

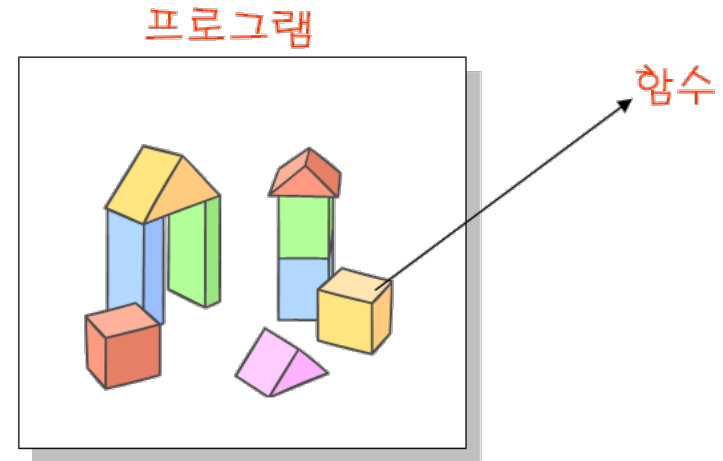
함수 functions 와 인자들 arguments

2023

국민대학교 소프트웨어학부

모듈의 개념

- **모듈(module)**
 - 독립되어 있는 프로그램의 일부분
- **모듈러 프로그래밍**
 - 모듈 개념을 사용하는 프로그래밍 기법
- **모듈러 프로그래밍의 장점**
 - 각 모듈들은 독자적으로 개발 가능
 - 다른 모듈과 독립적으로 변경 가능
 - 유지 보수가 쉬워진다.
 - 모듈의 재사용 가능
- C++에서는 **모듈==함수**



```
#include <iostream>
using namespace std;

int angleClock(int h, int m);

int main(void)
{
    int t;
    int h, m;

    cin >> t;

    for(int i=0; i<t; i++)
    {
        cin >> h >> m;
        cout << angleClock( h, m ) << endl;
    }

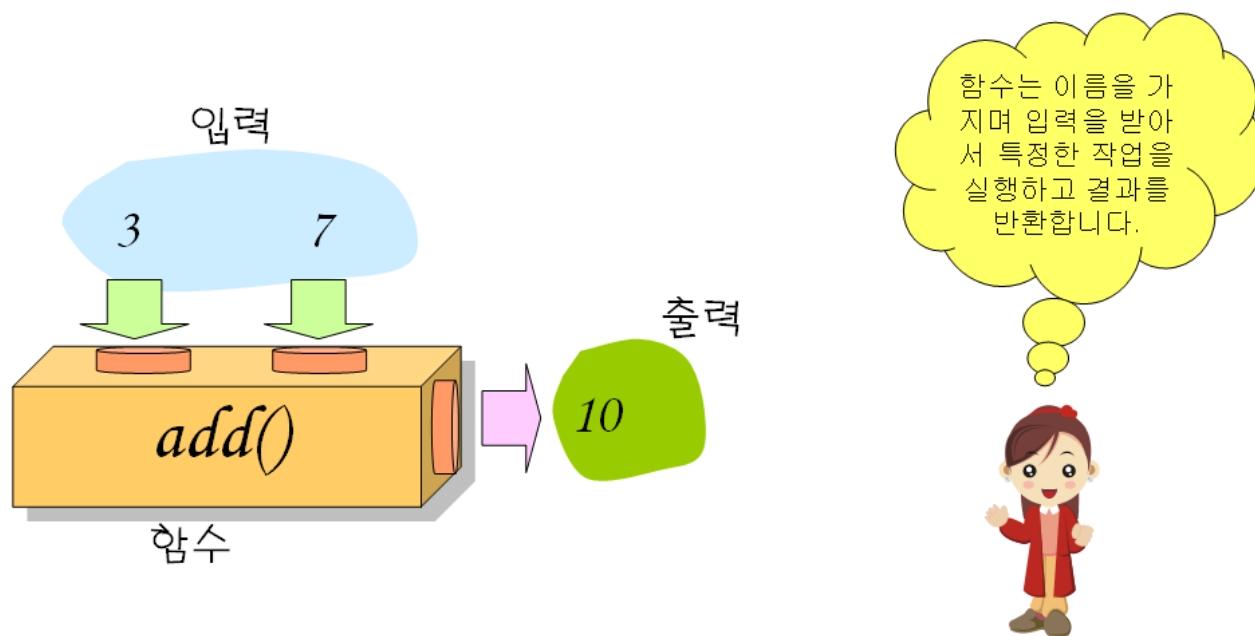
    return 0;
}

int angleClock(int h, int m)
{

}
```

