Power C++

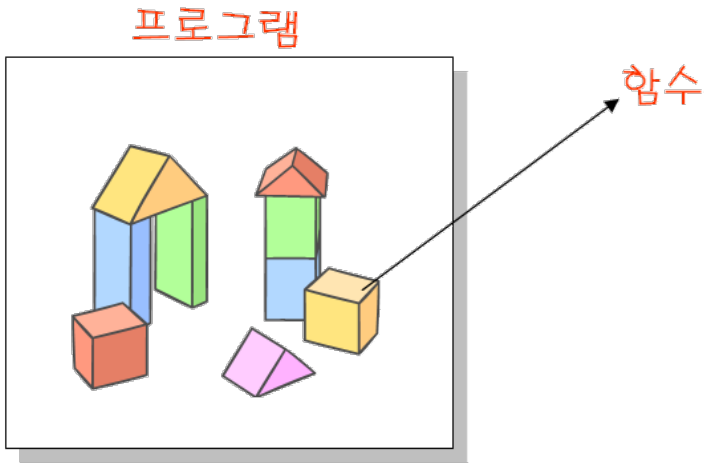
제4장 함수

이번 장에서 학습할 내용

- 함수의 선언 및 정의
- 매개 변수와 반환값
- 전역 변수
- 지역 변수
- 디폴트 매개 변수
- 중복 함수

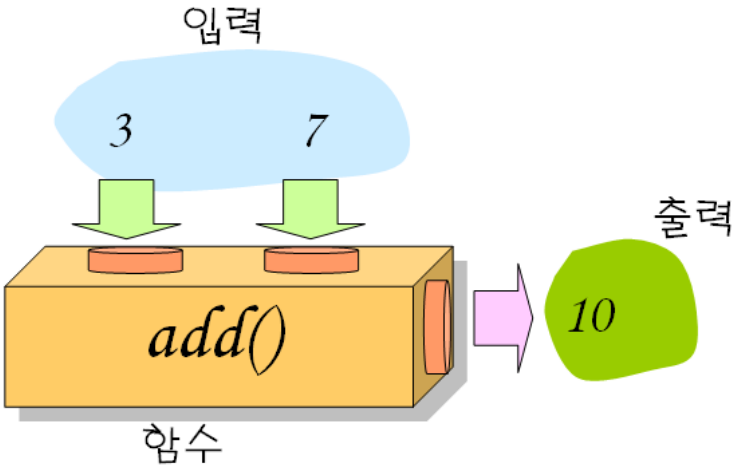
모듈의 개념

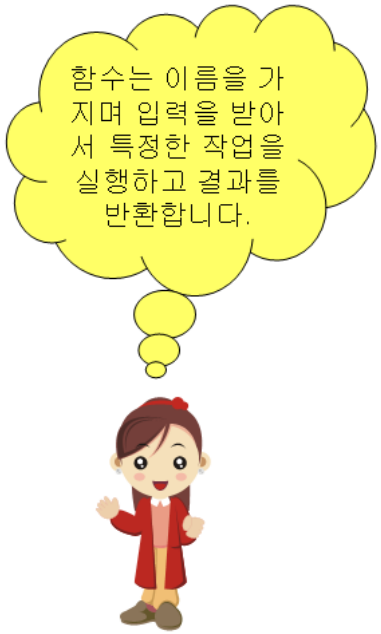
- *모듈(module)*
  - 독립되어 있는 프로그램의 일부분
- *모듈러 프로그래밍*
  - 모듈 개념을 사용하는 프로그래밍 기법
- *모듈러 프로그래밍의 장점*
  - 각 모듈들은 독자적으로 개발 가능
  - 다른 모듈과 독립적으로 변경 가능
  - 유지 보수가 쉬워진다.
  - 모듈의 재사용 가능
- C++에서는 *모듈==함수*



함수의 개념

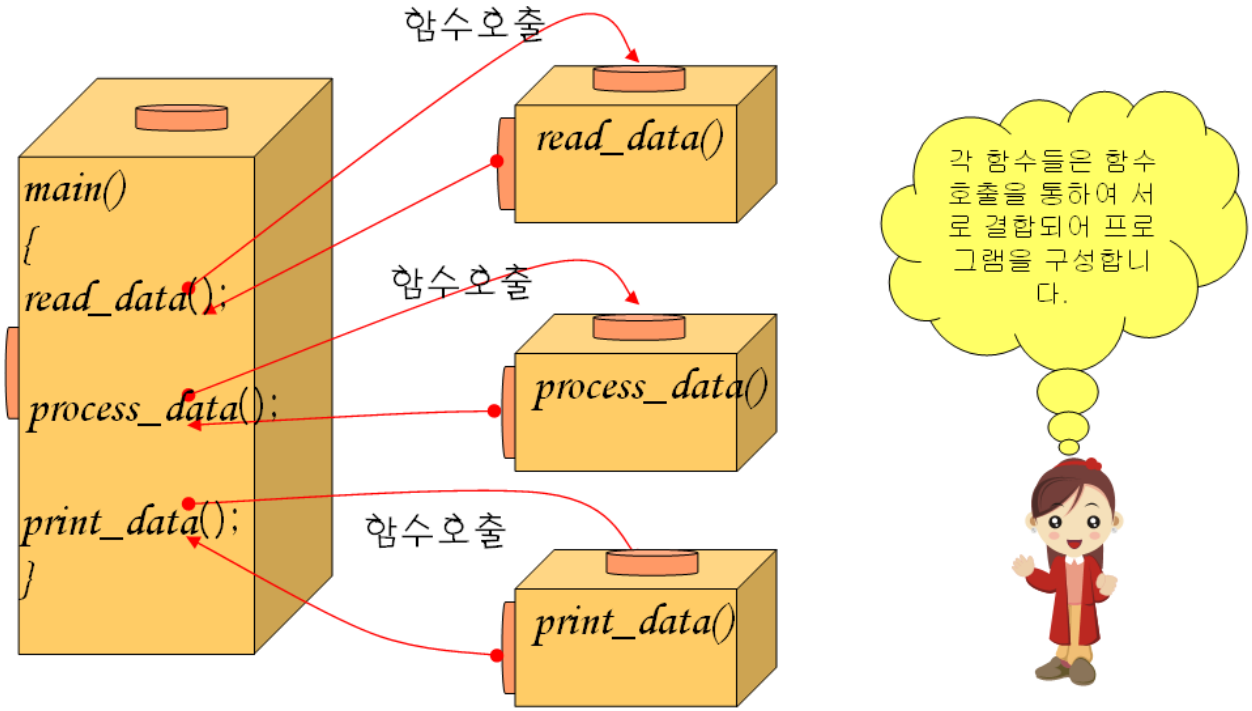
- *함수(function)*: 특정한 작업을 수행하는 독립적인 부분
- *함수 호출(function call)*: 함수를 호출하여 사용하는 것
- 함수는 입력을 받으며 출력을 생성한다.



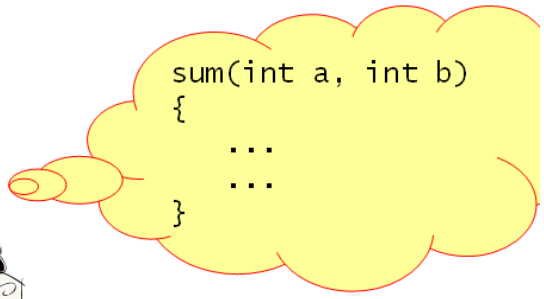


함수들의 연결

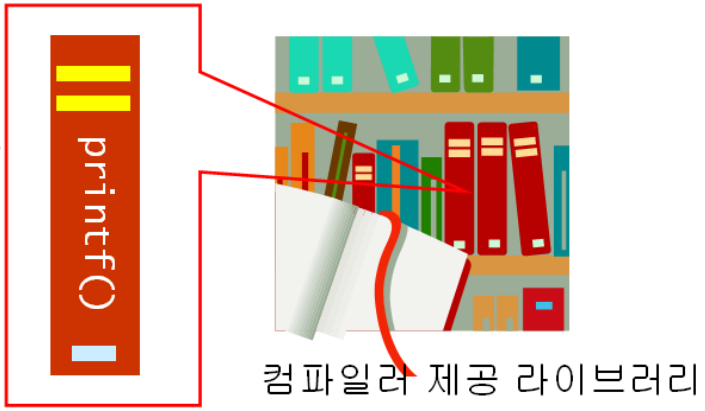
- 프로그램은 여러 개의 함수들로 이루어진다.
- 함수 호출을 통하여 서로 서로 연결된다.
- 제일 먼저 호출되는 함수는 main()이다.



