Programming technologies

LECTURER: TOLEGENOV AZAMAT MARATOVICH
DEPARTMENT: CE&T (ROOM 409)
EMAIL: AZICUS.IT@GMAIL.COM

Content

- Procedural Programming vs Object-Oriented Programming (advantages and disadvantages)
- OOP principles
- Encapsulation
- Introduction to objects
- Access specifiers
- Member functions
- Struct vs Class
- Information hiding benefits

Procedural Programming

- Procedural programming uses a list of instruction to tell the machine what to do step-by-step.
- Procedural programming relies on procedures, also known as routines or subroutines(sub-problems).

Object-Oriented Programming

- Object-oriented programming, or OOP, is an approach to problem-solving where all computations are carried out using **objects**.
- An object is a component of a program that knows how to perform certain actions and how to interact with other elements of the program.
- Objects are the basic units of object-oriented programming.

Comparing

- OOP or Object Oriented Programming is a good programming practice to create manageable projects more easily.
- Procedural programming means writing code without objects.
- OOP enlightens any language for better coding, for best performance and for writing very big projects without worrying a lot about managing them.
- OOP gives you facilities to create reusable objects that you or other developers can use in their projects without reinventing them again and again.
- OOP removes the hassles and difficulties of writing and managing big applications.

OOP vs POP

Procedure Oriented Programming	Object Oriented Programming
In POP, program is divided into small parts called **functions**.	In OOP, program is divided into parts called **objects**.
In POP, importance is not given to **data** but to functions as well as **sequence** of actions to be done.	In OOP, Importance is given to the data rather than procedures or functions because it works as a **real world**.
POP does not have any access specifier.	OOP has access specifiers named Public, Private, Protected, etc.
To add new data and function in POP is not so easy.	OOP provides an easy way to add new data and function.
Example of POP are : C, VB, FORTRAN, Pascal.	Example of OOP are : C++, JAVA, VB.NET, C#.NET.

OOP in C++

- The main purpose of C++ programming is to add object orientation to the C programming language and classes are the central feature of C++ that supports object-oriented programming and are often called **user-defined types**.
- A class is used to specify **the form of an object** and it combines **data** representation and **methods** for manipulating that data

DRY

- **DRY** (Don’t Repeat Yourself)

- If you feel some “déjà vu” while coding, it does not mean that you need to visit a psychiatrist, you only need to find the same piece of code you made before and place it in separate function, perhaps even in a class.

KISS

- **KISS** (Keep It Simple, Stupid)

- Indeed, why need to complicate things? The code should solve the problem and that’s all. Always know that you will return to this code to find some mistake you made? In a class, each method must solve an elementary task. If not, the code must be divided into two or more methods.

Advantages of OOP

- Reusability
- Refactoring
- Extensibility
- Efficiency

Reusability

- An object is an entity which has bundles of properties and methods and can interact with other objects.
- An object can be sufficient or it may have dependencies over other objects.
- But an object is usually developed to solve a specific set of problems.
- So when other developers suffer from the same set of problems, they can just incorporate your class to their project and use it without affecting their existing workflow.
- It prevents from DRY, which means Don't Repeat Yourself.
- In procedural programming, reusing is possible but complex.

Refactoring

- When you need to refactor your projects, OOP gives you the maximum benefit because all objects are small entities and contain its properties and methods as a part of itself.
- So refactoring is comparatively easier.

Extensibility

- If you need to add features to your project, you can achieve best results from OOP.
- One of the core OOP features is extensibility.
- You can refactor your object to add the feature.
- While doing it, you can still maintain backward compatibility of this object so that it works fine with an old code base.
- Or you can extend the object and create a totally new object that retains all the necessary properties and methods of the parent object from which it has been derived, and then expose new features.
- This is termed "inheritance" and is a very important feature of OOP.

Efficiency

- The concept of object oriented programming is actually developed for better efficiency and ease of development process.
- Several design patterns are developed to create better and efficient code.
- Moreover in OOP, you can think of your solution in a much better approach than procedural programming.
- Because you first split your problem into a small set of problems and then find solutions for each of them, the big problem is solved automatically.

OOP principles (concepts)

- Encapsulation (will cover soon)
- Abstraction
- Inheritance
- Polymorphism

So let’s start, too much theory

- A **class** is a programmer-defined data type that describes what an object of the class will look like when it is created.
- It consists of a set of variables and a set of functions.
- Here is the general format of a class declaration:

```
// Class declaration begins with the key word class and a name.
class ClassName {
        //Declarations for class member variables
        //and member functions go here.
```

- `}; // Notice the required semicolon.`

Encapsulation

- Encapsulation is one of the four fundamental OOP concepts.
- Encapsulation is a mechanism of **wrapping** the data (**variables**) and code acting on the data (**methods**) together as as single unit.
- In encapsulation the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class, therefore it is also known as data hiding.

Some rules

- Object Oriented Programming rules:
  - methods define in **public** access modifier.
  - instance variables define in **private** access modifier.
  - instance variables are accessed through methods (**setter** and **getter** methods).
- By default, all members of a class are private.

