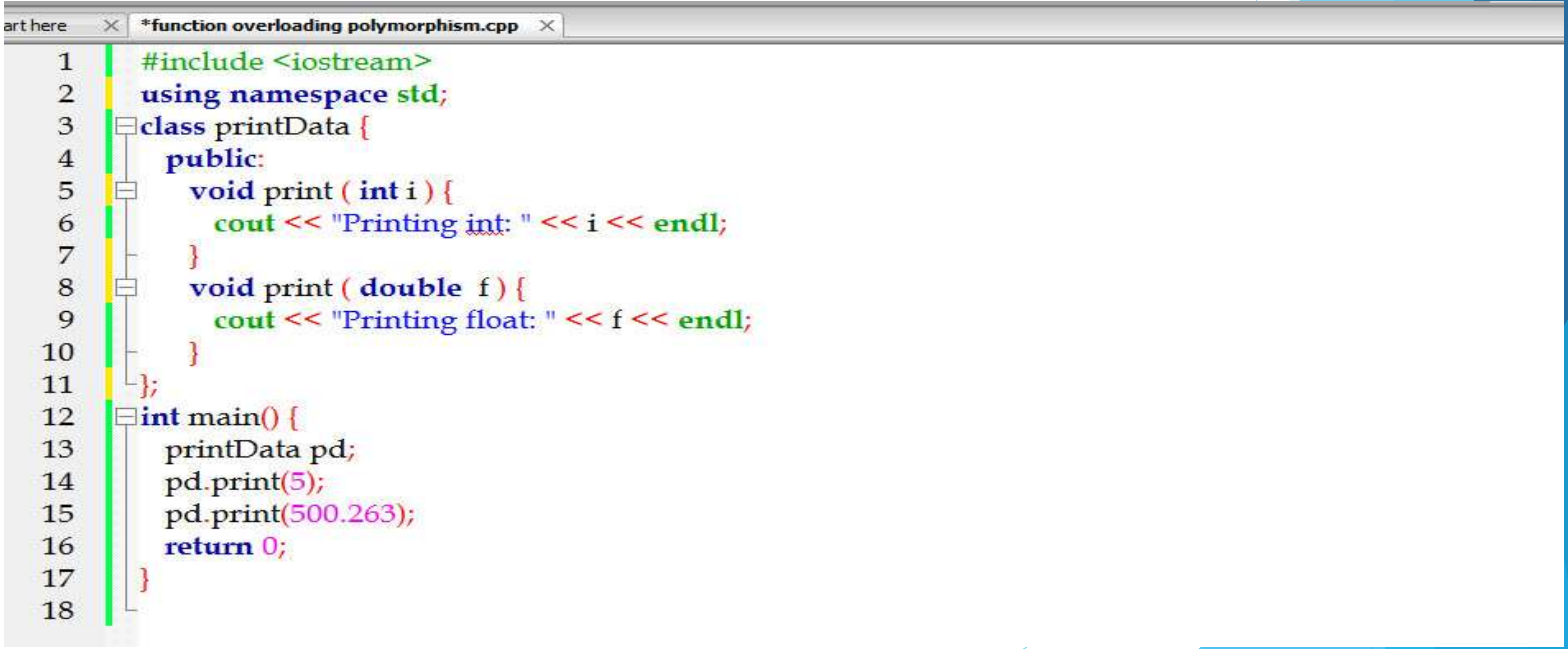
Fig: Types of polymorphism

Types of Polymorphism:

- Static Polymorphism: It takes place during Compile time. Function Overloading, Operator Overloading implements static polymorphism.
- Dynamic Polymorphism: It takes place during Run time. The member function whice change Their behavior during run time called Virtual Functions.

