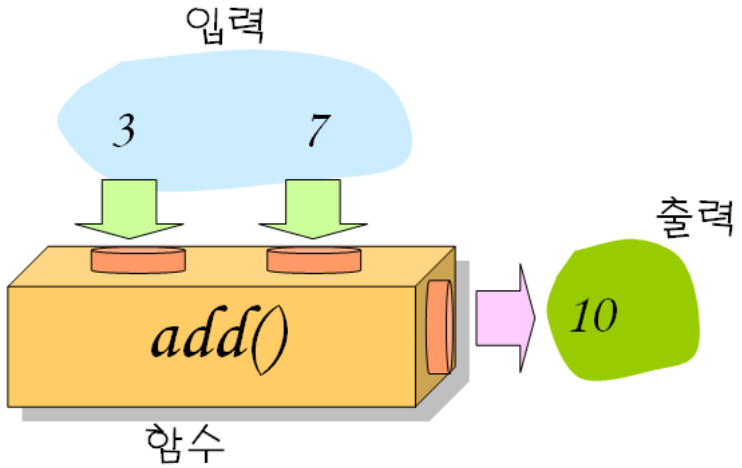
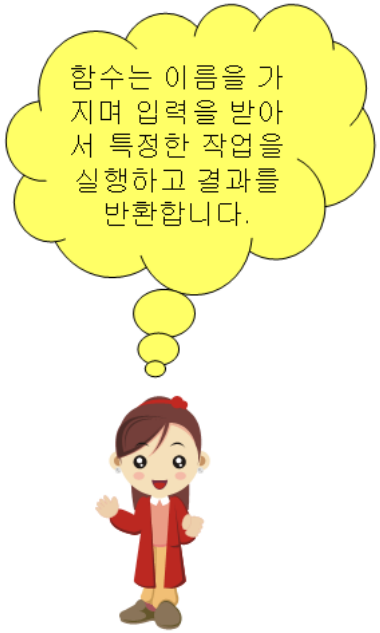
컴퓨터프로그래밍 및 실습
강의자료 2

함수 (function)

함수의 개념

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





함수의 필요성

```
#include <stdio.h>
int main(void)
{
        int i;
        for(i = 0; i < 10; i++)
                printf("*");
        printf("₩n");

        for(i = 0; i < 10; i++)
                printf("*");
        printf("₩n");
        for(i = 0; i < 10; i++)
                printf("*");
        printf("₩n");
        return 0;
}
```

함수의 필요성

```
#include <stdio.h>
void print_star()
{
        int i;
        for(i = 0; i < 10; i++)
                printf("*");
}
int main(void)
{
        print_star();
        printf("\n");
        print_star();
        printf("\n");
        print_star();
        printf("\n");
        return 0;
}
```

함수를 정의하였다. 함수는 한번 정의되면 여러 번 호출하여서 실행이 가능하다.

함수의 장점

- 함수를 사용하면 코드가 중복되는 것을 막을 수 있다.
- 한번 작성된 함수는 여러 번 재사용할 수 있다.
- 함수를 사용하면 전체 프로그램을 모듈로 나눌 수 있어서 개발 과정이 쉬워지고 보다 체계적이 되면서 유지보수도 쉬워진다.

함수들의 연결

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

```
#include <stdio.h>
void print_star()
{
        int i;
        for(i = 0; i < 10; i++)
                printf("*");
}
int main(void)
{
        print_star();
        ...
        print_star();
        ...
        print_star();
        return 0;
}
```

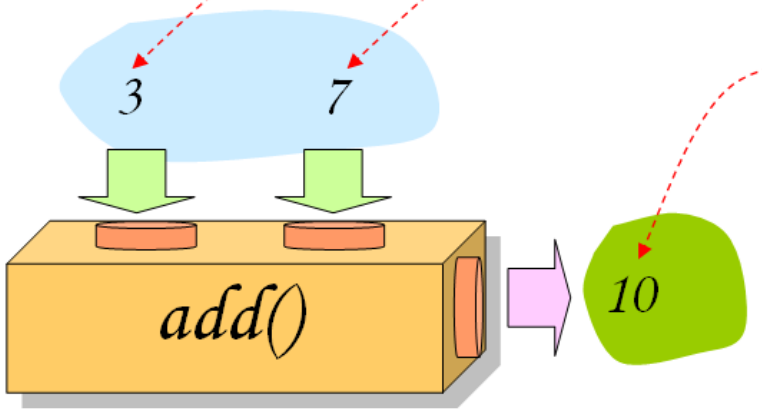
함수 유형

- 라이브러리 함수
  - 라이브러리 함수는 이미 정의되어 있고 컴파일 되어 있음
  - 표준 라이브러리 헤더 파일에서 원형을 제공
  - 함수의 원형(prototype)에 맞게 호출
  - 예: sqrt