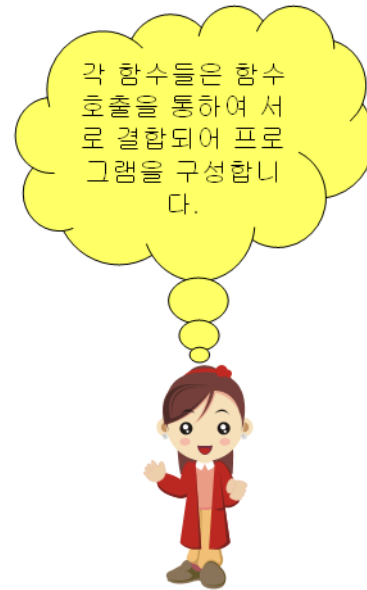
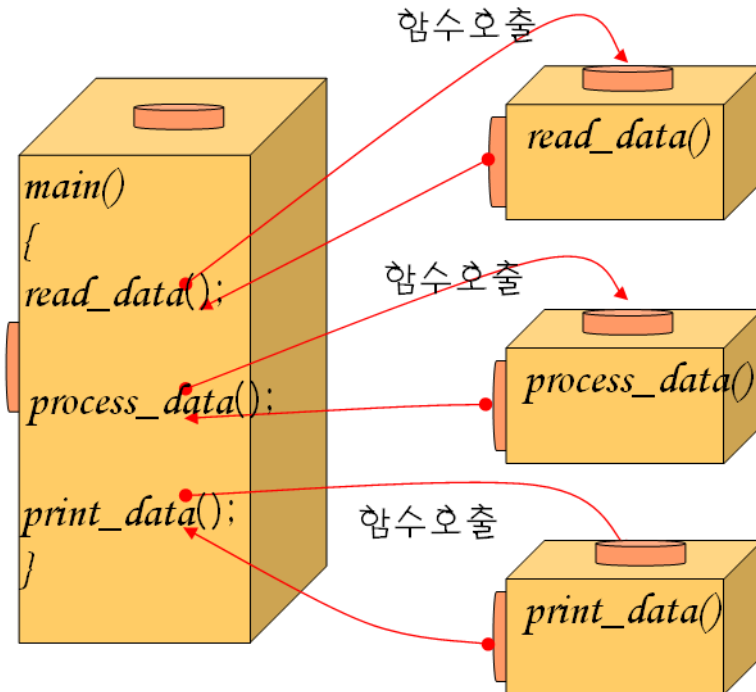
함수의 개념

- 함수(*function*): 특정한 작업을 수행하는 독립적인 부분
- 함수 호출(*function call*): 함수를 호출하여 사용하는 것
- 함수는 입력을 받으며 출력을 생성한다.

