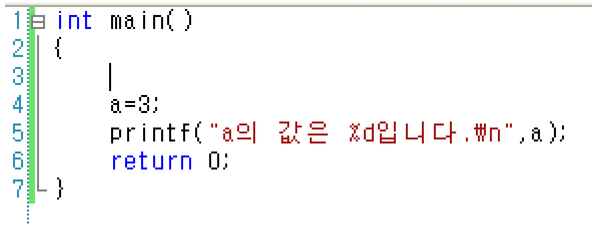
5

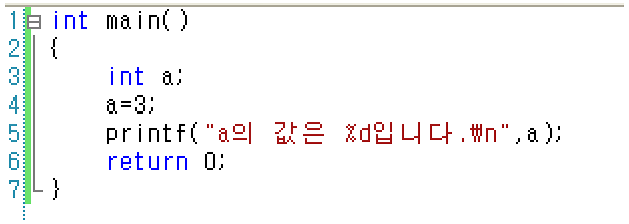
C Functions

함수 선언의 필요성



```
int main()
{
    |
    a=3;
    printf("a의 값은 %d입니다.₩n",a);
    return 0;
}
```

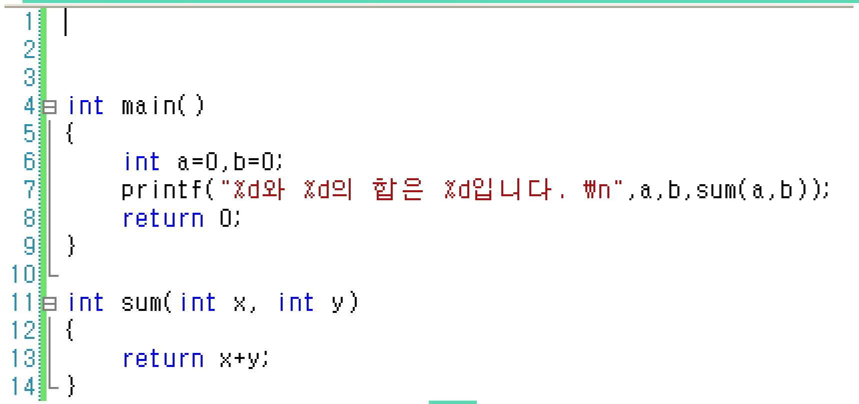
Error!
변수 a의 타입을
컴파일러가 알 수 없다!



```
int main()
{
    int a;
    a=3;
    printf("a의 값은 %d입니다.₩n",a);
    return 0;
}
```

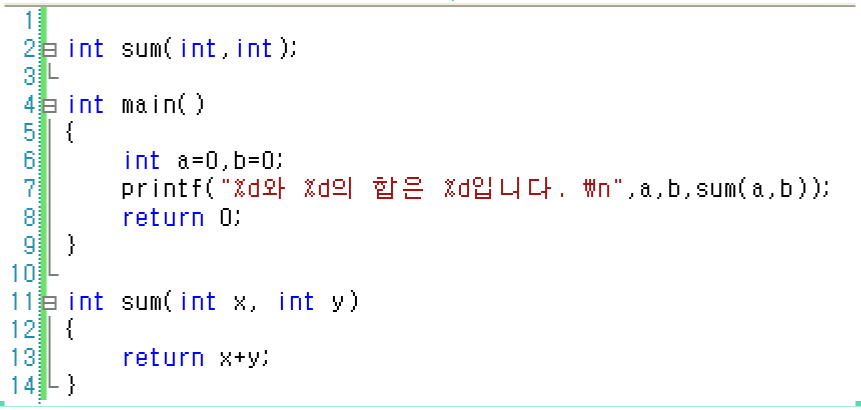
변수의 선언을 해 주어야 함

함수 선언의 필요성



```
 
 
 
int main()
{
	int a=0,b=0;
	printf("%d와 %d의 합은 %d입니다. ₩n",a,b,sum(a,b));
	return 0;
}

int sum(int x, int y)
{
	return x+y;
}
```

