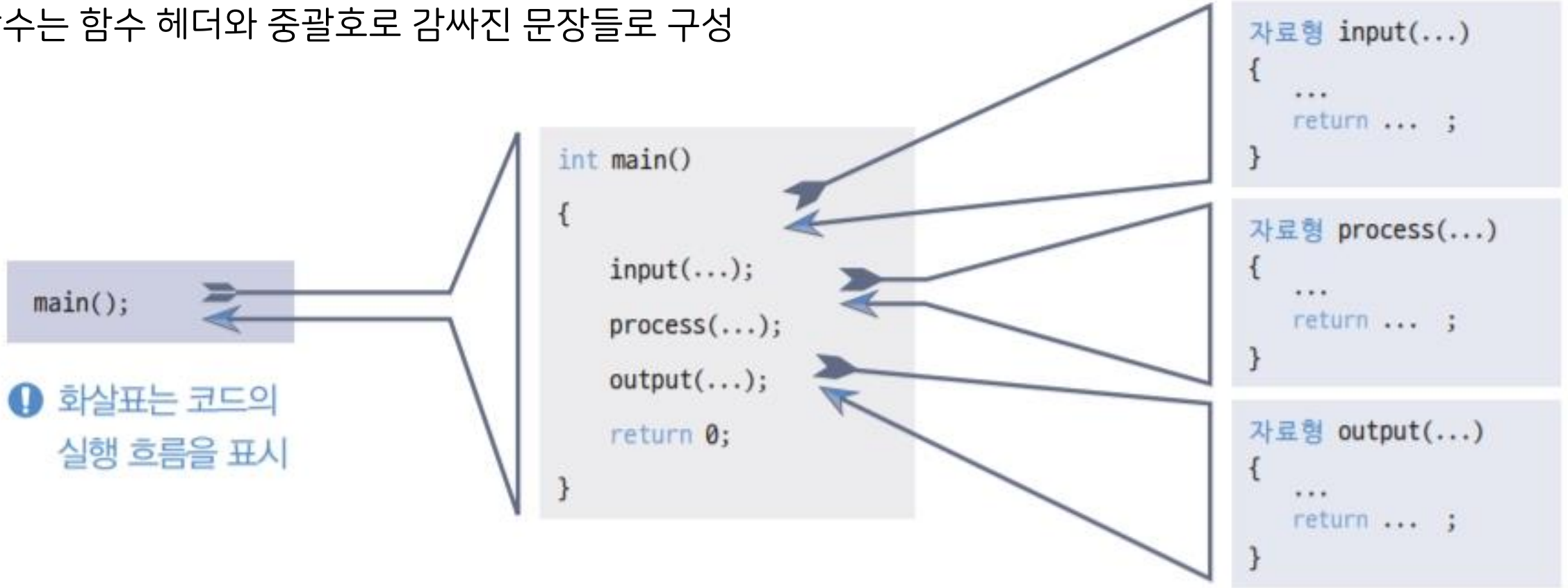
C++ / JAVA 프로그래밍

05주차 실습

함수 사용하기

함수란?

- 함수는 어떤 작업을 하기 위해 설계된 엔티티
- 함수는 함수 헤더와 중괄호로 감싸진 문장들로 구성



함수의 장점 (1/2)

- 함수를 사용하면 입력할 내용이 더 많아지는 것 같은데, 왜 함수로 나누어서 코드를 작성하는 이유는?
- 함수로 나누어서 작성할 때 얻을 수 있는 장점은?

- **분할 처리**
  - 복잡하고 어려운 일을 한 번에 하는 것보다 작은 일을 여러 번 하는 것이 더 쉬움
  - 자동차를 한 번에 처음부터 끝까지 만들기보다 차체, 타이어, 엔진을 따로 만들어서 조립하는 방법이 훨씬 쉬움

- **오류 확인(디버그)**
  - 오류를 찾는 일은 프로그래밍에서 가장 힘든 작업 중 하나
  - 그런데 프로그램을 작은 함수로 나누고 각각의 함수들을 모두 디버깅한 뒤에 다시 하나의 프로그램으로 조립하면?
  - 오류 검사가 더 쉬워지며 발견한 오류도 더 쉽게 해결할 수 있음

함수의 장점 (2/2)

- **재사용**
  - 작은 작업들은 더 다양한 곳에 활용할 수 있음
  - 예를 들어 자동차 타이어는 규격만 맞으면 여러 모델의 자동차에 장착이 가능
  - 따라서 작은 작업들을 분리해서 함수로 만들면, 다양한 프로그램에서 활용하여 쉽게 큰 프로그램을 만들 수 있음

- **함수 라이브러리**
  - 도서관에 인기 있는 책들이 많이 보관되어 있는 것처럼 함수 라이브러리는 잘 쓰여진 함수들이 보관되어 있는 곳
  - 운영체제 또는 컴퓨터 하드웨어와의 상호작용과 관련된 작업들은 함수 라이브러리로 미리 작성되어 있음

함수를 만들기(또는 사용하기) 위해 정해야 할 3가지

- 함수의 이름은 무엇인가?
  - 구현하려는 기능이 더하기라면? sum

} 함수명 (function name)