- 사용자 정의 함수
  - 사용자가 직접 함수 작성

함수의 정의

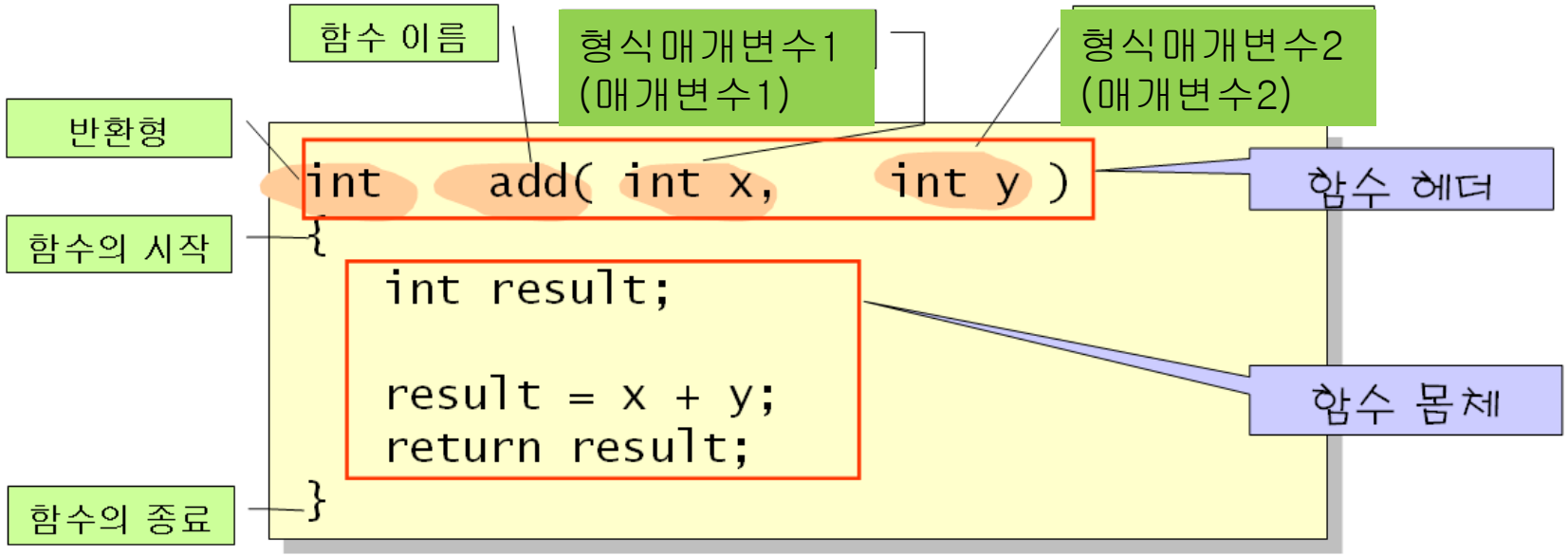
- 반환형(return type)
- 함수 헤더(function header)
- 함수 몸체(function body)




```
int add(int x, int y)
{
  int result;

  result = x + y;
  return result;
}
```

함수의 구조



- 함수정의 : 함수가 수행할 일을 기술한 코드
- 함수정의의 일반적인 형식
- 형식매개변수를 매개변수라 하기도 함

반환형 함수이름(**형식매개변수 리스트**)
{
    함수 몸체 // 문장들
}

함수에서 return 문장

- return 문

  (형식 1) return 식;

  의미 – 함수 결과로서 식의 값을 반환하고
  함수 수행을 끝낸다(함수를 빠져나간다).

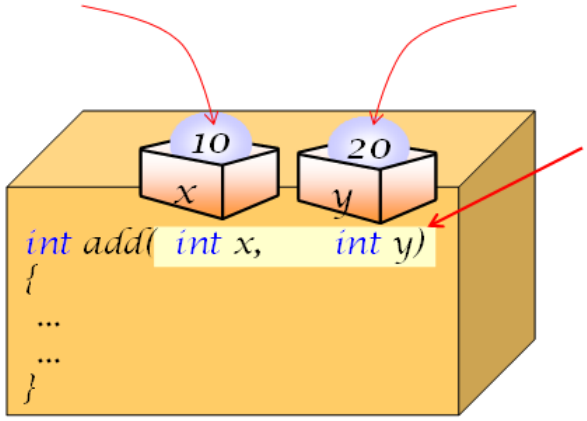
  (형식 2) return;

  함수의 반환형이 void

  의미 – 함수 수행을 끝낸다(함수를 빠져나간다).