컴파일러는
함수 sum에 대해 알지 못함!



```
 
int sum(int,int);

int main()
{
	int a=0,b=0;
	printf("%d와 %d의 합은 %d입니다. ₩n",a,b,sum(a,b));
	return 0;
}

int sum(int x, int y)
{
	return x+y;
}
```

프로그래머는
컴파일러에게 함수 sum에
대해 알려 주어야 한다

Outline

```
/* Fig. 5.4: fig05_04.c
   Finding the maximum of three integers */
#include <stdio.h>

int maximum( int x, int y, int z ); /* function prototype */

/* function main begins program execution */
int main( void )
{
   int number1; /* first integer */
   int number2; /* second integer */
   int number3; /* third integer */

   printf( "Enter three integers: " );
   scanf( "%d%d%d", &number1, &number2, &number3 );

   /* number1, number2 and number3 are arguments
      to the maximum function call */
   printf( "Maximum is: %d\n", maximum( number1, number2, number3 ) );

   return 0; /* indicates successful termination */

} /* end main */

```

Function prototype

Function call

fig05_04.c

(1 of 2)

Outline

fig05_04.c

(2 of 2)

```
/* Function maximum definition */
/* x, y and z are parameters */
int maximum( int x, int y, int z )
{
   int max = x;       /* assume x is largest */

   if ( y > max ) { /* if y is larger than max, assign y to max */
      max = y;
   } /* end if */

   if ( z > max ) { /* if z is larger than max, assign z to max */
      max = z;
   } /* end if */

   return max;        /* max is largest value */

} /* end function maximum */
```

Function definition

```
Enter three integers: 22 85 17
Maximum is: 85
```

```
Enter three integers: 85 22 17
Maximum is: 85
```

```
Enter three integers: 22 17 85
Maximum is: 85
```

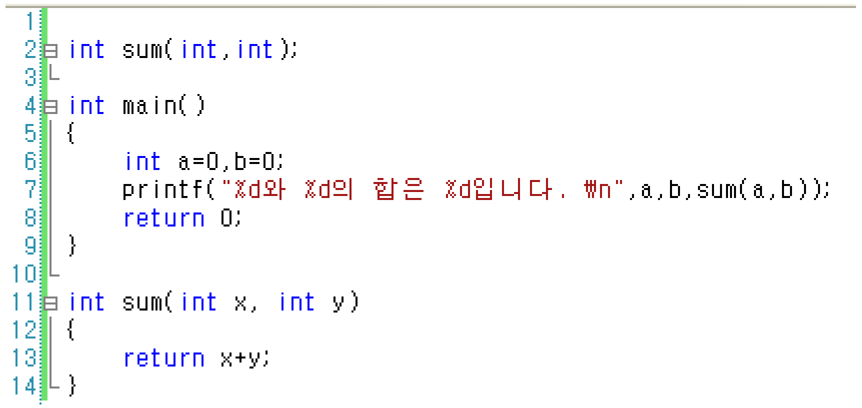
5.6 Function Prototypes

- Function prototype
  - Function name
  - Parameters – what the function takes in
  - Return type – data type function returns (default `int`)
  - Used to validate functions
  - Prototype only needed if function definition comes after use in program
  - The function with the prototype

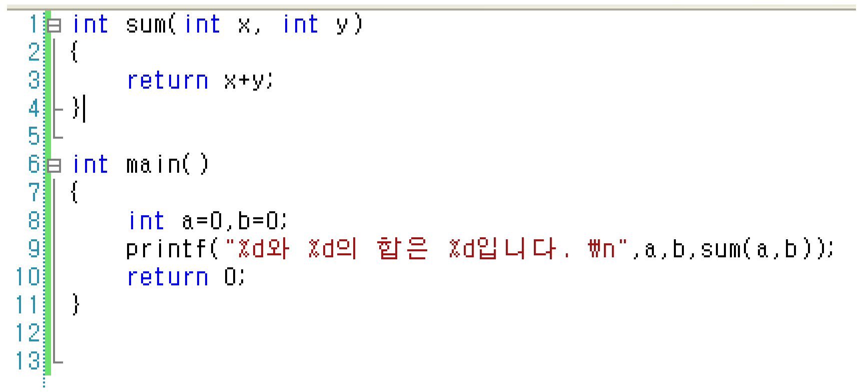
    ```
    int maximum( int x, int y, int z );
    ```