함수의 종류



함수
- 사용자 정의 함수
- 라이브러리 함수

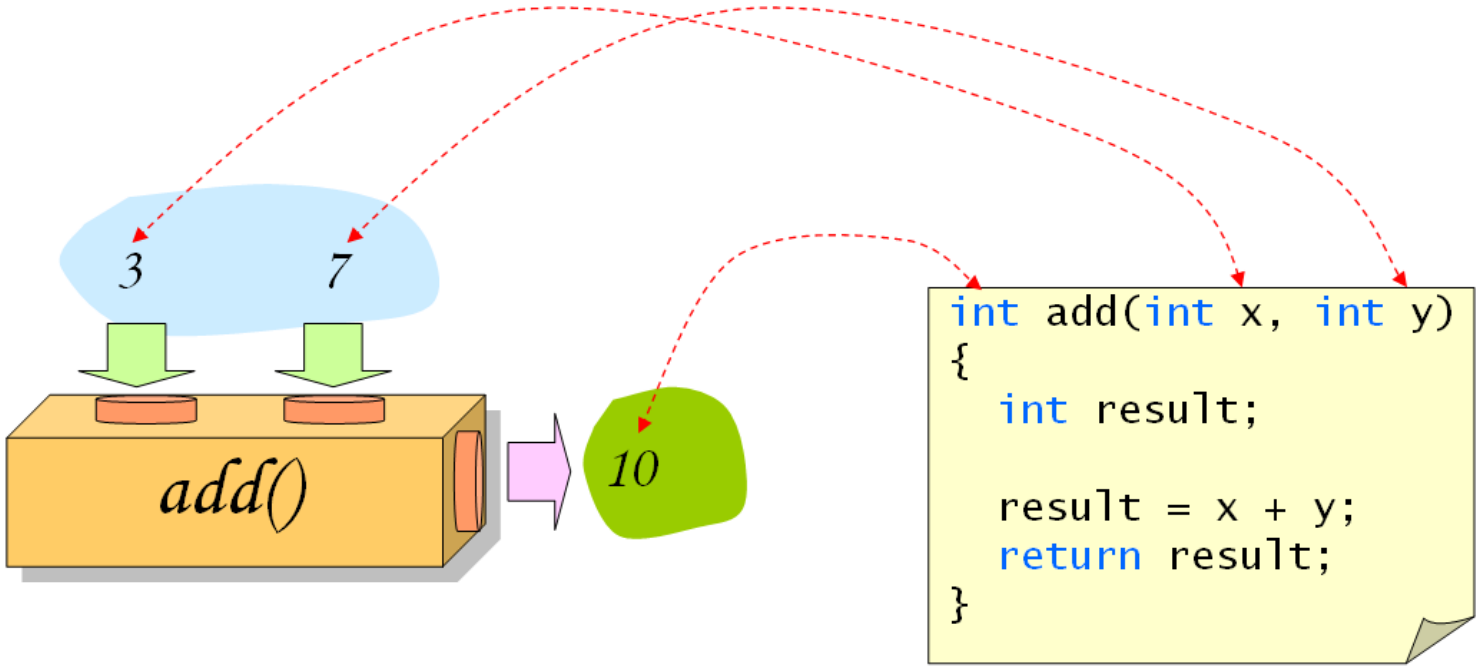


중간 점검 문제

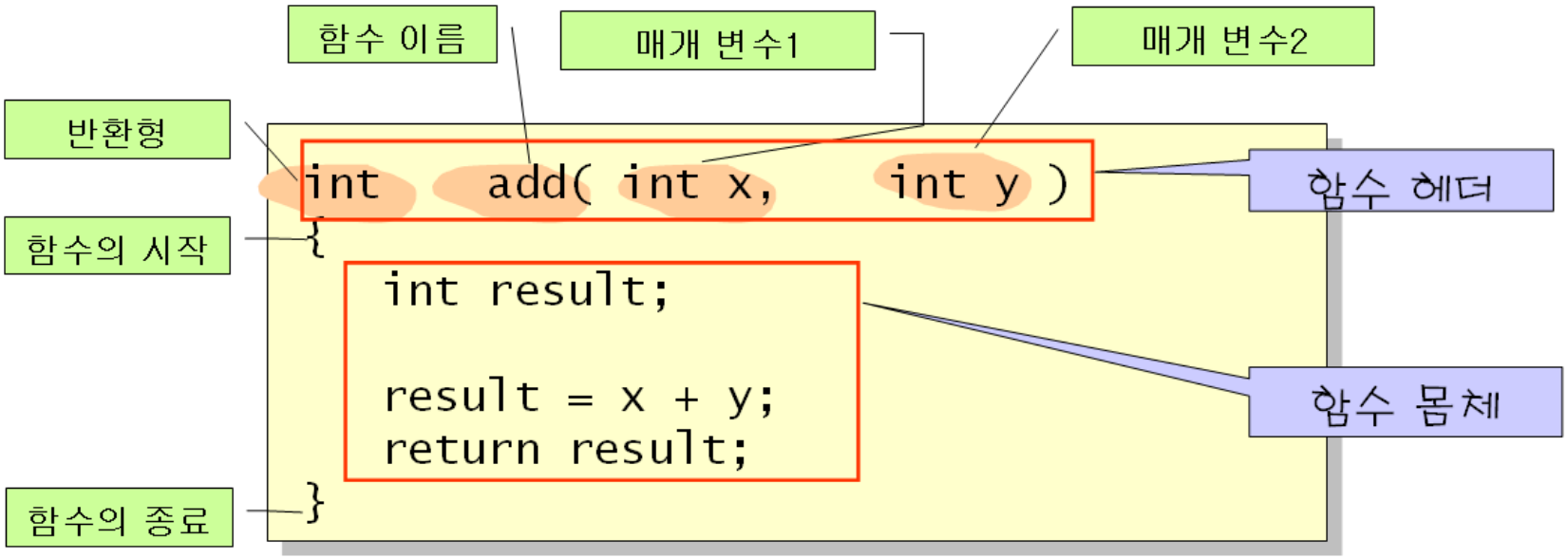
1. 함수와 프로그램의 관계는?
2. 컴파일러에서 지원되는 함수를 _____함수라고 한다.

함수의 정의

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

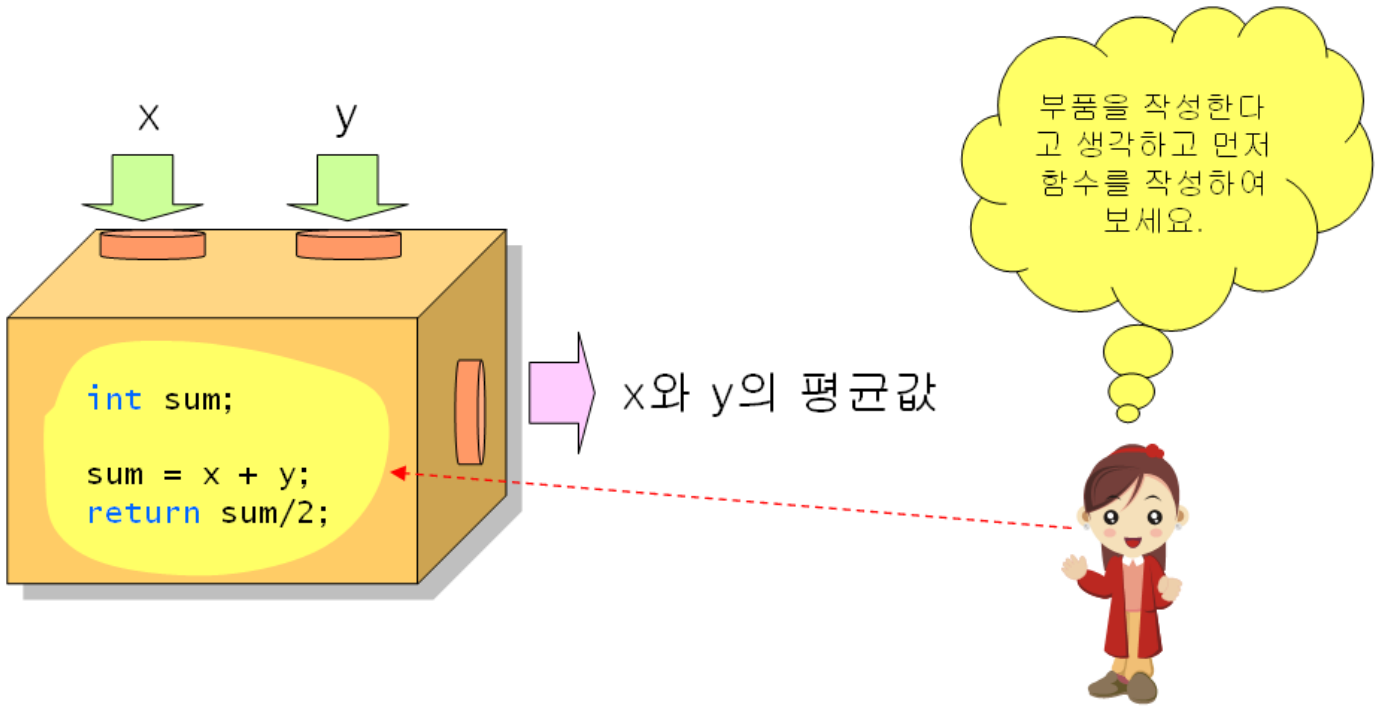


함수의 구조



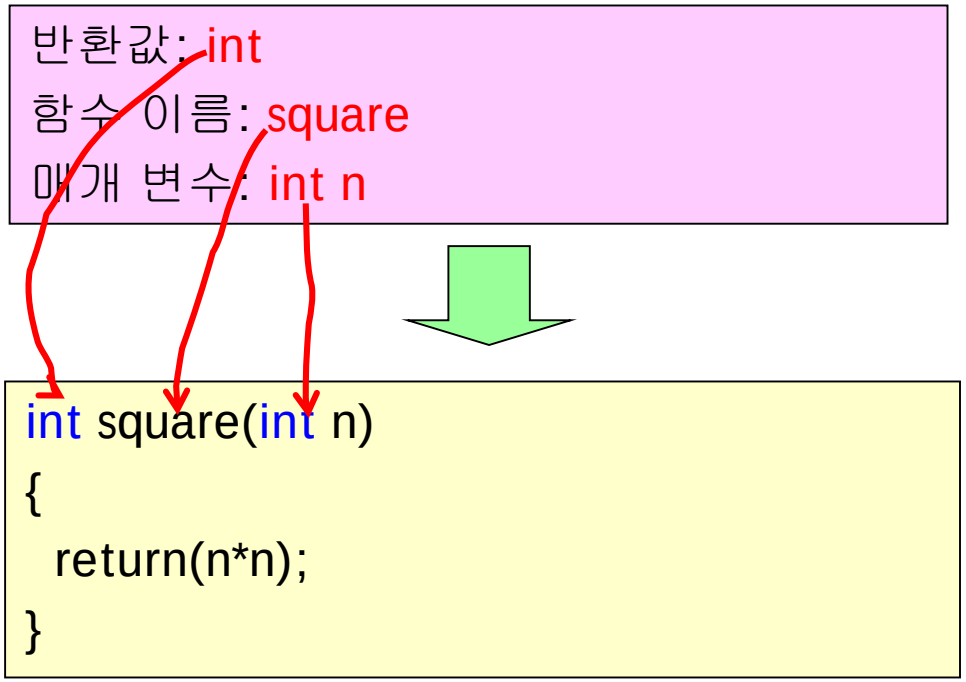
함수 정의 예제

- 함수를 프로그램을 이루는 부품이라고 가정하자.
- 입력을 받아서 작업한 후에 결과를 생성한다.