✓ 함수의 수행 중 함수의 마지막을 만나면 함수를 빠져나간다. – return 문장의 수행과 동일한 효과

형식매개변수(매개변수)와 실매개 변수(인자)



매개변수는 외부에서 전달되는 데이터가 저장되는 변수

형식매개변수(formal parameter)를 매개변수(parameter), 실매개변수(actual parameter)를 인자(argument)라 하기도 함

- 예:

형식매개변수(혹은 매개변수)들

```
int add(int x, int y)
{
   int result;
   result = x + y;
   return result;
}

int main()
{
   int value;
   value =  add(10, 30);
   printf("%d \n", value);
   return 0;
}
```

실매개변수(인자)들

함수 정의 예

- 예: 절대값을 구하는 함수

```
double abs(double x)
{
    if (x >= 0) return x;
    else return -x;
}
```

예제

- 두 개의 정수 중에서 큰 수를 계산하는 함수와 이를 이용하는 프로그램

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

```
int main ()
{
    int a, b, c;
    int largest;

    scanf("%d %d %d", &a, &b, &c);
    largest = get_max(a, b);
    largest = get_max(largest,c);
    printf("The largest value = %d ",
    largest);
    return 0;
}
```

예제

- 팩토리얼값(n!)을 계산하는 함수

```
int factorial(int n)
{
    int i;
    long result = 1;

   for(i = 1; i <= n; i++)
     result *= i;  // result = result * x

    return result;
}
```

```
int main ()
{
    int n;
    int fact;

    scanf("%d", &n);

    fact = factorial(n);
    printf("%d ! =  ", fact);
    return 0;
}
```

void 반환형

```
#include<stdio.h>
void print_hello(void)
{
    cout <<"Hello World!□n";
}

int main(void)
{
    print_hello();
    return 0;
}
```

- 결과값을 return하지 않는 함수
- 인자(실매개변수)를 받지 않는 함수를 정의할 때에는 인자가 들어갈 자리에 void를 사용
- void는 생략가능

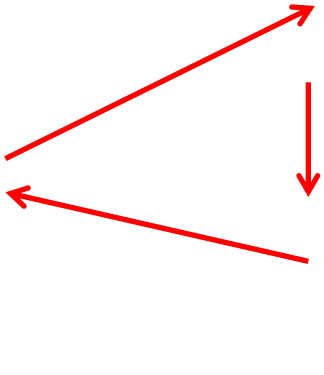
함수 호출과 반환

- 함수 호출(function call):
  - 함수 호출이 되면 (프로그램 실행시 함수를 만나면) 실매개변수로부터 형식매개변수로 전달이 일어난다.
  - 함수안의 문장들이 순차적으로 실행된다.
  - 문장의 실행이 끝나면(return 문을 만나든지 혹은 함수 몸체의 끝을 만나면) 호출한 위치로 되돌아 간다.
  - 결과값을 전달할 수 있다.

```
int main()
{
   int value;
   value =  add(10, 30);
   printf("%d \n", value);

   return 0;
}
```

```
int add(int x, int y)
{
   int result;
   result = x + y;
   return result;
}
```



