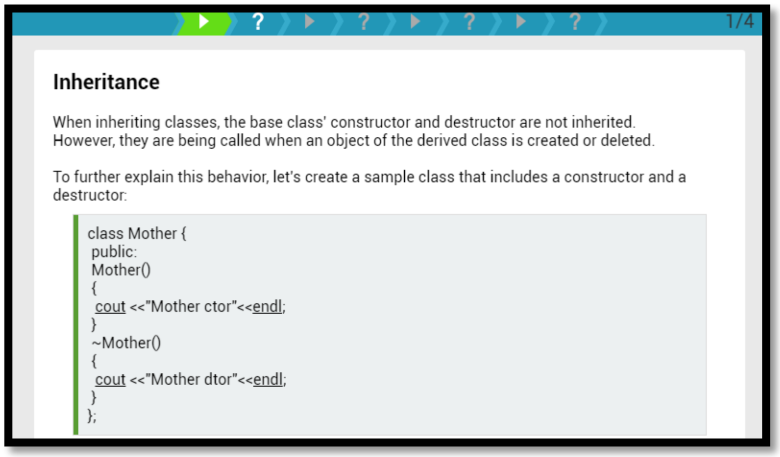
DAILY ASSESSMENT

Date:	25/06/2020	Name:	Davis S. Patel
Course:	Programming in C++	USN:	4AL16EC045
Topic:	Module 7 & 8	Semester & Section:	8TH - A
GitHub Repository:	Davis		

FORENOON SESSION DETAILS

Image of session



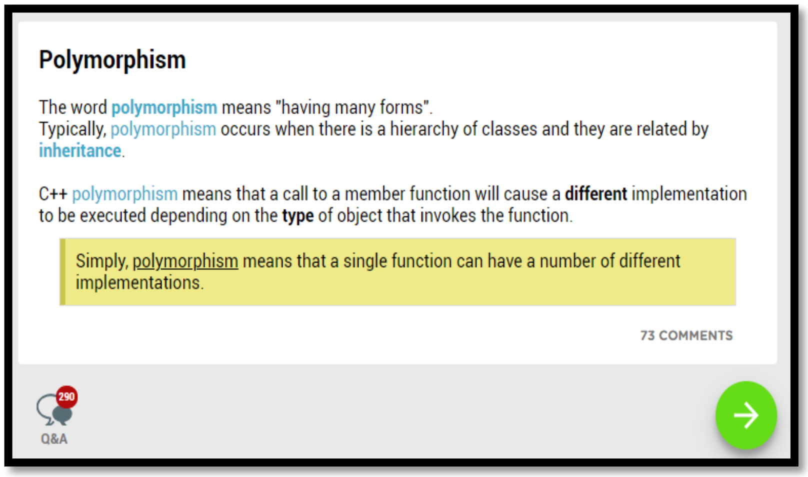
1/4

Inheritance

When inheriting classes, the base class' constructor and destructor are not inherited. However, they are being called when an object of the derived class is created or deleted.

To further explain this behavior, let's create a sample class that includes a constructor and a destructor:

```
class Mother {
 public:
 Mother()
 {
  cout <<"Mother ctor"<<endl;
 }
 ~Mother()
 {
  cout <<"Mother dtor"<<endl;
 }
};
```



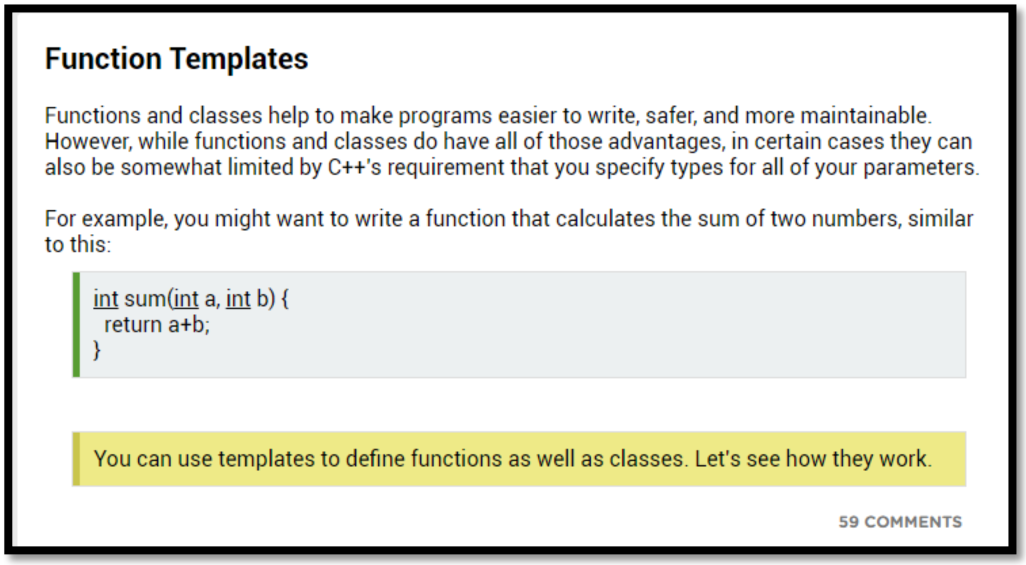
Polymorphism

The word **polymorphism** means "having many forms".
Typically, polymorphism occurs when there is a hierarchy of classes and they are related by **inheritance**.

