



누구나 즐기는 C언어 콘서트

제7장 함수





이번 장에서 학습할 내용

- 모듈화
- 함수의 개념, 역할
- 함수 작성 방법
- 반환값
- 인수 전달
- 함수를 사용하는 이유

규모가 큰
프로그램은 전체
문제를 보다
단순하고 이해하기
쉬운 함수로
나누어서
프로그램을
작성하여야 한다.





함수가 필요한 이유

- 같은 작업이 되풀이 되는 경우

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

```
int main(void)
```

```
{
```

```
    int i;
```

```
    for(i = 0; i < 10; i++)  
        printf("*");
```

```
    ...
```

```
    for(i = 0; i < 10; i++)  
        printf("*");
```

```
    ...
```

```
    for(i = 0; i < 10; i++)  
        printf("*");
```

```
}
```

10개의 *을 출력하는
코드

10개의 *을 출력하는
코드

10개의 *을 출력하는
코드



함수가 있다면

- 함수는 한번 작성되면 여러 번 사용(호출)이 가능하다.

```
#include <stdio.h>

void print_star()
{
    int i;
    for(i = 0; i < 10; i++)
        printf("*");
}

int main(void)
{
    print_star();
    ...
    print_star();
    ...
    print_star();
}
```

함수

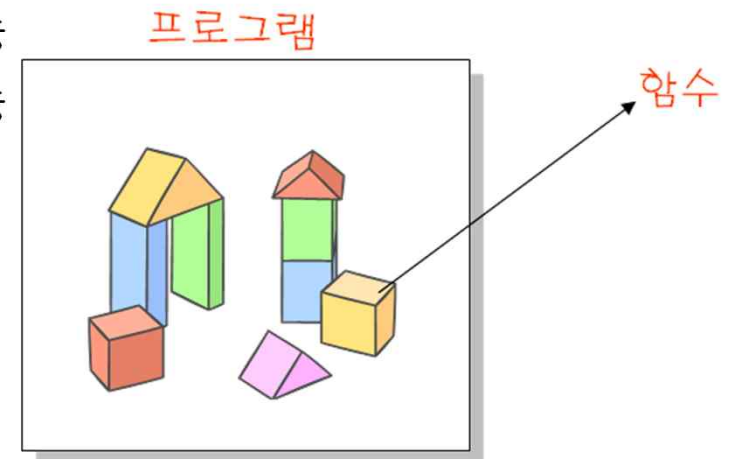
함수 호출



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

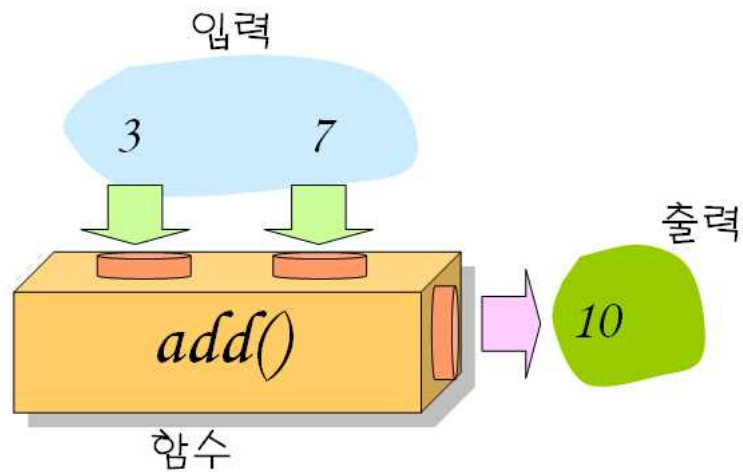
- C에서는 **모듈==함수**





함수의 개념

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



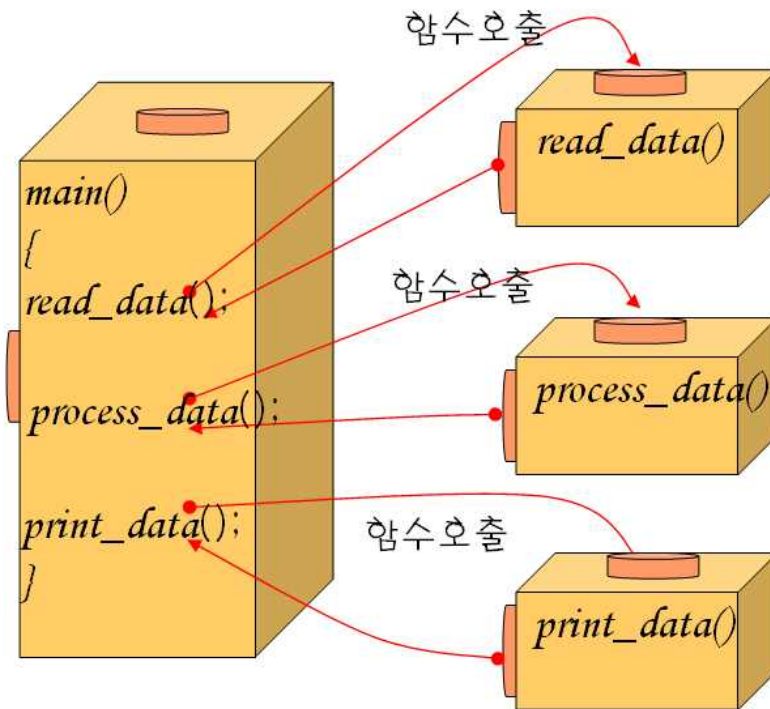
함수는 이름을 가지며 입력을 받아서 특정한 작업을 실행하고 결과를 반환합니다.





