# VG101: Introduction to Computer and Programming

## Week12 Checklist

### Welcome to C++

- `C++` has a large improvement in `C++11`, so C++11 is also called modern `C++`, and we will stick to `C++11`. Do remember to add flag `-std=c++11` to indicate you are using `C++11` when compiling.
- compiler: `g++` or `clang++`, similar usage as `gcc` and `clang`
- Features on C++
  - class (object-oriented programming)
  - more library
  - inheritance
  - template
  - rvalue reference
  - more...
- Note: please always remember to get out of comfort zone when learning `C++`. Don't use `C++` as "`C` with class"
  - Don't include `<xxx.h>`. If some library functions in `C` are indeed necessary, use `<cxxx>` instead. Some adjustments are applied to make these libraries more compatible with C++
  - Stay hungry, stay foolish.
  - Reference: https://en.cppreference.com/w/ . It contains almost all the functions and keywords that you may use.

### `namespace`

`namespace` is used to indicate the "space" of functions/data to avoid collision

- `using namespace std`: for all the standard C++ libraries
- `std::cout << "hey";` is equivalent to `using namespace std; cout << "hey";`
- You may use your own namespace when you define your own class

### I/O

- `#include <iostream>`

```
cout << "Hello " << "World!" << endl; // output: Hello World!
int a, b;
char c;
cin >> a >> b >> c;
cout << a << b << c;    // string, char, int. All can be outputed by cout
```

- `cin` and `cout` free you from the workload of specifying the format; it can automatically transfer the format for you. We will come back to explain how it realize such functions after `class` and overloading
- `endl` is similar to `\n`, but it will flush the buffer. More specifically, `std::cout << std::endl;` is equivalent to `std::cout << "\n" << std::flush;`
- There are some other functions and libraries (e.g. `getline`, `fstream`) to implement I/O in different situations.
- Note: Don't mix using `C`-style I/O and `C++`-style I/O! It can result in some problems. We actually almost never use `printf` or `scanf` in `C++` since `cin`, `cout` and other `iostream` are so powerful that we don't miss `printf` and `scanf` at all...

## bool

- In C++, we finally have a data type represent a logical value `true` or `false`
- `cout` outputs a bool value will get 1 or 0

## Overloading

Note that, an operator is essentially a function with syntax sugar. Overloading means defining functions/operators with the same name but different inputs. When calling the functions, the programming will check the input to invoke the corresponding functions

```
void func(int a) { cout << "It is an int!" << endl; }
void func(double a) { cout << "It is a double!" << endl; }
func(3); // It is an int!
func(3.0); // It is a double!
```

Note that, whenever calling functions, it may involve some data type conversions. In the case that there are function overloadings, there may still functions will be matched if data type conversion is applied. In such case, the closest function (less conversion) will be called.

```
// void func(int a) { cout << "It is an int!" << endl; }
void func(double a) { cout << "It is a double!" << endl; }
func(3); // It is a double!
func(3.0); // It is a double!
```

Operator overloading is similar to the function overloading, as long as you view the operator as a function. We will see more examples later.

## `class`

Welcome one of the greatest step of `C++`: `class`!

## Procedural Programming vs. Object-Oriented Programming (OOP)

- Procedural Programming: Everything is considered as a procedure, which means a program consists of several procedures (e.g. a function to implement some tasks). There is no ownship of data.
- Object-Oriented Programming: Everything is considered as an object. Data and tasks all belong to some specific object (ownship)
- Example: Bob kisses Alice
  - Procedural Programming: the core of this sinario is "kiss". It happens to be the case that, the "kiss" involves Bob and Alice this time. Define function as `kiss(Bob, Alice)`
  - Object-Oriented Programming: the core of this sinario is "Bob". Bob is the subject of the sentence, and kiss is just one of behaviors that Bob can do. Define function as `Bob.kiss(Alice)`
- OOP actually has many advantages over Procedural Programming, and it is commonly used in many modern programming language (e.g. Python)
  - Aside: pro and con for high level and low level programming language
- OOP allows many advantages feacture
  - Encapsulation: group data. Data are considered as the property of an object, and the object is responsible for mantaining the data. (e.g. Bob can keep some `int` to indicate `height`, `weight`, etc)
  - Inheritance: if an object `A` is an augmented version of `a`, we could let `A` inherit from `a`. It promotes the code reuse.
  - Polymorphism: If `A1`, `A2`, `A3` are all inherited from `a`, they work like `a` with different extension. If view from `a`, it is called polymorphism.

## `class`: the implmentation of OOP in C++

```
// in .h
class DynamicSizeArray
{
private:
    int* array;
    int capacity;
    int size;
public:
    DynamicSizeArray();
    ~DynamicSizeArray() { delete[] array; }
    int num_element() { return  size; }
    int available_capacity() { return capacity - size; }
    bool is_empty() { return size == 0; }
    void push_back(int x);              // you may take the declaration
also as definition
                                        // or separate it into a .cpp file
```

```
    bool operator==(const DynamicSizeArray& other); // operator
overloading
};

// in .cpp
void DynamicSizeArray::push_back(int x) // Notice that, here we lable the
namespace of push_back
{
    if (size == capacity)
    {
        int * new_array = new int[2*capacity];
        for (int i = 0; i < size; i++)
            new_array[i] = array[i];
        delete[] array;
        // new and delete are C++ version of malloc and free
        array = new_array;
    }
    array[size] = x;
    size++;
}

bool DynamicSizeArray::operator==(const DynamicSizeArray& other)
{
    if (size != other.size)
        return false;
    for (int i = 0; i < size; i++)
    {
        if (array[i] != other.array[i])
            return false;
    }
    return true;
}
```

`class` is similar like `struct` in `C`: they both pack up a series of data; but `class` has different intention: it also packs up functions to maintains these data and even set protection of data from accessing outside the class

- `public`, `private`, and `protected`: one of the most important syntax difference between `class` and `struct`. Default is private
- Note: We tend to put the definition of a class in a header file, and the function implement of class in `.cpp`
- Critical thinking: Why `cin` and `cout` could convert the content read to specific data type automatically?