## **Chapter 19: Function**

C++ code can be broken into smaller chunks called functions. This function will help our code more understandable and maintainable. Imaging you have a software product of 200.000 lines of code. It is impossible to maintain our code or debug a sudden error if they are not organized into chucks of function that do a specific task.

Furthermore, you can even reuse the code in many parts of the program.

Structure of the function in C++:

type function\_name(arguments) {

statement;

statement;

return something;

}

Type can be any c++ type such as: int, string, float, double, etc. Type void is a type that represents nothing, an empty set of values.

There are typically 3 ways of passing parameters:

1. Passing by copy the value:

#include <iostream>

using namespace std;

void printBirthday(int birthday)

{

cout << "Your birthday is: " << birthday;

}int main()

{

printBirthday(18);

}

1. Passing by reference (pointer):

#include <iostream>

using namespace std;

void printBirthday(int& birthday)

{

cout << "Your birthday is: " << birthday;

}int main()

{

int i = 18;

printBirthday(i);

}

1. Passing by reference to a constance:

#include <iostream>

using namespace std;

void printBirthday(const string& Name)

{

cout << "Your Name is: " << Name;

}int main()

{

string Name = "ducthang";

printBirthday(Name);

}

There are certain notes that should be aware when doing function:

1. A function must be declared before the main() function, so that it is available to be used in main func.
2. A function can be defined whether before main() or after, but the declaration step must be done before main func.

## **Chapter 21: Scope and Lifetime**

### **Scope**

There are various kinds of scope in C++. Scope is used to manage local - global identity of a variable. A variable can be clean by GC (for eg: the going 'i' in for loop) after the function contains the variable is end. Or global identity when the variable must be use universally and immediate value must return.

Let's dig in more to this topic.

1. Local Scope: the variable is done using after the function end.

void myfunction()

{

//Here begins the x's scope

int x = 123;

} // and here it ends

1. Block Scope:

int main()

{

int x = 123; // first x' scope begins here

{

int x = 456; // redefinition of x, second x' scope begins here

} // block ends, second x' scope ends here

// the first x resumes here

} // block ends, scope of first x's ends here

### **Lifetime**

The lifetime of an object is the time an object spends in memory. The lifetime is determined by a so-called storage duration. There are some kinds of duration:

1. Automatic Storage Duration: start at the beginning of the block; end at the end of the block.
2. Dynamic Storage Duration (Heap Memory): a manually set/ allocated/ deallocated in the memory.
3. Static Storage Duration: start at the beginning of the program; end at the end of the program.

### **New and Delete**

Operators new and delete: "new" allocates new space of memory for an object and "delete" removes it. This only perform on the free memory, or the "heap memory" of the computer.

#include <iostream>

using namespace std;

int main()

{

int\* p = new int;

\*p = 123;

cout << "The pointed-to value is: " << \*p;

delete p;

}

## **Chapter 23: Class**

### **Introduction**

Class is a user-defined type. A class consists of members. The members are data members and member functions. Just image like this: you have type int, and certain functions that goes with the type such as Atoi. The Class type you just created can alos be a type, and the function goes with it need to be related to the identity you give it.

Inside the class can be certain declaration of its members:

#include <iostream>

class University

{

int NumberOfClass;

int NumberOfStudent;

string SchoolName;

};

In this class called "University", there are some properties such as the number of class, the number of student, and the name of the university. Each one has its own type logically.

For function of a class, please look at the following example:

#include <iostream>

using namespace std;

class University

{

void PrintName() {

cout << "School Name is VGU" << endln;

}

};

To use the class, you can do like this:

#include <iostream>

using namespace std;

class University

{

public:

void PrintName() {

cout << "School Name is VGU" << endln;

}

};

int main() {

University university; // declare the variable according the to type here.

university.PrintName();

return 0;

}

### **Class Access Specifier**

You can set what can be access in the class by set the tag for section.

#include <iostream>

class MyClass

{

public:

// everything in here

// has public access level

protected:

// everything in here

// has protected access level

private:

// everything in here

// has private access level

};

### **Constructor and Default Construcutor**

A constructor is a member function that has the same name as the class. To initialize an object of a class, we use constructors. This help us the pass all the nessesary elemtents to the class.

Default Constructor is the constructor without any parameters. Example given:

#include <iostream>

using namespace std;

class University

{

void Unversity() {

cout << "School Name is VGU" << endln;

}

};

int main() {

University university;

}

output:

School Name is VGU

Example of constructor require parameters:

#include <iostream>

using namespace std;

class University

{

private:

string SchoolName;

int NumberOfStudent;

public:

void Unversity(int Student, string Name) {

this->SchoolName = Name;

this->NumberOfStudent = Student;

cout << "Init ok!" << endl;

}

};

int main() {

University university(12, "VGU");

}

output:

Init ok!

Member Initialization: when we intialize an object with existing object of the same class, we invoke a copy instructor:

University obj1(1,2);

University obj2 = obj1; //create a obj2 w/o constructor but by using a copy constructor from obj1

Destructor is a member function that gets invoked when an object is destroyed, usually when an object goes out of its scope.

#include <iostream>

class MyClass

{

public:

MyClass() {} // constructor

~MyClass() {} // destructor

};