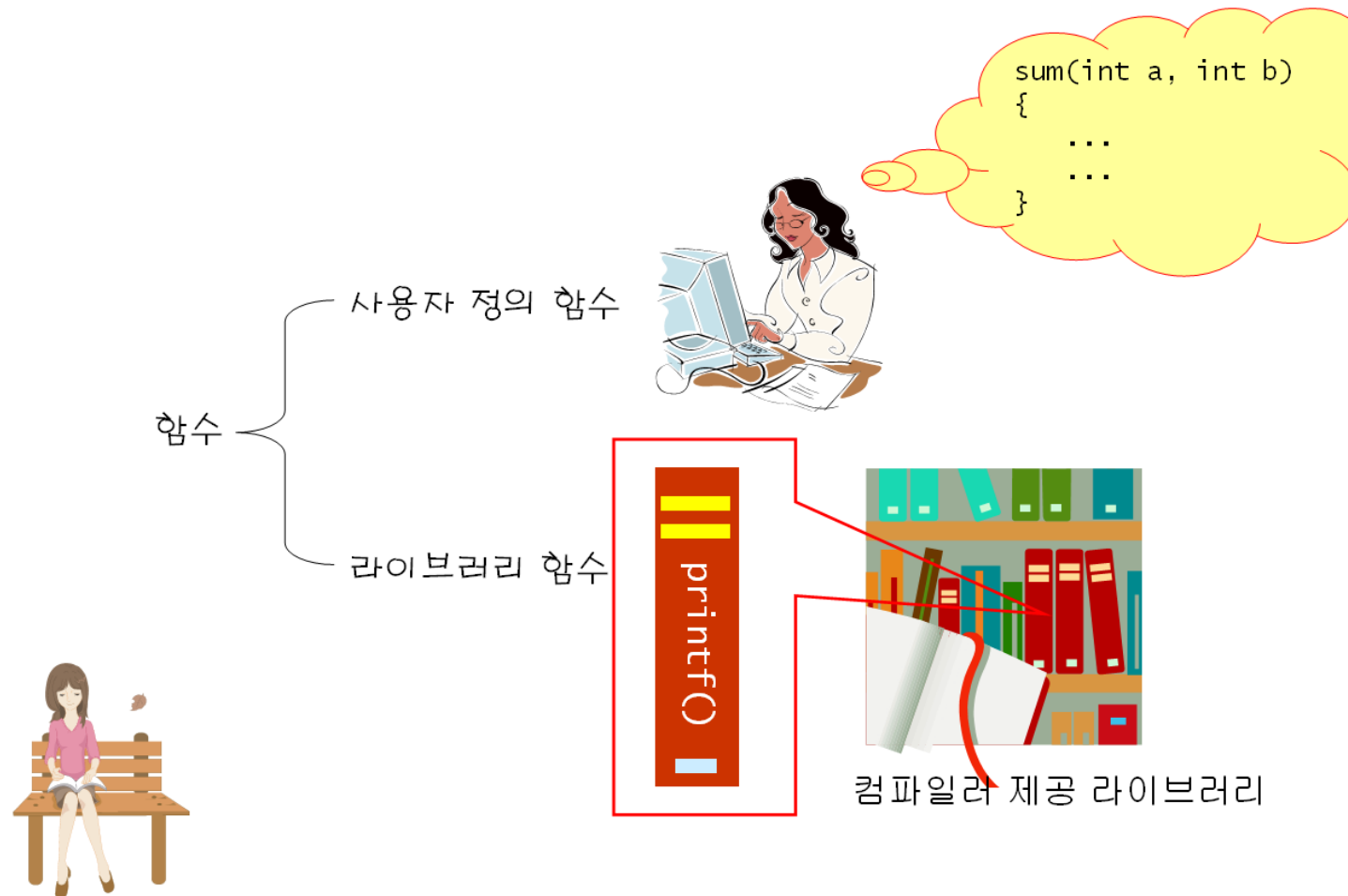


함수들의 연결

- 프로그램은 여러 개의 함수들로 이루어진다.
- 함수 호출을 통하여 서로 서로 연결된다.
- 제일 먼저 호출되는 함수는 `main()`이다.