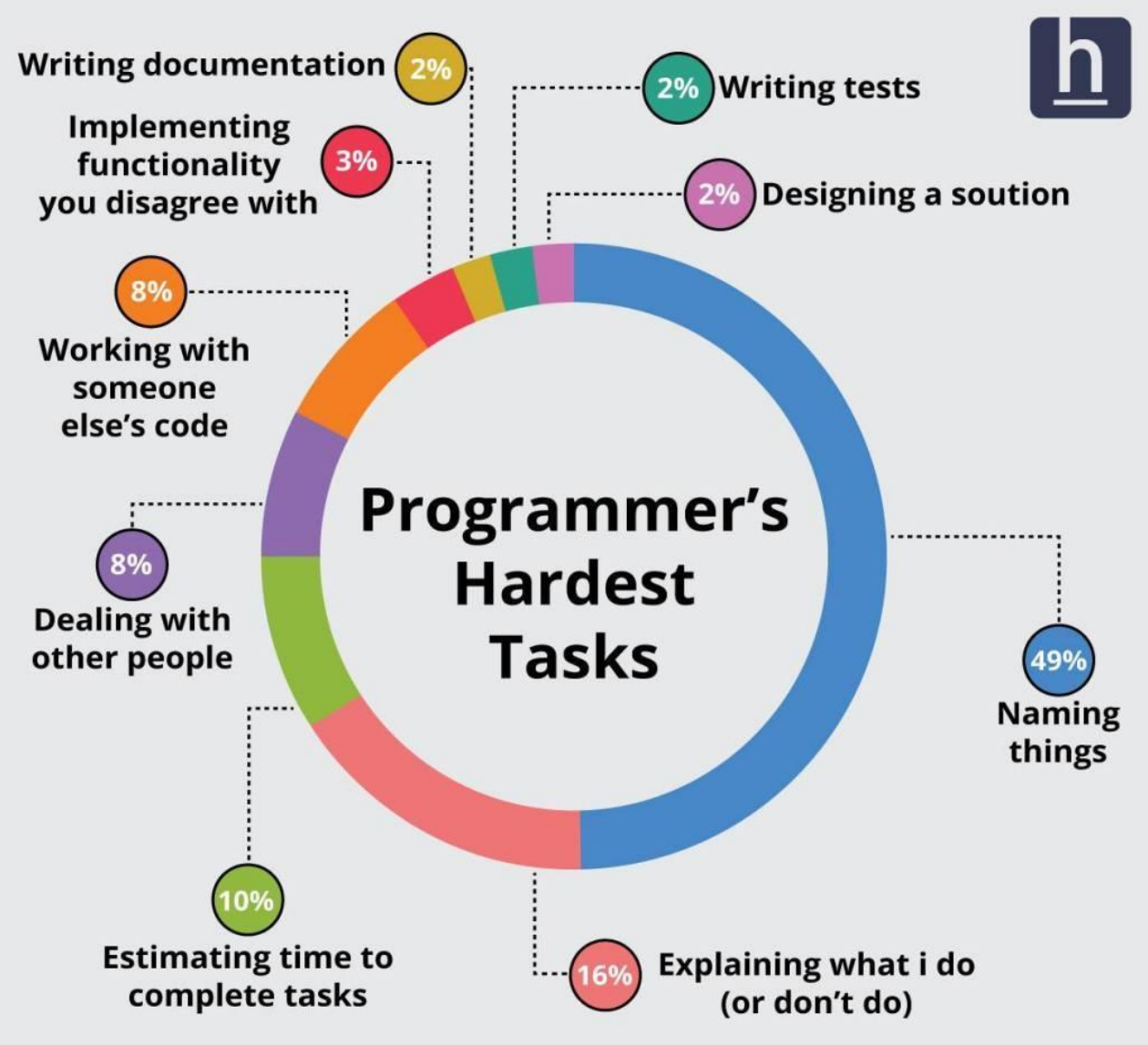
- 함수가 기능을 수행하는 데 필요한 데이터는 무엇인가?
  - 더하려는 정수 2개

} 매개변수 (parameters)

- 함수가 수행된 후의 결과는 어떤 값인가?
  - 정수 (입력 받은 정수 2개를 더한 값)

} 반환형 (return type)

프로그래머가 가장 힘들어하는 일



함수를 만들어 보자

- 함수 원형 (function prototype): 함수를 사용할 때 알아야 할 것만 모아놓은 것
- 함수 선언 (function declaration): 함수 원형을 적는 것

```
반환형 함수명(매개변수1 타입, 매개변수2 타입);
```

세미콜론을 붙여줌

함수 선언에서는 타입만 써도 OK

- 함수 정의(function definition): 함수의 실제 코드를 작성하고 기능을 구현하는 것

```
반환형 함수명(매개변수1 타입 매개변수1 이름, 매개변수2 타입 매개변수2 이름){

    // 함수가 처리할 명령 및 결괏값 반환
    함수 본체 (function body) = 두 중괄호 사이의 내용
}
```

형식 인자(formal parameters)
또는 매개변수(parametric variables)

- 함수 호출 (function call): 함수에 필요한 인자를 주고 결과를 얻는 것

함수의 기본 사용법 – 함수 정의

- 함수를 사용하려면 함수 정의, 함수 선언, 함수 호출이라는 엔티티가 필요
- 함수 정의(function definition)는 함수를 만드는 것을 의미

```
리턴_자료형 함수_이름(매개변수_리스트)
{
    본문
}
```

함수 정의 구문

함수의 기본 사용법 – 함수 정의 예제

- 두 정수 중 큰 값을 찾아서 리턴하는 함수

```
int larger(int first, int second)  // header
{
	int temp;
	if (first > second)
	{
		temp = first;
	}
	else
	{
		temp = second;
	}
	return temp;
}
```

함수 정의 시 주의 사항

- 들여쓰기(indentation)을 잘하자
  - 프로그래밍에서 들여쓰기는 소스코드의 가독성을 높이는 핵심 요소

```
int main(){int math; math=95; cout<<"Score is "<<math<<endl; return 0;}
```

  - 변수의 유효 범위를 쉽게 판단할 수 있음
  - 들여쓰기에도 일관성이 필요
    - 예를 들어 두 칸 들여쓰기, 네 칸 들여쓰기를 혼용한다면, 개발자로 하여금 혼란을 유발할 수 있음

