Lecture 2

C++ Basics II

Defining Functions

Seoul National University
Graphics & Media Lab

Contents

- Function (7.1-7.6)
- Recursive function (7.3.3)

Introduction to Function

- How to define a simple function ?

```
#include <iostream>

void main() {
   const int height = 3, width = 5, length = 7;
   std::cout << “Volume is " << height*width*length;
}
```

```
#include <iostream>

int volume(int h, int w, int l) { return h*w*l; }

void main() {
   const int height = 3, width = 5, length = 7;
   std::cout << “Volume is " << volume(height,width,length);
}
```

Function, 집중 해부

The data type of the returned value

The name of the function

The names of the parameters along with their data types

```
int volume(int h, int w, int l) {
    return h*w*l;
}
```

The expression whose value is to be returned

Indicates that a value is to be returned by the function

Function, a Means for Procedure Abstraction

- The function is a means for **procedure abstraction**.

```
#include <iostream>

int volume(int h, int w, int l) { return h*w*l; }
int area(int h, int w, int l) { return 2*(h*w + w*l + l*h); }

void main() {
   std::cout << "The volume of box1 is “<< volume(3,5,7) << std::endl;
   std::cout << "The area of box1 is " << area(3,5,7) << std::endl;
   std::cout << "The area of box2 is " << area(10,20,30) << std::endl;
   std::cout << "The area of box3 is " << area(15,25,35) << std::endl;
}
```

Default Arguments of a Function

- Default arguments
  - Default arguments allow some arguments to be omitted.
  - Use default argument values which are expected to be used most of the time.

```
#include <iostream>

int get_area(int h, int w=10, int l=10) { return 2*(h*w + w*l + l*h); }

void main() {
        std::cout << "The area is " << get_area(3) << std::endl;
        std::cout << "The area is " << get_area(3,11,12) << std::endl;
}

// get_area(3) ≡ get_area(3,10,10)
// get_area(3, 5) ≡ get_area(3,5,10)
```

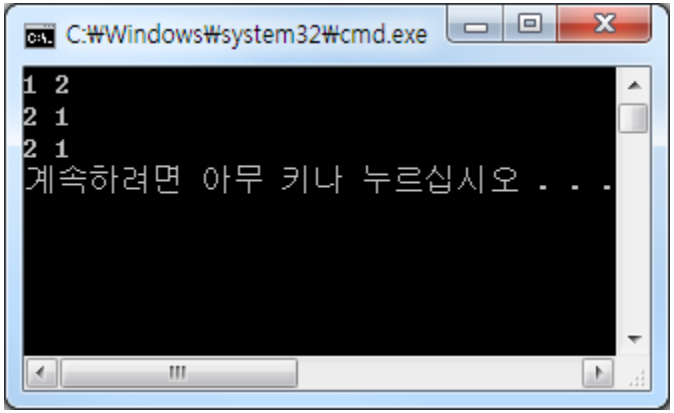
Argument Passing Mechanisms

- Call by value, call by reference, call by pointer

```
#include <iostream>

void swap_using_value(int a, int b) { int tmp = a; a = b; b = tmp;     }
void swap_using_ref(int& a, int& b) { int tmp = a; a = b; b = tmp;     }
void swap_using_ptr(int* a, int* b) { int tmp = *a; *a = *b; *b = tmp; }

void main() {
   int u,v;
   u = 1, v = 2; swap_using_value(u,v); std::cout << u << " " << v << "\n";
   u = 1, v = 2; swap_using_ref(u,v); std::cout << u << " " << v << "\n";
   u = 1, v = 2; swap_using_ptr(&u,&v); std::cout << u << " " << v << "\n";
}
```



Argument Passing Mechanisms

- Call by value, call by reference, call by pointer

```
#include <iostream>

void swap_using_value(int a, int b) { int tmp = a; a = b; b = tmp;     }
void swap_using_ref(int& a, int& b) { int tmp = a; a = b; b = tmp;     }
void swap_using_ptr(int* a, int* b) { int tmp = *a; *a = *b; *b = tmp; }
void swap_using_const_ref(const int& a, const int& b) {int tmp=a; a=b; b=tmp;}
void main() {
   int u,v;
   u = 1, v = 2; swap_using_value(u,v); std::cout << u << " " << v << "\n";
   u = 1, v = 2; swap_using_ref(u,v); std::cout << u << " " << v << "\n";
   u = 1, v = 2; swap_using_ptr(&u,&v); std::cout << u << " " << v << "\n";
   u = 1, v = 2; swap_using_const_ref(u,v); std::cout << u << " " << v << "\n";
}
```

Will produce a compile error

Inline Functions

- Inline functions
  - An **inline** function is expanded "in line" at each function call.
  - So there is no run-time overhead associated with the function call.

```
#include <iostream>

inline int get_area(int h, int w, int l) {
   return 2*(h*w + w*l + l*h);
}

void main() {
   int h0=3, w0=5, l0=7;
   int area = get_area(h0,w0,l0);
}
```

```
// this line would be expanded during
// compilation into something like

int area = 2*(h0*w0 + w0*l0 + l0*h0);
```

Recursive Functions

- A recursive function is a function which calls itself, either directly or indirectly.

```
#include <iostream>

int f(int n) {
   if(n==0 || n==1)
      return 1;
   else
      return f(n-1) + f(n-2);
}

void main() {
   std::cout << f(3) << std::endl;
   std::cout << f(10) << std::endl;
}
```

Recursive Functions