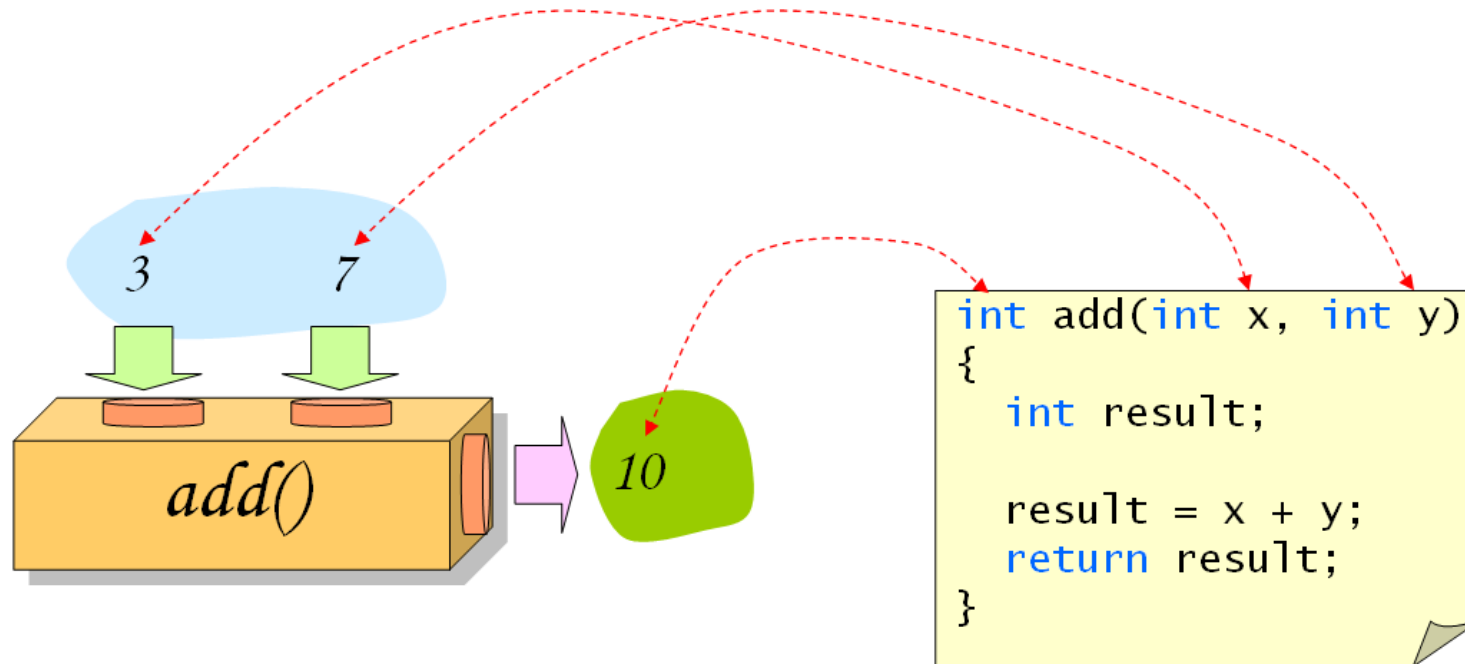


함수의 종류

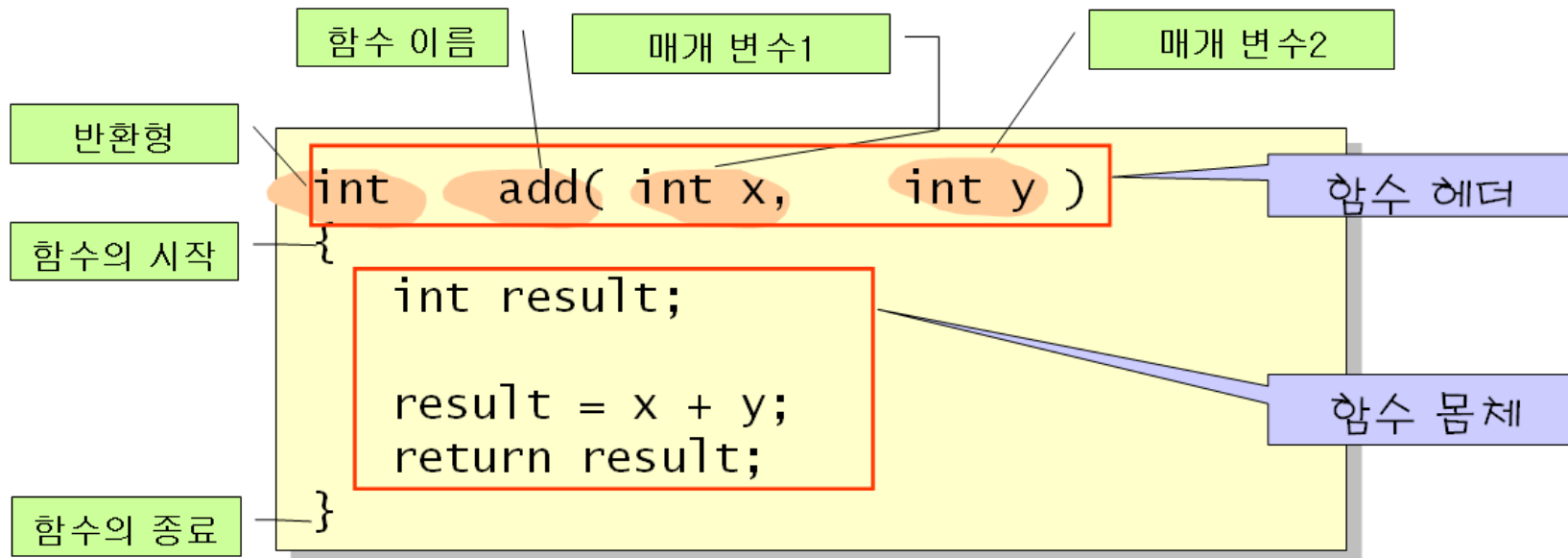


함수의 정의

- 반환형(return type)
- 함수 헤더(function header)
- 함수 몸체(function body)