Function Overloading

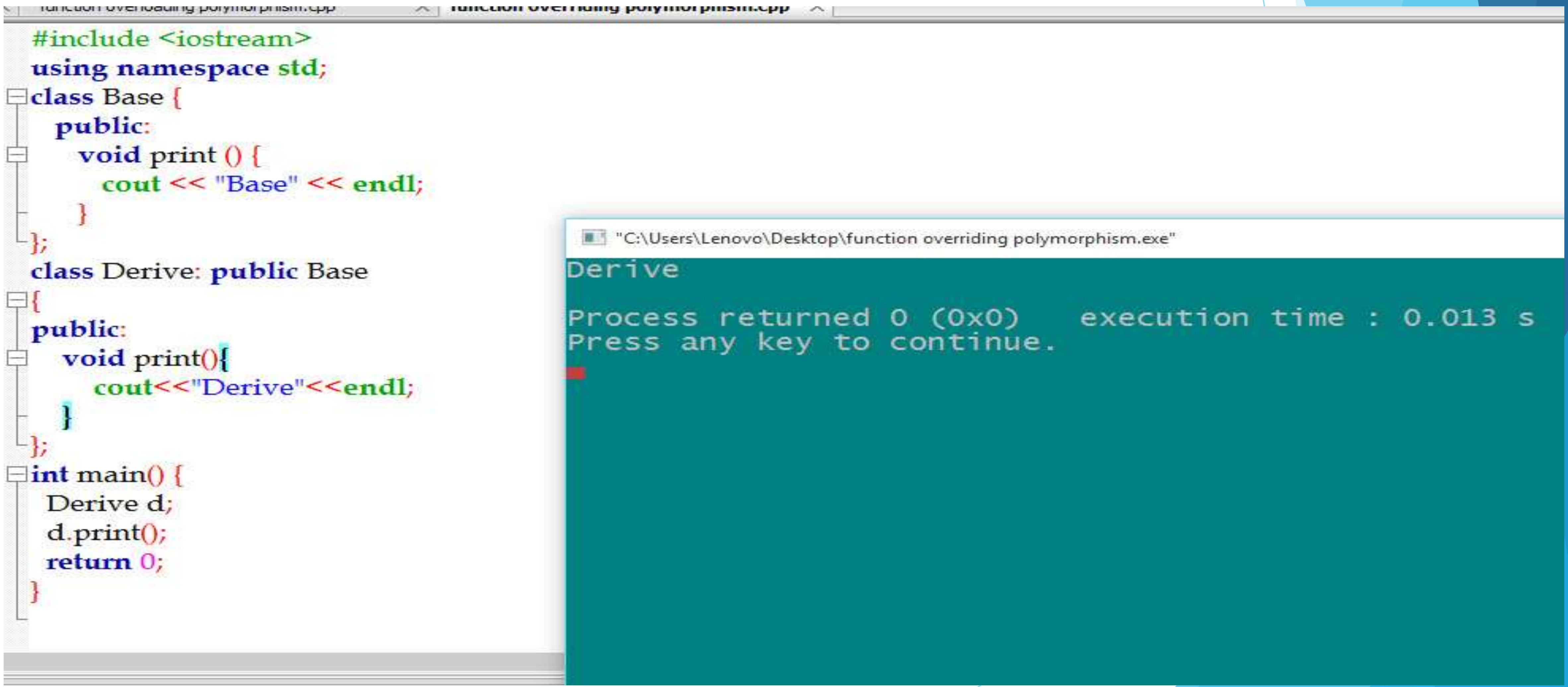
- Here we have multiple definitions for the same function name in the same scope. The definition of the function must differ from each other by the types and/or the number of arguments in the argument list.



```
#include <iostream>
using namespace std;
class printData {
   public:
      void print ( int i ) {
         cout << "Printing int: " << i << endl;
      }
      void print ( double  f ) {
         cout << "Printing float: " << f << endl;
      }
};
int main() {
   printData pd;
   pd.print(5);
   pd.print(500.263);
   return 0;
}
```

Function Overriding

- Overriding of functions happens when one class inherits another class. Here functions of both class have the same name and there world be same parameter list and return type of the functions should be same.



```cpp
#include <iostream>
using namespace std;
class Base {
  public:
    void print () {
      cout << "Base" << endl;
    }
};
class Derive: public Base
{
public:
   void print(){
      cout<<"Derive"<<endl;
   }
};
int main() {
  Derive d;
  d.print();
  return 0;
}
```

