Introduction to Programming (2)

C++ Basics - 2

Laihyuk Park (lhpark@seoultech.ac.kr)

Keyboard input and screen output

- C style

```
#pragma warning (disable: 4996)

#include <stdio.h>

void main() {
    char c0 = 'a';
    int i0 = 3, x;
    float f0 = 3.141592f;

    printf("%c %d %d %f₩n", c0, c0, i0, f0);
    scanf("%d", &x);
    printf("%d₩n", x);
}
```

ASCII Code Table

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

Keyboard input and screen output

- C++ Style

```
#include <iostream >
void main() {
    char c0 = 'a';
    int i0 = 3, x;
    float f0 = 3.141592f;
    std::cout << c0 << " " << (int) c0 << " " << i0 << " " << f0 << std::endl;
    std::cin >> x;
    std::cout << x << std::endl;
}
```

Typical Input Loop

- A typical way of getting user-inputs with a loop structure

```
#include <iostream>

void main() {
    int ival, sum = 0;

    while (std::cin >> ival, !std::cin.eof()) {
        // do something with ival…,
        // e.g., sum += ival;
    }
    std::cout << "Sum : " << sum << std::endl;
}
```

File Input and Output

- ifstream stands for ‘input file stream’.
- ofstream stands for ‘output file stream’.
- fstream can used as either an input or an output file stream.

```
#include <iostream>
#include <fstream>

void main() {
    std::ifstream fs_1("a.txt");
    std::ofstream fs_2("b.txt");
    std::fstream out_fs("test.txt", std::fstream::out);
    int i;

    fs_1 >> i;
    fs_2 << "Programming Methdology" << std::endl;
    out_fs << "is easy";
    if (fs_1.is_open())
        std::cout << i << std::endl;
    else
        std::cout << "File is not found" << std::endl;
    fs_1.close(); fs_2.close(); out_fs.close();
}
```

Header file handling

- Header file is the feature which allows programmers to reuse certain portion of the source code.
- Encountering #include, the preprocessor of C++ compiler inserts the source of the header file at that location.

box.h

```
#include <iostream>

class Box {
public:
	void print() { std::cout << height << " " << width << " " << length; }
	double height, width, length;
};
```

main.cpp

```
#include "box.h"


int main()
{
	Box box;
	box.height = 3; box.width = 5; box.length = 7;
	box.print();

	return 0;
}
```

Header file handling

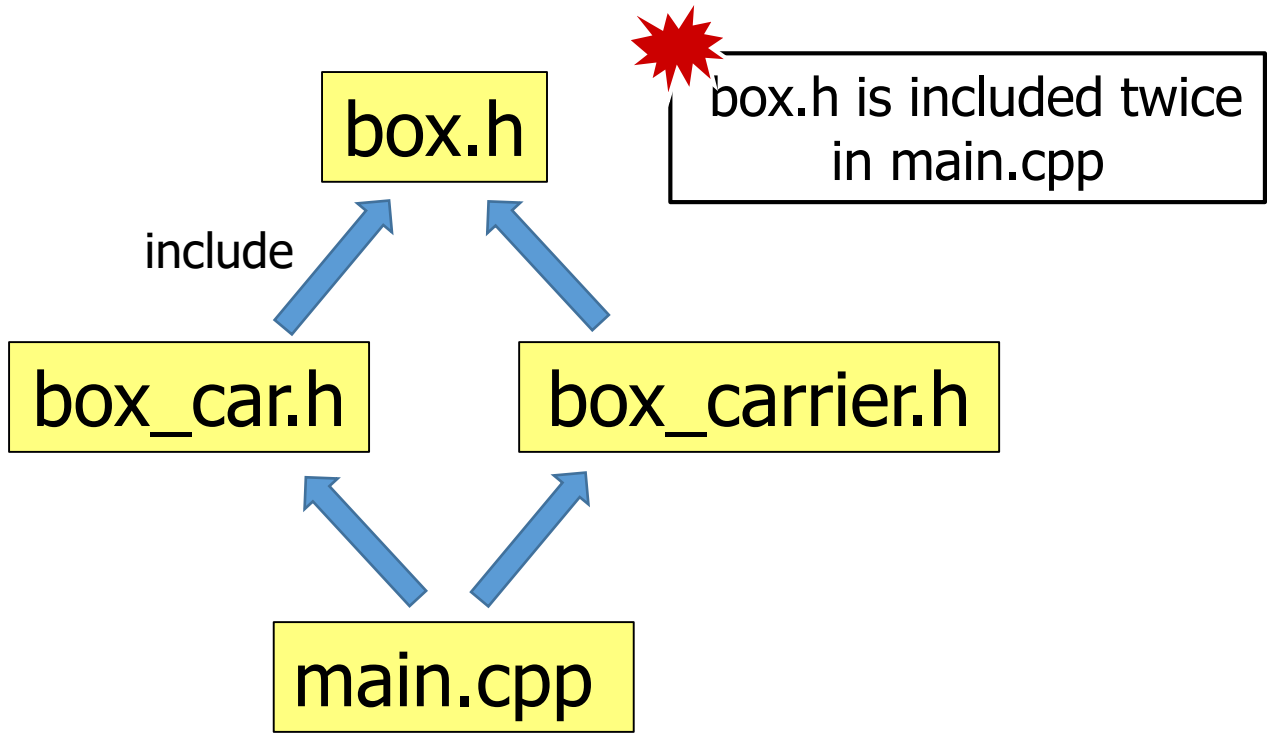
- Headers are for **declarations**, not definitions.
  - Because headers are included in multiple source files, they should not contain definitions of variables or functions.

box.h

```
#include <iostream>

class Box {
public:
    void print() { std::cout << height << " " << width << " " << length; }
    double height, width, length;
};

Box box;// definition (x)
int integer0;// definition (x)
```

Header file handling

- Including a header file more than once causes **multiple definitions** of the classes and objects that the header file defines.
  - It causes compilation errors.