함수들의 연결

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



각 함수들은 함수 호출을 통하여 서로 결합되어 프로그램을 구성합니다.





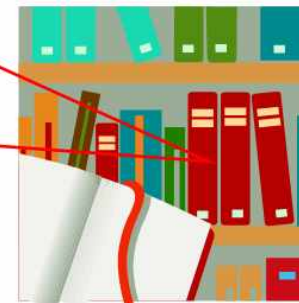
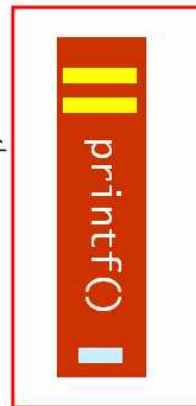
함수의 종류

함수

- 사용자 정의 함수
- 라이브러리 함수



```
sum(int a, int b)
{
    ...
    ...
}
```



컴파일러 제공 라이브러리





중간 점검

1. 함수가 필요한 이유는 무엇인가?
2. 함수와 프로그램의 관계는?
3. 컴파일러에서 지원되는 함수를 _____ 함수라고 한다.





이번 장에서 학습할 내용

- 모듈화
- 함수의 개념, 역할
- 함수 작성 방법
- 반환값
- 인수 전달
- 함수를 사용하는 이유

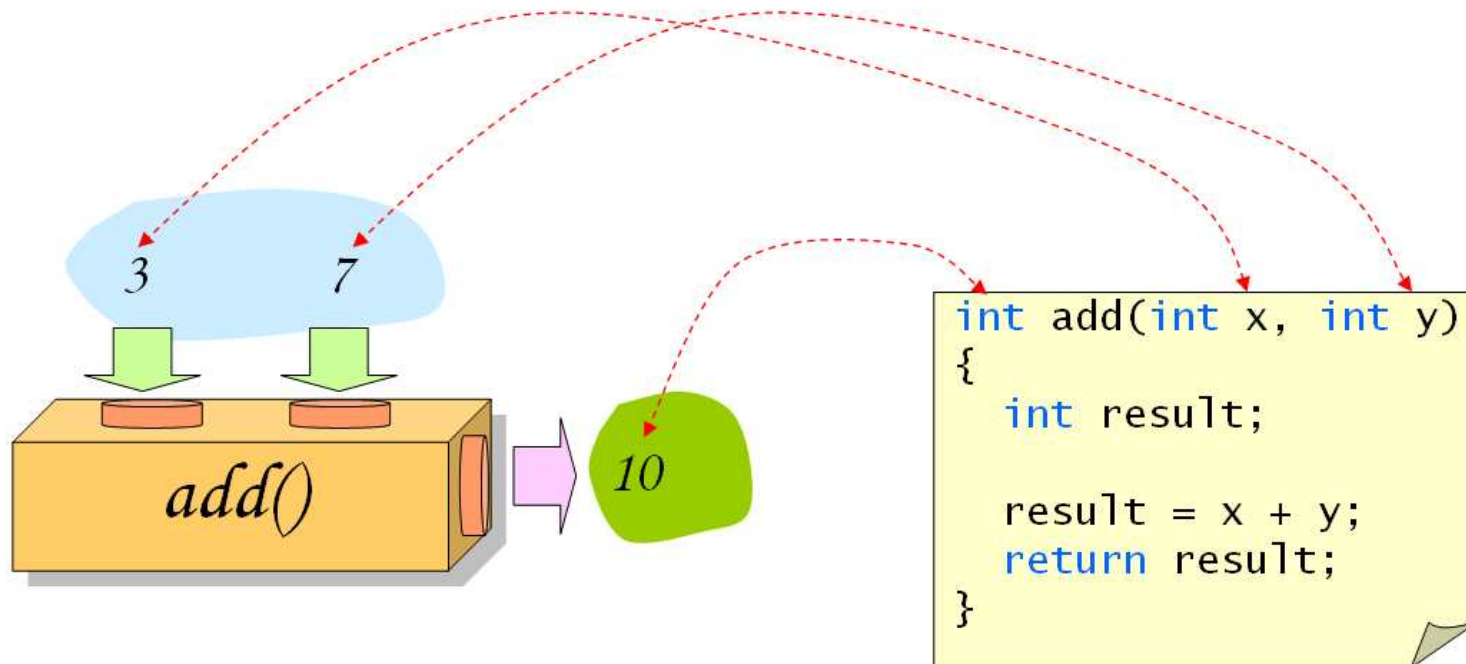
규모가 큰
프로그램은 전체
문제를 보다
단순하고 이해하기
쉬운 함수로
나누어서
프로그램을
작성하여야 한다.





