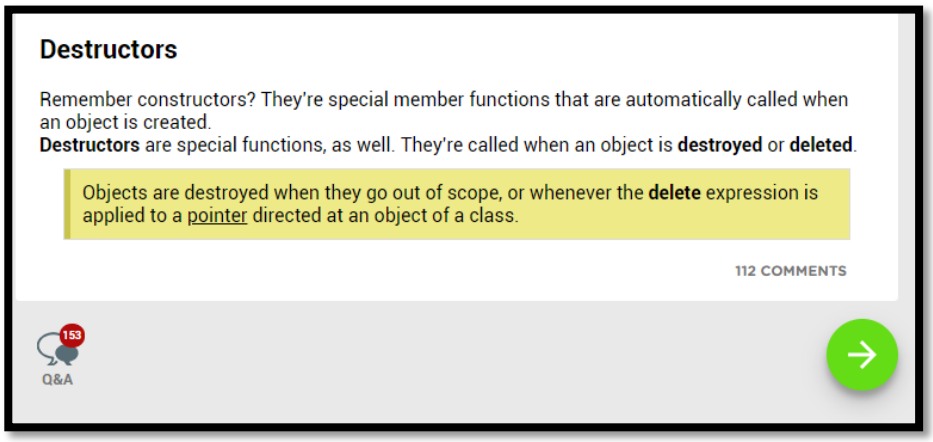
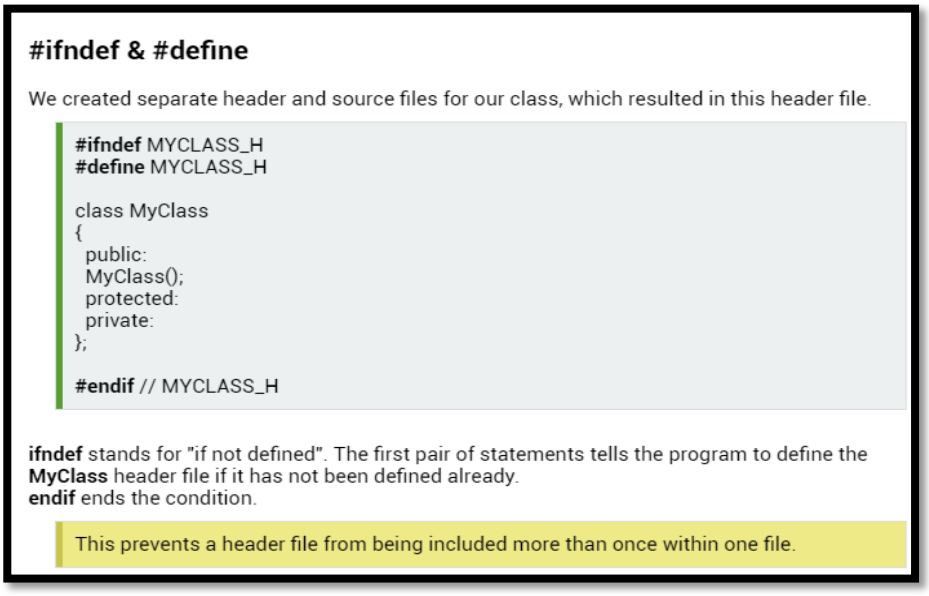
DAILY ASSESSMENT

Date:	**24/06/2020**	**Name:**	**Davis S. Patel**
Course:	**Programming in C++**	**USN:**	**4AL16EC045**
Topic:	**Module 5 & 6**	**Semester & Section:**	**8th - A**
GitHub Repository:	**Davis**		

FORENOON SESSION DETAILS

Image of session





Member Functions

Let's create a sample function called **myPrint()** in our class.
MyClass.h

```
class MyClass
{
  public:
    MyClass();
    void myPrint();
};
```

MyClass.cpp

```
#include "MyClass.h"
#include <iostream>
using namespace std;

MyClass::MyClass() {
}

void MyClass::myPrint() {
  cout <<"Hello"<<endl;
}
```

Dot Operator

Next, we'll create an object of the type **MyClass**, and call its **myPrint()** function using the dot (.) operator:

```
#include "MyClass.h"

int main() {
  MyClass obj;
  obj.myPrint();
}

// Outputs "Hello"
```

Try It Yourself

REPORT –

C++ Separate Header and Implementation Files C++ classes (and often function prototypes) are normally split up into two files. The header file has the extension of .h and contains class definitions and functions. The implementation of the class goes into the .cpp file. By doing this, if your class implementation doesn't change then it won't need to be recompiled. Most IDE's will do this for you – they will only recompile the classes that have changed. This is possible when they are split up this way, but it isn't possible if everything is in one file (or if the implementation is all part of the header file).

Using #ifndef Sometimes we can end up including a header file multiple times. C++ doesn't like this if it ends up redefining something again. In our previous example this didn't happen because main.cpp and Num.cpp are compiled separately so the inclusion of Num.h causes no problem. But consider if we had another class in main that uses Num:

Destructors in C++

Destructors in C++ are members functions in a class that delete an object. They are called when the class object goes out of scope such as when the function ends, the program ends, a delete variable is called etc.

Destructors are different from normal member functions as they don't take any argument and don't return anything. Also, destructors have the same name as their class and their name is preceded by a tilde(~).

A program that demonstrates destructors in C++ is given as follows.

```
#include<iostream>

using namespace std;

class Demo {

   private:
```

```
   int num1, num2;
   public:
   Demo(int n1, int n2) {
      cout<<"Inside Constructor"<<endl;
      num1 = n1;
      num2 = n2;
   }
   void display() {
      cout<<"num1 = "<< num1 <<endl;
      cout<<"num2 = "<< num2 <<endl;
   }
   ~Demo() {
      cout<<"Inside Destructor";
   }
};
int main() {
   Demo obj1(10, 20);
   obj1.display();
   return 0;
}
```

Output

Inside Constructor

num1 = 10

num2 = 20

Inside Destructor

Const member functions in C++

Like member functions and member function arguments, the objects of a class can also be declared as **const**. an object declared as const cannot be modified and hence, can invoke only const member functions as these functions ensure not to modify the object.
A const object can be created by prefixing the const keyword to the object declaration. Any attempt to change the data member of const objects results in a compile-time.

Syntax:

const Class_Name Object_name;

- When a function is declared as const, it can be called on any type of object, const object as well as non-const objects.
- Whenever an object is declared as const, it needs to be initialized at the time of declaration. However, the object initialization while declaring is possible only with the help of constructors.

C++ Friend Functions

A friend function of a class is defined outside that class' scope but it has the right to access all private and protected members of the class. Even though the prototypes for friend functions appear in the class definition, friends are not member functions.

A friend can be a function, function template, or member function, or a class or class template, in which case the entire class and all of its members are friends.

To declare a function as a friend of a class, precede the function prototype in the class definition with keyword **friend** as follows −

```
class Box {
   double width;

   public:
      double length;
      friend void printWidth( Box box );
      void setWidth( double wid );
};
```

C++ allows you to specify more than one definition for a function name or an operator in the same scope, which is called function overloading and operator overloading respectively.

An overloaded declaration is a declaration that is declared with the same name as a previously declared declaration in the same scope, except that both declarations have different arguments and obviously different definition (implementation).

When you call an overloaded function or operator, the compiler determines the most appropriate definition to use, by comparing the argument types you have used to call the function or operator with the parameter types specified in the definitions. The process of selecting the most appropriate overloaded function or operator is called overload resolution.