예제 #1

- 정수의 제곱값을 계산하는 함수



예제 #2

- 두개의 정수중에서 큰 수를 계산하는 함수

반환값: int
함수 이름: get_max
매개 변수: int x, int y

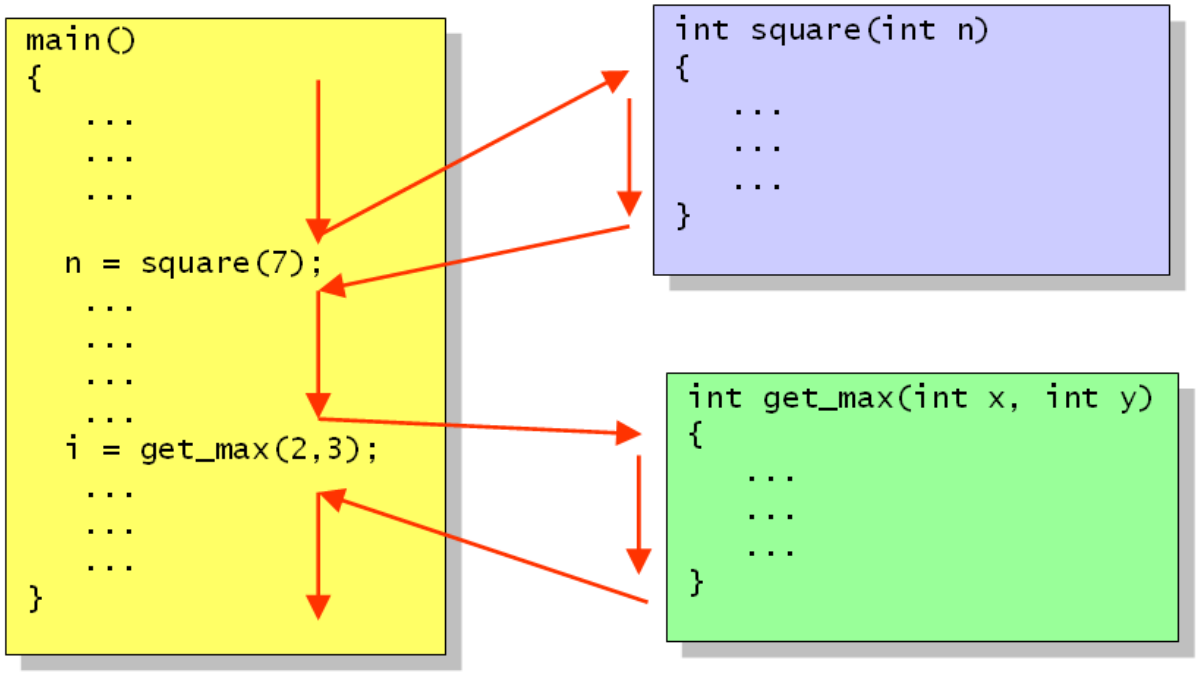
```
int get_max(int x, int y)
{
  if( x > y ) return(x);
  else return(y);
}
```

중간 점검 문제

1. 함수 이름 앞에 void가 있다면 무슨 의미인가?
2. 함수가 작업을 수행하는데 필요한 데이터로서 외부에서 주어지는 것을 무엇이라고 하는가?
3. 함수 몸체는 어떤 기호로 둘러 싸여 있는가?
4. 함수의 몸체 안에서 정의되는 변수를 무엇이라고 하는가?

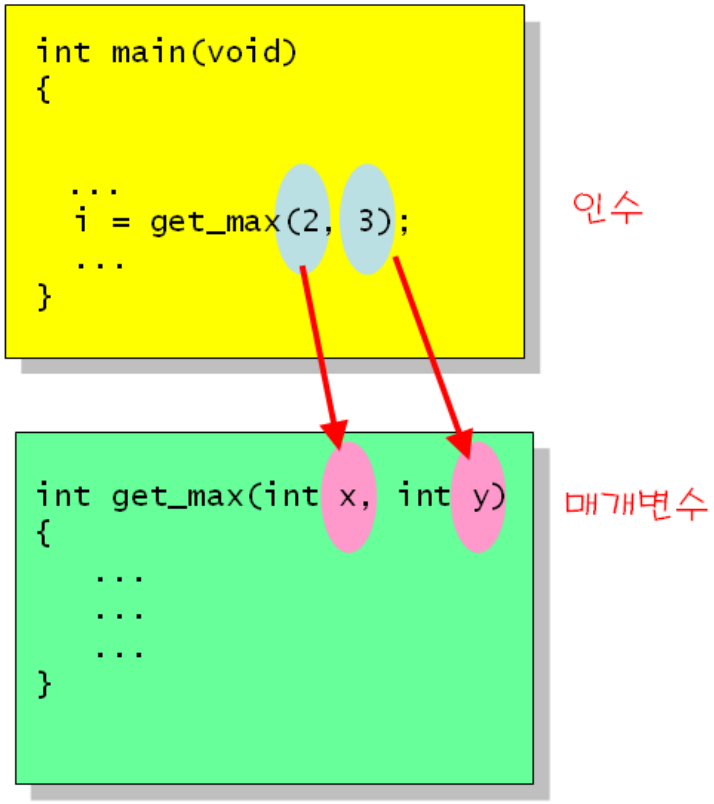
함수 호출과 반환

- *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

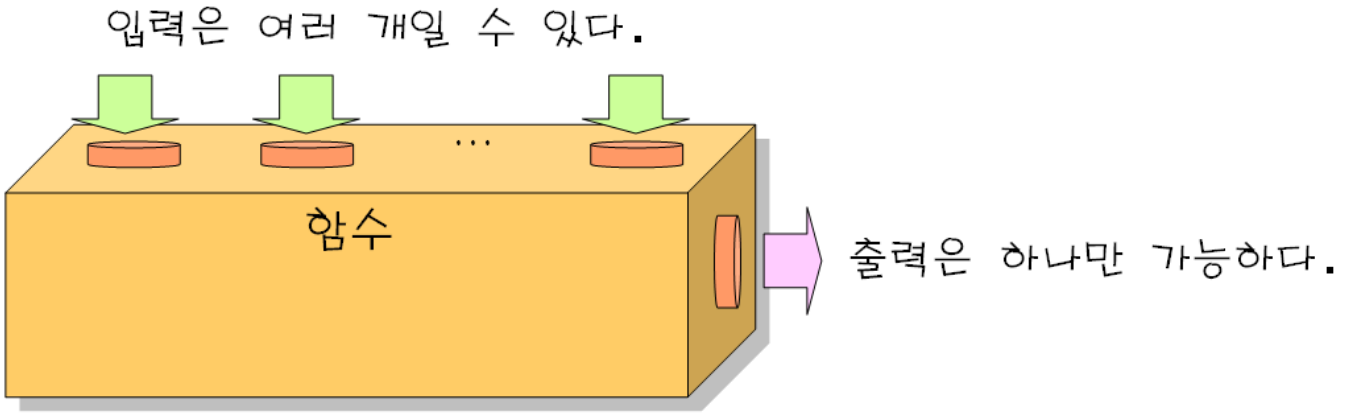
- *인자(argument):* 실인수, 실매개 변수라고도 한다.
- *매개 변수(parameter*): 형식 인수, 형식 매개 변수라고도 한다.




```
#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);                                  // 함수 원형
int main()
{
        int i, result;
        for(i = 0; i < 5; i++)
        {
                result = square(i);                 // 함수 호출
                cout <<  result << endl;
        }
        return 0;
}
int square(int n)                                   // 함수 정의
{
        return(n * n);
}
```

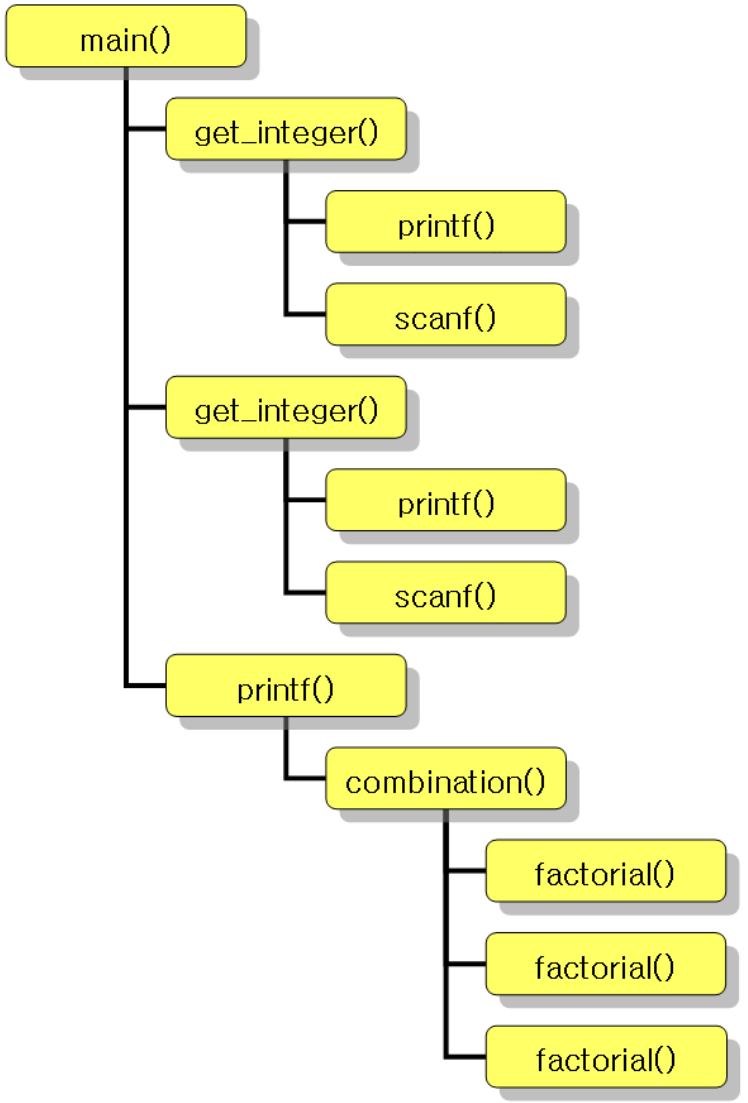
함수 원형

조합(combination) 계산 함수

$$C(n,r) = \frac{n!}{(n-r)!r!}$$

$$C(3,2) = \frac{3!}{(3-2)!2!} = \frac{6}{2} = 3$$

- 팩토리얼 계산 함수와 get_integer() 함수를 호출하여 조합을 계산한다



예제

```
// 수학적인 조합 값을 구하는 예제
#include <iostream>
using namespace std;

// 함수 원형 정의
int get_integer(void);
int combination(int n, int r);
int factorial(int n);

int main()
{
    int a, b;

    a = get_integer();
    b = get_integer();

    cout << "C(" << a << "," << b << ") = " << combination(a, b) << endl;
    return 0;
}
```

예제

```
int combination(int n, int r)
{
      return (factorial(n)/(factorial(r) * factorial(n-r)));
}
int get_integer(void)
{
      int n;
      cout << "정수를 입력하시오: ";
      cin >> n;
      return n;
}
int factorial(int n)
{
      int i;  long result = 1;
      for(i = 1; i <= n; i++)
            result *= i;                        // result = result * i
      return result;
}
```

```
정수를 입력하시오: 10
정수를 입력하시오: 3
C(10, 3) = 120
```

중간 점검 문제

1. 실인수와 형식 매개 변수는 어떤 관계가 있는가?
2. 인수와 매개 변수가 값을 주고받는 방법에는 ___에 의한 호출과 ___에 의한 호출의 2가지 방법이 있다.
3. 반환값이 실수로 정의된 함수에서 실수로 정수를 반환하면 어떤 일이 발생하는가?