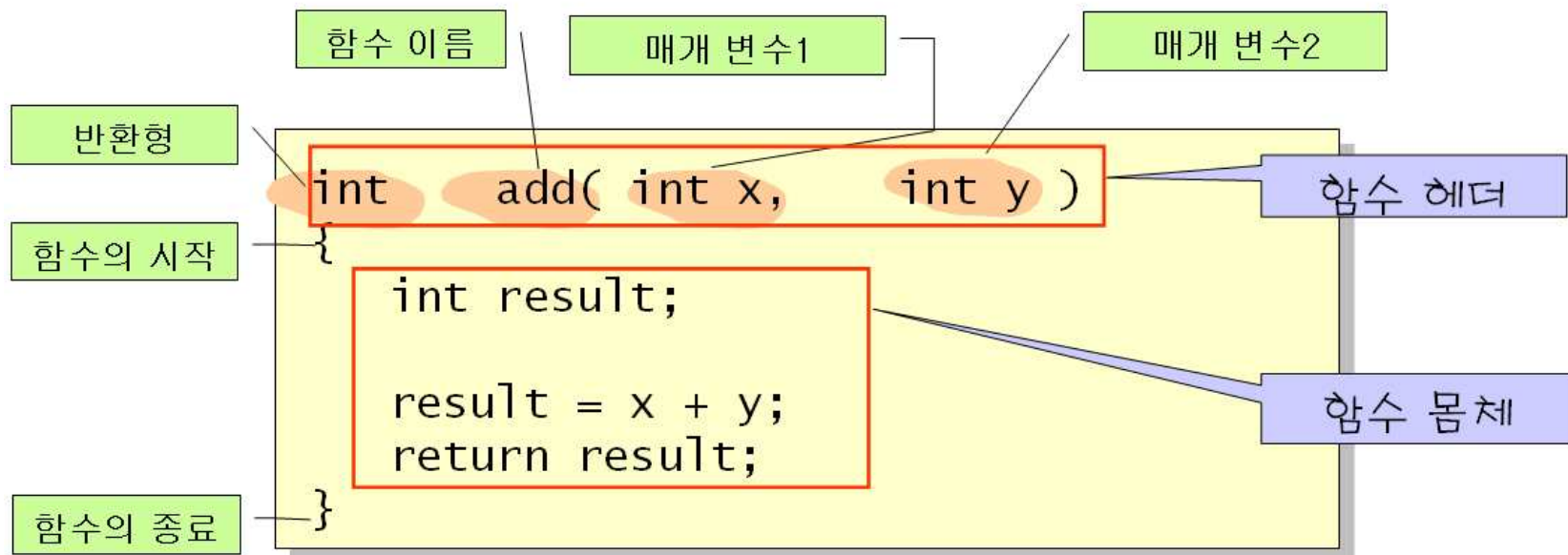
함수의 정의

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





함수의 구조





반환형

- 함수 이름 앞에 반환하는 데이터의 유형을 표시한다.
- `char`, `int`, `long`, `double` ... 등이 가능하다.
- 반환형이 없으면 `void`로 표시

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

```
void print_info()  
{  
    ...  
}
```



함수 이름

- 일반적으로 **동사+명사**
- (예)
 - compute_average(),
 - get_integer()
 - set_speed()