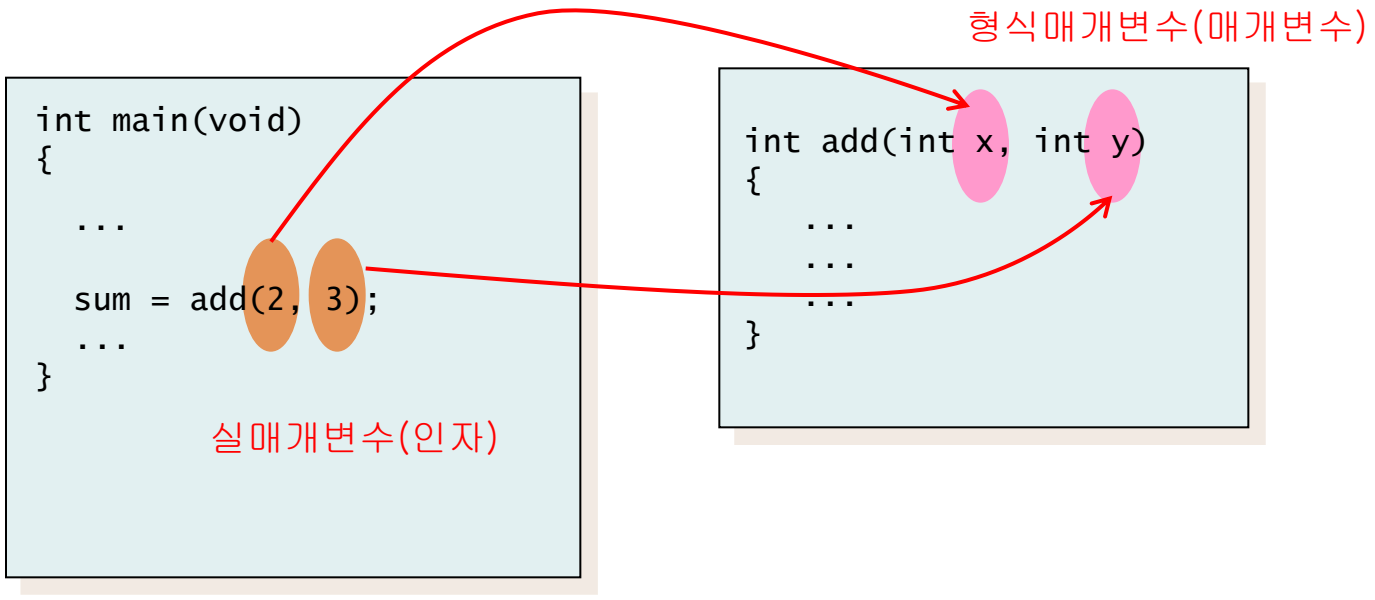
함수의 매개변수

- 매개변수
  - 함수가 자료를 전달받기 위하여 이용

✓ 실매개변수(인자)
- 함수 사용(호출)에 있는 인자

✓ 형식매개변수 (매개변수)
- 함수 정의에 있는 인자

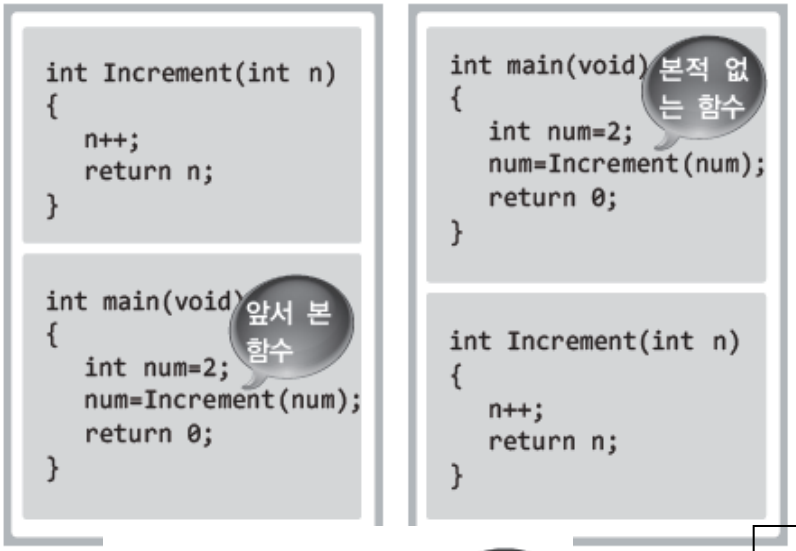
매개변수


형식매개변수(매개변수)
int main(void)
{
...
sum = add(2, 3);
...
}
실매개변수(인자)
int add(int x, int y)
{
...
...
...
}

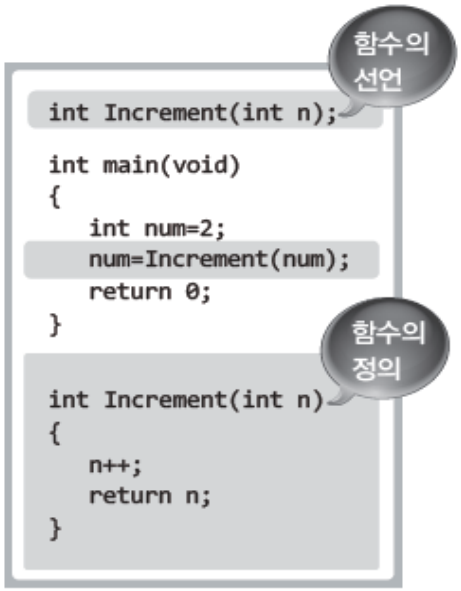
함수 원형(prototype)

- 함수의 사용방법에 대한 기술
- 컴파일러에게 함수에 대하여 미리 알리는 것
- 함수정의가 함수사용 이후에 나올 경우, 함수 원형이 함수사용 전에 나와 있어야 한다.