함수 원형

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

```
반환형   함수이름(매개변수1,  매개변수2, ... );
```

```
#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);

함수 원형 예제

```
#include <iostream>
using namespace std;
// 함수 원형
int compute_sum(int n);
int main(void)
{
	int n, sum;
	cout << "정수를 입력하시오: ";
	cin >> n;
	sum = compute_sum(n);			// 함수 사용
	cout << "1부터 " << n << "까지의 합은 " << sum << "입니다. \n";
	return 0;
}
int compute_sum(int n)
{
	int i;
	int result = 0;

	for(i = 1; i <= n; i++)
		result += i;
	return result;
}
```

```
정수를 입력하시오: 10
1부터 10까지의 합은 55입니다.
```

중간 점검 문제

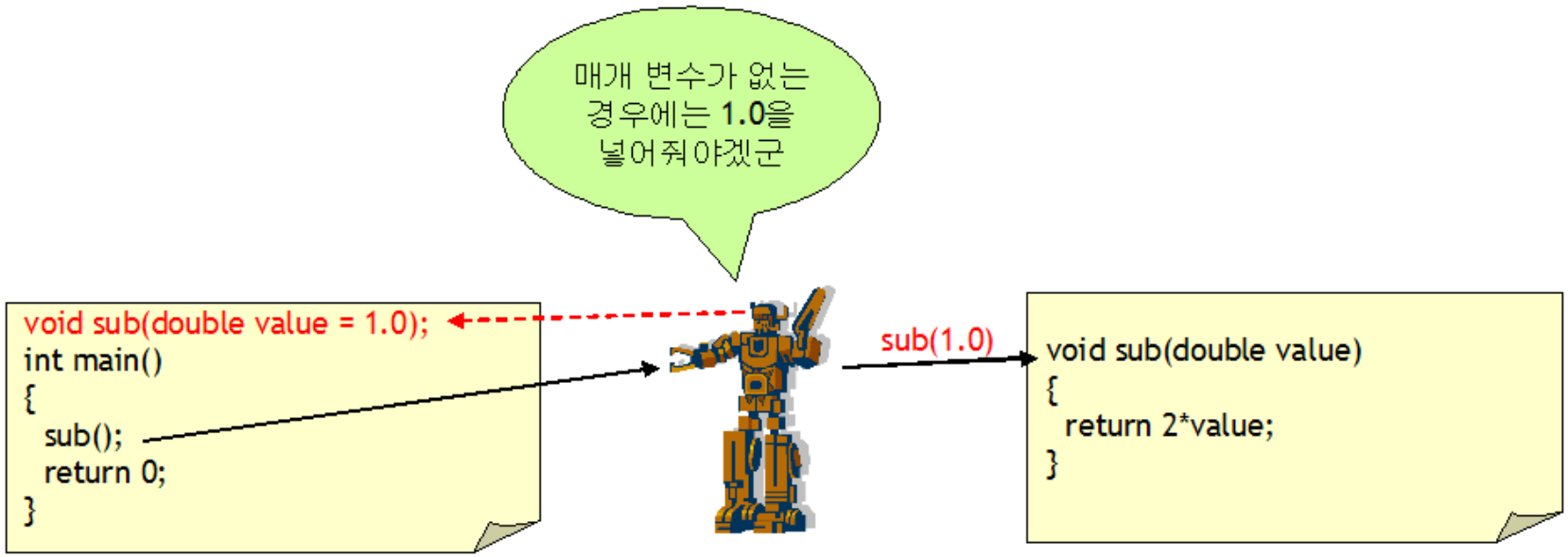
1. 함수 원형을 사용하지 않으려면 어떻게 하면 되는가?
2. 함수 정의의 첫 번째 줄에는 어떤 정보들이 포함되는가? 이것을 무엇이라고 부르는가?
3. 함수가 반환할 수 있는 값의 개수는?
4. 함수가 값을 반환하지 않는다면 반환형은 어떻게 정의되어야 하는가?
5. 함수 정의와 함수 원형의 차이점은 무엇인가?
6. 함수 원형에 반드시 필요한 것은 아니지만 대개 매개 변수들의 이름을 추가하는 이유는 무엇인가?
7. 다음과 같은 함수 원형을 보고 우리가 알 수 있는 정보는 어떤 것들인가?

```
double pow(double, double);
```

디폴트 매개 변수

- 디폴트 매개 변수(default parameter): 인자를 전달하지 않아도 디폴트값을 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!
```

예 제

```
#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 제곱하는 함수
int square(int i)
{
    return i*i;
}
// 실수값을 제곱하는 함수
double square(double i)
{
    return i*i;
}
```

예 제

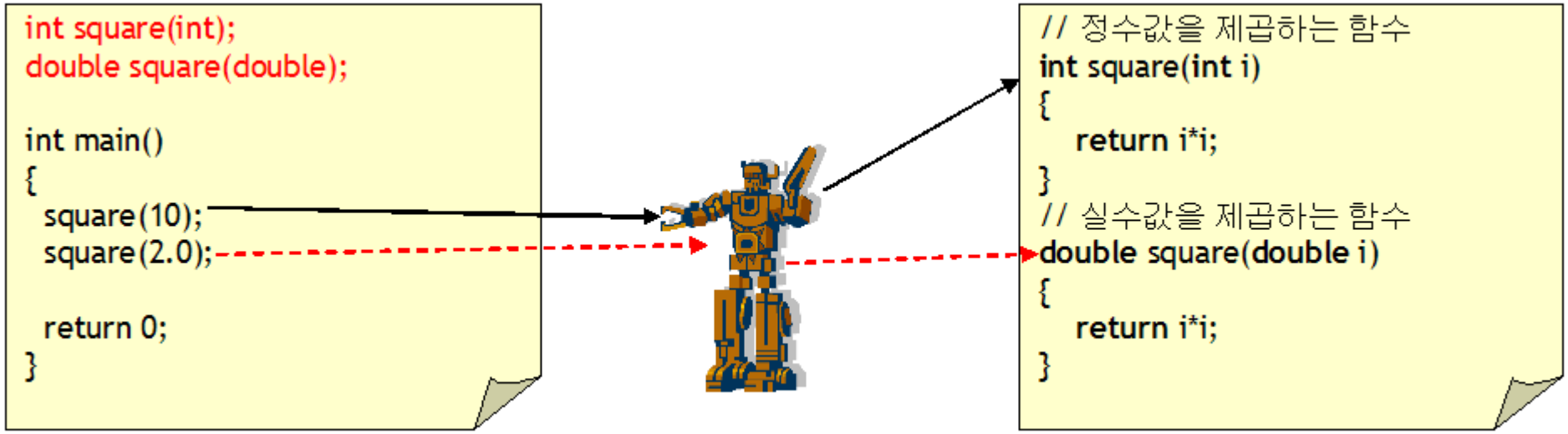


그림 4.10 중복 함수의 개념

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);

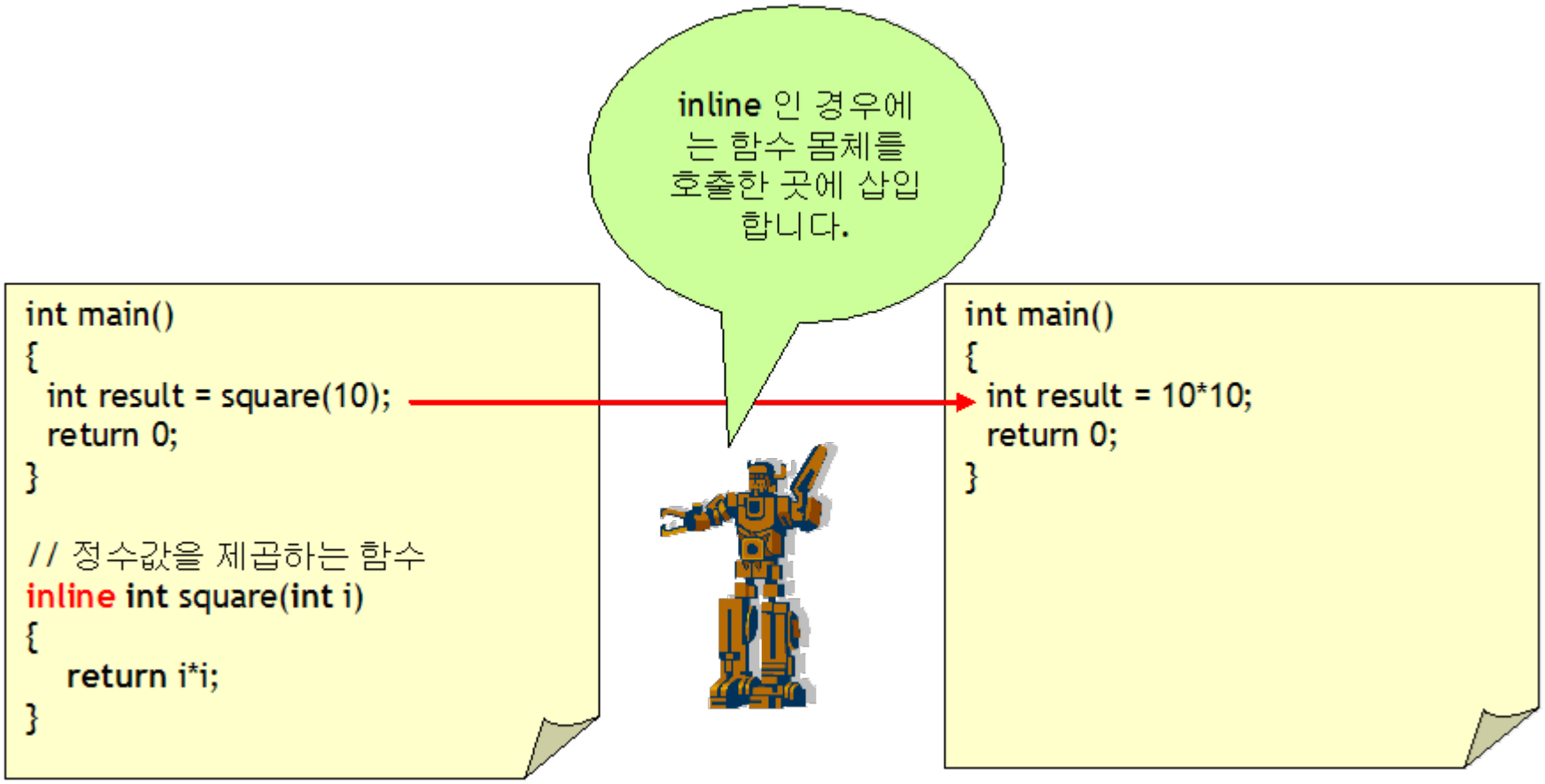
- int sub(int, int);// 중복 가능!
- int sub(int, double);// 중복 가능!
- double sub(double);// 중복 가능!

- double sub(int);// 오류!! - 반환형이 다르더라도 중복 안됨!
- float sub(int, int);// 오류!! - 반환형이 다르더라도 중복 안됨!