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

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



매개 변수

- **매개 변수(*parameter*)**: 함수가 외부로부터 전달받는 데이터를 가지고 있는 변수

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

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



지역 변수

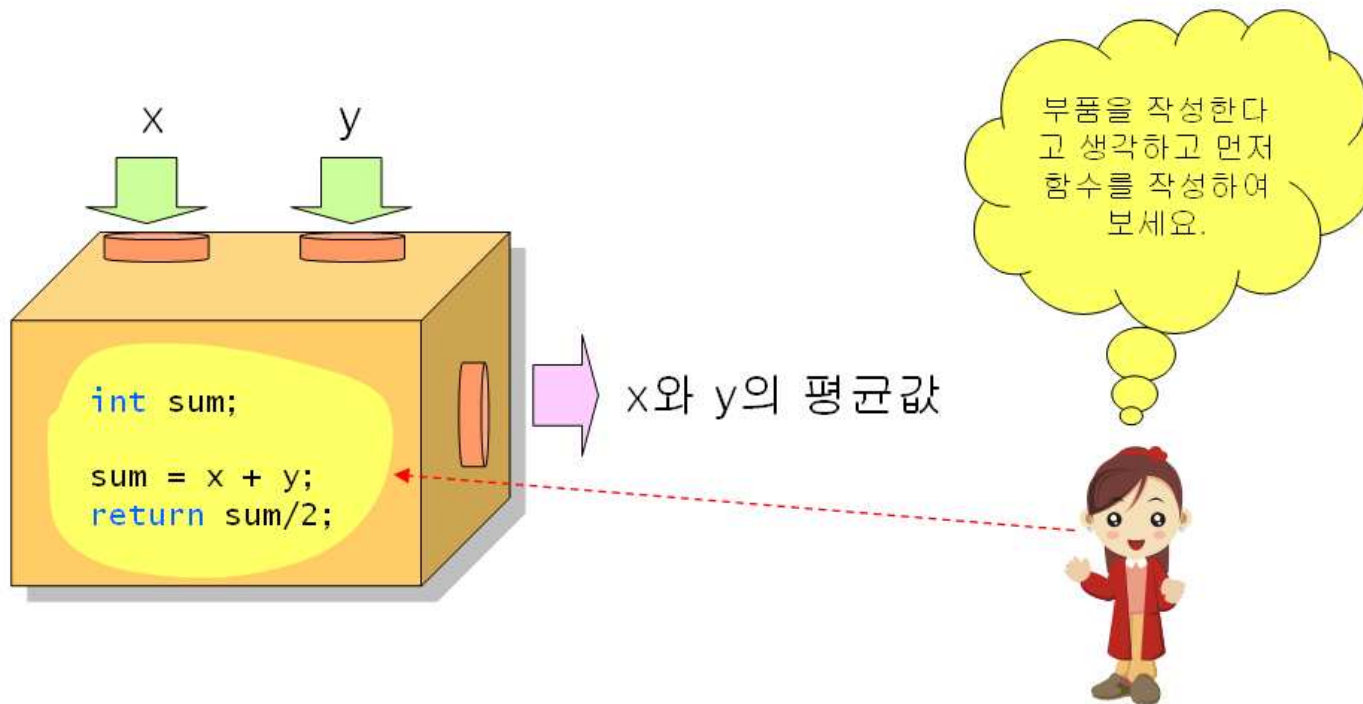
- 지역 변수(local variable): 함수 안에서 정의되는 변수

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




함수 정의 예제

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

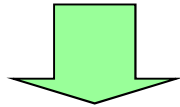




예제 #1

- 정수의 제곱값을 계산하는 함수

반환값: **int**
함수 이름: **square**
매개 변수: **int n**



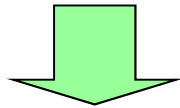
```
int square(int n)  
{  
    return(n*n);  
}
```



예제 #2

- 두개의 정수중에서 큰 수를 계산하는 함수

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



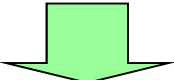
```
int get_max(int x, int y)
{
    if( x > y ) return(x);
    else return(y);
}
```



예제 #3

- 정수의 거듭 제곱값(x^y)을 계산하는 함수

반환값: `int`
함수 이름: `power`
매개 변수: `int x, int y`



```
int power(int x, int y)
{
    int i;
    long result = 1;

    for(i = 0; i < y; i++)
        result *= x;
    return result;
}
```



중간 점검

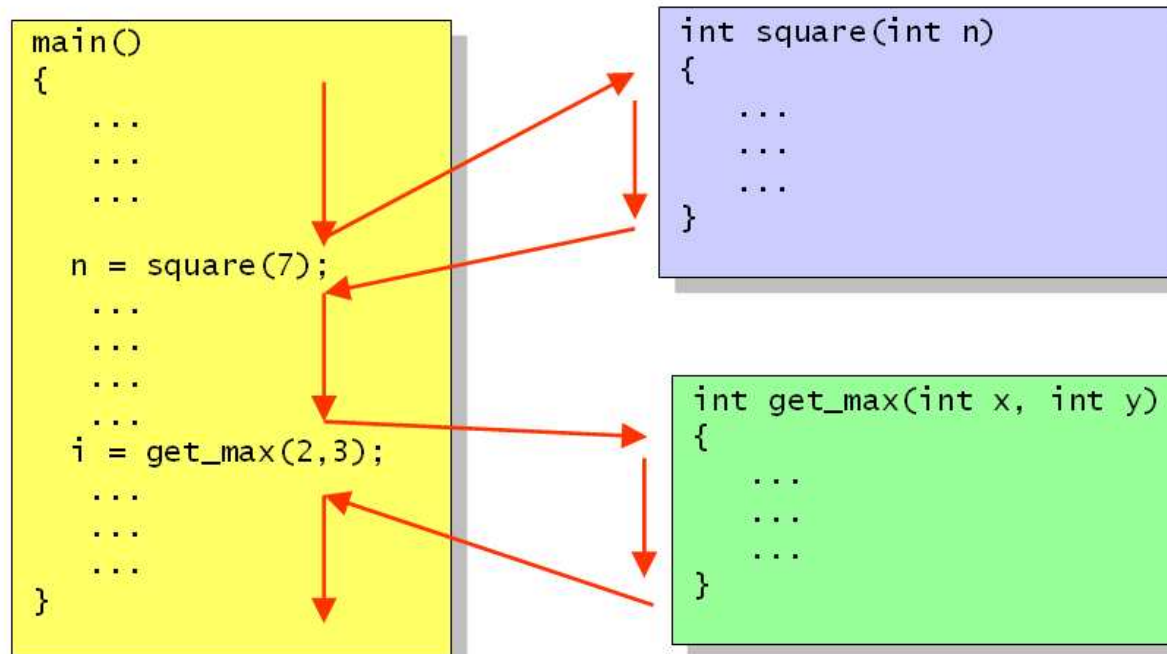
1. 함수 이름 앞에 **void**가 있다면 무슨 의미인가?
2. 함수가 작업을 수행하는데 필요한 데이터로서 외부에서 주어지는 것을 무엇이라고 하는가?
3. 함수 몸체는 어떤 기호로 둘러 싸여 있는가?
4. 함수의 몸체 안에서 정의되는 변수를 무엇이라고 하는가?