- 빈 줄을 적절히 이용하자
  - 오로지 가독성을 위해 사용하는 것
  - 빈 줄은 컴파일 과정에서 모두 사라지므로 많이 넣어도 상관 없음

함수의 기본 사용법 – 함수 선언

- 함수 선언(function declaration)또는 함수 프로토타입(function prototype)
  - 함수의 헤더와 세미콜론만 조합된 구문
  - 함수 헤더에서 매개변수의 이름은 입력하지 않아도 됨

```
int larger(int first, int second);   // 매개변수에 이름을 넣은 경우

int larger(int, int);                // 매개변수에 이름을 넣지 않은 경우
```

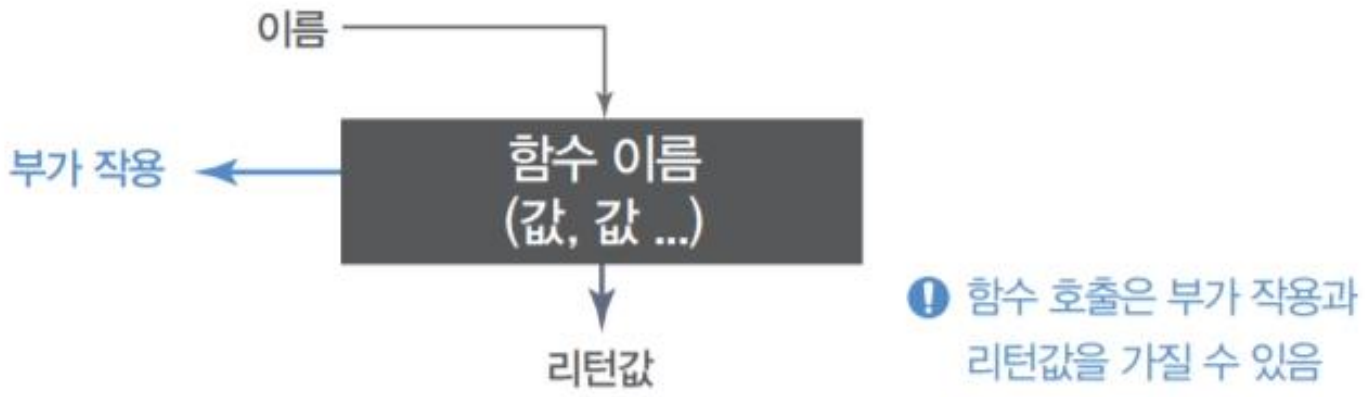
함수 선언 시 주의사항

- 변수를 사용하기 전에 선언하듯이 함수도 사용하기 전에 선언해야 함
  - C++가 단방향 컴파일러(one pass compiler)를 사용하기 때문
  - 단방향 컴파일러는 소스코드를 위에서 아래로 쭉 읽어 내려 감
  - 읽어 내려가던 중, 미리 선언되지 않은 함수를 만나면 오류(또는 경고)를 출력함
  - (단방향이 아니라) 여러 번 읽어 내려 간다면 사용 후에 선언되어 있어도 처리할 수 있음

- 함수명은 짧고 의미 있게 짓자 (이름 짓는 방법은 변수와 동일)
  - 이름이 길면 입력이 힘들고, 오타도 자주 남
  - 함수명을 통해 무엇을 하는 함수인지 알 수 있다면, 사용하기 쉬워짐 (영어 사전을 자주 보자)
  - 대소문자에 민감하므로, 자신만의 일관성이 필요함
    - check_sum(), checksum(), checkSum(), CheckSum()

함수의 기본 사용법 – 함수 호출

- 함수 호출(function call)



함수 호출 동작

```
int main()
{
        cout << larger(3, 13);
        cout << larger(10, 12);
        cout << larger(2, 12);
        return 0;
}
```

함수에 주석 달기

- 우리는 함수 원형만 알면 함수를 호출할 수 있음
  - 다만 무슨 기능을 하는 함수인지 알아야 함
  - 이름만으로 기능을 쉽게 알 수 있으면 좋지만, 이것만으로 부족한 경우가 있음
  - 주석을 적절히 이용하면 함수를 호출하는 사람에게 다음과 같은 내용을 알려줄 수 있음
    - 함수의 기능이 무엇인지
    - 인자로 사용해야할 값은 어떤 것인지
    - 함수를 통해 어떤 값을 반환하는지
    - 함수와 관련된 다른 함수는 무엇인지
    - …
  - 한 줄 주석을 달 때는 슬래쉬 기호 두 개(//)를 이용함
  - 여러 줄 주석을 달 때는 /* 으로 시작해서 */으로 끝을 표시

```
11  /*
12   * 덧셈 함수
13   * @param x 더할 값 1
14   * @param y 더할 값 2
15   * @return 두 값을 더한 결과
16   */
17  int sum(int x, int y){
18      int tmp;    // 임시 변수
19      tmp = x+y;
20      return tmp;
21  }
```

argument와 parameter

- parameter는 함수 정의에 있는 변수 선언을 뜻함
- argument는 함수를 호출할 때 매개변수를 초기화하는 값

- parameter는 변수를 할당할 때 왼쪽에 위치하는 변수라고 할 수 있음
- argument는 오른쪽에 위치하는 값이라고 할 수 있음

```
void fun(int x) // 함수 정의에서 x가 parameter
{
        …
}
```

```
int main()
{
        …
        fun(5);  // 함수 호출에서 5가 argument
        …
}
```

함수의 종류

- 함수는 크게 라이브러리 함수와 사용자 정의 함수로 구분

- C++ 라이브러리에는 미리 정의된 함수가 있음
  - 라이브러리에 필요한 함수가 미리 정의되어 있다면 이를 활용하면 됨
  - 라이브러리 내부의 함수를 사용하려면, 라이브러리를 읽어 들이고 함수를 호출하기만 하면 됨
  - 라이브러리 함수를 잘 활용하려면 어떤 함수가 어떻게 선언되었는지 알아야 함