    - Takes in 3 `int`s
    - Returns an `int`
- Promotion rules and conversions
  - Converting to lower types can lead to errors

Prototype 필요 없는 경우



```
int sum(int,int);

int main()
{
	int a=0,b=0;
	printf("%d와 %d의 합은 %d입니다. ₩n",a,b,sum(a,b));
	return 0;
}

int sum(int x, int y)
{
	return x+y;
}
```



```
int sum(int x, int y)
{
	return x+y;
}

int main()
{
	int a=0,b=0;
	printf("%d와 %d의 합은 %d입니다. ₩n",a,b,sum(a,b));
	return 0;
}
```

Main에서 sum을 call하기 전에 Sum이 정의되어 있으므로, 컴파일러는 sum에 대해 미리 알고 있다.

이 경우 따로, prototype이 필요하지는 않다!

Prototype을 써야 되는 이유

Good Programming Practice 5.7

Include function prototypes for all functions to take advantage of C’s type-checking capabilities. Use `#include` preprocessor directives to obtain function prototypes for the standard library functions from the headers for the appropriate libraries, or to obtain headers containing function prototypes for functions developed by you and/or your group members.

Good Programming Practice 5.8

Parameter names are sometimes included in function prototypes (our preference) for documentation purposes. The compiler ignores these names.

Common Programming Error 5.8

Forgetting the semicolon at the end of a function prototype is a syntax error.

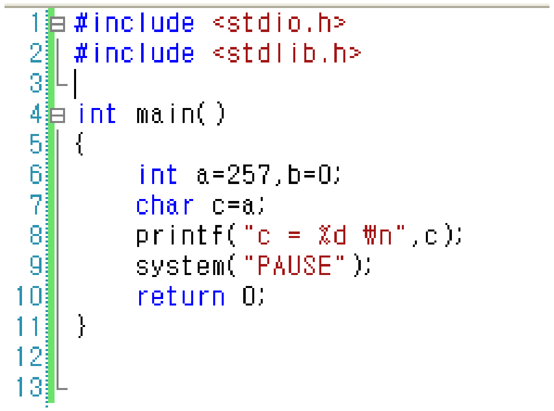
Data type	printf conversion specification	scanf conversion specification
`Long double`	`%Lf`	`%Lf`
`double`	`%f`	`%lf`
`float`	`%f`	`%f`
`Unsigned long int`	`%lu`	`%lu`
`long int`	`%ld`	`%ld`
`unsigned int`	`%u`	`%u`
`int`	`%d`	`%d`
`unsigned short`	`%hu`	`%hu`
`short`	`%hd`	`%hd`
`char`	`%c`	`%c`

Fig. 5.5 | Promotion hierarchy for data types.

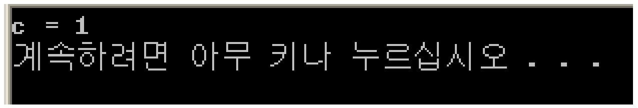
Common Programming Error 5.9

Converting from a higher data type in the promotion hierarchy to a lower type can change the data value.

