OOAIA

Rupesh Nasre.
rupesh@iitm.ac.in

January 2018

In these lectures

- Introduction to OOP
- Classes and Objects
- Operator Overloading
- Inheritance
- Templates

Concepts are applicable in general.
We will use C++ and Linux as the environments.

Prerequisite:

- Programming experience

References:

- *The C++ Programming Language*, Bjarne Stroustrup, 4e, Pearson
- *C++ Primer Plus*, Stephen Prata, 6e, Pearson

Hello World!

hello.c

```
#include <stdio.h>
int main() {
      printf("Hello World!\n");
      return 0;
}
```

```
$ gcc hello.c
$ a.out
Hello World!
$ cp hello.c hello.cpp
$ g++ hello.cpp
$ a.out
Hello World!
```

hello2.cpp

```
#include <iostream>
int main() {
      std::cout << "Hello World!\n";
      return 0;
}
```

```
$ g++ hello2.cpp
$ a.out
Hello World!
```

hello.java

```
class Message {
      public static void main(String[] x) {
            System.out.print("Hello World!\n");
}     }
```

```
$ javac hello.java
$ java Message
Hello World!
```


Homework: Check what happens with *gcc* and *g++* when *main* is declared as *void main()*.

OO Hello World!

Procedural

```
#include <iostream>
int main() {
        std::cout << “Hello World!\n”;
        return 0;
}
```

Object-oriented

```
#include <iostream>
…
int main() {
        Message msg(“Hello World!”);
        msg.print();
        return 0;
}
```

- In procedural style, such as usual C programs, we solve problems using algorithmic steps.
- In OO style, such as good C++ programs, we solve problems by casting them into objects and interactions among them.

Procedural vs. OO

- One can write procedural programs in C++; one can write object-oriented programs in C.
- OO allows us to build a program in application's vocabulary.
  - e.g., student, teacher, lecture, exam, question, …
  - e.g., car, brake, accelerator, wheel, seat, key, …
- Procedural is often top-down (from programs to functions); OO resembles bottom-up design (from classes to programs). But both are iterative.
- Instead of concentrating on tasks, OOP allows us to concentrate on concepts.

Role of C++

- Helps in enforcing data hiding.
  - public, private, protected
- Allows reuse of functionality.
  - inheritance
- Enables change of behavior under different contexts.
  - polymorphism
- Allows creation of generic functionality
  - templates

Let's make tea.

Procedural C++

```
...
int main() {
    Pot pot;
    addWater(pot, 1);
    addTealeaves(pot, 1);
    startBurner();
    boil(pot, 2, false);
    addSugar(pot, 1);
    addMilk(pot, 0.5);
    boil(pot, 2, true);
    stopBurner();
    std::cout << “Tea is ready.\n”;
    return 0;
}
```

Object-oriented C++

```
...
int main() {
    Pot pot;
    Burner burner;
    Water water(1);
    Tealeaves tealeaves(1)
    Sugar sugar(1);
    Milk milk(0.5);

    burner.start(pot);
    pot.add(water);
    pot.add(tealeaves);
    burner.boil(2, false);
    pot.add(sugar);
    pot.add(milk);
    burner.boil(2, true);
    burner.stop();
    std::cout << “Tea is ready.\n”;
    return 0;
}
```

Now let's make coffee.

Procedural C++

```
...
int main() {
        Pot pot;
        addMilk(pot, 1);
        addCoffeepowder(pot, 0.5);
        addSugar(pot, 1);
        addMilk(pot, 0.5);
        startBurner();
        boil(pot, 2, true);
        stopBurner();
        std::cout << “Coffee is ready.\n”;
        return 0;
}
```

Object-oriented C++

```
...
int main() {
        Pot pot;
        Burner burner;
        Water water(1);
        Coffeepowder cpowder(0.5)
        Sugar sugar(1);
        Milk milk(0.5);

        burner.start(pot);
        pot.add(water);
        pot.add(cpowder);
        pot.add(sugar);
        pot.add(milk);
        burner.boil(2, true);
        burner.stop();
        std::cout << “Coffee is ready.\n”;
        return 0;
}
```

Pot interface may not change in OO C++.

Interface

- Behavior visible to the outside world.
- What clients need to know.
- Hides implementation details.
- Allows changing implementation without changing the behavior.
- e.g., an electric switch, strlen function in C, etc.

Interface Drinks:
add(drink)
remove(drink)
heatAll()
freezeAll()

- We need not know how drinks are stored internally (array, vector, heap, …).
- A client can continue to call heatAll even if the internal representation changes.
- Interfaces help in **data hiding**.

Why C++? Why not C?

- If one can write object-oriented programs in C, why design a new language?
- In C, a client may change server interface and inner details; but it need not change.
- In C++, a client cannot change server interface and inner details, unless server allows.
- C++ also supports other useful mechanisms.
  - code reuse with inheritance
  - operator overloading with polymorphism
  - generic programming with templates
  - support for exceptions

Abstraction

- Abstraction simplifies complexity.
  - As a user, we need to know of only switch-on and -off; and not about ground, live and neutral wires.
  - When we drive a two-wheeler, we need not know how the engine operates and about cylinders, valve control and turbo.
  - We know to click gmail send button; we need not know how UDP packets are transmitted.
- Interface defines an abstraction.
- A type is an abstraction for a service.
  - Number of bits, interpretation of bits, operations on bits