Class construction

- Before creating a class:
  - think how to name a class.
  - determine what member variables and member functions the class should have.
- Once the class has been designed, the next step is to write the class declaration.

Circle class as an example

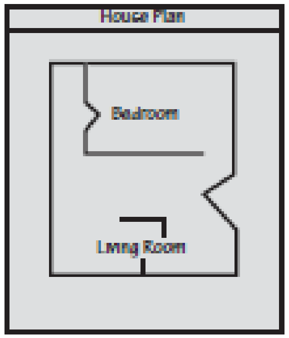
```
class Circle{
   private: //all members after this will be private
         double radius;
   public: //all members after this will be public
         void setRadius(double r){ radius = r; }
         double getRadius(){return radius; }
         double getArea(){
                  return 3.14 * pow(radius, 2);
         }
};
```

Introduction to objects

- A class declaration is similar to the **blueprint** for a house.
- The blueprint itself is not a house, but is a detailed **description** of a house.
- When we use the blueprint to build an actual house, we could say we are constructing an **instance** of the house described by the blueprint.
- If we wish, we can construct several identical houses from the same blueprint.
- Each house is a **separate instance** of the house described by the blueprint.

Classes and objects

Blueprint that describes a house.



Instances of the house described by the blueprint.

Classes and objects

- A class declaration serves a similar purpose. It describes what the objects created from the class will look like when they are constructed.
- Each object created from it is called an ***instance** of the class*.

Defining objects

- Class **objects** are created with simple definition statements, *just like variables*.
- For example, the following statement defines **circle1** and **circle2** to be two objects of the Circle class:

```
Circle circle1, circle2;
```

- Both of the objects **circle1** and **circle2** will have their own copy of data members.

Defining objects

- Also dynamic memory allocation can be used to create an instance of a class.

```
Circle *cptr = new Circle;
```

Access specifiers

- Data hiding is one of the important features of Object Oriented Programming which allows preventing the functions of a program to access directly the internal representation of a class type.
- The access restriction to the class members is specified by the labeled **public**, **private**, and **protected** sections within the class body.
- The keywords public, private, and protected are called **access specifiers**.

The public members

- A **public** member is accessible from anywhere outside the class but within a program.
- You can set and get the value of public variables without any member function.

The private members

- A **private** member variable or function cannot be accessed, or even viewed from outside the class.
- Only the class and friend functions can access private members.
- By default all the members of a class would be private

The protected members

- A **protected** member variable or function is very similar to a private member but it provided one additional benefit that they can be accessed in child classes which are called **derived classes**.
- You will learn derived classes and inheritance in the 5-6th weeks.

Box example

```
class Box {
        double width;
    public:
        double length;
        void setWidth( double wid ){...};
        double getWidth( void ){...};
};
```

- Class member **width** is private.

Member functions

- A member function of a class is a function that has its **definition** or its **prototype within the class** definition like any other variable.
- It operates on any object of the class of which it is a member, and **has access to all the members** of a class for that object.
- Member functions can be defined within the class definition or separately using **scope resolution operator, ::**.
- Defining a member function within the class definition declares the function **inline**, even if you do not use the inline specifier.

Definition of a function(inside a class)

```
class Box {
   public:
         double length; // Length of a box
         double breadth; // Breadth of a box
         double height; // Height of a box
         double getVolume(void) {
                  return length * breadth * height;
         }
};
```

Definition of a function(outside of class)

```
class Box {
   public:
         double length; // Length of a box
         double breadth; // Breadth of a box
         double height; // Height of a box
         double getVolume(void) ;
};

double Box::getVolume(void) {
   return length * breadth * height;
}
```

Struct vs Class

- Let’s create a **class** and a **struct** for date representation and see the difference.
- Also we need to provide a function that show the date in some format.

Definition of class and struct

```
struct DateStruct{
   int month;
   int day;
   int year;
};

class DateClass{
   public:
      int month;
      int day;
      int year;
};
```

Adding a function to the struct

```
struct DateStruct{
    int month;
    int day;
    int year;
};

void print(DateStruct &date){
    std::cout << date.month << "/" << date.day << "/" << date.year;
}

int main(){
    DateStruct today { 10, 14, 2020 }; // use uniform initialization

    today.day = 16; // use member selection operator to select a member of the struct
    print(today);

    return 0;
}
```

Adding a function to the class

```
class DateClass{
      public:
          int month;
          int day;
          int year;

    void print(){
        std::cout << month << "/" << day << "/" << year;
    }
};

int main(){
    DateClass today { 10, 14, 2020 };

    today.day = 16; // use member selection operator to select a member variable of the class
    today.print(); // use member selection operator to select a member function of the class

    return 0;
}
```

Information hiding benefits

- In example with Circle class if radius is public anyone can give negative value. So setter function must check for negative value before assigning it.
- In example with PhoneBook class, let's assume you use an array of Record objects, and you will need size variable also. After giving a size to array, if the size variable is public anyone can change it. With the function you can control it.

End of a lecture 1

Thank you for attention.