함수 호출과 반환

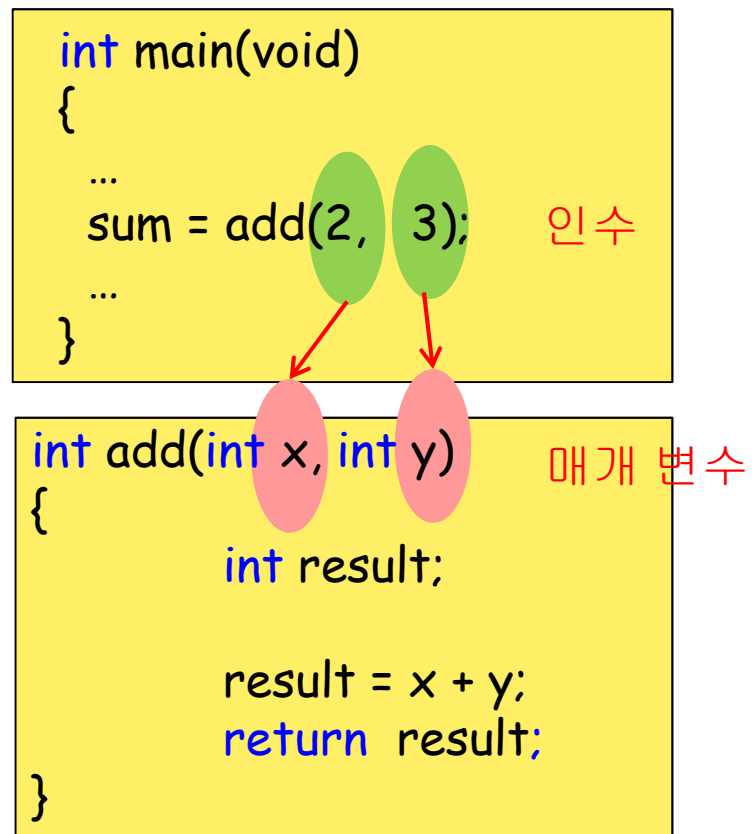
- 함수 호출(*function call*):
 - 함수를 사용하기 위하여 함수의 이름을 적어주는 것
 - 함수안의 문장들이 순차적으로 실행된다.
 - 문장의 실행이 끝나면 호출한 위치로 되돌아 간다.
 - 결과값을 전달할 수 있다.





인수와 매개 변수

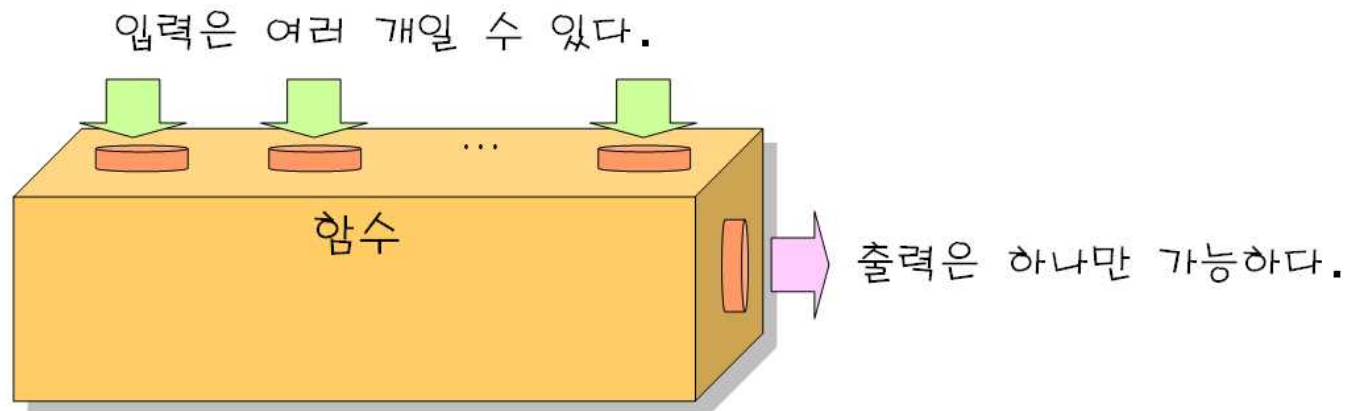
- 인수(*argument*): 실인수, 실매개 변수라고도 한다.
- 매개 변수(*parameter*): 형식 인수, 형식 매개 변수라고도 한다.





반환값

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





값을 반환하는 문장

- `return` 문장을 사용하여 값을 반환한다.

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

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

- `return` 문장의 사용예

```
return 0;
return (x);
return x+y;
```



함수 원형

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

```
// 정수의 제곱을 계산하는 함수 예제
#include <stdio.h>
int square(int n);           // 함수 원형

int main(void)
{
    ...
    result = square(i);      // 함수 호출
}

int square(int n)            // 함수 정의
{
    return(n * n);
}
```

함수 원형



예제

// 정수의 제곱을 계산하는 함수 예제

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

```
int square(int n); // 함수 원형
```

```
int main(void)
```

```
{
```

```
    int i, result;
```

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

```
    {
```

```
        result = square(i); // 함수 호출
```

```
        printf("%d \n", result);
```

```
    }
```

```
    return 0;
```

```
}
```

```
int square(int n) // 함수 정의
```

```
{
```

