Lecture #03

Function

SE271 Object-Oriented Programming (2020)

Yeseong Kim

Original slides from Prof. Shin at DGIST

Short Notice

- The first homework is coming …
  - Will be released after the Wednesday class (highly possibly during this Friday)

- Let's clarify your questions first

Operator Precedence

- (L->R) Type parentheses, post-increment/decrement, Type cast
- (R->L) sign, c-style cast, pre-increment/decrement
- (L->R) *, /, %
- (L->R) +, -
- (L->R) <<, >>
- (L->R) <, <=, >, >=
- (L->R) ==, !=
- (L->R) bit operators(&)
- …
- (L->R) logical operators(&&)
- (R->L) assignment

Use ( ) if necessary!

```
int x = 1, y = 2;
cout << (++x > 2 && ++y > 2);
cout << "x=" << x << ", y=" << y << endl;
cout << (++x > 2 || ++y > 2);
cout << "x=" << x << ", y=" << y << endl;
```

https://www.learncpp.com/cpp-tutorial/operator-precedence-and-associativity/

Today's Topic

- Function
  - Declaration (prototype)
  - Definition
    - Parameters – call by value
    - Return (void)
  - Inline function
- Variable scope
- Global, local variables, static variable
- Header file
- Preprocess
- Namespace

Example: Function

```
#include <iostream>
using namespace std;
string order(int menu) {
    if (menu == 1) return static_cast<string>("sold out");
    else return static_cast<string>("successfully ordered");
}
int main(){
    int iMenu{ 1 };
    cout << "1. Coffee\n2. Juice\n3. Quit\n";
    do {
        cout << "Select Menu? ";
        cin >> iMenu;
        cout << order(iMenu) << endl;
    } while (iMenu != 3);
     return 0;
}
```

```
1. Coffee
2. Juice
3. Quit
Select Menu?  1
sold out
Select Menu?
```

Function

- What is function in C/C++?
  - A reusable sequence of statement(s) designed to a particular job
- Why define your own function?
  - Readability: sqrt(5) is clearer than copy-pasting in an algorithm to compute the square root
  - Maintainability: To change the algorithm, just change the function (vs changing it everywhere you ever used it)
  - Code reuse: Lets other people use algorithms you've implemented
- main() is called (or invoked) after initialization of non-local objects, i.e., the entry point of program execution

Example: Function

```
#include <iostream>
using namespace std;
int main() {
    int iVal(0);    double dVal{0};
    cout << "Enter the radius? ";
    cin >> iVal;
    dVal = iVal * 2 * 3.14;
    cout << dVal << endl;
    dVal = iVal * iVal * 3.14;
    cout << dVal << endl;
    cout << "Enter the radius? ";
    cin >> iVal;
    dVal = iVal * 2 * 3.14;
    cout << dVal << endl;
    dVal = iVal * iVal * 3.14;
    cout << dVal << endl;
    return 0; }
```

```
cout << "Enter the radius? ";
cin >> iVal;
dVal = iVal * 2 * 3.14;
cout << dVal << endl;
dVal = iVal * iVal * 3.14;
cout << dVal << endl;
```

```
cout << "Enter the radius? ";
cin >> iVal;
```

```
dVal = iVal * 2 * 3.14;
cout << dVal << endl;
```

```
dVal = iVal * iVal * 3.14;
cout << dVal << endl;
```

Function Declaration and Definition

- Function Definition

```
① return_type  ② function_name (③ parameters )
{
    ④ function body
    [ return return_value ] // optional
}
```

```
double calArea ( int radius )
{
    double dVal;
    dVal = radius * radius * 3.14;
    return dVal;
}
```

③ default parameters

Example: Function

```
#include <iostream>
using namespace std;
int main() {
    int iVal(0);    double dVal{0};
    cout << "Enter the radius? ";
    cin >> iVal;
    dVal = calArea( iVal );
    cout << dVal << endl;
    return 0;
}
double calArea ( int radius )
{
    double dVal;
    dVal = radius * radius * 3.14;
    return dVal;
}
```

```
#include <iostream>
using namespace std;
double calArea ( int radius )
{
    double dVal;
    dVal = radius * radius * 3.14;
    return dVal;
}
int main() {
    int iVal(0);    double dVal{0};
    cout << "Enter the radius? ";
    cin >> iVal;
    dVal = calArea( iVal );
    cout << dVal << endl;
    return 0;
}
```

Function Declaration and Definition

- Function Declaration (prototype)

```
① return type  ② function name ( ③ parameters );
```

```
double calArea ( int radius );
```

```
double calArea ( int radius )
{
    double dVal;
    dVal = radius * radius * 3.14;
    return dVal;
}
```

Example: Function

```
#include <iostream>

// function declaration
int Multiply ( int , int  = 1);

int main() {
    std::cout << Multiply (10) ;
    std::cout << Multiply (10, 20) ;
    return 0;
}

// function definition
int Multiply ( int  iNum1, int  iNum2) {
    return iNum1 * iNum2;
}
```

Q. Is Stack OK? (memory size, copying time)

```
#include <iostream>
long calFact ( int = 0);
int main() {
    int iVal{ 0 }; long dVal{0};
    std::cout << "Enter the number? ";
    std::cin >> iVal;

    dVal = calFact( iVal );
<< std::endl;
    return 0;
}
long calFact ( int num ) {
    if (num == 0) return 1;
    else return num* calFact ( num-1 );
}
```