- 사용자 정의 함수는 함수 내용을 직접 정의해서 사용해야 함
  - 각자 필요한 함수가 모두 라이브러리에 정의되어 있지는 않기 때문에 필요한 함수는 따로 정의해서 사용해야 함

수학함수

- 라이브러리 함수 중에는 수학과 관련된 것들도 있음
  - 수학 관련 함수들은 <cmath> 헤더에 포함되어 있음
  - 앞에 c가 붙은 것으로 보아 C언어에서부터 사용하던 것임을 알 수 있음

함수 선언	설명
type abs(type x);	x의 절대값을 리턴
type ceil(type x);	x보다 작거나 같은 가장 큰 정수를 리턴
type floor(type x);	x보다 크거나 같은 가장 작은 정수를 리턴
type log(type x);	x의 자연 로그(밑이 e)를 리턴
type log10(type x);	x의 상용 로그(밑이 10)를 리턴
type exp(type x);	e^x를 리턴
type pow(type x, type y);	x^y를 리턴
type sqrt(type x);	x의 루트($x^{1/2}$)를 리턴

예제 1. 수학 함수 사용하기

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

int main ( ) {
    // abs 함수 사용
    cout << abs(8) << endl;
    cout << abs(-8) << endl;
    // floor, ceil 함수 사용
    cout << floor(12.78) << endl;
    cout << ceil(12.78) << endl;
    // log, log10 함수 사용
    cout << log(100) << endl;
    cout << log10(100) << endl;
    // exp, pow 함수 사용
    cout << exp(5) << endl;
    cout << pow(2,3) << endl;
    // sqrt 함수 사용
    cout << sqrt(100) << endl;
    return 0;
}
```

결과

```
8
8
12
13
4.60517
2
148.413
8
10
```

연습문제 1. 이차 방정식의 근 찾기

- $y = ax^2 + bx + c$라 할 때, 계수 a, b, c 값을 받아 이차 방정식의 근을 찾는 프로그램을 작성해봅시다.
  - 사용자로부터 3개의 값을 입력 받는다.
  - 입력 받은 값을 이용하여 판별식을 계산한다.
  - 이차 방정식의 판별식은 $b^2 - 4ac$이다.
    - 만일 판별식이 0보다 작으면 근이 존재하지 않는다.
    - 만일 판별식이 0이라면 중근을 갖는다.
    - 만일 판별식이 0보다 크면 2개의 근을 갖는다.
  - 근이 없을 경우 "근이 없습니다."라고 출력한다.
  - 근이 있을 경우, 근의 공식 $\frac{-b \pm \sqrt{b^2-4ac}}{2a}$을 이용해 근을 계산한다.
  - 근이 여러 개일 경우, 각각 한 줄 씩 출력한다.

결과

```
3개의 계수를 a, b, c 순으로 입력하세요: 3 5 4
근이 없습니다.

3개의 계수를 a, b, c 순으로 입력하세요: 1 2 1
x1 = x2 = -1

3개의 계수를 a, b, c 순으로 입력하세요: 4 -9 2
x1 = 2
x2 = 0.25
```

삼각 함수

- C++에서는 다양한 형태의 삼각함수를 사용할 수 있음
  - 삼각함수도 수학함수인 <cmath>에 포함되어 있음
  - 인자로 넣은 값에 대한 삼각함수 값을 반환함

함수 선언	설명	매개변수 단위	리턴값의 범위
type cos(type x);	코사인을 리턴	라디안	−1∼+1
type sin(type x);	사인을 리턴	라디안	−1∼+1
type tan(type x);	탄젠트를 리턴	라디안	부동 소수점
type acos(type x);	역 코사인을 리턴	−1∼+1	0∼π
type asin(type x);	역 사인을 리턴	−1∼+1	0∼π
type atan(type x);	역 탄젠트를 리턴	부동 소수점	−π/2∼+π/2

문자 처리 함수

- C++에는 문자를 처리할 때 활용할 수 있는 여러 라이브러리 함수가 있음
- 모든 문자 처리 함수는 <cctype> 라이브러리(C Character Type의 약자)에 있음

문자 구분 함수

함수 선언	역할
int isalnum(int x);	매개변수가 알파벳 또는 숫자인지 확인
int isalpha(int x);	매개변수가 알파벳인지 확인
int iscntrl(int x);	매개변수가 control 문자인지 확인

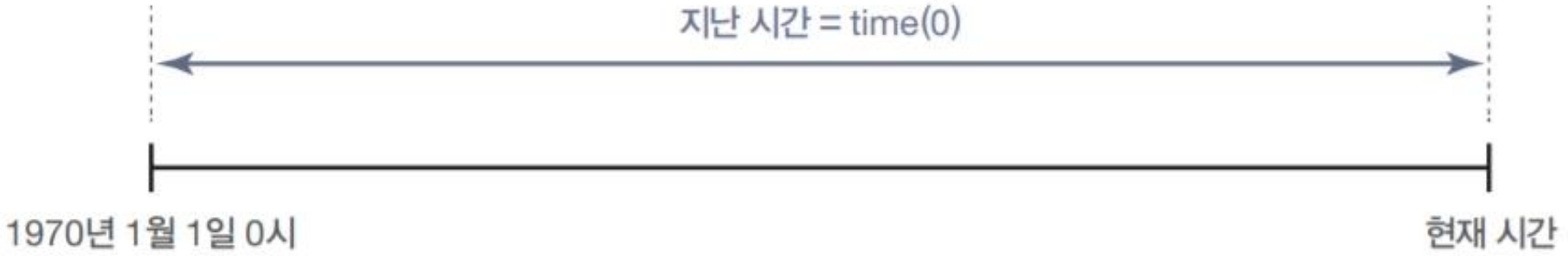
문자 변환 함수

함수 선언	역할	함수 선언	역할
int tolower(int x)	매개변수를 소문자로 변환해서 리턴	int toupper(int x)	매개변수를 대문자로 변환해서 리턴

시간 함수

- 자주 사용되는 라이브러리 함수로 <ctime> 헤더 파일에 정의되어 있는 time() 함수가 있음
- time(0)는 유닉스 타임(Unix Time)을 리턴
  - 유닉스 타임이란 1970년 1월 1일 0시를 기준으로 지난 초 단위의 시간을 말함

time(0) 함수의 리턴값



예제 2. 현재 시간 찾기 (1/2)