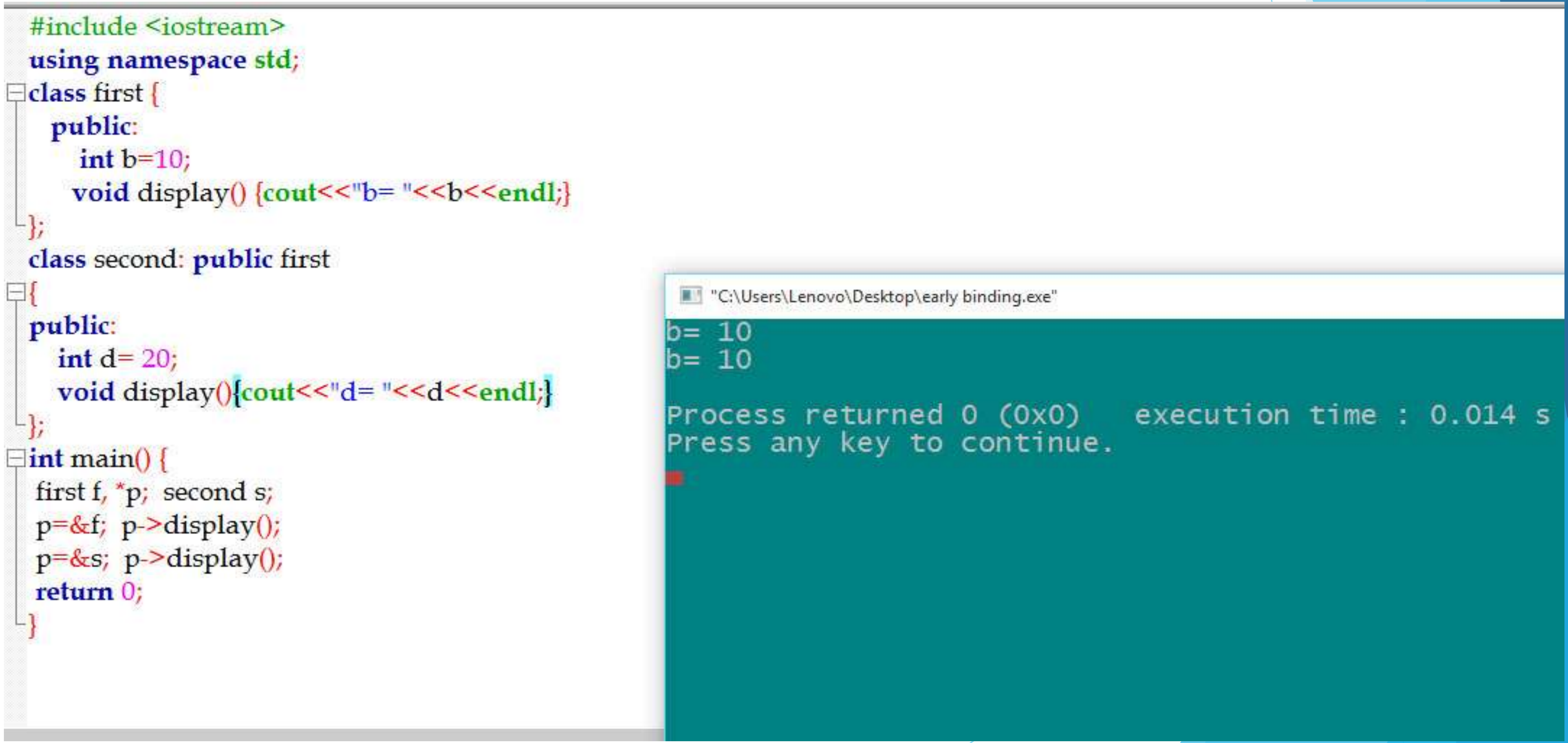
"C:\Users\Lenovo\Desktop\function overriding polymorphism.exe"

```
Derive

Process returned 0 (0x0)   execution time : 0.013 s
Press any key to continue.
```

Early Binding

- Events occurring at compile time are known as early binding. In the process of early binding all info which is required for a function call is known at compile time. Examples of early binding: function calls, overloaded function calls, and overloaded operators.



```
#include <iostream>
using namespace std;
class first {
   public:
      int b=10;
     void display() {cout<<"b= "<<b<<endl;}
};
class second: public first
{
public:
    int d= 20;
    void display(){cout<<"d= "<<d<<endl;}
};
int main() {
 first f, *p;  second s;
 p=&f;  p->display();
 p=&s;  p->display();
 return 0;
}
```