Common Programming Error 5.9



```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
	int a=257,b=0;
	char c=a;
	printf("c = %d ₩n",c);
	system("PAUSE");
	return 0;
}

```

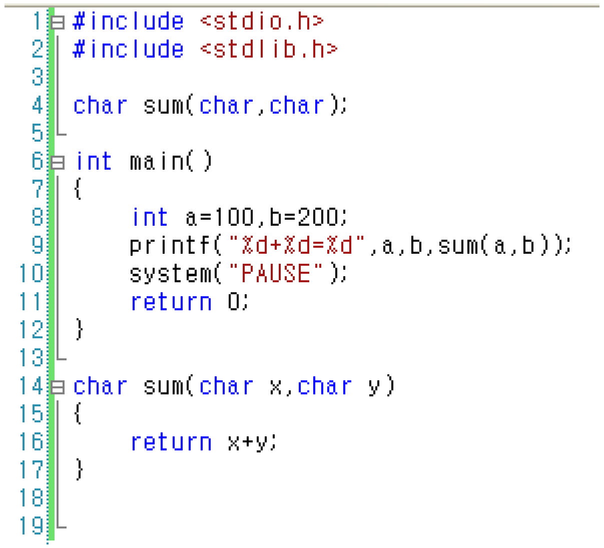


```
c = 1
계속하려면 아무 키나 누르십시오 . . .
```

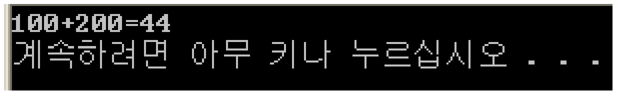
a	00000000	00000000	00000001	00000001

b	00000001

Common Programming Error 5.9



```c
#include <stdio.h>
#include <stdlib.h>

char sum(char,char);

int main()
{
        int a=100,b=200;
        printf("%d+%d=%d",a,b,sum(a,b));
        system("PAUSE");
        return 0;
}

char sum(char x,char y)
{
        return x+y;
}

```



```
100+200=44
계속하려면 아무 키나 누르십시오 . . .
```

Common Programming Error 5.10

Forgetting a function prototype causes a syntax error if the return type of the function is not `int` and the function definition appears after the function call in the program. Otherwise, forgetting a function prototype may cause a runtime error or an unexpected result.

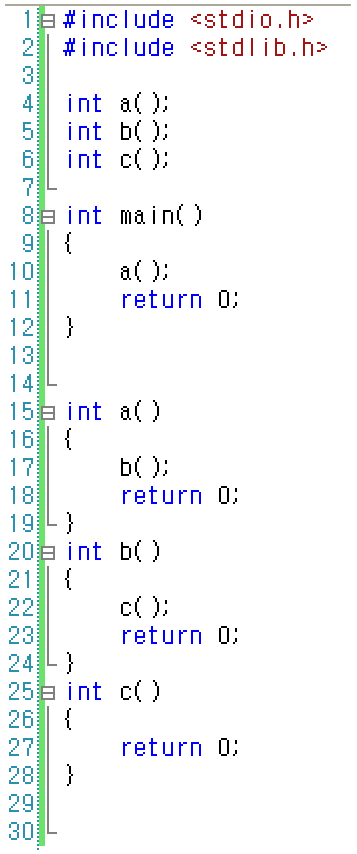
Software Engineering Observation 5.9

A function prototype placed outside any function definition applies to all calls to the function appearing after the function prototype in the file. A function prototype placed in a function applies only to calls made in that function.

