

DAILY ASSESSMENT FORMAT

Date:	25-06-2020	Name:	Rajeshwari Gadagi
Course:	C++ Programming	USN:	4AL17EC076
Topic:	Module 7	Semester & Section:	6 SEM & 'B' SEC
Github Repository:	Rajeshwari-gadagi		

FORENOON SESSION DETAILS

Image of session

Inheritance & Polymorphism
Inheritance XP 233 1/4

Inheritance

Inheritance is one of the most important concepts of object-oriented programming. Inheritance allows us to define a class based on another class. This facilitates greater ease in creating and maintaining an application.

The class whose properties are inherited by another class is called the **Base** class. The class which inherits the properties is called the **Derived** class. For example, the **Daughter** class (derived) can be inherited from the **Mother** class (base). The derived class inherits all feature from the base class, and can have its own additional features.

Base Class

base class

Derived Class
(inherited from base class)

base class features

Inheritance & Polymorphism
Inheritance XP 233 2/4

Inheritance

To demonstrate **inheritance**, let's create a **Mother** class and a **Daughter** class:


```

class Mother
{
public:
    Mother() {};
    void sayHi() {
        cout << "Hi";
    }
};

class Daughter
{
public:
    Daughter() {};
};
    
```

The Mother class has a public method called **sayHi()**.

The next step is to **inherit** (derive) the Daughter from the Mother

 Inheritance & Polymorphism
Inheritance

XP 233

4/4


Inheritance

As all public members of the Mother class become public members for the Daughter class, we can create an object of type Daughter and call the **sayHi()** function of the Mother class for that object.

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

class Mother
{
public:
    Mother() {}
    void sayHi() {
        cout << "Hi";
    }
};

class Daughter: public Mother
{
public:
    Daughter() {}
};
```

 Inheritance & Polymorphism
Protected Members

XP 238


1/3


Access Specifiers

Up to this point, we have worked exclusively with **public** and **private** access specifiers. Public members may be accessed from anywhere outside of the class, while access to private members is limited to their class and friend functions.

As we've seen previously, it's a good practice to use public methods to access private class variables.

31 COMMENTS

 533 Q&A



Report – Report can be typed or hand written for up to two pages.

<p><u>C++ Programming</u></p> <p>25/06/2020 Thursday</p> <p>→ <u>Module 8</u></p> <p>→ <u>Inheritance & Polymorphism</u></p> <p><u>Inheritance</u></p> <ul style="list-style-type: none"> It is one of the most important concepts of OOP Inheritance allows us to define a class based on another class. This provides greater reuse in creating and maintaining an application The base class is specified using a colon and an access specifier. public means that all public members of the base class are public in the derived class. <p>> <u>Protected Member</u></p> <p><u>Access specifier:</u></p> <p><u>Protected</u></p> <p>There are two access specifier, protected & protected member variable & function is very similar to private member, with the difference - when accessed with the derived class.</p> <p>> <u>Derived class Constructor & Destructor</u></p> <p>When inheriting classes, the base class constructor is not inherited. However, they are being called when an object</p>	<p>of the derived class is created & deleted</p> <p><u>Constructors:</u></p> <p>The base class constructor is called first</p> <p><u>Destructors:</u></p> <p>The derived class destructor is called first, then the base class destructor gets called.</p> <p>> <u>Polymorphism</u></p> <ul style="list-style-type: none"> It means when there is inheritance of classes & they have overloaded definitions One 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. <p>> <u>Virtual Functions</u></p> <p>Defining a virtual function in the base class with a corresponding version in a derived class allows polymorphism using function pointers to call the derived class's function</p> <p>> <u>Abstract classes</u></p> <p>Virtual functions are also have their implementation in the base class.</p> <p><u>Pure virtual function</u></p> <p>The virtual member functions without definitions are known as pure virtual function. A pure virtual function basically declares that the derived classes will have this function defined on their own.</p>
--	--

Date:	25-06-2020	Name:	Rajeshwari Gadagi
Course:	C++ Programming	USN:	4AL17EC076
Topic:	Module 8	Semester & Section:	6 SEM & 'B' SEC
Github Repository:	Rajeshwari-gadagi		

AFTERNOON SESSION DETAILS
Image of session



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



Function Templates

Now we can use our generic data type `T` in the function:

```
template <class T>  
T sum(T a, T b) {  
    return a+b;  
}  
  
int main () {  
    int x=7, y=15;  
    cout << sum(x, y) << endl;  
}  
  
// Outputs 22
```

Try It Yourself

The function returns a value of the generic type `T`, taking two parameters, also of type `T`.

Our new function worked exactly as the previous one for integer values did.



Function Templates

Template functions can save a lot of time, because they are written only once, and work with different types.

Template functions reduce code maintenance, because duplicate code is reduced significantly.

Enhanced safety is another advantage in using template functions, since it's not necessary to manually copy functions and change types.

51 COMMENTS





Function Templates

Function templates also make it possible to work with **multiple** generic data types. Define the data types using a comma-separated list. Let's create a function that compares arguments of varying data types (an **int** and a **double**), and prints the smaller one.

```
template <class T, class U>
```

As you can see, this template declares two different generic data types, **T** and **U**.

31 COMMENTS



Q&A



Report – Report can be typed or hand written for up to two pages.

C++ Programming

15/01/2020 - Thursday

Module:

> Template, Exceptions & File

> Function Template

• Functions & classes help to make programs easier to write, safer & more maintainable

• The function works as expected, but in limited solely to integers

> Function Template

• It becomes necessary to write a new function for each data type, such as integers, double, even (double & 1/2) even odd & 1/2

• Using function template, the basic idea is to avoid the necessity of defining an exact type for each variable. Instead C++ provides us with the capability of defining functions using placeholder types, called template type parameters.

• Template functions can take a lot of data, because they are written once and work with different types

> Function Template

• It is also possible to work with generic data types. Define the data type using some pre-defined data

• This is the idea of generic and it is used to define type parameters

• Don't always use T, because you can declare a type parameter using any identifier that works for you

> How Template:

• Syntax:

```
template <class T>  
class MyClass  
{  
};
```

• A specific syntax is required when you define a function template which is as follows

• The bigger function defines the greater value of the number variable

> Template Specialization

• It allows for the use of function of a different implementation. In a template, a specific type is passed as a template argument

> Exceptions

• Problems that occur when program execution are called exceptions. C++ exception handling is built upon keywords: try, catch & throw