```
// 입출력을 위한 헤더를 포함시킨다.
#include <iostream>
// 시간을 다루기 위한 헤더를 포함시킨다.
#include <ctime>
using namespace std;

int main ( )
{
    // 경과한 초 단위 시간과 현재 초 찾기
    // 1970년 1월 1일 0시를 기준으로 지난 초 단위의 시간을 얻어온다.
    long ut = time (0);
    // ut를 60으로 나눈 나머지는 현재 초를 의미한다.
    int sec = ut % 60;
```

예제 2. 현재 시간 찾기 (2/2)

```
    // 경과한 분 단위 시간과 현재 분 찾기
    // ut를 60으로 나눈 몫은 지난 시간을 분 단위로 변환한 것이다.
    long ut60 = ut / 60;
    // ut60을 60으로 나눈 나머지는 현재 분을 의미한다.
    int min = ut60 % 60;

    // 경과한 시간과 시 단위 시간 찾기
    // ut60을 60으로 나눈 몫은 지난 시간을 시 단위로 변환한 것이다.
    long ut3600 = ut60 / 60;
    ut3600 += 9;
    // 우리나라 시간으로 맞추기 위해 ut3600에 9를 더한다.
    // ut3600을 24으로 나눈 나머지는 현재 시를 의미한다.
    int hour = ut3600 % 24;

    // 현재 시간을 출력을 출력한다.
    cout << hour << " : " << min << " : " << sec << endl;
    return 0;
}
```

결과

```
09 : 55 : 45
```

랜덤 숫자 관련 함수

- 랜덤 관련 함수는 <cstdlib>에 포함되어 있으며 rand()와 srand()로 구성되어 있음

- int rand (void)
  - 0 ~ RAND_MAX 사이의 값 중 하나를 임의로 선택하는 함수
  - RAND_MAX의 기본 값은 32767
- void srand (unsigned int seed)
  - rand()함수에 사용될 값을 초기화하는 함수
  - 동일한 seed를 사용하면 rand()의 결과 값이 동일하게 출력됨
  - 따라서 seed는 매번 다른 값을 사용하려 하는데, 이를 위해 time(0)가 seed로 많이 사용됨

- 일반적으로 srand(time(0))을 호출해 초기화한 후, rand()를 연속적으로 호출해 랜덤 숫자를 생성함

랜덤 숫자의 범위를 조절하는 방법

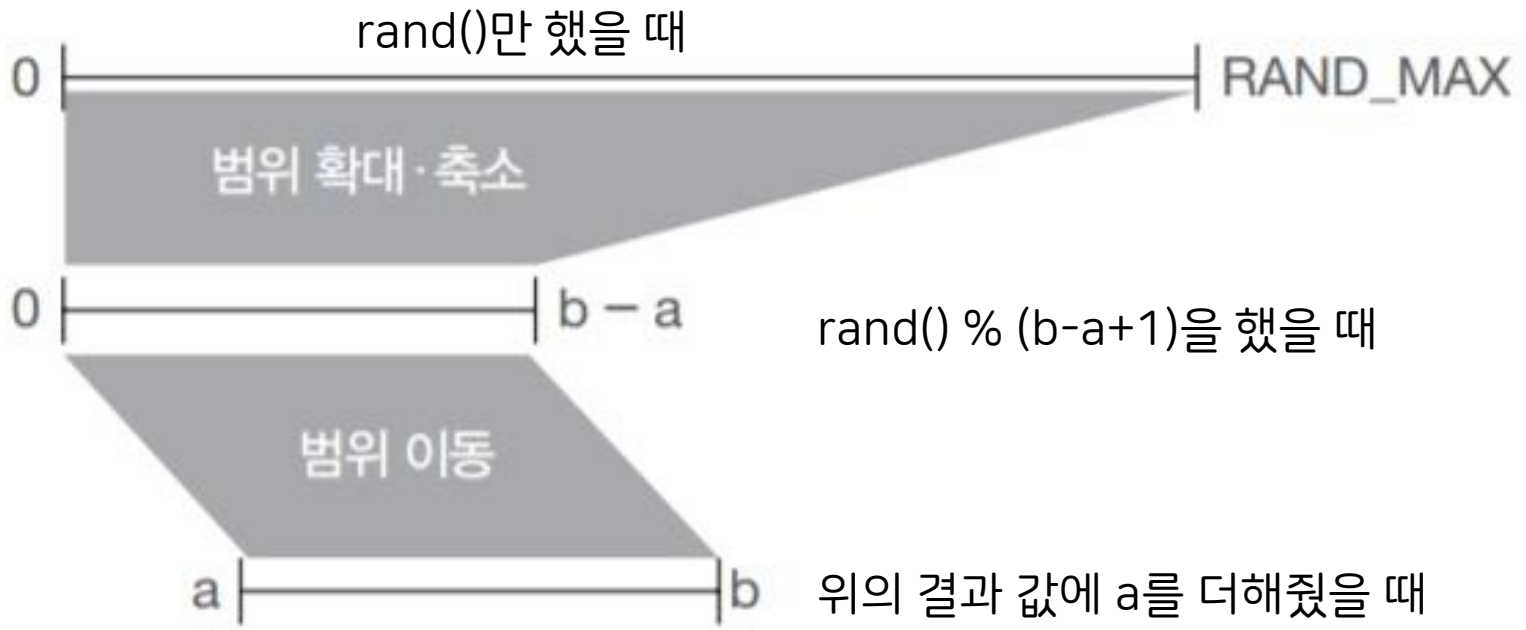
- 랜덤 값을 생성한 후, 범위를 조절하여 원하는 범주의 값이 나오도록 만들 수 있음
  - 주로 나눗셈 연산과 덧셈, 뺄셈 등이 사용됨

범위 확대·축소