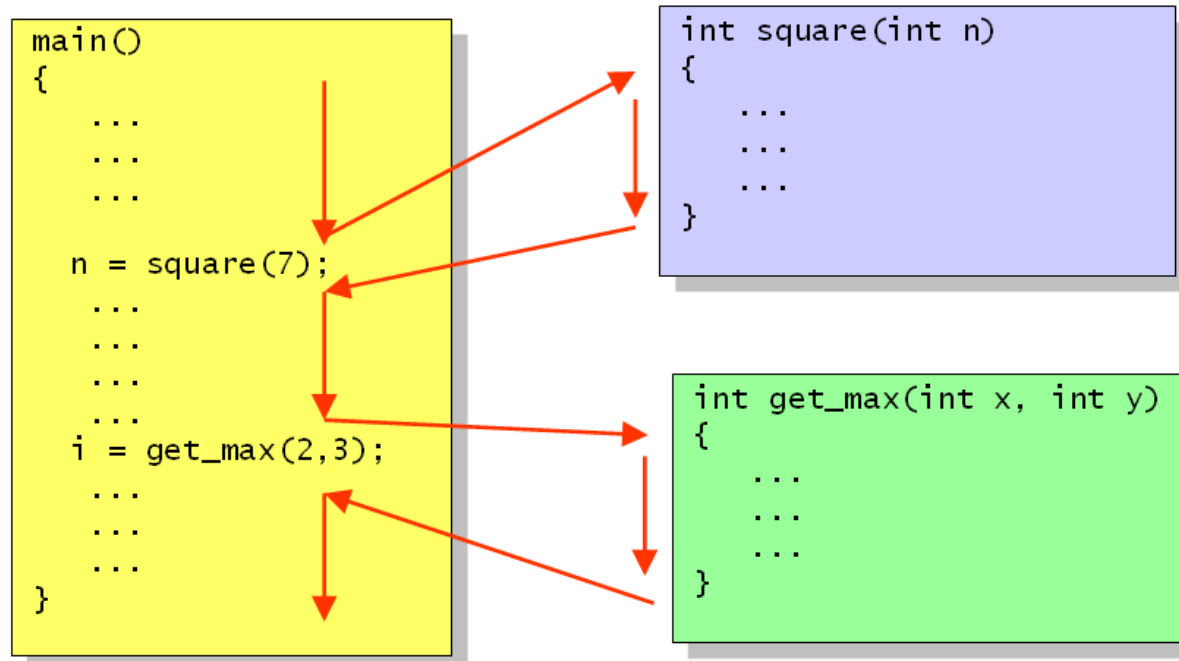


함수의 구조



함수 호출과 반환

- 함수 호출(*function call*):
 - 함수를 사용하기 위하여 함수의 이름을 적어주는 것
 - 함수안의 문장들이 순차적으로 실행된다.
 - 문장의 실행이 끝나면 호출한 위치로 되돌아 간다.
 - 결과값을 전달할 수 있다.



인자와 매개 변수

- **인자(argument)**: 실인수, 실매개 변수라고도 한다.
- **매개 변수(parameter)**: 형식 인수, 형식 매개 변수라고도 한다.

```
int main(void)
{
    ...
    i = get_max(2, 3);
    ...
}
```

인수

```
int get_max(int x, int y)
{
    ...
    ...
    ...
}
```

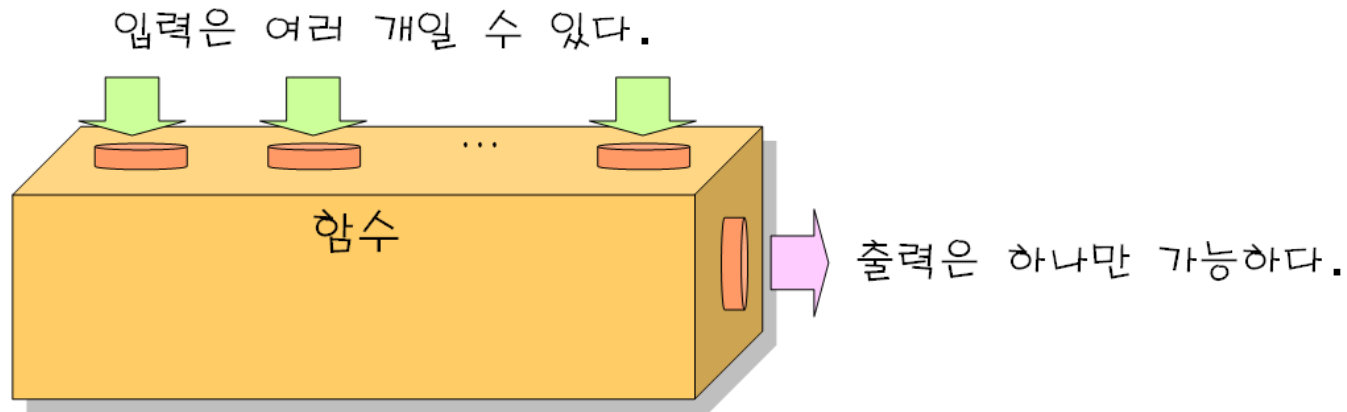
매개변수

```
#include <iostream>
using namespace std;
int add(int x, int y)
{
    return (x + y);
}
int main()
{
    // 2와 3이 add()의 인자가 된다.
    add(2, 3);

    // 5와 6이 add()의 인자가 된다.
    add(5, 6);
    return 0;
}
```

반환값

- 반환값(*return value*): 호출된 함수가 호출한 곳으로 작업의 결과값을 전달하는 것
- 인수는 여러 개가 가능하나 반환값은 하나만 가능



```
return 0;  
return(0);  
return x;  
return x+y;  
return x*x+2*x+1;
```

함수 원형

- 함수 원형(*function prototyping*): 컴파일러에게 함수에 대하여 미리 알리는 것

```
#include <iostream>
using namespace std;
```

```
int square(int n);
```

```
int main()
```

```
{
```

```
    int i, result;
```

```
    for(i = 0; i < 5; i++)
```

```
    {
```

```
        result = square(i);
```

```
        cout << result << endl;
```

```
    }
```

```
    return 0;
```

```
}
```

```
int square(int n)
```

```
{
```

```
    return(n * n);
```

```
}
```

// 함수 원형

함수 원형

// 함수 호출

// 함수 정의

함수 원형

- 함수 원형(*function prototype*) : 미리 컴파일러에게 함수에 대한 정보를 알리는 것

반환형 함수이름 (매개변수 1, 매개변수 2, ...);

```
#include <iostream>
using namespace std;
int compute_sum(int n);

int main()
{
    ...
    sum = compute_sum(100);
    ...
}

int compute_sum(int n)
{
    ...
}
```

- int compute_sum(int n);
- int get_integer(void);
- int combination(int n, int r);
- void draw_rect(int side);