인라인 함수

- 인라인 함수(inline function):

함수 호출을 하지 않고 코드를 복사하여서 넣는 것



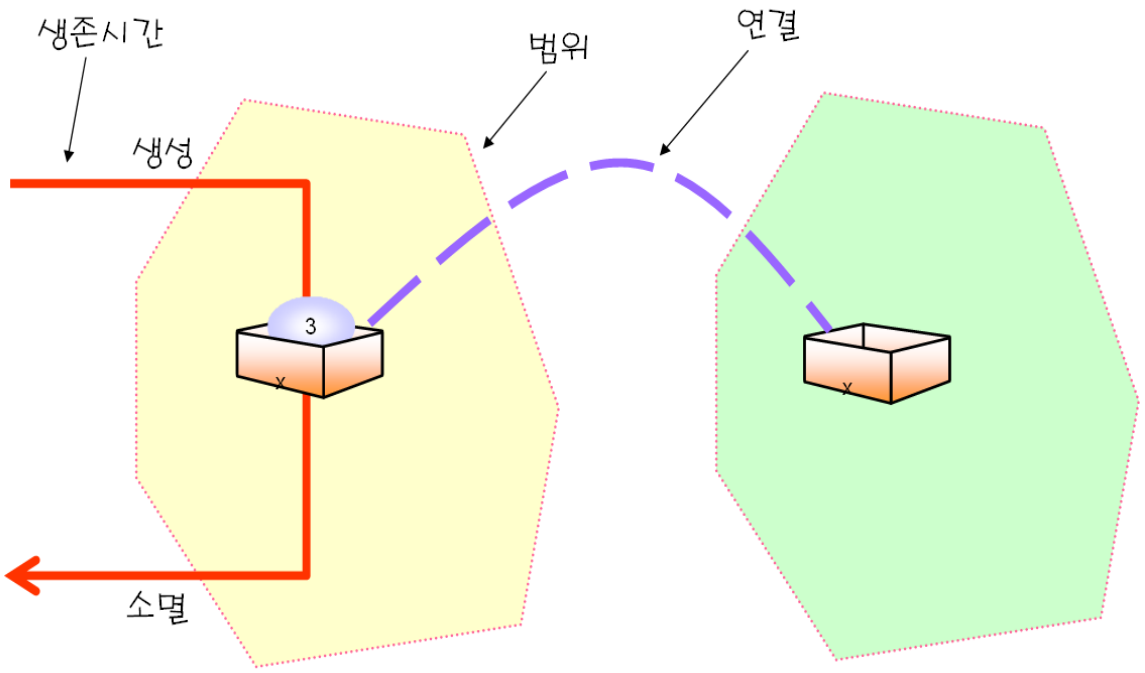
예제

```
#include <iostream>
using namespace std;
// 정수값을 제곱하는 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 . . .
```

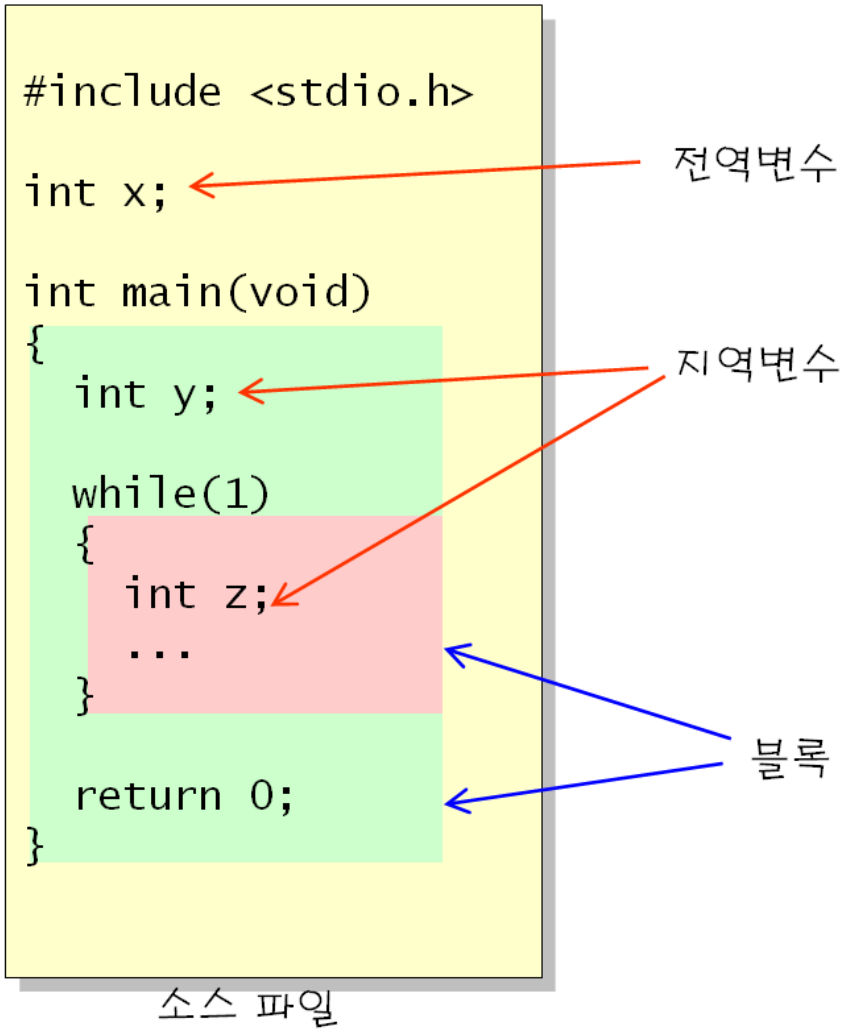
변수의 속성

- 변수의 속성 : 이름, 타입, 크기, 값 + 범위, 생존 시간, 연결
  - 범위(scope) : 변수가 사용 가능한 범위, 가시성
  - 생존 시간(lifetime): 메모리에 존재하는 시간
  - 연결(linkage): 다른 영역에 있는 변수와의 연결 상태



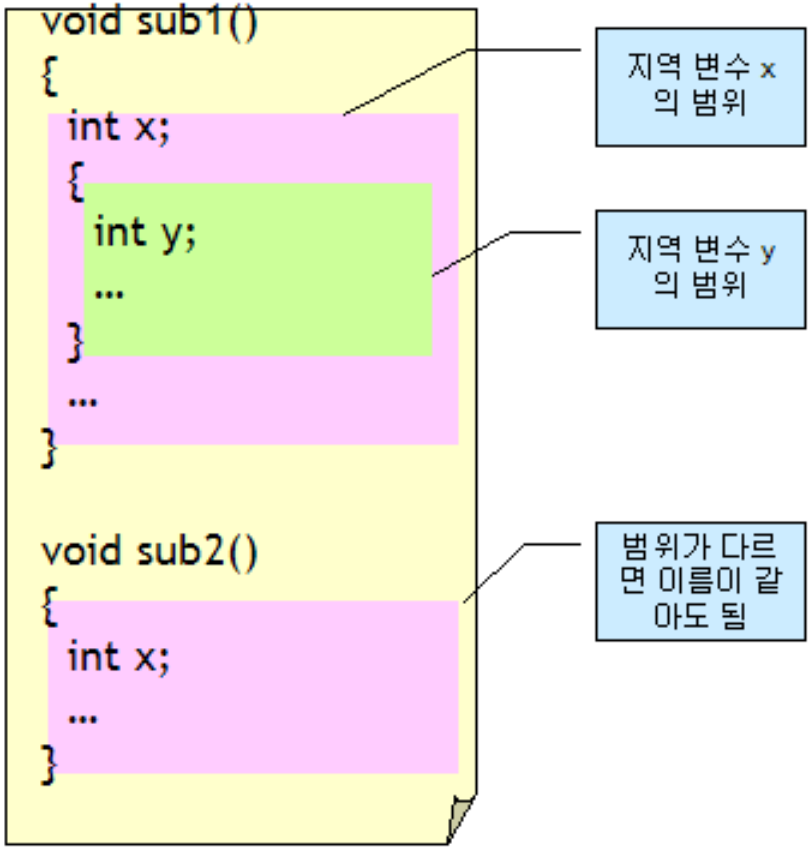
범위

- 범위(scope): 변수가 접근되고 사용되는 범위
- 범위는 선언되는 위치에 따라서 결정된다.
- 범위의 종류
  - 전역 변수: 함수의 외부에서 선언,
  - 지역 변수: 블록 안에서 선언



지역 변수

- 지역 변수(local variable): 블록 안에서 선언되는 변수



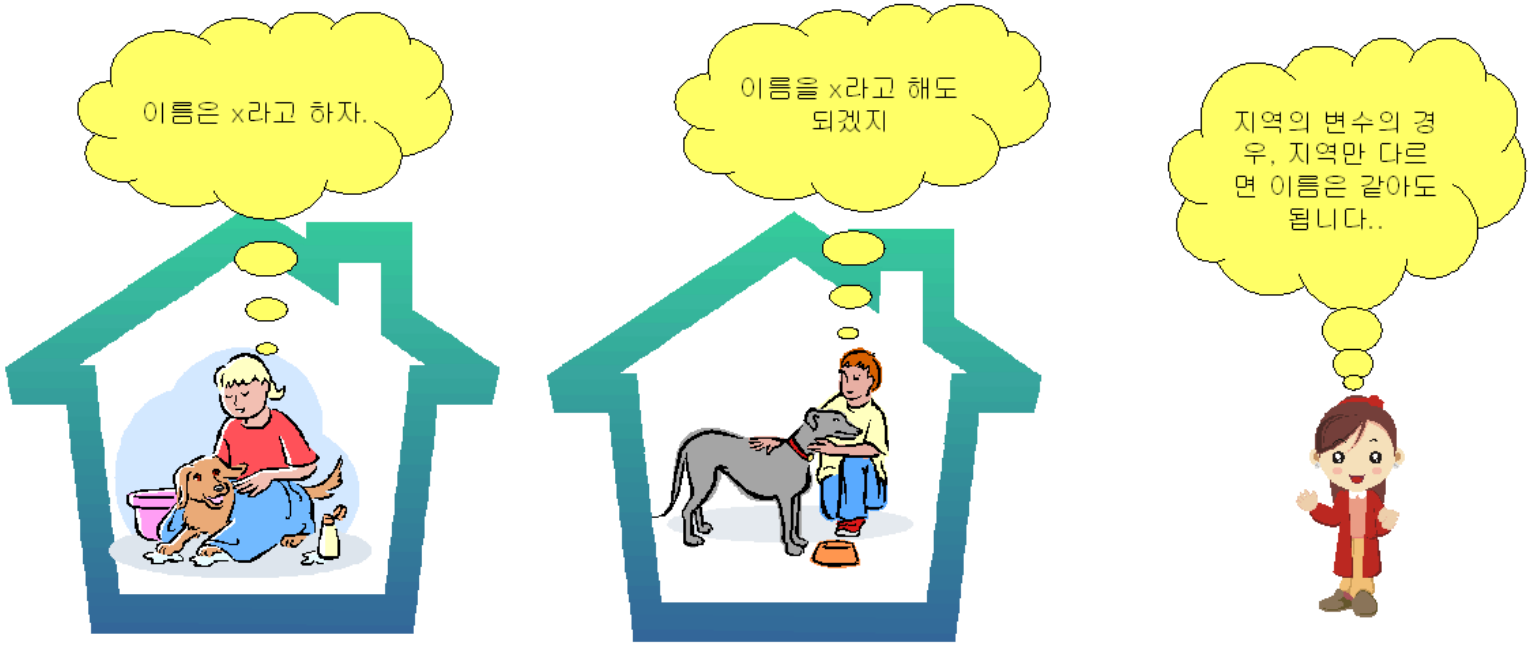
예제

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

int main()
{
    int i;