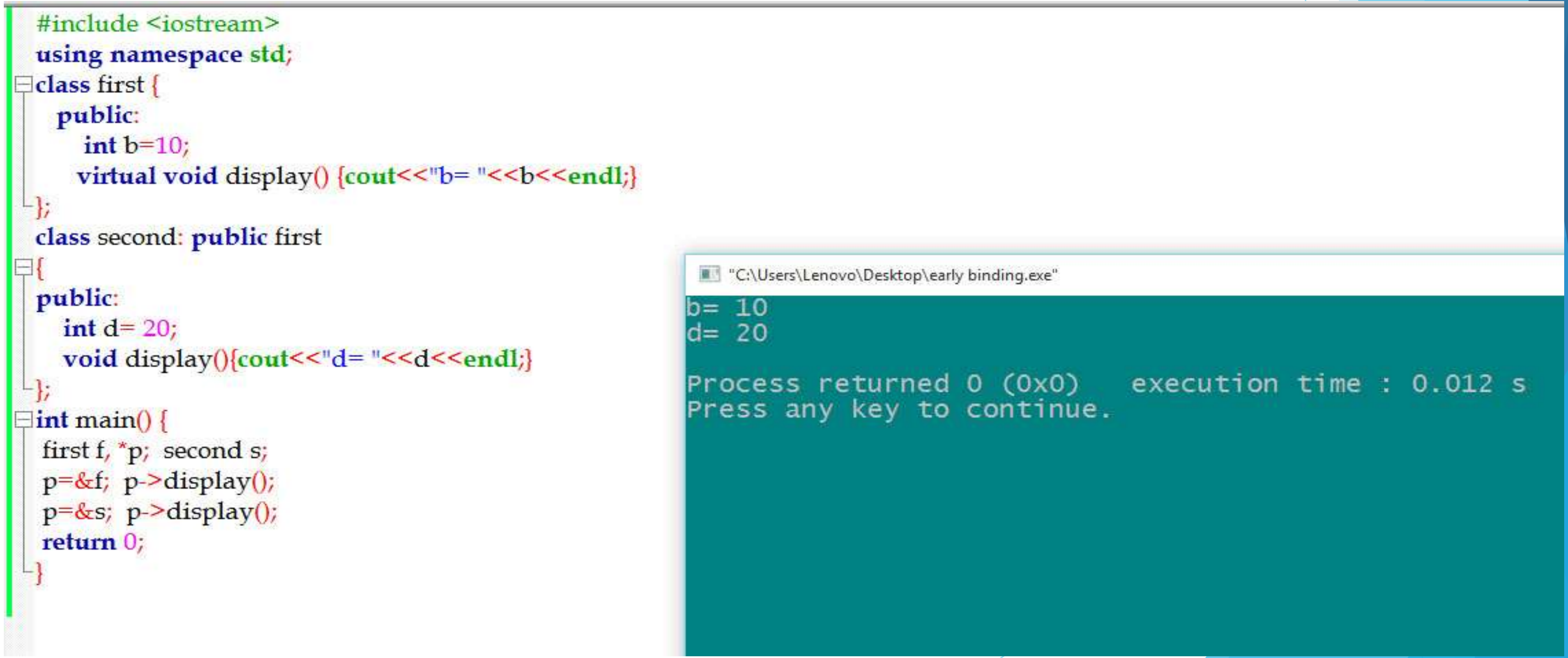
"C:\Users\Lenovo\Desktop\early binding.exe"

```
b= 10
b= 10

Process returned 0 (0x0)   execution time : 0.014 s
Press any key to continue.
```

Late Binding and Virtual Function:

- All the Code is understood at the Time of Execution This is called as Late Binding. As we Know that Late Binding is Performed By using the virtual Functions. Virtual Means Function must be Override. Virtual Function is a function in base class, which is overrided in the derived class, and which tells the compiler to perform **Late Binding** on this function.



```
#include <iostream>
using namespace std;
class first {
  public:
    int b=10;
   virtual void display() {cout<<"b= "<<b<<endl;}
};
class second: public first
{
public:
   int d= 20;
   void display(){cout<<"d= "<<d<<endl;}
};
int main() {
 first f, *p;  second s;
 p=&f;  p->display();
 p=&s;  p->display();
 return 0;
}
```

"C:\Users\Lenovo\Desktop\early binding.exe"

```
b= 10
d= 20

Process returned 0 (0x0)   execution time : 0.012 s
Press any key to continue.
```

Pure Virtual Function

- The virtual function that is only declare But Not Define In The In The Base Class.
- General Syntax:
- virtual function_name()=0;
- The class that contains the pure virtual function exists only to act as base or parent classes.

thank
you