Classes and Objects in C++ programming language

"Classes, Objects, Functions, Data Hiding"

Fundamentals of OOPs

Shakirullah Waseeb
shakir.waseeb@gmail.com

Nangarhar University

October 9, 2017

Agenda

Agenda

Introduction

- Sometimes it is convenient to initialize one or more member data upon creation of objects
- Consider the Counter example, which hold a counter variable, an increment and get_counter function; and can be used as a general purpose counter
- Initially upon creation of Counter instance variable the counter must be initialized with 0, so it would be convenient to have such a behavior
- This can be achieved using **constructor**; a special member function called automatically upon object instantiation

Syntax

- Constructor has same syntax with that of a function except; has no return type and its name is same with class name
- It must always be declared under public access specifier scope

counter.cpp

```
#include <iostream >
using namespace std;
class Counter {
    private:
        int counter;
    public:
        Counter () {
            counter = 0;
        }
        void increment() {
            counter += 1;
        }
        int get_counter() {
            return counter;
        }
};
```

Syntax - - - continue

- We can also use the Initializer list syntax to initialize data members with constructor as:

```
Counter() : counter(0) {}
SomeOtherClass() : Id(5), Price(2.4), Comment("Hi") {}
```

Agenda

Constructor Overloading

- When we don't define a no-argument constructor explicitly, an implicit no-argument constructor is built in to the program by compiler automatically
- This constructor is called *default constructor*, which enable us to creates the object
- We can overload constructor the way we overload the functions
- In case of overloaded constructors, while, creating the objects we must specify which constructor we wants to initialize the object

```
 Counter(): counter(0) {}
 Counter(int c): counter(c) {}
 SomeOtherClass(int id, float price, string comment): Id(id),
Price(price), Comment(comment) {}
```

Instantiation:

```
 Counter counter1, counter2(4);
```

Agenda

Default Copy Constructor

- We studied two ways to initialize objects:
  no-argument constructor: initialize data members to constant values
  multi-argument constructor: initialize dat members to values passed as arguments
- Another way to initialize an object! : *initialize it with another object of the same data type*
- Surprisingly, there exists a built into special constructor for all classes that do this for us:
  default copy constructor: a one argument constructor with an argument object of the same class as the constructor

Agenda

Destructors

- A function is called upon creation of an object automatically, similarly, there is also a function which is called automatically when an object is destroyed
- Such function is called *destructor*
- Its syntax is similar with that of constructor, only it is preceded by a tilde as:

  $\sim$Counter (){}

Agenda

Functions defined outside the class

- A class member function can also be defined outside the class
- It must be declared inside the class
- Its definition outside from the class can be associated with it via its class name and the scope resolution operator

Example

```
#include <iostream >
using namespace std;
class Simple {
    private:
        int numb;
    public:
        Simple (): numb(0) {}
        void print();
};
void Simple::print() {
            numb = simp1.numb + simp2.numb;
        }
int main() {
    Simple simp;
    simp.print();
}
```

Agenda

- Objects can be passed to function as argument
- The syntax for object arguments is same with that of simple data type
- Since a function of the class is its member function, hence it can access the private data of in any object of the same class supplied to it as an argument
- A member function is always given access to the object for which it is called (object connected to it with *dot* operator), but it may be able to access other objects

Example

Example

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

class Simple {
    private:
        int numb;
    public:
        Simple (): numb(0) {}
        Simple (int n): numb(n) {}

        void AddSimples(Simple simp1, Simple simp2) {
            numb = simp1.numb + simp2.numb;
        }

        int get_numb() {
            return numb;
        }
};

int main() {
    Simple simp1(35), simp2(45), simp3;
    simp3.AddSimples(simp1, simp2);
}
```

Agenda

- We can also return an object from a function

Example

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

class Simple {
    private:
        int numb;
    public:
        Simple (): numb(0) {}
        Simple (int n): numb(n) {}

        Simple AddSimples(Simple simp) {
            Simple temp;
            temp.numb = numb + simp.numb;
            return temp;
        }

        int get_numb() {
            return numb;
        }
};

int main() {
    Simple simp1(35), simp2(45), simp3;
    simp3 = simp1.AddSimples(simp2);
}
```

What's next?

Inheritance

a) Derived and Base Classes, Derived Class Constructors

Your Turn: Time to hear from you!

[1]

[1] https://fensafitters.files.wordpress.com/2013/07/3d095.jpg

References

Robert Lafore
Object-Oriented Programming in C++, 4th Edition .
2002.

Piyush Kumar
Object oriented Programming (Using C++)
http://www.compgeom.com/ piyush/teach/3330