```
temp = rand( ) % (b – a + 1)
```

범위 이동

```
result = temp + a
```



예제 3. 숫자 추측 게임 (1/4)

```
#define INSTRUCTION "1~15 사이의 정수를 입력하세요: "
#define SMALL_NUM "더 작은 숫자입니다."
#define BIG_NUM "더 큰 숫자입니다."
#define SUCCESS "성공"
#define FAIL "실패.. 답 = "
// 입출력 함수를 사용하기 위한 헤더 파일을 포함시킨다.
#include <iostream>
// 랜덤 함수를 사용하기 위한 헤더 파일을 포함시킨다.
#include <cstdlib>
// 시간 함수를 사용하기 위한 헤더 파일을 포함시킨다.
#include <ctime>
using namespace std;

int main () {

    // 하한값을 저장할 변수를 선언하고 1로 초기화한다.
    int low = 1;
    // 상한값을 저장할 변수를 선언하고 15로 초기화한다.
    int high = 15;
```

예제 3. 숫자 추측 게임 (2/4)

```
// 재시도 횟수를 저장할 변수를 선언하고 5로 초기화한다.
int tryLimit = 5;

// srand와 time(0)를 이용해 랜덤 숫자를 초기화 한다.
srand(time(0));
// 랜덤 숫자를 저장할 변수를 선언하고, rand()로 랜덤 숫자를 생성해 저장한다.
int num = rand();
// 랜덤 숫자가 상한값과 하한값 사이의 값이 되도록 조절한다.
num = num % (high - low +1 ) + low;

// 시도 횟수를 세기 위한 카운터 변수를 생성하고 0으로 초기화 한다.
int counter = 0;
// 정답 여부를 관리할 부울타입 변수를 생성하고 false로 초기화 한다.
bool found = false;
```

예제 3. 숫자 추측 게임 (3/4)

```
// 반복문 시작
// 시도 횟수 변수가 재시도 횟수 변수보다 작고, 맞추지 못한 동안 반복한다.
while (counter < tryLimit && !found) {
    // 입력 안내문을 출력한다.
    cout << INSTRUCTION;
    // 사용자의 입력을 저장할 변수를 선언한다.
    int guess;
    // 사용자로부터 값을 입력받아 변수에 저장한다.
    cin >> guess;

    // 만일 사용자의 입력이 랜덤 숫자와 같다면
    if (guess == num) {
        // 정답 여부 관리 변수를 true로 변경한다.
        found = true;
    // 만일 사용자의 입력이 랜덤 숫자보다 크다면
    } else if (guess > num) {
        // SMALL_NUM을 출력한다.
        cout << SMALL_NUM << endl;
```

예제 3. 숫자 추측 게임 (4/4)

```
        // 그 외의 경우
        } else {
            // BIG_NUM을 출력한다.
            cout << BIG_NUM << endl;
        }
        // 시도 횟수 변수를 하나 증가시킨다.
        counter++;
    }
    // 추측에 성공한 경우 (= 정답 여부 관리 변수가 true인 경우)
    if (found) {
        // SUCCESS를 출력하고, 뒤에 endl을 출력한다.
        cout << SUCCESS << endl;
    // 추측에 실패한 경우
    } else {
        // FAIL을 출력하고, 뒤에 정답과 endl을 출력한다.
        cout << FAIL << num << endl;
    }
    return 0;
}
```

결과

```
1~15 사이의 정수를 입력하세요: 5
더 큰 숫자입니다.
1~15 사이의 정수를 입력하세요: 10
더 큰 숫자입니다.
1~15 사이의 정수를 입력하세요: 13
더 작은 숫자입니다.
1~15 사이의 정수를 입력하세요: 12
성공
```

사용자 정의 함수

- 사용자가 필요로 하는 모든 함수가 C++ 라이브러리에 정의되어 있지는 않음
- 따라서 필요한 함수를 직접 만들어야 하는 경우도 있음
- C++의 함수는 아래 4가지 패턴 중 하나

```
void 이름()
{
    ...
    return;
}
```

(a) 매개변수가 없는 void 함수

```
void 이름(자료형 parameter, ... )
{
    ...
    return;
}
```

(b) 매개변수가 있는 void 함수

```
자료형 이름()
{
    ...
    return value;
}
```

(c) 매개변수가 없지만 리턴값이 있는 함수

```
자료형 이름(자료형 parameter, ... )
{
    ...
    return value;
}
```

(d) 매개변수와 리턴값이 있는 함수

매개변수가 없는 void 함수

- 가장 기본적인 패턴은 매개변수가 없는 void 함수
- 이름 그대로 매개변수도 없고, 리턴값도 없음
- 이 패턴의 함수는 함수 내부에서 부가 작용을 일으키기 위해서만 사용
- 부가 작용을 일으키지 않는다면 함수에 의미가 없음
- 이러한 함수는 값을 리턴하지 않으므로, 값이 필요한 위치에 활용할 수 없음

```
void greeting() {
    cout <<"********************" << endl;
    cout <<"* 안녕하세요!*" << endl;
    cout <<"********************" ;
    return;
}
```

```
int main() {
    greeting(); //greeting 함수 호출
    return 0;
}
```

연습문제 2. 매개변수가 없는 void 함수 작성 및 호출

- Hello World 문자열 위아래로 30개의 *을 출력해봅시다.
  - 가로로 30개의 `*`을 출력하고 줄바꿈 문자를 출력하는 `void print_stars()`함수를 작성합니다.
  - print_stars()를 Hello World 문자열의 위와 아래에서 호출해봅시다.
  - 이 프로그램의 실행결과는 다음과 같아야 합니다.