함수의 정의와 그에 따른 원형의 선언



▪ 컴파일이 위에서 아래로 진행이 되기 때문에 함수의 배치순서는 중요하다. 컴파일되지 않은 함수는 호출이 불가능하다



- 이후에 등장하는 함수에 대한 정보를 컴파일러에게 제공해서 이후에 등장하는 함수의 호출문장이 컴파일 가능하게 도울 수 있다
- 이렇게 제공되는 함수의 정보를 가리켜 '함수의 원형'이라 한다.
- int Increment(int n); // 함수 원형 선언
- int Increment(int); // 위와 동일한 함수 원형선언, 매개변수 이름 생략가능

함수 원형을 사용하지 않는 예제

```
int compute_sum(int n)
{
        int i;
        int result = 0;
        for(i = 1; i <= n; i++)
                result += i;
        return result;
}

int main(void)
{
        int sum;
        sum = compute_sum(100);
        printf("sum=%d \n", sum);
}
```

함수 정의가 함수 호출보다 먼저 오면 함수 원형을 정의하지 않아도 된다.

함수 원형을 사용하는 예제

```
int compute_sum(int n); // 함수 원형

int main(void)
{
        int sum;
        sum = compute_sum(100);
        printf("sum=%d \n", sum);
}

int compute_sum(int n)
{
        int i;
        int result = 0;
        for(i = 1; i <= n; i++)
                result += i;
        return result;
}
```

함수 정의가 함수 호출보다 먼저 오면 함수 원형을 선언해야 한다.

함수 원형과 헤더 파일

- 보통은 헤더 파일에 함수 원형이 선언되어 있음

```c
/* 두개의 숫자의 합을 계산하는 프로그램 */
#include <stdio.h>


int main(void)
{
  int n1;   /* 첫번째 숫자 */
  int n2;   /* 두번째 숫자 */
  int sum;  /* 두개의 숫자의 합을 저장 */

  printf("첫번째 숫자를 입력하시오:");
  scanf("%d", &n1);


  printf("두번째 숫자를 입력하시오:");
  scanf("%d", &n2);


  sum = n1 + n2;
  printf("두수의 합: %d", sum);


  return 0;
}
```

add.c

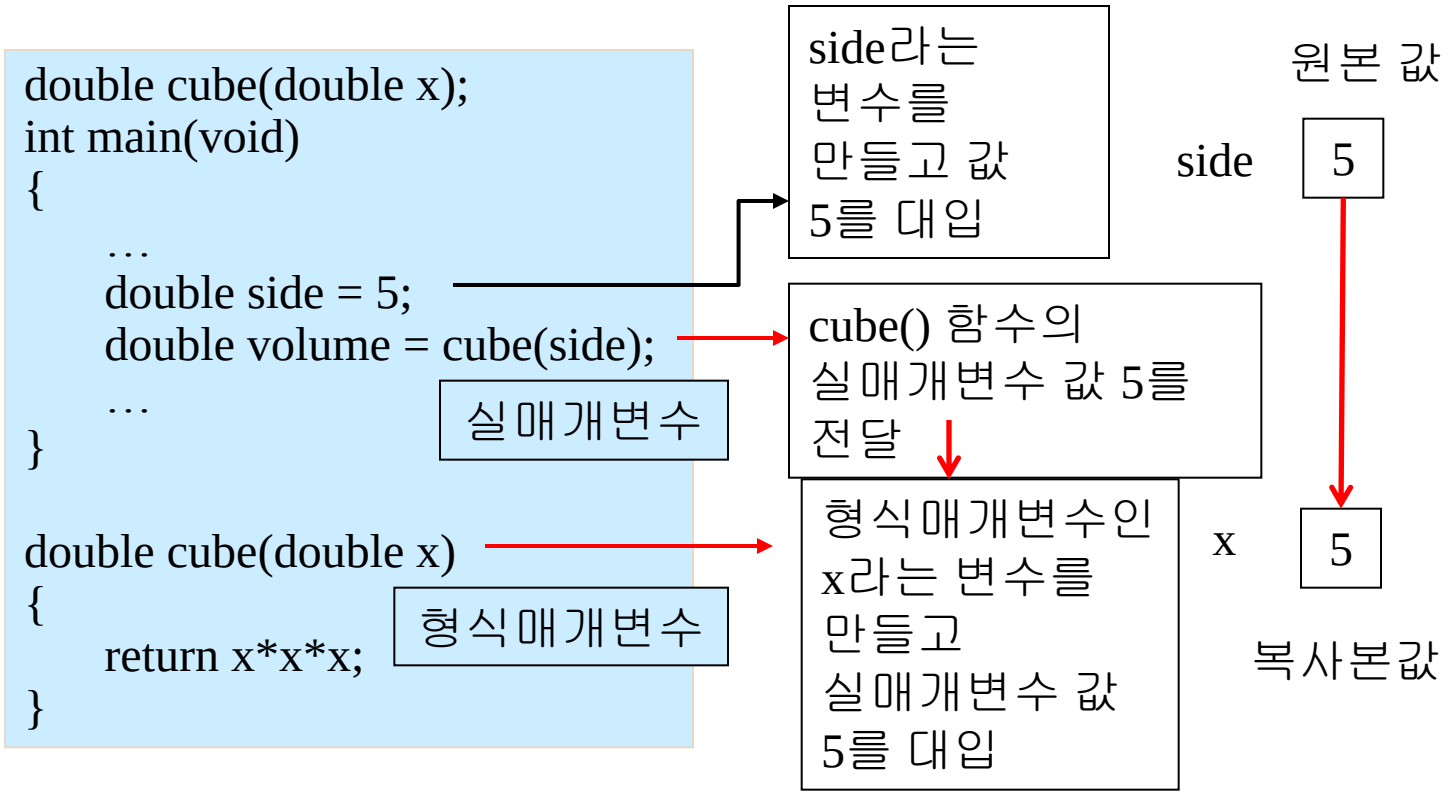
```c
/***
*stdio.h - definitions/declarations for
standard I/O routines
*
*
****/

...
_CRTIMP int __cdecl printf(const char
*, ...);
...
_CRTIMP int __cdecl scanf(const char
*, ...);
...
```

