EEET2482/COSC2082

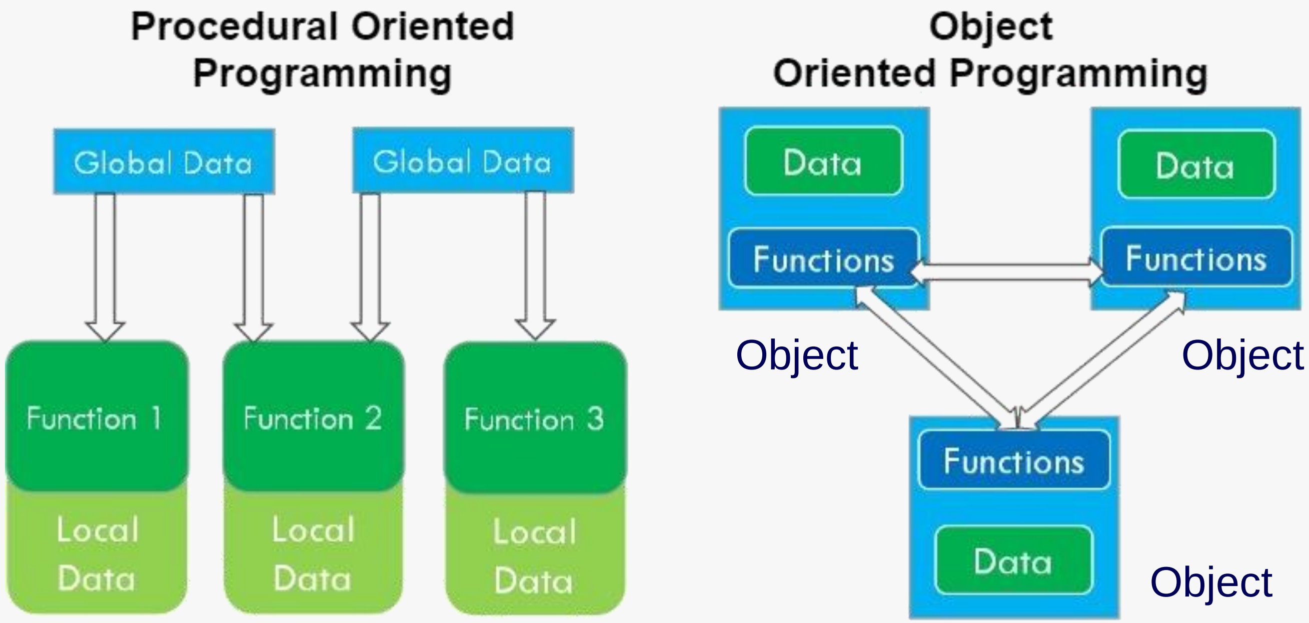
SOFTWARE ENGINEERING DESIGN, ADVANCED PROGRAMMING TECHNIQUES

WEEK 3 – CLASSES AND OBJECTS

LECTURER: LINH TRAN

Overview

- C++ support both **Procedural** and **Object Oriented Programming (OOP)**.
- Besides using *functions*, we can use *classes*, *objects* and other OOP concepts for better *code modularity, flexibility and reuse*.