    for(i = 0;i < 5; i++)
    {
        int temp = 1;
        cout << "temp = " << temp << endl;
        temp++;
    }
}
```

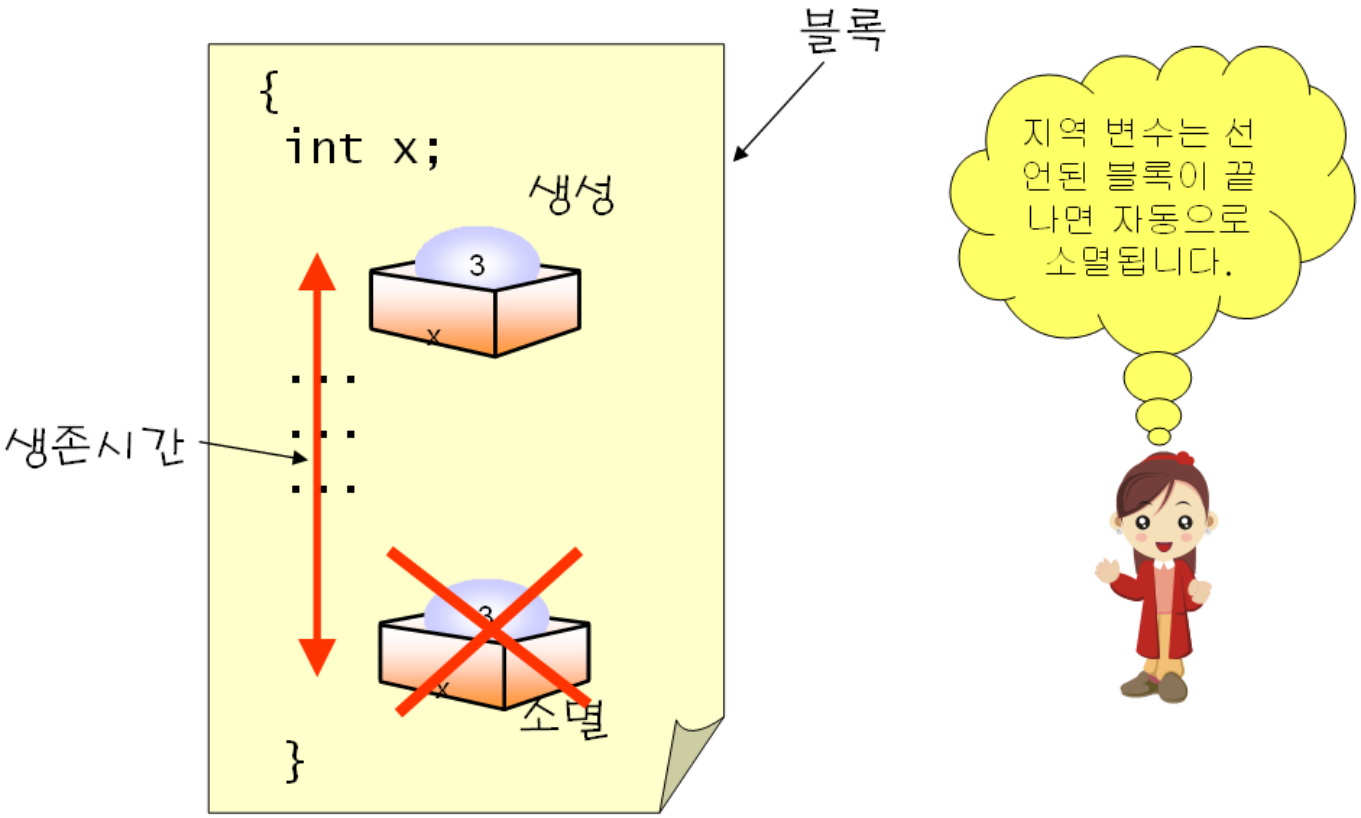
```
temp = 1
temp = 1
temp = 1
temp = 1
temp = 1
```

블록이 시작할 때마다 생성되어 초기화된다.

같은 이름의 지역 변수



지역 변수의 생존 기간



함수의 매개 변수

```
#include <iostream>
using namespace std;
int inc(int counter);
int main()
{
      int i;
      i = 10;
      cout << "함수 호출전 i=" << i << endl;
      inc(i);
      cout << "함수 호출후 i=" << i << endl;
      return 0;
}
int inc(int counter)
{
      counter++;
      return counter;
}
```

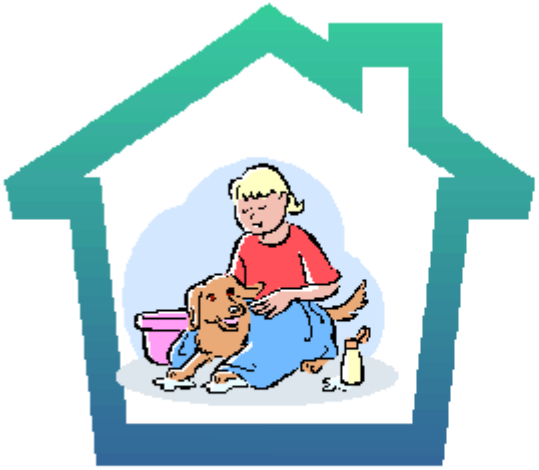
값에 의한 호출
(call by value)

매개 변수도 일종의
지역 변수임

```
함수 호출전 i=10
함수 호출후 i=10
```

전역 변수

- 전역 변수(global variable): 함수의 외부에서 선언되는 변수



지역변수

전역변수

전역 변수

```
#include <iostream>
using namespace std;
int x = 123;            // 전역 변수
void sub1()
{
    cout << "In sub1()  x=" << x << endl;  // 전역 변수 x 접근
}
void sub2()
{
    cout << "In sub2()  x=" << x << endl;  // 전역 변수 x 접근
}
int main(void)
{
    sub1();
    sub2();
    return 0;
}
```

전역 변수:
함수의 외부에
선언되는 변수

```
In sub1()   x=123
In sub2()   x=123
```

전역 변수의 초기값과 생존 기간

```
// 전역 변수의 초기값과 생존 기간
#include <iostream>
using namespace std;
int counter;          // 전역 변수
void set_counter(int i)
{
    counter = i;                  // 직접 사용 가능
}
int main(void)
{
    cout << "counter=" << counter << endl;

    counter = 100;                // 직접 사용 가능
    cout << "counter=" << counter << endl;

    set_counter(20);
    cout << "counter=" << counter << endl;
    return 0;
}
```

```
counter=0
counter=100
counter=20
```

* 전역 변수의 초기값은 0

*생존 기간은 프로그램 시작부터 종료

전역 변수의 사용

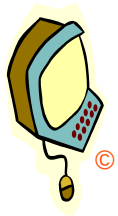
```
// 전역 변수를 사용하여 프로그램이 복잡해지는 경우
#include <iostream>
using namespace std;
void f(void);
int i;
int main(void)
{
        for(i = 0;i < 5; i++)
        {
                f();
        }
        return 0;
}
void f(void)
{
        for(i = 0;i < 10; i++)
                cout << "#";
}
```


```
##########
```

같은 이름의 전역 변수와 지역 변수

```
// 동일한 이름의 전역 변수와 지역 변수
#include <iostream>
using namespace std;

int sum = 1;                    // 전역 변수
int main()
{
    int i = 0;
    int sum = 0;                // 지역 변수

    for(i = 0; i <= 10; i++)
    {
        sum += i;               // 지역 변수가 전역 변수를 가린다.
    }
    cout << "sum = " << sum << endl;

    return 0;
}
```

지역 변수가
전역 변수를
가린다.



```
sum = 55
```

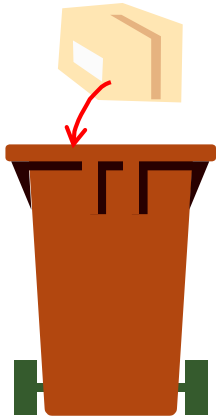
중간 점검 문제

1. 변수의 범위는 대개 무엇으로 결정되는가?
2. 변수의 범위에는 몇 가지의 종류가 있는가?
3. 파일 범위를 가지는 변수를 무엇이라고 하는가?
4. 블록 범위를 가지는 변수를 무엇이라고 하는가?
5. 지역 변수를 블록의 중간에서 정의할 수 있는가?
6. 똑같은 이름의 지역 변수가 서로 다른 함수 안에 정의될 수 있는가?
7. 지역 변수가 선언된 블록이 종료되면 지역 변수는 어떻게 되는가?
8. 지역 변수의 초기값은 얼마인가?
9. 함수의 매개 변수도 지역 변수인가?
10. 전역 변수는 어디에 선언되는가?
11. 전역 변수의 생존 기간과 초기값은?
12. 똑같은 이름의 전역 변수와 지역 변수가 동시에 존재 는가?

생존 기간

- 정적 할당(static allocation):
  - 프로그램 실행 시간 동안 계속 유지
- 자동 할당(automatic allocation):
  - 블록에 들어갈때 생성
  - 블록에서 나올때 소멸
- 생존 기간을 결정하는 요인
  - 변수가 선언된 위치
  - 저장 유형 지정자
- 저장 유형 지정자
  - auto
  - register
  - static
  - extern

변수 생성



변수 소멸

저장 유형 지정자 auto

- 변수를 선언한 위치에서 자동으로 만들어지고 블록을 벗어나게 되며 자동으로 소멸되는 저장 유형을 지정
- 지역 변수는 auto가 생략되어도 자동 변수가 된다.