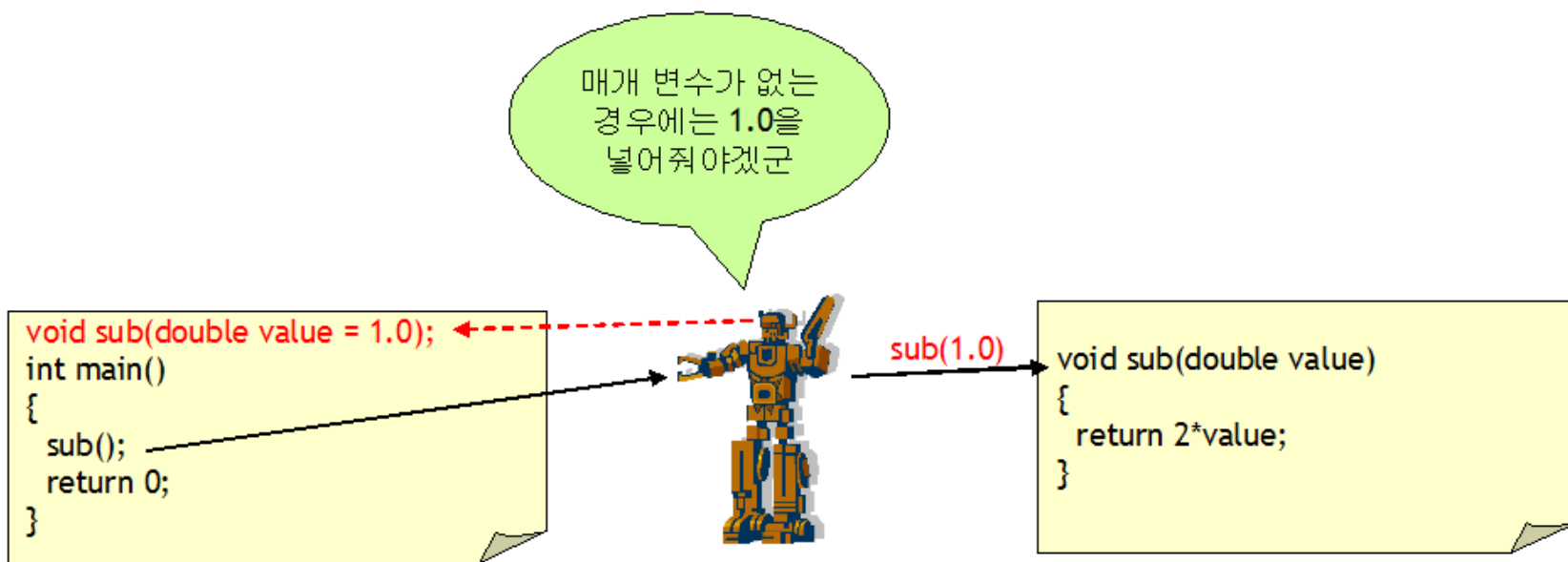
OR

- int compute_sum(int);
- int get_integer(void);
- int combination(int, int);
- void draw_rect(int);

디폴트 매개 변수

- 디폴트 매개 변수(default parameter): 인자를 전달하지 않아도 디폴트 값을 대신 넣어주는 기능

```
void sub(double value = 1.0); // 함수 원형 정의시
```



주의할 점

- 디폴트 매개 변수는 뒤에서부터 앞쪽으로만 정의할 수 있다.

`void sub(int p1, int p2, int p3=30);// OK!`

`void sub(int p1, int p2=20, int p3=30);// OK!`

`void sub(int p1=10, int p2=20, int p3=30);// OK!`

`void sub(int p1, int p2=20, int p3);// 컴파일 오류!`

`void sub(int p1=10, int p2, int p3=30);// 컴파일 오류!`

예제



```
#include <iostream>
using namespace std;
```

```
int calc_deposit(int salary=300, int month=12);
```

```
int main()
{
    cout << "0개의 디폴트 매개 변수 사용" << endl;
    cout << calc_deposit(200, 6) << endl;

    cout << "1개의 디폴트 매개 변수 사용" << endl;
    cout << calc_deposit(200) << endl;

    cout << "2개의 디폴트 매개 변수 사용" << endl;
    cout << calc_deposit() << endl;
    return 0;
}
```

```
int calc_deposit(int salary, int month)
{
    return salary*month;
}
```



0개의 디폴트 매개 변수 사용
1200
1개의 디폴트 매개 변수 사용
2400
2개의 디폴트 매개 변수 사용
3600
계속하려면 아무 키나 누르십시오 ...