stdio.h

함수호출에서의 매개변수 전달 방법

- 매개변수 전달방법
- call by value (값에 의한 매개변수 전달)로 전달함
  실매개변수의 값을 형식매개변수로 복사하여 전달



- Call by value (값에 의한 호출)

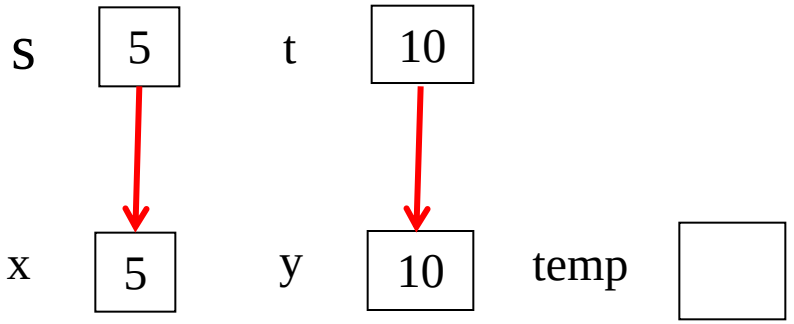
  - 실매개변수의 값이 호출되는(called) 함수의 형식매개변수에 복사됨

  - 호출되는(Called) 함수에서 형식매개변수의 값을 변경해도 호출하는 함수(caller) 쪽에서의 실매개변수 값에 영향을 미치지 않음

```
void swap(int x, int y) {
  int temp;
  temp = x;
  x = y;
  y = temp;
}

int main() {
  int s = 5, t = 10;
  cout <<"s=" << s <<" t=" <<t
       << endl;
  swap(s, t);
  cout <<"s = " << s <<" t = " <<t
       << endl;
  return 0;
}
```

- 함수 swap(s,t)를 호출하면 실매개변수 s와 t의 값이 함수 swap의 형식매개변수 x와 y로 전달된다



- 함수 swap에서 x와 y의 값을 바꾸지만, 수행이 끝난 후, 호출한 곳으로 돌아오면 S와 t의 값은 변함이 없다.