Header file handling

- Including a header file more than once causes **multiple definitions** of the classes and objects that the header file defines.
  - It causes compilation errors.
  - We can solve the problem using #ifndef

```
#ifndef _BOX_CAR_
#define _BOX_CAR_

#include <iostream>

class Box {
public:
    void print() { std::cout << height << " " << width << " " << length; }
    double height, width, length;
};


#endif // !_BOX_CAR_
```

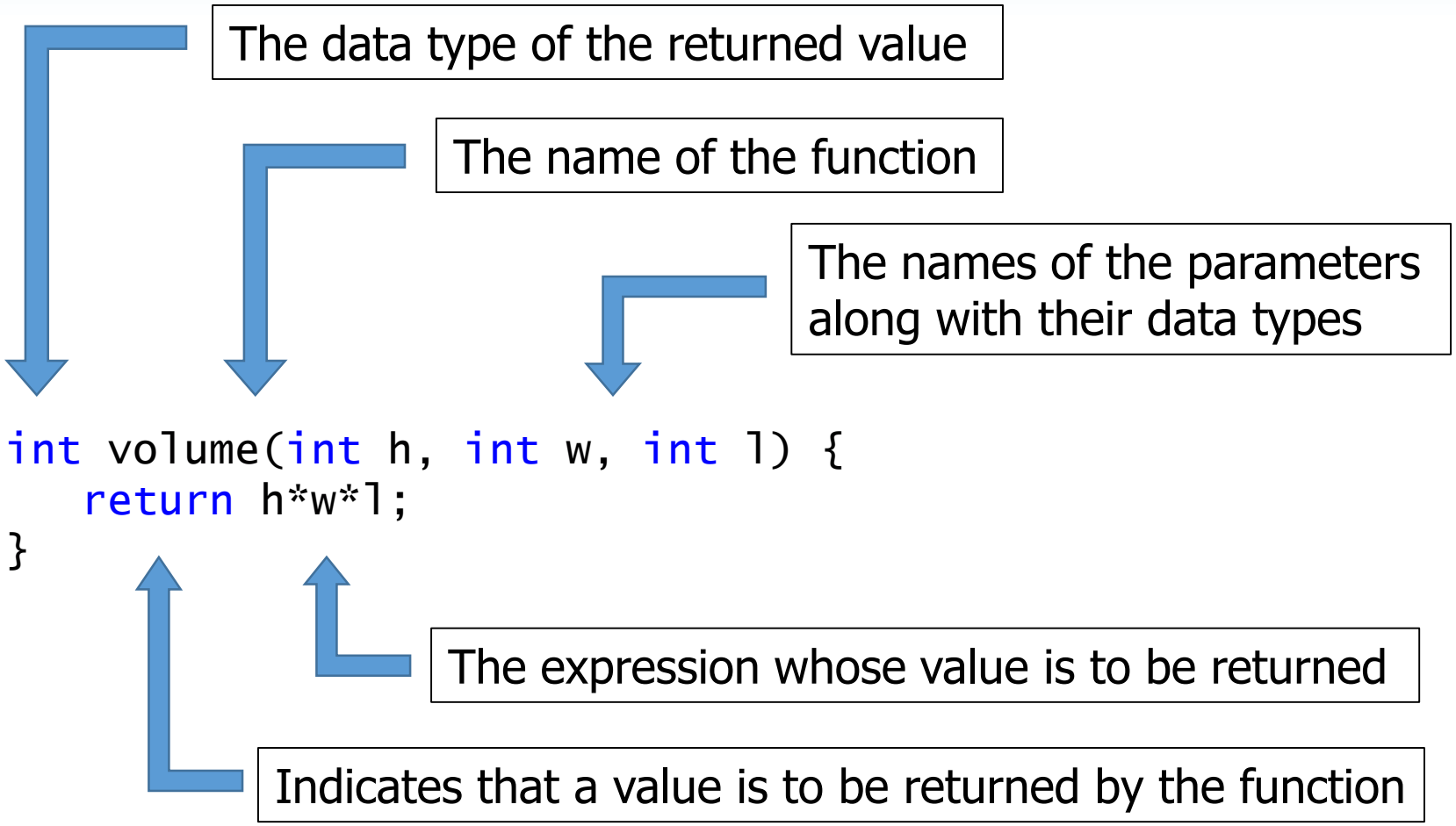
Function

- 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 << std::endl;
}
```

```
#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) << std::endl;
}
```

Function


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

- 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;
}
```

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)
```

Function

- 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 function

- 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 function