```
**********
Hello World
**********
```

main 함수와 greeting 함수 사이의 커뮤니케이션

```
int main()
{
    greeting();
    return 0;
}
```

```
void greeting()
{
    ...
    cout << ... ;
    ...
    return;
}
```

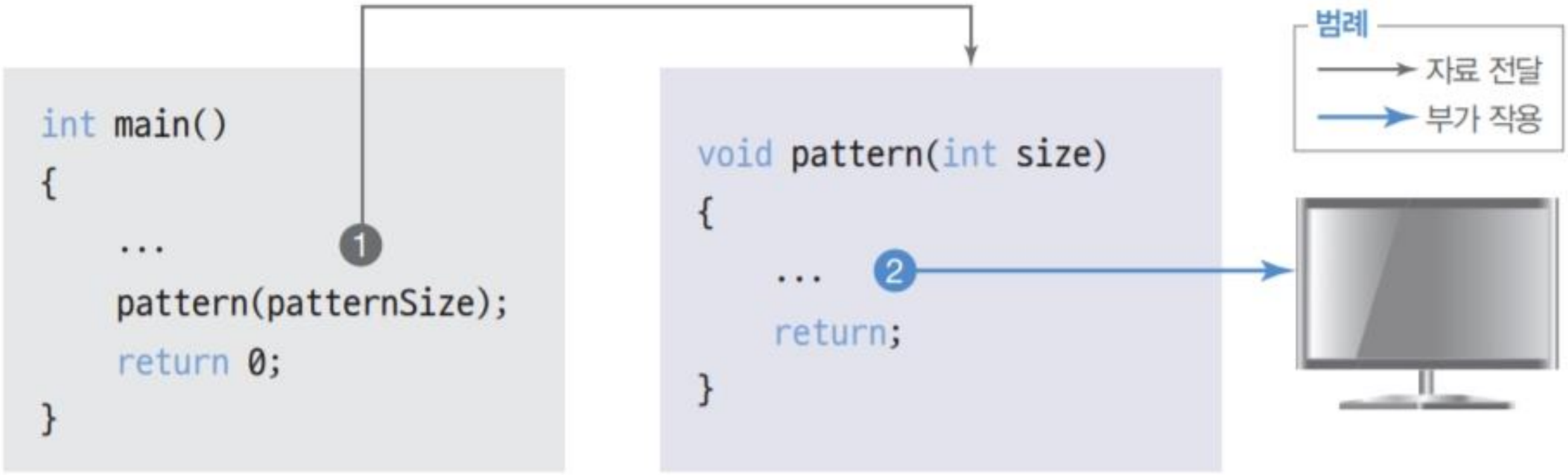
매개변수가 있는 void 함수

- 이 함수는 매개변수(parameter)로 값(argument)을 전달해서 활용하는 함수
- 이러한 함수는 부가 작용을 일으키지만, 리턴값을 리턴하지는 않음
- 함수가 부가 작용을 일으키고, 이 부가 작용에 추가적인 정보가 필요한 경우에 사용하는 패턴
- 일반적으로 매개변수가 있는 void 함수는 '입력-처리-출력'이라는 설계에서 출력 부분에 많이 사용

```
void pattern (int size) {
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            cout << "*" ;
        }
        cout << endl;
    }
    return;
}
```

```
int main() {
    int patternSize; // 함수에 전달할 값
    // 입력 유효성 검사
    do {
        cout << "패턴의 크기를 입력하세요:";
        cin >> patternSize;
    } while (patternSize <=0);
    // 함수 호출
    // 이때 patternSize는 인수(argument)
    pattern (patternSize);
    return 0;
}
```

main 함수와 pattern 함수의 커뮤니케이션



매개변수가 없지만 리턴값이 있는 함수

- 이번 패턴은 리턴값을 목적으로 사용하는 함수
- 이러한 함수는 일반적으로 함수 내부에서 입력을 받고 이를 리턴
- '입력-처리-출력'이라는 설계에서 입력 부분에 많이 사용

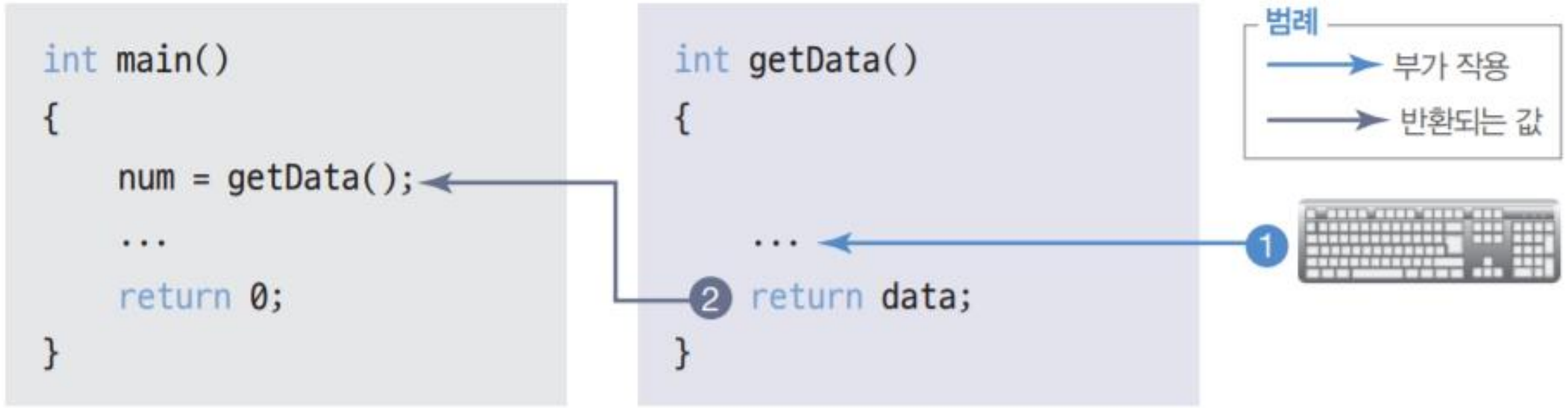
```
int getData() {
    int data;
    do {
        cout << "양의 정수를 입력하세요: ";
        cin >> data;
    } while (data <= 0);
    return data;
}
```