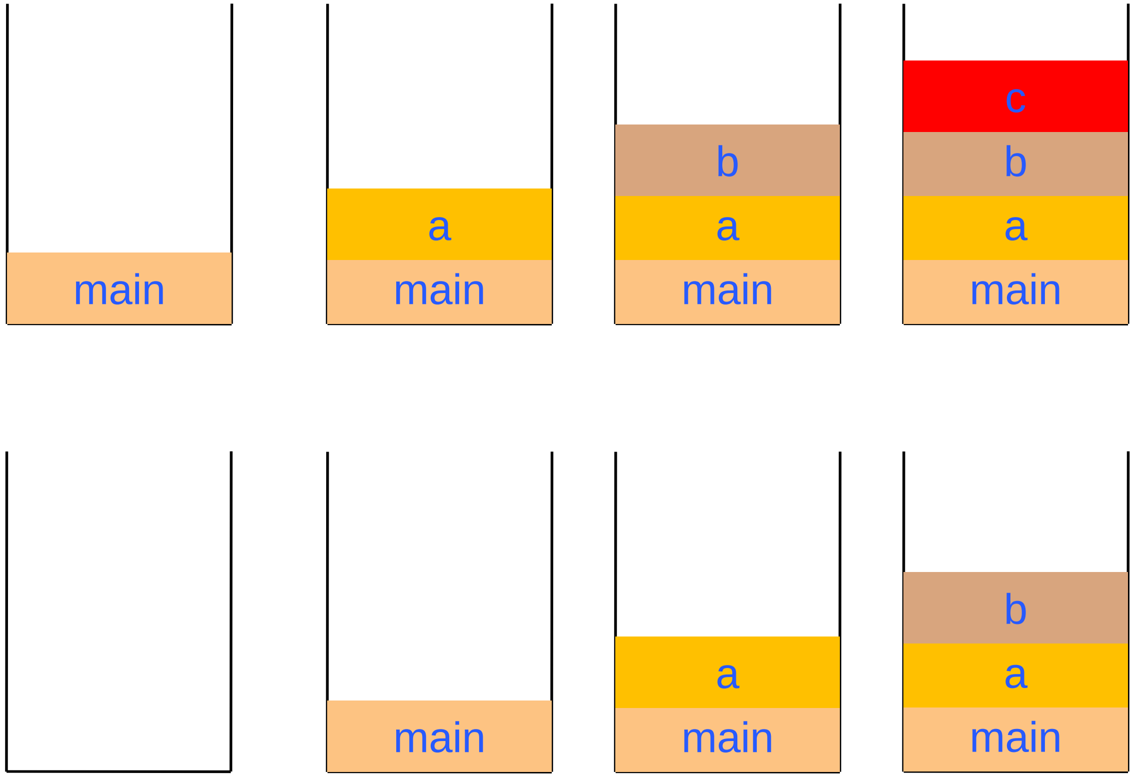
5.7 Function Call Stack and Activation Records

- **Program execution stack**
  - **A stack is a last-in, first-out (LIFO) data structure**
    - **Anything put into the stack is placed "on top"**
    - **The only data that can be taken out is the data on top**
  - **C uses a program execution stack to keep track of which functions have been called**
    - **When a function is called, it is placed on top of the stack**
    - **When a function ends, it is taken off the stack and control returns to the function immediately below it**
  - **Calling more functions than C can handle at once is known as a "stack overflow error"**

5.7 Function Call Stack and Activation Records



```
#include <stdio.h>
#include <stdlib.h>

int a( );
int b( );
int c( );

int main( )
{
        a( );
        return 0;
}


int a( )
{
        b( );
        return 0;
}
int b( )
{
        c( );
        return 0;
}
int c( )
{
        return 0;
}

```



5.8 Headers

- **Header files**
  - **Contain function prototypes for library functions**
  - `<stdlib.h>` , `<math.h>` **, etc**
  - **Load with** `#include <filename>`

    `#include <math.h>`
- **Custom header files**
  - **Create file with functions**
  - **Save as** `filename.h`
  - **Load in other files with** `#include "filename.h"`
  - **Reuse functions**

Standard library header	Explanation
`<assert.h>`	Contains macros and information for adding diagnostics that aid program debugging.
`<ctype.h>`	Contains function prototypes for functions that test characters for certain properties, and function prototypes for functions that can be used to convert lowercase letters to uppercase letters and vice versa.
`<errno.h>`	Defines macros that are useful for reporting error conditions.
`<float.h>`	Contains the floating-point size limits of the system.
`<limits.h>`	Contains the integral size limits of the system.
`<locale.h>`	Contains function prototypes and other information that enables a program to be modified for the current locale on which it is running. The notion of locale enables the computer system to handle different conventions for expressing data like dates, times, dollar amounts and large numbers throughout the world.