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

```
}
```



0
1
4
9
16



예제

```
// 두수 중에서 큰 수를 찾는 함수 예제
#include <stdio.h>
int get_max(int x, int y);

int main(void)
{
    int a, b;
    printf("두개의 정수를 입력하시오: ");
    scanf("%d %d", &a, &b);
    printf("두수 중에서 큰 수는 %d입니다.\n", get_max(a, b));
    return 0;
}

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



두개의 정수를 입력하시오: 2 3
두 수 중에서 큰 수는 3입니다.



조합(combination) 계산 함수

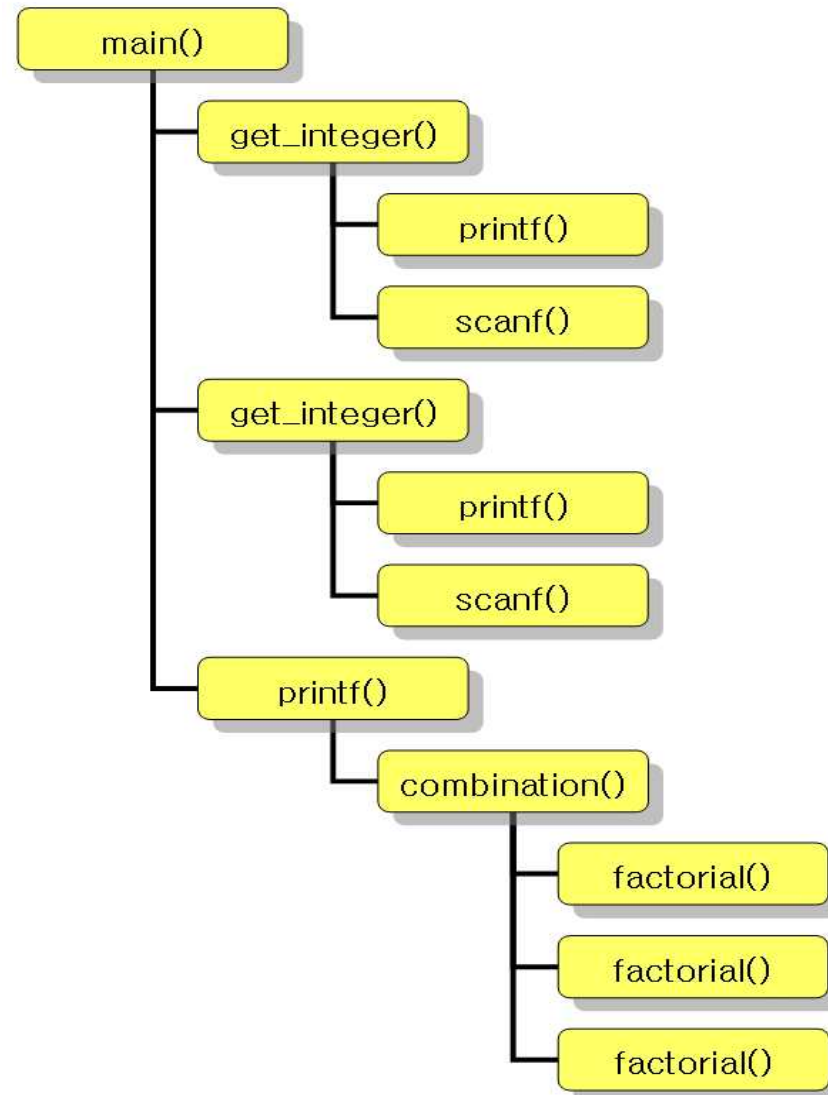
$$C(n, r) = \frac{n!}{(n-r)!r!}$$

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

- 팩토리얼 계산 함수와 `get_integer()` 함수를 호출하여 조합을 계산한다



함수 호출 계층 구조





예제

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

```
int get_integer(void);
```

```
int combination(int n, int r);
```

```
int factorial(int n);
```

```
int main(void)
```

```
{
```

```
    int a, b;
```

```
    a = get_integer();
```

```
    b = get_integer();
```

```
    printf("C(%d, %d) = %d \n", a, b, combination(a, b));
```

```
    return 0;
```

```
}
```

```
int combination(int n, int r)
```

```
{
```

```
    return (factorial(n)/(factorial(r) * factorial(n-r)));
```

```
}
```



예제

```
int get_integer(void)
{
    int n;

    printf("정수를 입력하시오: ");
    scanf("%d", &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

1. 인수와 매개 변수는 어떤 관계가 있는가?
2. 반환값이 **double**형으로 정의된 함수에서 정수를 반환하면 어떤 일이 발생하는가?





함수 원형

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

반환형 함수이름 (매개변수 1, 매개변수 2, ...);

```
int compute_sum(int n);  
  
int main(void)  
{  
    //...  
    sum = compute_sum(100);  
    //...  
}  
  
int compute_sum(int n)  
{  
    int i;  
    int result = 0;  
  
    for(i = 1; i <= n; i++)  
        result += i;  
  
    return result;  
}
```

함수 원형

함수 호출

함수 정의



함수 원형 예제

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

```
// 함수 원형
```

```
int compute_sum(int n);
```

```
int main(void)
```

```
{
```

```
    int n, sum;
```

```
    printf("정수를 입력하시오: ");
```

```
    scanf("%d", &n);
```

```
    sum = compute_sum(n);           // 함수 사용
```

```
    printf("1부터 %d까지의 합은 %d입니다. \n", n, sum);
```

```
}
```

```
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.



함수 원형을 사용하지 않으려면

- 함수가 미리 정의되면 된다.
- 그러나 특수한 경우에는 이것이 불가능하다. 따라서 함수 원형을 사용하는 것이 바람직하다.



함수 원형을 사용하지 않는 예제

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