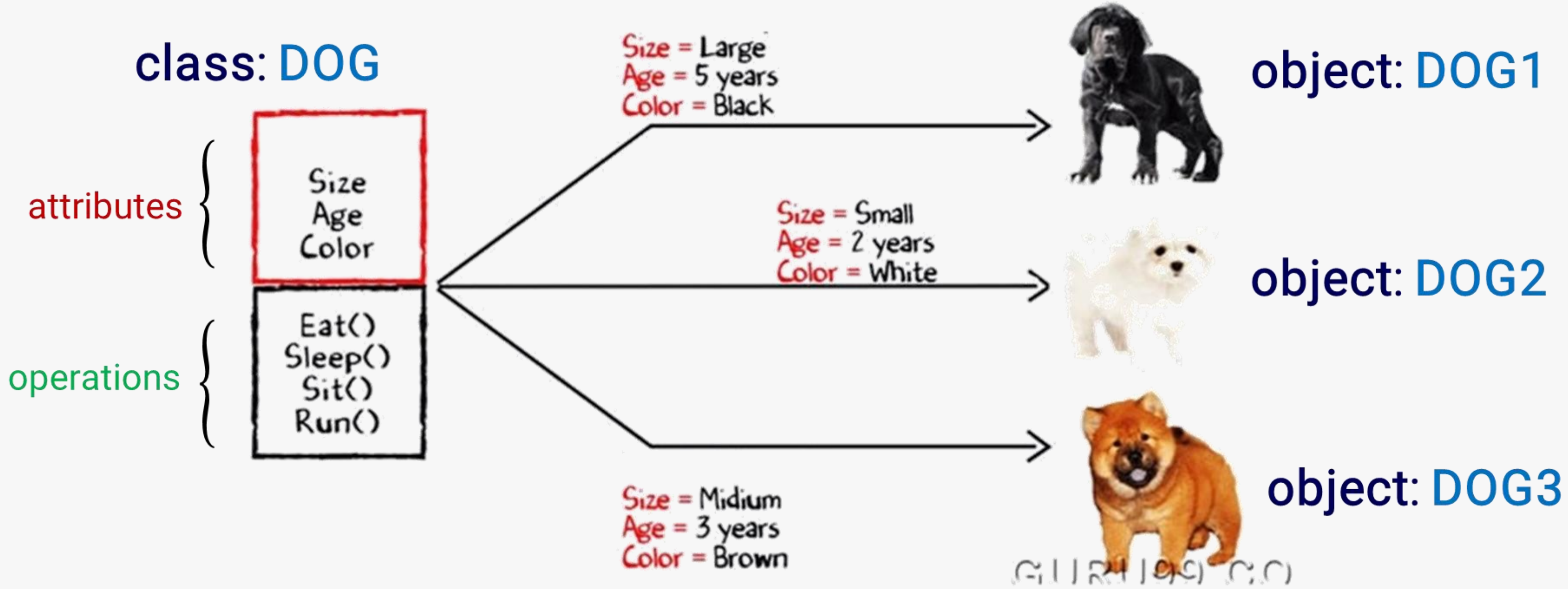
Object

Object

Object

Classes and Objects

- A **class** is general type of entities with **common characteristics** *(**attributes** and **methods/ operations**).*
- An **object** is a single entity (an instance of the class).



Create a Class in C++

Definition Syntax:

```
class ClassName {
    Access_Specifier://can be public, private or protected
        Attributes;  //variables
        Methods;//functions
}; //class definition ends with semicolon
```

- Attributes and methods arebasically **variables** and **functions** that belongs to the class. These are often referred to as "***class members***".
- Public members can be accessed directly via dot (.) operator.

Example

```cpp
#include <iostream>

class Dog {       // The class
  public:             // Access specifier
    int age;        // Attributes (variables)
    bool hungry = true;
    bool happy;

    void eat() {// Method (function)
        hungry = false; happy = true;
    }

    void info() {// Method (function)
        std::cout   << "age = " << age
                    << " hungry = " << hungry
                    << " happy = " << happy << "\n";
    }
};
```

```cpp
int main() {
    Dog Dog1, Dog2;  //Create objects of Dog

    //Access their variables and functions
    Dog1.age = 5;
    Dog1.eat();
    std::cout << "Dog 1's info: ";
    Dog1.info();

    Dog2.age = 10;
    std::cout << "Dog 2's info: ";
    Dog2.info();

    return 0;
}
```

Class Methods

- Two ways to define functions that belongs to a class:
  - **Inside class definition** *(as in the previous slide)*
  - **Outside class definition** (*preceded by **class name** and **scope resolution :: operator***)
- Can have input parameters and return value.