```
int main()
{
  auto int sum = 0;
  int i = 0;
  ...
  ...
}
```

전부 자동 변수로서 함수가 시작되면 생성되고 끝나면 소멸된다.

저장 유형 지정자 static

```
#include <iostream>
using namespace std;
void sub(void);
int main()
{
      int i;
      for(i = 0;i < 5; i++)
            sub();
      return 0;
}
void sub(void)
{
      int auto_count = 0;
      static int static_count = 0;

      auto_count++;
      static_count++;
      cout << "auto_count=" << auto_count << endl;
      cout << "static_count=" << static_count << endl;
}
```

```
auto_count=1
static_count=1
auto_count=1
static_count=2
auto_count=1
static_count=3
...
```

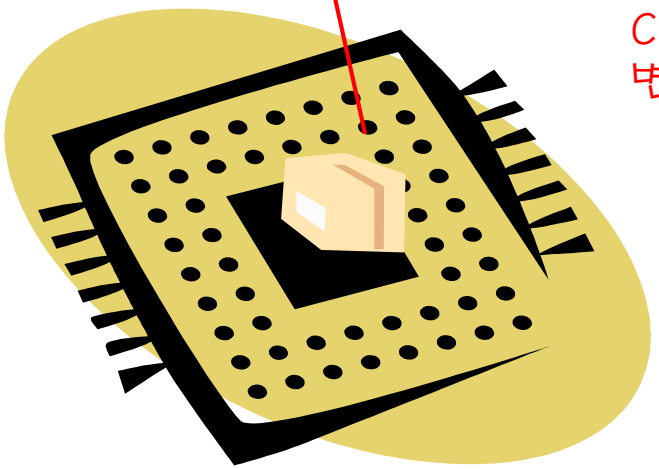
자동 지역 변수

정적 지역 변수로서
static을 붙이면 지역 변수가
정적 변수로 된다.

저장 유형 지정자 register

- 레지스터(register)에 변수를 저장.

```
register int i;
for(i = 0;i < 100; i++)
  sum += i;
```



CPU안의 레지스터에 변수가 저장됨

저장 유형 지정자 extern

extern1.c

컴파일러에게 변수가 다른 곳에서 선언되었음을 알린다

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

int x;              // 전역 변수
extern int y;       // 현재 소스 파일의 뒷부분에 선언된 변수
extern int z;       // 다른 소스 파일의 변수
int main(void)
{
        extern int x;    // 전역 변수 x를 참조한다. 없어도된다.
        x = 10;
        y = 20;
        z = 30;

        return 0;
}
int y;          // 전역 변수
```

extern2.c

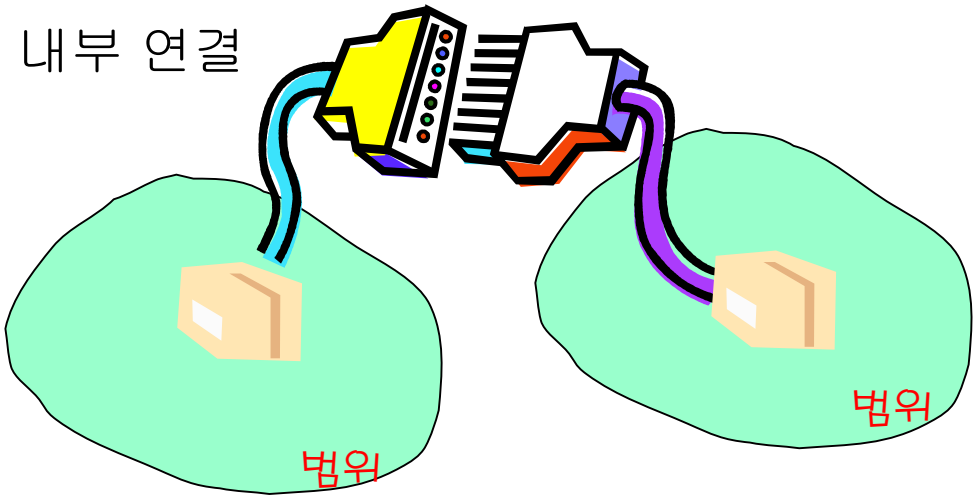
```
int z;
```

중간 점검 문제

1. 정적 할당과 자동 할당의 차이점은 무엇인가?
2. 변수에 공간을 할당하는 방법은 어떤 요인에 의하여 달라지는가?
3. 저장 유형 지정자에는 어떤 것들이 있는가?
4. 지역 변수를 정적 변수로 만들려면 어떤 지정자를 붙여야 하는가?
5. 변수를 CPU 내부의 레지스터에 저장시키는 지정자는?
6. 컴파일러에게 변수가 외부에 선언되어 있다고 알리는 지정자는?
7. extern 지정자를 변수 앞에 붙이면 무엇을 의미하는가?
8. static 지정자를 변수 앞에 붙이면 무엇을 의미하는가?

연결

- *연결(linkage):* 다른 범위에 속하는 변수들을 서로 연결하는 것
  - 외부 연결
  - 내부 연결
  - 무연결
- 전역 변수만이 연결을 가질 수 있다.
- static 지정자를 사용한다.
  - static이 없으면 외부 연결
  - static이 있으면 내부 연결



연결 예제

linkage1.c

```
#include <iostream>
using namespace std;
int all_files;          // 다른 소스 파일에서도 사용할 수 있는 전역 변수
static int this_file; // 현재의 소스 파일에서만 사용할 수 있는 전역 변수
extern void sub();

int main()
{
      sub();
      cout << all_files << endl;
      return 0;
}
```

연결

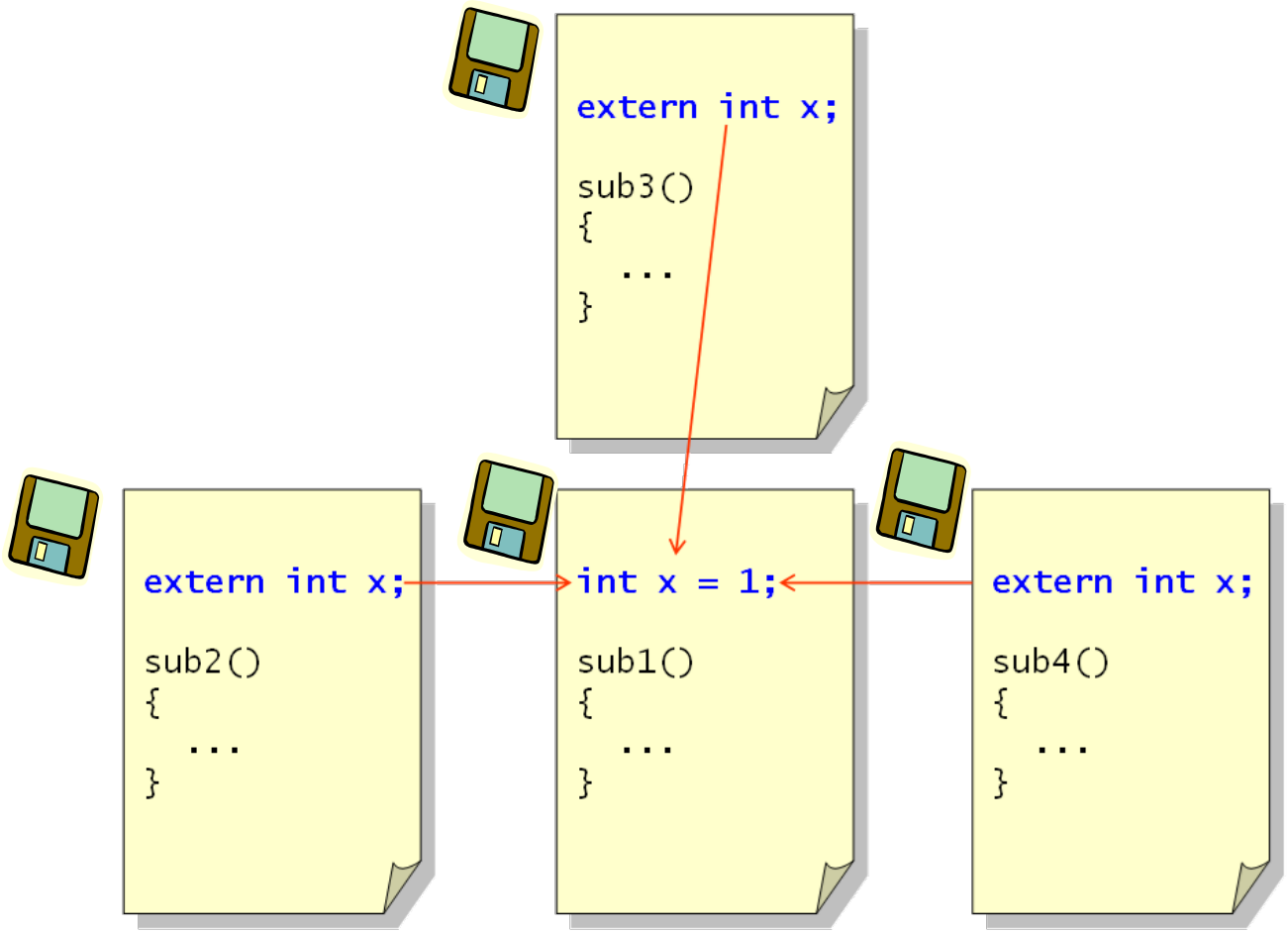
linkage2.c

```
extern int all_files;
void sub()
{
        all_files = 10;
}
```

```
10
```

다중 소스 파일

- 연결은 흔히 다중 소스 파일에서 변수들을 연결하는데 사용된다.



함수앞의 static

main.c

```
#include <iostream>
using namespace std;
extern void f2();
int main(void)
{
	f2();
	return 0;
}
```

static이 붙는 함수는 파일 안에서만 사용할 수 있다

sub.c

```
#include <iostream>
using namespace std;
static void f1()
{
	cout << "f1()이 호출되었습니다.\n";
}

void f2()
{
	f1();
	cout << "f2()가 호출되었습니다.\n";
}
```

저장 유형 정리

- 일반적으로는 *자동 저장 유형* 사용 권장
- 자주 사용되는 변수는 *레지스터 유형*
- 변수의 값이 함수 호출이 끝나도 그 값을 유지하여야 할 필요가 있다면 *지역 정적*
- 만약 많은 함수에서 공유되어야 하는 변수라면 *외부 참조 변수*

저장 유형	*키워드*	*정의되는 위치*	*범위*	*생존 시간*
자동	auto	함수 내부	지역	임시
레지스터	register	함수 내부	지역	임시
정적 지역	static	함수 내부	지역	영구
전역	없음	함수 외부	모든 소스 파일	영구
정적 전역	static	함수 외부	하나의 소스 파일	영구
외부 참조	extern	함수 외부	모든 소스 파일	영구

예제

main.c

```
#include <iostream>
using namespace std;
unsigned random_i(void);
double random_f(void);

extern unsigned call_count;    // 외부 참조 변수
int main()
{
    register int i;                // 레지스터 변수

    for(i = 0; i < 10; i++)
            cout << random_i() << " ";

    cout << endl;

    for(i = 0; i < 10; i++)
            cout << random_f() << " ";

    cout << "\n함수가 호출된 횟수= " << call_count;
    return 0;
}
```

예제

random.c

```
#define SEED 17
#define MULT 25173
#define INC  13849
#define MOD  65536
unsigned int call_count = 0     // 전역 변수
static unsigned seed = SEED;    // 정적 전역 변수
unsigned random_i(void)
{
    seed = (MULT*seed + INC) % MOD;
    call_count++;
    return seed;
}
double random_f(void)
{
    seed = (MULT*seed + INC) % MOD;
    call_count++;

    return seed / (double) MOD;
}
```

48574 61999 40372 31453 39802 35227 15504 29161
14966 52039
0.885437 0.317215 0.463654 0.762497 0.546997
0.768570 0.422577 0.739731 0.455627 0.720901
함수가 호출된 횟수 = 20

중간 점검 문제

1. 변수를 하나의 파일에서만 사용할 수 있도록 만드는 키워드는 무엇인가?
2. 변수를 여러 파일에 걸쳐 사용할 수 있도록 만드는 키워드는 무엇인가?
3. 지역 변수들은 외부와 연계가 가능한가?
4. 키워드 extern의 용도는 무엇인가?
5. 만약 함수 안에 선언된 정수 변수의 값이 함수 호출 사이에 유지되기를 바란다면 어떻게 정의하여야 하는가?
6. 지역 변수를 프로그램이 실행되는 동안 계속 유지되는 변수로 만드는 키워드는 무엇인가?