```
// 함수 정의
```

```
int compute_sum(int n)
```

```
{
```

```
    int i;
```

```
    int result = 0;
```

```
    for(i = 1; i <= n; i++)
```

```
        result += i;
```

```
    return result;
```

```
}
```

```
int main(void)
```

```
{
```

```
    int n, sum;
```

```
    printf("정수를 입력하시오: ");
```

```
    scanf("%d", &n);
```

```
    sum = compute_sum(n);           // 함수 사용
```

```
    printf("1부터 %d까지의 합은 %d입니다. \n", n, sum);
```

```
    return 0;
```

```
}  
©
```



정수를 입력하시오: 10

1부터 10까지의 합은 55입니다.



함수 원형과 헤더 파일

- 보통은 헤더 파일에 함수 원형이 선언되어 있음

```
/* 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;
}
```

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

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

stdio.h



중간 점검

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

`double pow(double, double);`





함수를 사용하는 이유

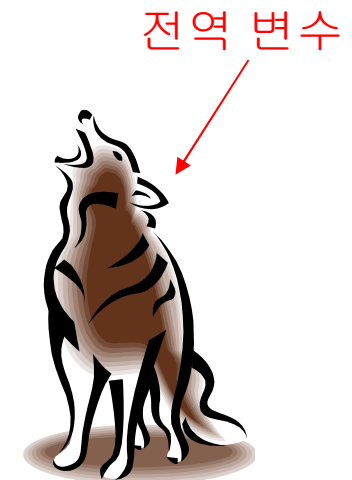
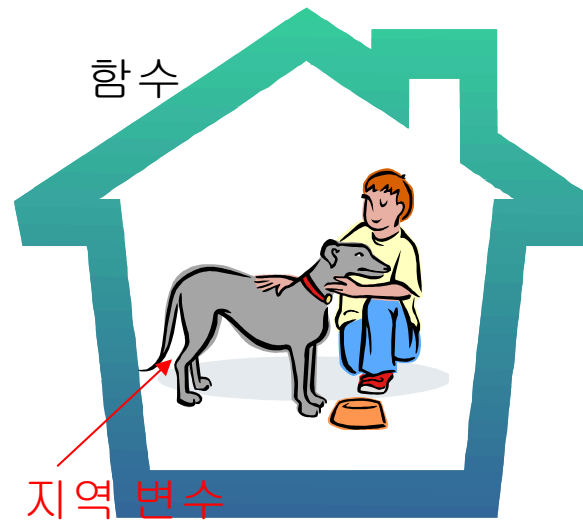
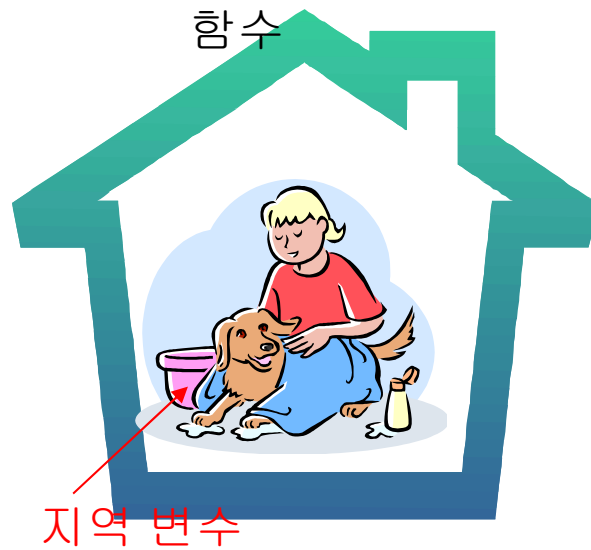
- 소스 코드의 중복을 없애준다.
 - 한번 만들어진 함수를 여러 번 호출하여 사용할 수 있다.
- 한번 작성된 함수를 다른 프로그램에서도 사용할 수 있다.
- 복잡한 문제를 단순한 부분으로 분해할 수 있다.

```
void print_heading(void)
{
    printf("*****");
    printf(" NAME ADDRESS PHONE ");
    printf("*****");
}
int main(void)
{
    // 출력이 필요한 위치 #1
    print_heading();
    ...
    // 출력이 필요한 위치 #2
    print_heading();
    ...
    ...
}
```

```
int main(void)
{
    ...
    read_list();
    sort_list();
    print_list();
    ...
}
```




변수의 범위





지역 변수

- 지역 변수(local variable): 함수나 블록 안에 선언되는 변수

```
int compute_sum(int n)
```

```
{
```

```
    int i, result = 0;
```

```
    for(i = 1; i <= n; i++)
```

```
        result += i;
```

```
    return result;
```

```
}
```

지역 변수

지역 변수는 여기서 소멸된다.



지역 변수의 사용 범위

```
int sub1()
```

```
{
```

```
    int x;
```

```
    ...
```

```
}
```

```
void sub2()
```

```
{
```

```
    printf("%d\n", x); // 컴파일 오류!
```

```
}
```

지역 변수 x의 범위.



블록 안에서의 지역 변수

```
int sub1()
```

```
{
```

```
    int x;
```

```
    while(1)
```

```
    {
```

```
        int y;
```

```
        ...
```

```
    }
```

```
    ...
```

```
}
```

지역 변수 x의 범위.

지역 변수 y의 범위.



지역 변수의 생존 기간

- 정의된 블록이나 함수 안에서만 생존한다.