중복 함수

- 중복 함수(overloading functions):
같은 이름을 가지는 함수를 여러 개 정의하는 것

```
// 정수값을 제공하는 함수
```

```
int square(int i)
```

```
{
```

```
    return i*i;
```

```
}
```

```
// 실수값을 제공하는 함수
```

```
double square(double i)
```

```
{
```

```
    return i*i;
```

```
}
```

예제

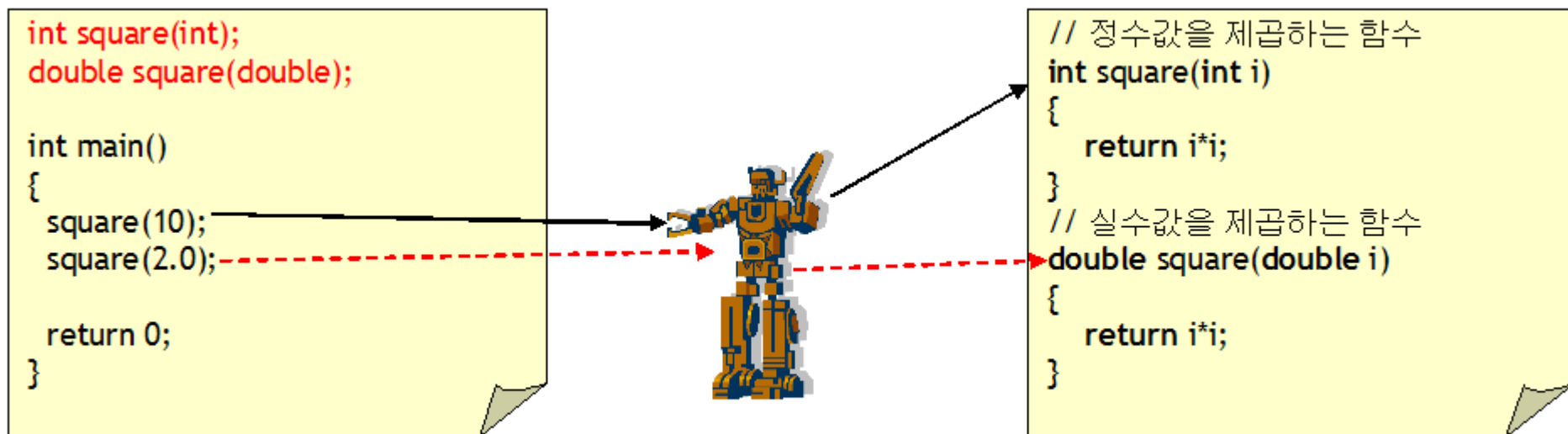


그림 4.10 중복 함수의 개념

중복 함수의 장점

- 중복 함수를 사용하지 않은 경우:

```
square_int(int int);
```

```
square_double(double int);
```

```
square_short(short int);
```

- 중복 함수를 사용하는 경우

```
square(int int);
```

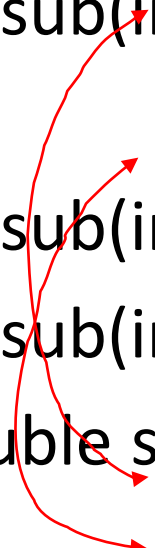
```
square(double int);
```

```
square(short int);
```



함수 이름의 재사용이 가능

주의할 점

- `int sub(int);`
 - `int sub(int, int);`// 중복 가능!
 - `int sub(int, double);`// 중복 가능!
 - `double sub(double);`// 중복 가능!
 - `double sub(int);`// 오류!! - 반환형이 다르더라도 중복 안됨!
 - `float sub(int, int);`// 오류!! - 반환형이 다르더라도 중복 안됨!
- 
- A diagram consisting of three red curved arrows. The first arrow starts from the 'int' parameter in the second line and points to the 'int' parameter in the first line. The second arrow starts from the 'double' parameter in the third line and points to the 'double' parameter in the fourth line. The third arrow starts from the 'double' parameter in the fourth line and points to the 'double' parameter in the fifth line. These arrows illustrate that function signatures are considered identical only if both the parameter list and the return type match.

인라인 함수

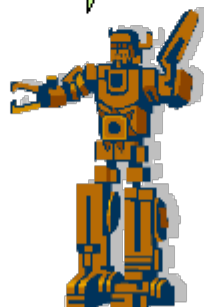
- 인라인 함수(inline function):

함수 호출을 하지 않고 코드를 복사하여서 넣는 것

inline 인 경우에는
함수 몸체를
호출한 곳에 삽입
합니다.

```
int main()
{
    int result = square(10);
    return 0;
}

// 정수값을 제공하는 함수
inline int square(int i)
{
    return i*i;
}
```



```
int main()
{
    int result = 10*10;
    return 0;
}
```

예제



```
#include <iostream>
using namespace std;
// 정수값을 제공하는 함수
inline double square(double i)
{
    return i*i;
}
int main()
{
    double result;
    cout << "2.0의 제곱은 ";
    result = square(2.0);
    cout << result << endl;
    cout << "3.0의 제곱은 ";
    result = square(3.0);
    cout << result << endl;
    return 0;
}
```



2.0의 제곱은 4
3.0의 제곱은 9
계속하려면 아무 키나 누르십시오 ...