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

class car {
    public:
        int speed = 100; //default value is 100
        int add_speed(int num);
};

int car::add_speed(int num) {
    speed += num;
    return speed;
}

int main() {
    car my_car; // Create an object of Car
    cout << "my_car's new speed: " << my_car.add_speed(200);
    return 0;
}
```

Constructor Method

- Constructor is a **special method** that is *automatically called when an object of the class is created.*
- Must have the same name with the class name.
- Can be defined inside/outside class definition
- Can have parameters, but NO return value.

```
#include <iostream>

class car {        // The class
    public:         // Access specifier
        int speed;      // Attribute
        int price;
        car (int speed_val, int price_val) { // Constructor with parameters
            speed = speed_val;
            price = price_val;
        }
};

int main() {
    // Create Car objects and call the constructor with different values
    car car1(500, 10000);
    car car2(200, 5000);

    // Print values
    std::cout << "Car1: speed & price: " << car1.speed << " "
                                         << car1.price << "\n";
    std::cout << "Car2: speed & price: "    << car2.speed << " "
                                            << car2.price << "\n";
  return 0;
}
```

Access Specifier & Data Encapsulation

- **Access Specifier** defines *how members (attributes and methods) of a class can be accessed.*
  - `public` - members can be from outside the class
  - `private` - members CANNOT be accessed from outside the class
  - `protected` - members CANNOT be accessed from outside the class, however, they can be accessed in inherited classes

- Make class members `private` to **encapsulate and hide data** (*sensitive such as password, game settings, personal info…*)

- May still provide access to private data through `public` **functions** (usually **get/set**)

Example

```cpp
#include <iostream>

#define DEFAULT_VAL 100
#define PASSWORD_KEY 12345
class myclass {
    private: // Private attribute (cannot be accessed directly)
        int num = DEFAULT_VAL;

    public: // Public functions (to manage access)
        int get_num(){ return num; };

        void set_num(int new_val){
            int pwd;
            std::cout << "Enter password: ";  std::cin >> pwd;
            if (pwd == PASSWORD_KEY) {
                num = new_val;
                std::cout << "Set new value successfully ! \n";
            } else {
                std::cerr << "Incorrect password ! \n";
            }
        }
};
```

```cpp
int main() {
    myclass object1;
    std::cout   << "Current num value: "
                << object1.get_num() << "\n";

    object1.set_num(200);
    std::cout   << "New num value: "
                << object1.get_num();

    return 0;
}
```

C++ string class

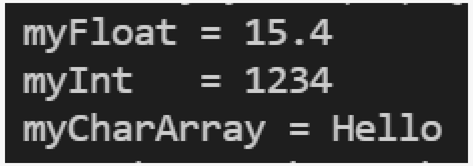
- Besides C-type string (i.e. character array), we can use **C++ string class** to work with string. Note: C++ string is not terminated by ‘\0’
- Some basic functions of string class (include <string> library to use):

Type	Syntax	Example
Constructor	string (const string& str);	std::string str ("Hello World");
Assignment	=	str = "ABC";
Concatenation (appending)	+ +=	str = str + " DE";
Length	length()	std::cout << "length: " << str.length() << "\n";
Element indexing	[ ]	std::cout << "1st char: " << str[0] << "\n";
Comparision	== != < <= > >=	std::cout << ((str == "ABC DE") ? "YES \n" : "NO \n") ;
Get a line into a string	getline(cin, string);	std::getline(std::cin, str); std::cout << str << "\n";
Manipulate string content	substr(), erase(), insert(), replace()	std::cout << "sub str: " << str.substr(1, 3);

StringStream Class

- Another type of IOS stream (similar with cin, cout), but **input/ ouput will be written to/ reading from a string** with >> and << operators**.**
- Can be used conveniently to ***extract/ insert data from/ to a string.***