```
int main() {
    int number = getData(); // 매개변수 없이 함수 호출
    cout << "가장 오른쪽의 숫자 =  " << number % 10;
    return 0;
}
```

main 함수와 getData 함수의 커뮤니케이션

```
int main()
{
    num = getData();
    ...
    return 0;
}
```

```
int getData()
{
    ...
    return data;
}
```



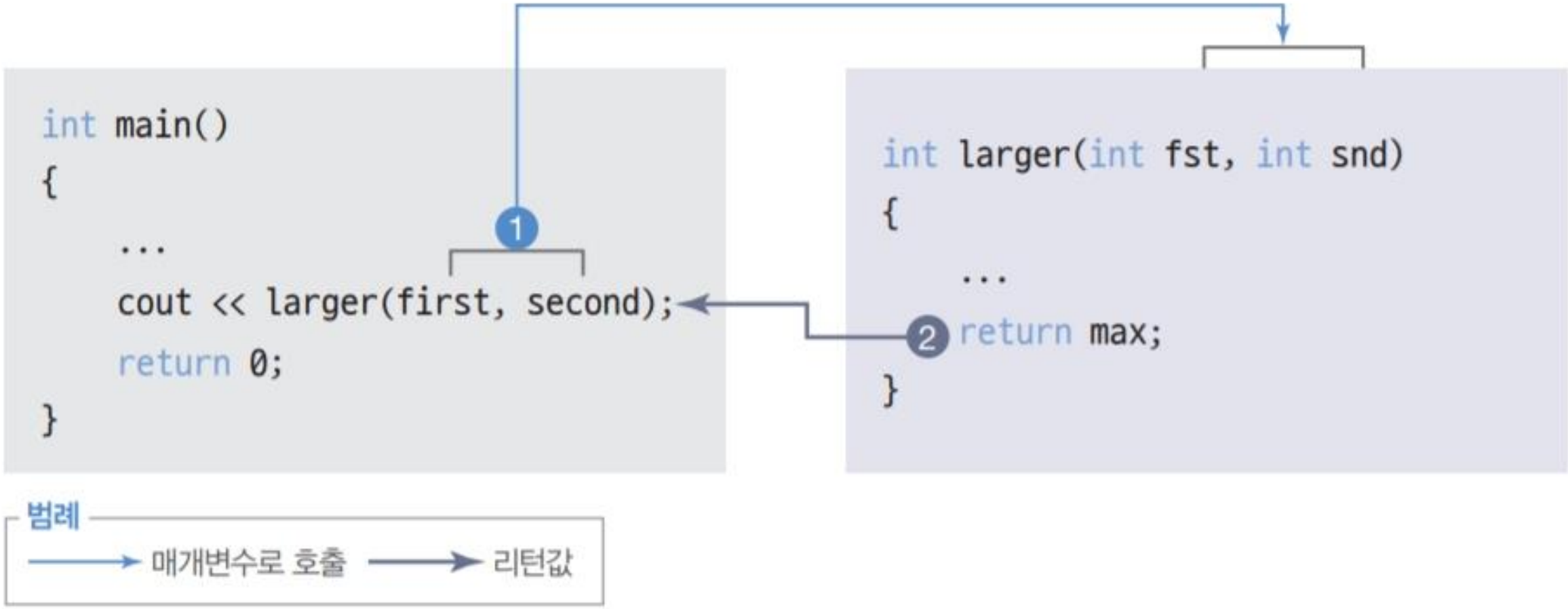
매개변수와 리턴값이 있는 함수

- 입력 값에 따라 결과가 달라지는 경우, 그 결과 값을 돌려받기 위해 사용
- '입력-처리-출력'이라는 설계에서 출력 부분에 많이 사용

```
int larger (int fst, int snd) {
    int max;
    if (fst > snd) {
        max = fst;
    } else {
        max = snd;
    }
    return (max);
}
```

```
int main() {
    // 선언
    int first, second;
    // 입력받기
    cout << "첫 번째 숫자를 입력하세요: ";
    cin >> first;
    cout << "두 번째 숫자를 입력하세요: ";
    cin >> second;
    // 함수 호출
    cout << "두 수 중에 큰 것 =  " << larger (first, second);
    return 0;
}
```

main 함수와 larger 함수의 커뮤니케이션



연습문제 3. 두 수를 합해 반환하는 함수

- 사용자로부터 두 수를 입력받은 후 함수를 이용해 덧셈을 처리하는 프로그램을 작성해봅시다.

  - get_sum()이라는 함수를 정의합니다.
    - get_sum() 함수는 인자로 두 정수를 받고, 그들을 더한 결과를 정수 형태로 반환합니다.

  - main() 함수에서 사용자로부터 두 정수를 입력받습니다.
  - 입력받은 정수를 get_sum()의 인자로 넘겨주고 덧셈의 결과를 돌려받습니다.
  - 돌려받은 결과를 화면에 출력합니다.

```
> ./main
두 수를 입력하세요(a b): 3 8
두 수의 합: 11
> ./main
두 수를 입력하세요(a b): 11 20
두 수의 합: 31
```


HANYANG
UNIVERSITY