따라서 출력 결과값은 다음과 같다

s = 5 t = 10

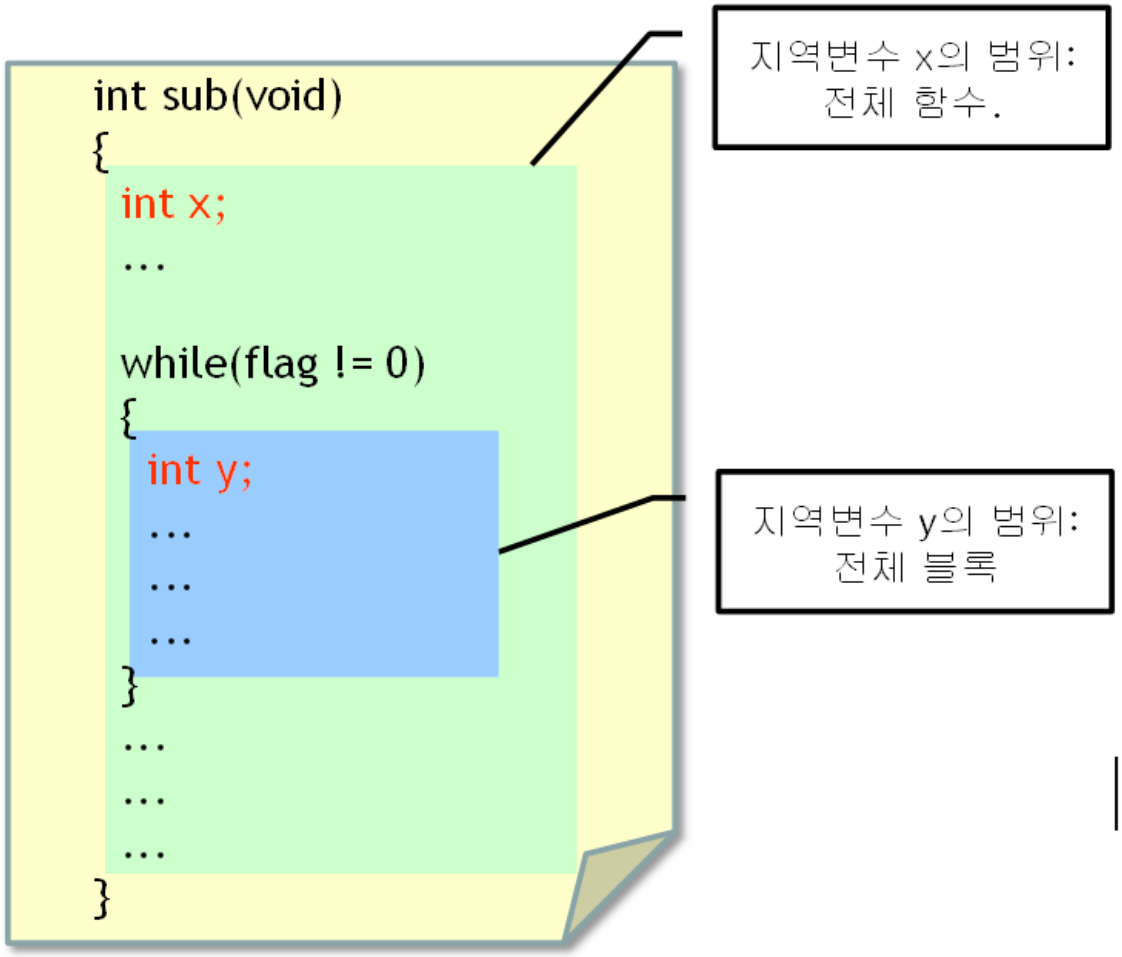
변수의 속성

- 변수의 범위와 생존 시간

  - **범위(scope)** : 변수가 사용 가능한 범위, 가시성
  - **생존 시간(lifetime)** : 메모리에 존재하는 시간

지역변수

- 지역 변수(local variable)는 블록 안에 선언되는 변수



지역변수

- 함수내에 선언되는 변수를 가리켜 지역변수(local variable)라 한다.
- 지역변수는 선언된 이후로부터 함수내에서만 접근이 가능하다.
- 한 지역(함수, 블럭) 내에 동일한 이름의 변수선언 불가능하다.
- 다른 지역에 동일한 이름의 변수선언 가능하다.
- 해당 지역을 빠져나가면 지역변수는 소멸된다. 그리고 호출될 때마다 새롭게 할당된다.