```
#include <iostream>
#include <sstream>

int main() {
    std::string str = "15.40 1234 Hello World"; // create string
    std::stringstream ss;        // create a stringstream object

    ss << str;              // put the content of string into the stringstream

    //Extract data from the stringstream
    float myFloat;    int   myInt;   char  myCharArray[20];
    ss >> myFloat >> myInt >> myCharArray;

    std::cout << "myFloat = " <<  myFloat << "\n"
        << "myInt    = " <<  myInt << "\n"
        << "myCharArray = " <<  myCharArray << "\n";

    return 0;
}
```



```
myFloat = 15.4
myInt   = 1234
myCharArray = Hello
```

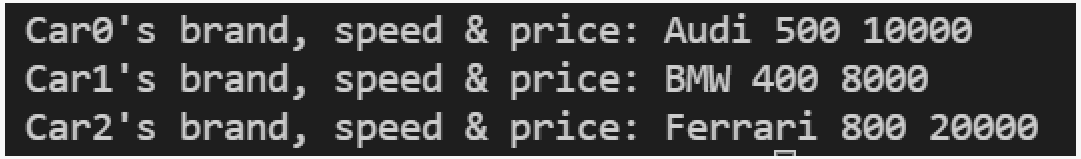
Array of Objects

- An array of objects can be declared in the same manner as normal

```
//Array of 3 Car objects
Car MyCars[3];
```

- Use the class contructor to initialise each element of the array

```
#include <iostream>

class Car {
    public:
        std::string brand;
        int speed, price;

        Car (std::string brand_info, int speed_val, int price_val) {
            brand =  brand_info; speed = speed_val; price = price_val;
        }
};


int main() {
    Car MyCars[3] = { Car("Audi", 500, 10000),
                      Car("BMW", 400, 8000),
                      Car("Ferrari", 800, 20000) };

    for (int i = 0; i < 3; i++) {
        std::cout << "Car" << i << "'s brand, speed & price: "
                  << MyCars[i].brand << " "
                  << MyCars[i].speed << " "
                  << MyCars[i].price << "\n";
    }

  return 0;
}
```



```
Car0's brand, speed & price: Audi 500 10000
Car1's brand, speed & price: BMW 400 8000
Car2's brand, speed & price: Ferrari 800 20000
```

Structures (keyword: struct)

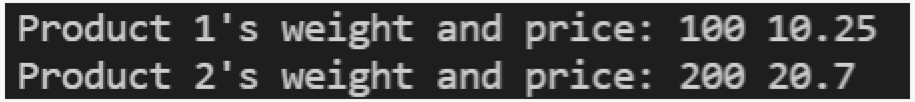
- A structure is a collection of variables of different data types under a single name.

- In C++, **struct** and **class** are equivalent (both defines a class type).

  The only difference is that *by default all members are **public** in a struct, and **private** in a class.*

- Since struct is originated from C, in practice, people **usually only use struct for C-type structures (i.e. with attributes, but no member methods).**

```
#include <iostream>

struct product {
  int weight;
  double price;
};

int main() {
    product pd1, pd2;
    pd1.weight = 100; pd1.price = 10.25;
    pd2.weight = 200; pd2.price = 20.7;

    std::cout << "Product 1's weight and price: "
              << pd1.weight << " " << pd1.price << "\n";
    std::cout << "Product 2's weight and price: "
              << pd2.weight << " " << pd2.price << "\n";
}
```



```
Product 1's weight and price: 100 10.25
Product 2's weight and price: 200 20.7
```

Why OOP?

For large projects, OOP techniques allow a great deal of flexibility.

Specifically, OOP is used for the following reasons:

- **Encapsulation**: we cannot alter the value of the private member data other than by calling a public member function to do it for us. This helps us hiding sensitive data, eliminating potential bugs, and in large programs it makes the code easier to understand.
- **Modularity**: Different programmers or teams can work on different independent classes at the same time.
- **Code-reuse**: This is done by using more advanced techniques such as Inheritance and the fact that a well designed class can be re-used very easily in other programs you write in the future.