C++ polymorphism means that a call to a member function will cause a **different** implementation to be executed depending on the **type** of object that invokes the function.

Simply, polymorphism means that a single function can have a number of different implementations.

73 COMMENTS



Function Templates

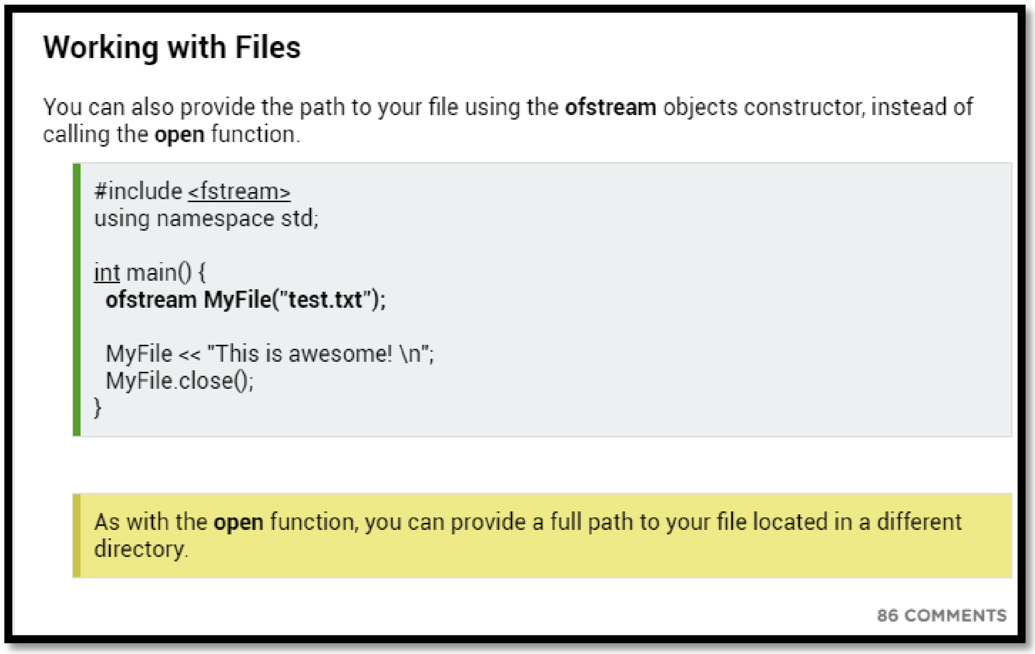
Functions and classes help to make programs easier to write, safer, and more maintainable. However, while functions and classes do have all of those advantages, in certain cases they can also be somewhat limited by C++'s requirement that you specify types for all of your parameters.

For example, you might want to write a function that calculates the sum of two numbers, similar to this:

```
int sum(int a, int b) {
  return a+b;
}
```

You can use templates to define functions as well as classes. Let's see how they work.

59 COMMENTS



Working with Files

You can also provide the path to your file using the **ofstream** objects constructor, instead of calling the **open** function.

```
#include <fstream>
using namespace std;

int main() {
  ofstream MyFile("test.txt");

  MyFile << "This is awesome! \n";
  MyFile.close();
}
```

As with the **open** function, you can provide a full path to your file located in a different directory.

86 COMMENTS

REPORT:

C++ Inheritance

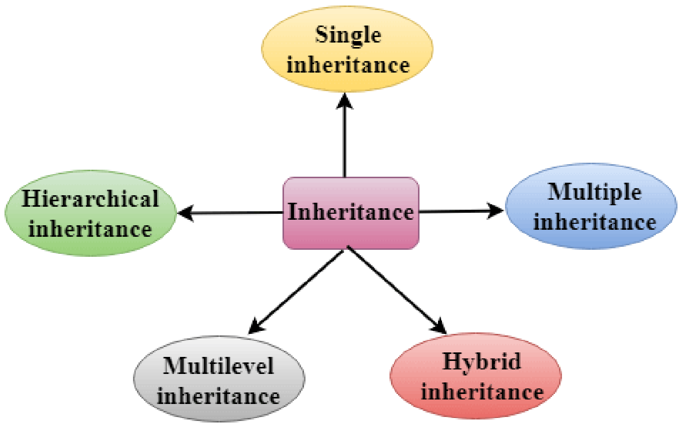
In C++, inheritance is a process in which one object acquires all the properties and behaviors of its parent object automatically. In such way, you can reuse, extend or modify the attributes and behaviors which are defined in other class.

In C++, the class which inherits the members of another class is called derived class and the class whose members are inherited is called base class. The derived class is the specialized class for the base class.