라이브러리 함수

- 라이브러리 함수(*library function*): 컴파일러에서 제공하는 함수
 - 표준 입출력
 - 수학 연산
 - 문자열 처리
 - 시간 처리
 - 오류 처리
 - 데이터 검색과 정렬

수학 라이브러리 함수

분류	함수	설명
삼각함수	<code>double sin(double x)</code>	사인값 계산
	<code>double cos(double x)</code>	코사인값 계산
	<code>double tan(double x)</code>	탄젠트값 계산
역삼각함수	<code>double acos(double x)</code>	역코사인값 계산 결과값 범위 $[0, \pi]$
	<code>double asin(double x)</code>	역사인값 계산 결과값 범위 $[-\pi/2, \pi]$
	<code>double atan(double x)</code>	역탄젠트값 계산 결과값 범위 $[-\pi/2, \pi]$
쌍곡선함수	<code>double cosh(double x)</code>	쌍곡선 코사인
	<code>double sinh(double x)</code>	쌍곡선 사인
	<code>double tanh(double x)</code>	쌍곡선 탄젠트
지수함수	<code>double exp(double x)</code>	e^x
	<code>double log(double x)</code>	$\log_e x$
	<code>double log10(double x)</code>	$\log_{10} x$
기타함수	<code>double ceil(double x)</code>	x보다 작지 않은 가장 작은 정수
	<code>double floor(double x)</code>	x보다 크지 않은 가장 큰 정수
	<code>double fabs(double x)</code>	x의 절대값
	<code>double pow(double x, double y)</code>	x^y
	<code>double sqrt(double x)</code>	\sqrt{x}

overloaded abs()

- `int abs(int n);`
- `long abs(long n);`
- `float abs(float arg);`
- `double abs(double arg);`

난수 생성 라이브러리 함수

- **int rand()**
 - 난수를 생성하는 함수
 - 0부터 RAND_MAX까지의 난수를 생성



```
#include <iostream>
#include <cmath>
#include <ctime>
#include <cstdlib>
using namespace std;
// 0에서 RAND_MAX 까지 n개의 난수를 화면에 출력한다.
void get_random( int n )
{
    int i;
    for( i = 0; i < n; i++ )
        cout << rand() << endl;
}
int main()
{
    // 일반적으로 난수 발생기의 시드(seed)를 현재 시간으로 설정한다.
    // 현재 시간은 수행할 때마다 달라지기 때문이다.
    srand( (unsigned)time( NULL ) );
    get_random( 10 );
    return 0;
}
```



```
21783
14153
4693
13117
21900
19957
15212
20710
4357
16495
```

search rand at cplusplus.com

C library:

- <cassert> (assert.h)
- <cctype> (ctype.h)
- <cerrno> (errno.h)
- <cfenv> (fenv.h)
- <cfloat> (float.h)
- <inttypes> (inttypes.h)
- <iso646> (iso646.h)
- <limits> (limits.h)
- <locale> (locale.h)
- <cmath> (math.h)
- <setjmp> (setjmp.h)
- <signal> (signal.h)
- <stdarg> (stdarg.h)
- <stdbool> (stdbool.h)
- <stddef> (stddef.h)
- <stdint> (stdint.h)
- <stdio> (stdio.h)
- <stdlib> (stdlib.h)**
- <string> (string.h)
- <tgmath> (tgmath.h)
- <time> (time.h)
- <uchar> (uchar.h)
- <wchar> (wchar.h)
- <wctype> (wctype.h)

Containers:

Input/Output:

Multi-threading:

Other:

<stdlib>

rand

Generate random number

Returns a pseudo-random integral number in the range between 0 and `RAND_MAX`.

This number is generated by an algorithm that returns a sequence of apparently non-related numbers each time it is called. This algorithm uses a seed to generate the series, which should be initialized to some distinctive value using function `srand`.

`RAND_MAX` is a constant defined in `<stdlib>`.

A typical way to generate trivial pseudo-random numbers in a determined range using `rand` is to use the modulo of the returned value by the range span and add the initial value of the range:

```
1 v1 = rand() % 100;           // v1 in the range 0 to 99
2 v2 = rand() % 100 + 1;       // v2 in the range 1 to 100
3 v3 = rand() % 30 + 1985;      // v3 in the range 1985-2014
```

Notice though that this modulo operation does not generate uniformly distributed random numbers in the span (since in most cases this operation makes lower numbers slightly more likely).

C++ supports a wide range of powerful tools to generate random and pseudo-random numbers (see `<random>` for more info).

Parameters

(none)

Return Value

An integer value between 0 and `RAND_MAX`.

Example

```
1 /* rand example: guess the number */
2 #include <stdio.h>           /* printf, scanf, puts, NULL */
3 #include <stdlib.h>          /* srand, rand */
4 #include <time.h>            /* time */
```