재귀(recursion)이란?

- 알고리즘이나 함수가 수행 도중에 자기 자신을 다시 호출하여 문제를 해결하는 기법

- 팩토리얼의 정의

$$n! = \begin{cases} 1 & n = 1 \\ n*(n-1)! & n \geq 2 \end{cases}$$

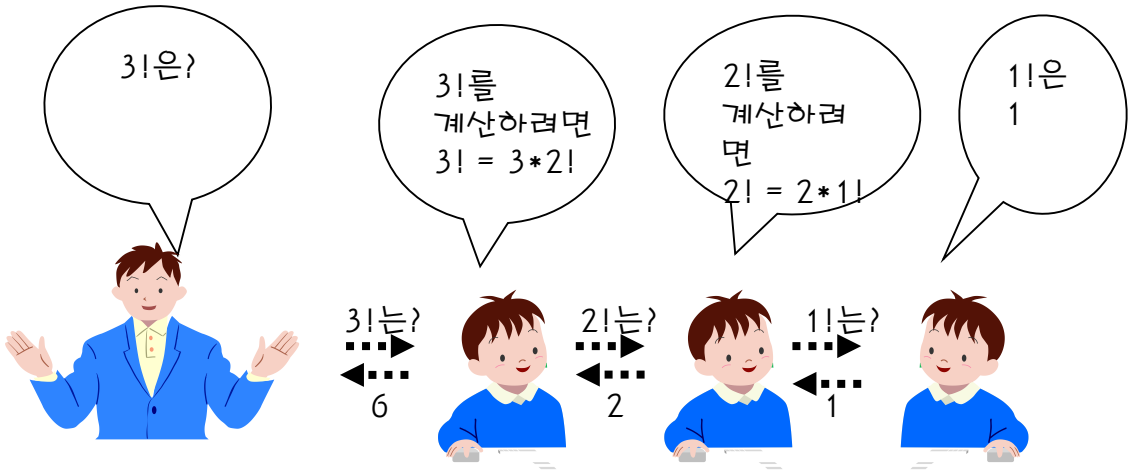
- 팩토리얼 프로그래밍 #1: (n-1)! 팩토리얼을 구하는 서브 함수를 따로 제작

```
int factorial(int n)
{
    if( n == 1 ) return(1);
    else return (n * factorial_n_1(n-1) );
}
```

팩토리얼 구하기 #1

- 팩토리얼 프로그래밍 #2: (n-1)! 팩토리얼을 현재 작성중인 함수를 다시 호출하여 계산(순환 호출)

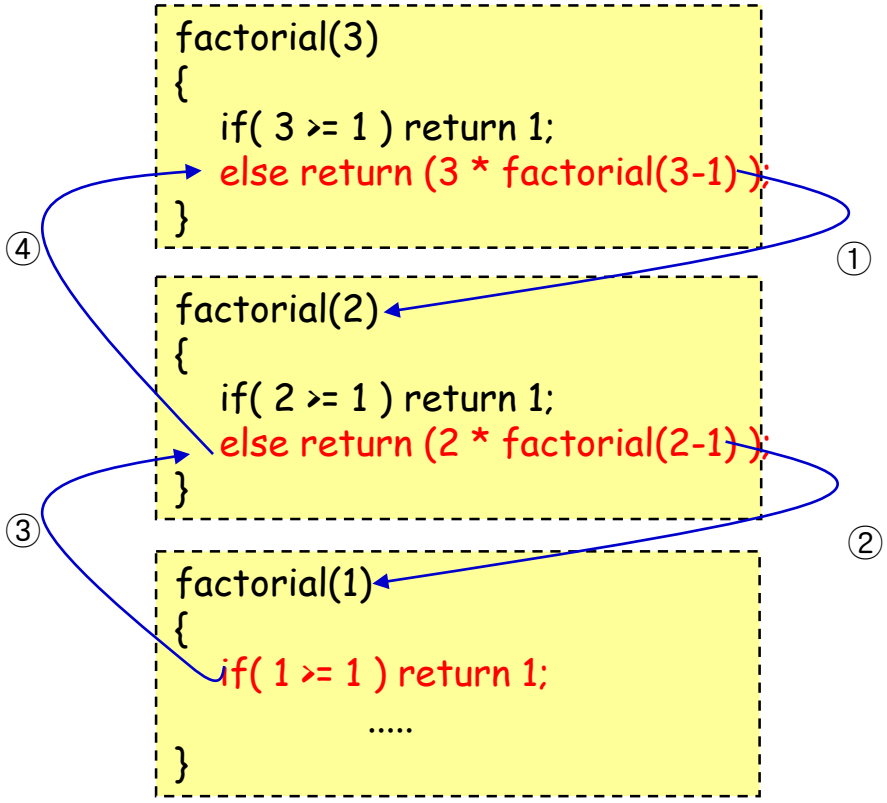
```
int factorial(int n)
{
    if( n >= 1 ) return(1);
    else return (n * factorial(n-1) );
}
```



팩토리얼 구하기 #2

- 팩토리얼의 호출 순서

factorial(3) = 3 * factorial(2)
= 3 * 2 * factorial(1)
= 3 * 2 * 1
= 3 * 2
= 6



순환 알고리즘의 구조

- 순환 알고리즘은 다음과 같은 부분들을 포함한다.
  - 순환 호출을 하는 부분
  - 순환 호출을 멈추는 부분

```
int factorial(int n)
{
    if( n == 1 )  return 1
    else return n * factorial(n-1);
}
```

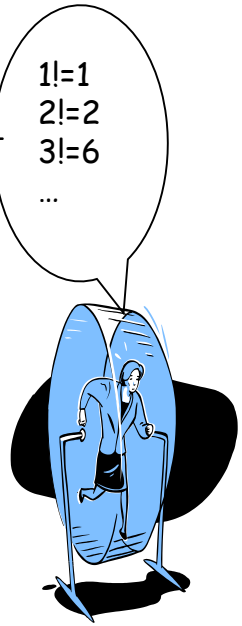
순환을 멈추는 부분

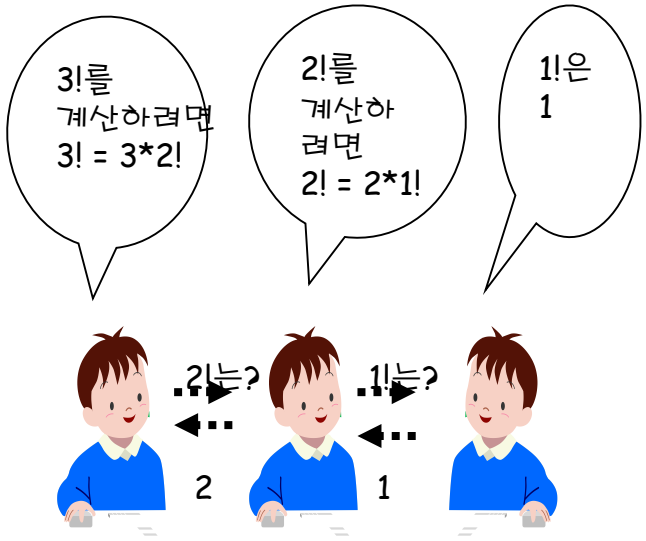
순환호출을 하는 부분

- 만약 순환 호출을 멈추는 부분이 없다면?.
  - 시스템 오류가 발생할 때까지 무한정 호출하게 된다.

순환 <-> 반복

- 컴퓨터에서의 되풀이
  - 순환(recursion): 순환 호출 이용
  - 반복(iteration): for나 while을 이용한 반복
- 대부분의 순환은 반복으로 바꾸어 작성할 수 있다.
- 순환
  - 순환적인 문제에서는 자연스러운 방법
  - 함수 호출의 오버헤드
- 반복
  - 수행속도가 빠르다.
  - 순환적인 문제에 대해서는 프로그램 작성이 아주 어려울 수도 있다.





팩토리얼의 반복적 구현

$$n! = \begin{cases} 1 & n = 1 \\ n*(n-1)*(n-2)*\cdots*1 & n \geq 2 \end{cases}$$

```
int factorial_iter(int n)
{
    int k, value=1;
    for(k=1; k<=n; k++)
        value = value*k;
    return(value);
}
```

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

예제

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

int main()
{
    double pi = 3.1415926535;

    cout << "sin()=" << sin(pi/2.0) << endl;
    cout << "cos()=" << cos(pi/2.0) << endl;
    cout << "tan()=" << tan(0.5) << endl;
    cout << "log()=" << log(10.0) << endl;
    cout << "log10()=" << log10(100.0) << endl;
    cout << "exp()=" << exp(10.0) << endl;
    return 0;
}
```

```
sin()=1
cos()=4.48966e-011
tan()=0.546302
log()=2.30259
log10()=2
exp()=22026.5
```

직각 삼각형 예제

```
#include <iostream>
#include <cmath>
using namespace std;
#define RAD_TO_DEG (45.0/atan(1))

int main()
{
    double w, h, r, theta;
    cout << "밑변을 입력하시오: ";
    cin >> w;
    cout << "높이를 입력하시오: ";
    cin >> h;
    r = sqrt(w * w + h * h);
    theta = RAD_TO_DEG * atan2(h, w);
    cout << "빗변= " << r << " 각도= " << theta << endl;
    return 0;
}
```

밑변과 높이를 입력하시오: 10.0 10.0
빗변= 14.142136 각도= 45.000000

난수 생성 라이브러리 함수

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

```
#include <iostream>
#include <cmath>
#include <ctime>
#include <cstdlib>
using namespace std;
// 0에서 9까지의 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
```

중간 점검 문제

1. 90도에서의 싸인값을 계산하는 문장을 작성하여 보라.
2. rand() % 10 이 계산하는 값의 범위는?

Q & A