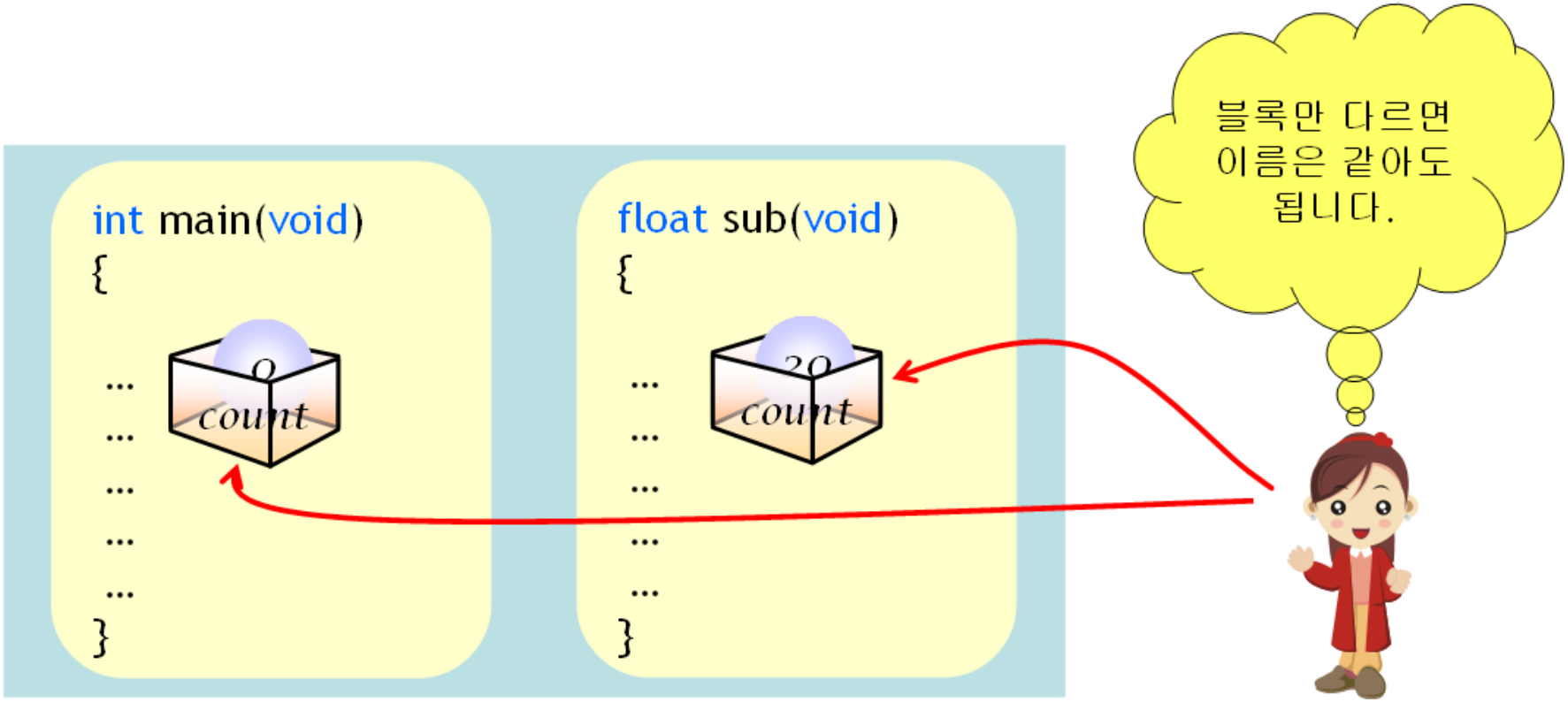
```
int SimpleFuncOne(void)
{
    int num=10;    // 이후부터 SimpleFuncOne의 num 유효
    num++;
    printf("SimpleFuncOne num: %d \n", num);
    return 0;    // SimpleFuncOne의 num이 유효한 마지막 문장
}

int SimpleFuncTwo(void)
{
    int num1=20;    // 이후부터 num1 유효
    int num2=30;    // 이후부터 num2 유효
    num1++, num2--;
    printf("num1 & num2: %d %d \n", num1, num2);
    return 0;    // num1, num2 유효한 마지막 문장
}
```

```
int main(void)
{
    int num=17;    // 이후부터 main의 num 유효
    SimpleFuncOne();
    SimpleFuncTwo();
    printf("main num: %d \n", num);
    return 0;    // main의 num이 유효한 마지막 문장
}
```

이름이 같은 지역변수

- 지역변수의 이름이 같아도 다른 블록에 있으면, 이들은 다른 변수로 취급한다.



정적(static) 변수

```
#include <stdio.h>

int f(void)
{
   static int count = 0;
   count++;
   printf("count = %d\n", count)
   return count;
}

int main(void)
{
   f();
   f();

   return 0;
}
```

- 변수 선언시 자료형 앞에 static를 둠
- 함수안에서 선언되는 정적 변수는 함수 수행이 종료되더라도 메모리 해제가 되지 않고 프로그램 종료시까지 메모리가 할당되어 있음
- 정적 변수가 선언된 함수가 다시 수행되면 정적 변수의 이전 값을 유지함

지역변수의 일종인 형식매개변수(매개변수)

- 형식매개변수도 선언된 함수 내에서만 접근이 가능하다.
- 선언된 함수가 반환을 하면, 지역변수와 마찬가지로 형식매개변수도 소멸된다.

전역변수(global variable)의 이해와 선언방법

- 전역변수(global variable)는 함수외부에 선언된다.
- 프로그램의 시작과 동시에 메모리 공간이 할당되어 종료시까지 존재한다.
- 별도의 값으로 초기화하지 않으면 0으로 초기화된다.
- 프로그램 전체영역 어디서든 접근이 가능하다

```
void Add(int val);
int num;    // 전역변수는 기본 0으로 초기화됨

int main(void)
{
    printf("num: %d \n", num);
    Add(3);
    printf("num: %d \n", num);
    num++;   // 전역변수 num의 값 1 증가
    printf("num: %d \n", num);
    return 0;
}

void Add(int val)
{
    num += val;    // 전역변수 num의 값 val만큼 증가
}
```

```
num: 0
num: 3
num: 4
```