Fig. 5.6 | Some of the standard library headers. (Part 1 of 3.)

Standard library header	Explanation
`<math.h>`	Contains function prototypes for math library functions.
`<setjmp.h>`	Contains function prototypes for functions that allow bypassing of the usual function call and return sequence.
`<signal.h>`	Contains function prototypes and macros to handle various conditions that may arise during program execution.
`<stdarg.h>`	Defines macros for dealing with a list of arguments to a function whose number and types are unknown.
`<stddef.h>`	Contains common definitions of types used by C for performing certain calculations.

Fig. 5.6 | Some of the standard library headers. (Part 2 of 3.)

Standard library header	Explanation
`<stdio.h>`	Contains function prototypes for the standard input/output library functions, and information used by them.
`<stdlib.h>`	Contains function prototypes for conversions of numbers to text and text to numbers, memory allocation, random numbers, and other utility functions.
`<string.h>`	Contains function prototypes for string-processing functions.
`<time.h>`	Contains function prototypes and types for manipulating the time and date.

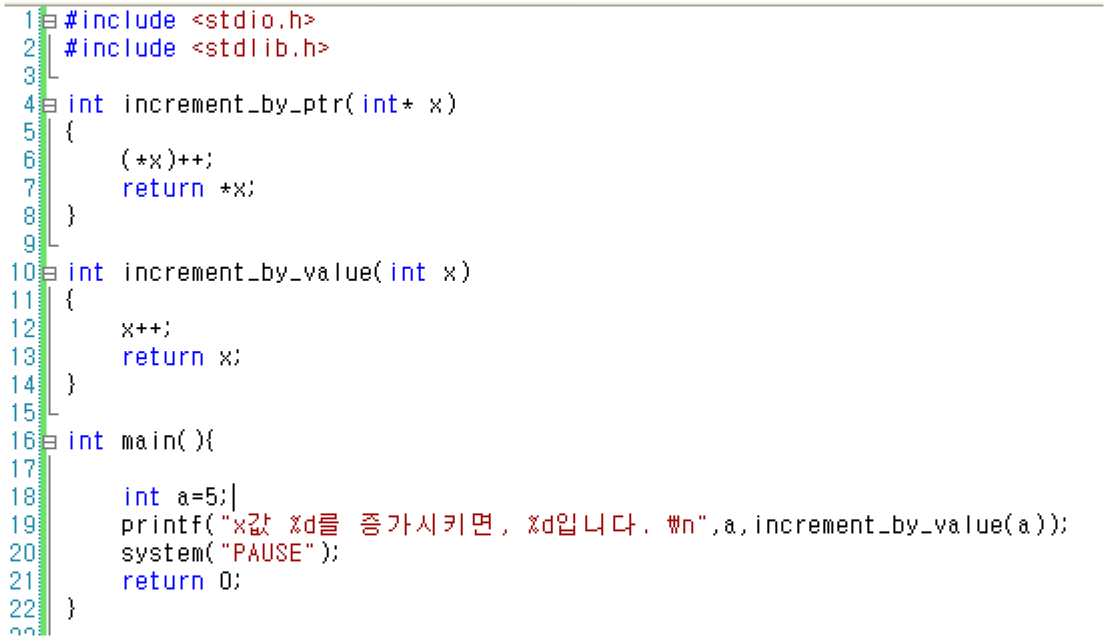
Fig. 5.6 | Some of the standard library headers. (Part 3 of 3.)