Types of Inheritance

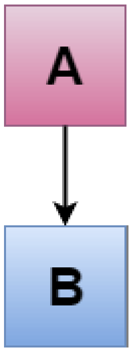
C++ supports five types of inheritance:

- Single inheritance
- Multiple inheritance
- Hierarchical inheritance
- Multilevel inheritance
- Hybrid inheritance



C++ Single Inheritance

Single inheritance is defined as the inheritance in which a derived class is inherited from the only one base class.



Where 'A' is the base class, and 'B' is the derived class.

Example –

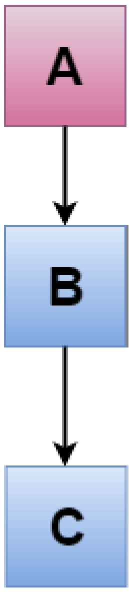
```
include <iostream>
using namespace std;
 class Account {
   public:
   float salary = 60000;
 };
   class Programmer: public Account {
   public:
   float bonus = 5000;
   };
int main(void) {
     Programmer p1;
     cout<<"Salary: "<<p1.salary<<endl;
     cout<<"Bonus: "<<p1.bonus<<endl;
    return 0;
}
```

Output:

```
Salary: 60000
Bonus: 5000
```

C++ Multilevel Inheritance

Multilevel inheritance is a process of deriving a class from another derived class.



Example –

```
#include <iostream>
using namespace std;
 class Animal {
   public:
 void eat() {
    cout<<"Eating..."<<endl;
 }
   };
   class Dog: public Animal
   {
       public:
     void bark(){
    cout<<"Barking..."<<endl;
```

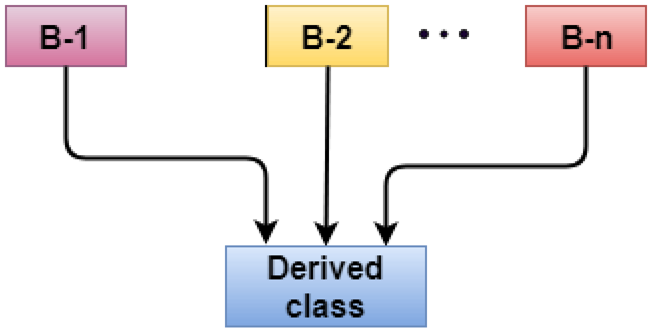
```
     }
   };
   class BabyDog: public Dog
   {
       public:
     void weep() {
    cout<<"Weeping...";
     }
   };
int main(void) {
    BabyDog d1;
    d1.eat();
    d1.bark();
     d1.weep();
     return 0;
}
```

Output:

```
Eating...
Barking...
Weeping...
```

C++ Multiple Inheritance

Multiple inheritance is the process of deriving a new class that inherits the attributes from two or more classes.



Example –

```
#include <iostream>
using namespace std;
class A
{
    protected:
     int a;
    public:
    void get_a(int n)
    {
        a = n;
    }
};

class B
{
    protected:
    int b;
    public:
    void get_b(int n)
    {
        b = n;
    }
};
class C : public A,public B
{
   public:
    void display()
    {
        std::cout << "The value of a is : " <<a<< std::endl;
        std::cout << "The value of b is : " <<b<< std::endl;
        cout<<"Addition of a and b is : "<<a+b;
    }
};
int main()
{
   C c;
   c.get_a(10);
   c.get_b(20);
   c.display();
```

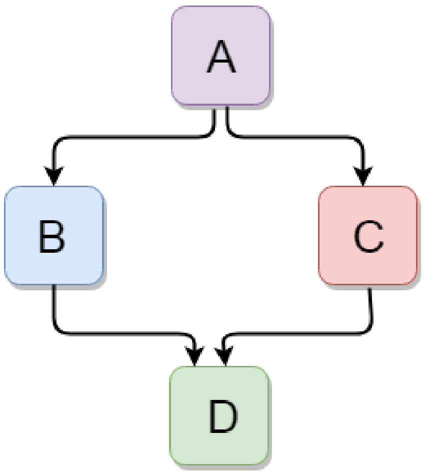
```
    return 0;
}
```

Output:

```
The value of a is : 10
The value of b is : 20
Addition of a and b is : 30
```

C++ Hybrid Inheritance

Hybrid inheritance is a combination of more than one type of inheritance.



Function templates

Function templates are special functions that can operate with *generic types*. This allows us to create a function template whose functionality can be adapted to more than one type or class without repeating the entire code for each type.

In C++ this can be achieved using *template parameters*. A template parameter is a special kind of parameter that can be used to pass a type as argument: just like regular function parameters can be used to pass values to a function, template parameters allow to pass also types to a function. These function templates can use these parameters as if they were any other regular type.

The format for declaring function templates with type parameters is:

```
template <class identifier> function_declaration;
template <typename identifier> function_declaration;
```

The only difference between both prototypes is the use of either the keyword class or the keyword typename. Its use is indistinct, since both expressions have exactly the same meaning and behave exactly the same way.