Solution 1 – Macro function

```
① Macro Constant
#define  PI  3.14

② Macro Function
#define NAME(Parameter) Replacement
```

```
#define  Multiply(x, y)  x*y

int a = Multiply(3, 2);
```

Note 1. macro constant vs. function

Note 2. simple replacement

Solution 2 – inline function

```
#include <iostream>

// function declaration
inline int Multiply ( int , int  = 1);

int main() {
    std::cout << Multiply (10) ;
    std::cout << Multiply (10, 20) ;
    return 0;
}

// function definition
int Multiply ( int  iNum1, int  iNum2) {
    return iNum1 * iNum2;
}
```

```
#include <iostream>

int main() {
    std::cout << 10 * 1 ;
    std::cout << 10 * 20 ;
    return 0;
}
```

Local VS Global Variable, Static duration variable

- Local Variable: variables defined inside the function body
  - Lifetime: until the end of the set of curly braces(**function/block { })**
  - Scope: until the end of the set of curly braces(**function/block { })**
- Global variable: variables defined outside the function body
  - Lifetime: until the end of the program
  - Scope: until the end of the file

- Static variable (keyword: static)
  - Lifetime: until the end of the program
  - Scope: until the end of the set of curly braces

Local VS Global Variable, Static Variable

```
#include <iostream>
double g_count=0;       // global variable

void counter() {
    static int iCount{ 0 }; // static variable
    iCount++;
    g_count = iCount;
    std::cout << iCount << std::endl;
}

int main() {
    counter();
    counter();
    counter();
    std::cout  << g_count << std::endl;
    std::cout  << iCount << std::endl; // error
    return 0;
}
```

```
1
2
3
3
```

Header File

- .h file VS. .cpp file

```cpp
#include <iostream>

// function declaration
int Multiply ( int , int  = 1);

int main() {
    std::cout << Multiply (10);
    std::cout << Multiply (10, 20) ;
    return 0;
}

// function definition
int Multiply ( int  iNum1, int  iNum2) {
    return iNum1 * iNum2;
}
```

File: multi.h

```cpp
// function declaration
int Multiply ( int , int  = 1);
```

multi.obj

File: multi.cpp

```cpp
#include "multi.h"
// function definition
int Multiply ( int  iNum1, int  iNum2) {
    return iNum1 * iNum2;
}
```

File: main.cpp

```cpp
#include <iostream>
#include "multi.h"
int main() {
    std::cout << Multiply (10) ;
    std::cout << Multiply (10, 20) ;
    return 0;
}
```

main.obj

Header File

File: multi.h

```
// function declaration
int Multiply ( int , int  = 1);
```

File: multi.cpp

```
#include “multi.h”
// function definition
int Multiply ( int  iNum1, int  iNum2) {
    return iNum1 * iNum2;
}
```

File: main.cpp

```
#include <iostream>
#include “multi.cpp”
int main() {
    std::cout << Multiply (10) ;
    std::cout << Multiply (10, 20) ;
    return 0;
}
```

1. Reduce compiling time
2. Provide library without source file
3. Avoid duplication of definitions

Preprocess

- #include
- #define
  - Constant
  - Macro function
- #ifndef DEFINE
  #endif

  …

File: multi.h

```
#ifndef MULTI_H
#define MULTI_H

int Multiply ( int , int  = 1);

#endif
```

File: multi.cpp

```
#include "multi.h"
// function definition
int Multiply ( int  iNum1, int  iNum2) {
    return iNum1 * iNum2;
}
```

File: main.cpp

```
#include <iostream>
#include "multi.h"
int main() {
    std::cout << Multiply (10) ;
    std::cout << Multiply (10, 20) ;
    return 0; }
```

* Note: can use "#pragma once" for the same purpose

Namespace

- namespace NAME
  {
  functions
  variables
  }

- Usage
  - NAME::functions
  - NAME::variables

- using keyword
  - using namespace NAME

```
namespace CIRCLE
{
    double PI = 3.14;
    double calArea(int radius) {
        double dVal = radius * radius * PI;
        return dVal;
    }
    double calCircumference(int radius) {
        double dVal = 2 * radius * PI;
        return dVal;
    }
}
int main() {
    cout << CIRCLE::PI << endl;
    cout << CIRCLE::calArea(1) << endl;

    return 0;
}
```

SHAPE::CIRCLE::PI

References

- Learn c++
  - https://www.learncpp.com/
  - Chapter 21.~2.4, 2.11~2.12
  - S.4.2, S.4.3
  - 7.1~7.5

Example: swap operation?

Q2. More than 2 return values?

```
#include <iostream>
void intSwap1(int num1, int num2) {
    int temp{num1};
    num1 = num2;
    num2 = temp; }
void intSwap2(int* num1, int* num2) {
    int temp{*num1};
    *num1 = *num2;
    *num2 = temp;}
int main(){
    int iNum1{ 1 };
    int iNum2{ 3 };
    std::cout << iNum1 << " " << iNum2 << std::endl;
    intSwap1(iNum1, iNum2);
    std::cout << iNum1 << " " << iNum2 << std::endl;
    intSwap2(&iNum1, &iNum2);
    std::cout << iNum1 << " " << iNum2 << std::endl;

     return 0;
}
```

← Call by value

← Call by reference

```
1 3
1 3
3 1
```

ANY QUESTIONS?