5.9 Calling Functions: Call-by-Value and Call-by-Reference

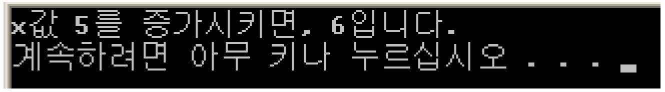
- Call by value
  - Copy of argument passed to function
  - Changes in function do not effect original
  - Use when function does not need to modify argument
    - Avoids accidental changes
- Call by reference
  - Passes original argument
  - Changes in function effect original
  - Only used with trusted functions
- For now, we focus on call by value

5.9 Calling Functions: Call-by-Value and Call-by-Reference

- **Call by value**



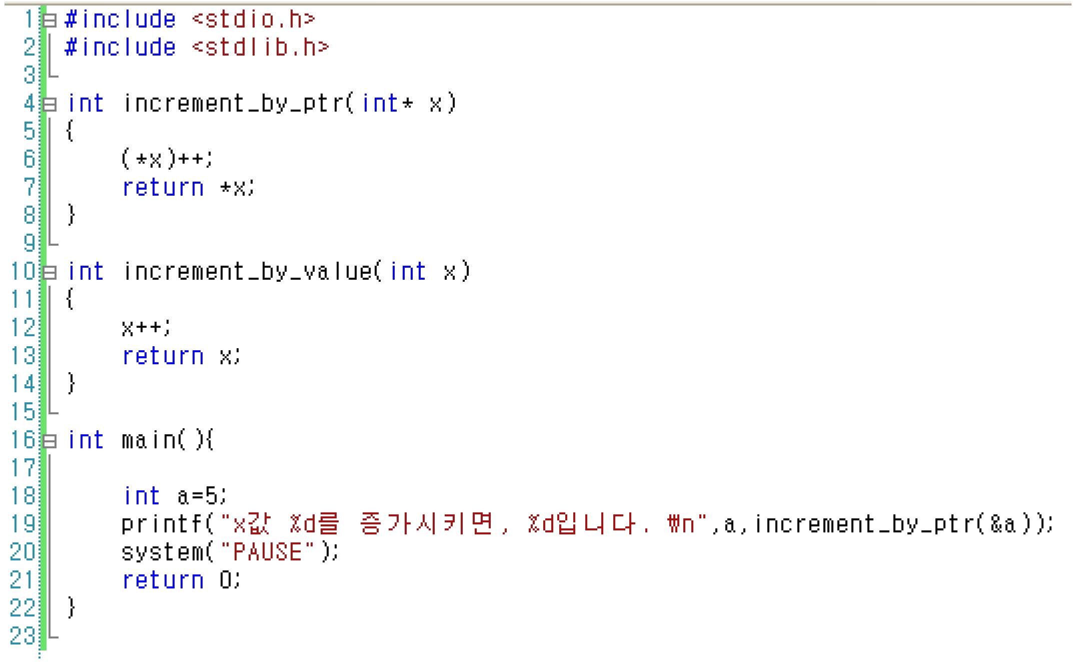
```
#include <stdio.h>
#include <stdlib.h>

int increment_by_ptr(int* x)
{
    (*x)++;
    return *x;
}

int increment_by_value(int x)
{
    x++;
    return x;
}

int main(){

    int a=5;
    printf("x값 %d를 증가시키면, %d입니다. ₩n",a,increment_by_value(a));
    system("PAUSE");
    return 0;
}
```



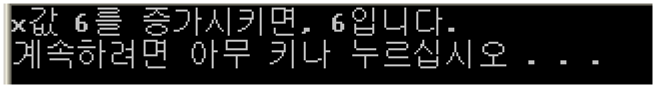
```
x값 5를 증가시키면, 6입니다.
계속하려면 아무 키나 누르십시오 . . .
```

5.9 Calling Functions: Call-by-Value and Call-by-Reference

- **Call by reference**



```c
#include <stdio.h>
#include <stdlib.h>

int increment_by_ptr( int* x)
{
    (*x)++;
    return *x;
}

int increment_by_value( int x)
{
    x++;
    return x;
}

int main(){

    int a=5;
    printf("x값 %d를 증가시키면, %d입니다. ₩n",a,increment_by_ptr(&a));
    system("PAUSE");
    return 0;
}
```



```
x값 6를 증가시키면, 6입니다.
계속하려면 아무 키나 누르십시오 . . .
```