```
int sub()  
{  
    int i = 0;  
  
    ...  
    return result;  
}
```

지역 변수 생성

지역 변수 소멸



지역 변수의 초기값

```
#include <stdio.h>

int main(void)
{
    int temp;

    printf("temp = %d\n", temp);
}
```

초기화시키지 않으면 쓰레기값

실행 결과

temp = -858993460





함수의 매개 변수

- 매개 변수도 일종의 지역 변수
- 함수를 호출할 때 넣어주는 인수 값으로 초기화

```
int inc(int counter)
{
    counter++;
    return counter;
}
```

일종의 지역 변수,

함수 호출시 인수값으로 초기화된다.



함수의 매개 변수



```
#include <stdio.h>
int inc(int counter);
```

```
int main(void)
{
    int i;
```

```
    i = 10;
    printf("함수 호출전 i=%d\n", i);
    inc(i);
    printf("함수 호출후 i=%d\n", i);
    return 0;
```

```
}
```

```
int inc(int counter)
```

```
{
    counter++;
    return counter;
}
```

변수 i의 값이 counter로 복사된다.



함수 호출전 i=10
함수 호출후 i=10

main 함수 안의 변수 i의 값은
변경되지 않음



같은 이름의 지역 변수

```
int sub1()
```

```
{
```

```
    int x = 0;
```

```
    ...
```

```
}
```

```
int sub2()
```

```
{
```

```
    int x = 0;
```

```
    ...
```

```
}
```

이름이 같아도 각각 다른 변수이다.



전역 변수

- 전역 변수(global variable): 함수의 외부에 선언되는 변수
- 초기값을 주지 않으면 0이다.

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

```
int global = 123;
```

전역 변수

```
void sub1()
```

```
{
```

```
    printf("%d\n", global);
```

```
}
```

전역 변수는 소스 파일의 어디서나 사용 가능

```
void sub2()
```

```
{
```

```
    printf("%d\n", global);
```

```
}
```



전역 변수의 초기값과 생존 기간



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

```
int counter; // 전역 변수
```

```
void set_counter(int i)
```

```
{
```

```
    counter = i;    // 직접 사용 가능
```

```
}
```

```
int main(void)
```

```
{
```

```
    printf("counter=%d\n", counter);
```

```
    counter = 100;    // 직접 사용 가능  
    printf("counter=%d\n", counter);
```

```
    set_counter(20);  
    printf("counter=%d\n", counter);
```

```
    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 <stdio.h>
```

```
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++)
```

```
        printf("#");
```

```
}
```

출력은
어떻게
될까요?



```
#####
```



같은 이름의 전역 변수와 지역 변수



```
// 동일한 이름의 전역 변수와 지역 변수  
#include <stdio.h>
```

```
int sum = 1;    // 전역 변수
```

```
int main(void)  
{
```

```
    int i = 0;
```

```
    int sum = 0;    // 지역 변수
```

```
    for(i = 0; i <= 10; i++)
```

```
    {
```

```
        sum += i;
```

```
    }
```

```
    printf("sum = %d\n", sum);
```

```
    return 0;
```

```
}
```

지역 변수가
전역 변수를
가린다.



```
sum = 55
```



중간 점검

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



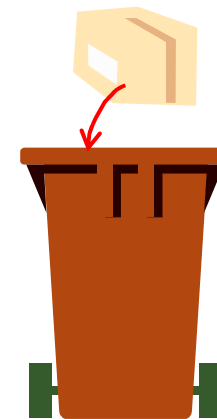


생존 기간

- 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
 - **register**
 - **static**
 - **extern**



변수 생성



변수 소멸



저장 유형 지정자 **static**

- 정적 변수: 블록에서만 사용되지만 블록을 벗어나도 자동으로 삭제되지 않는 변수
- 앞에 **static**을 붙인다.

```
void sub()
```

```
{
```

```
    static int count;
```

```
    ....
```

```
    return;
```

```
}
```

정적 변수



저장 유형 지정자 **static**



```
#include <stdio.h>
void sub(void);
```

```
int main(void)
{
    int i;
    for(i = 0; i < 3; i++)
        sub();
    return 0;
}
```

```
void sub(void)
{
```

```
    int auto_count = 0;
```

```
    static int static_count = 0;
```

```
    auto_count++;
```

```
    static_count++;
```

```
    printf("auto_count=%d\n", auto_count);
```

```
    printf("static_count=%d\n", static_count);
```

```
}
```



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

자동 지역 변수

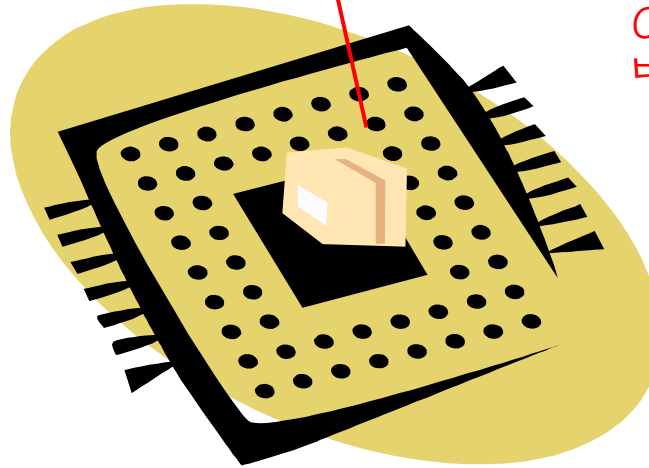
정적 지역 변수로서
static을 붙이면 지역 변수가
정적 변수로 된다.



저장 유형 지정자 register

- 레지스터(register)에 변수를 저장.

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

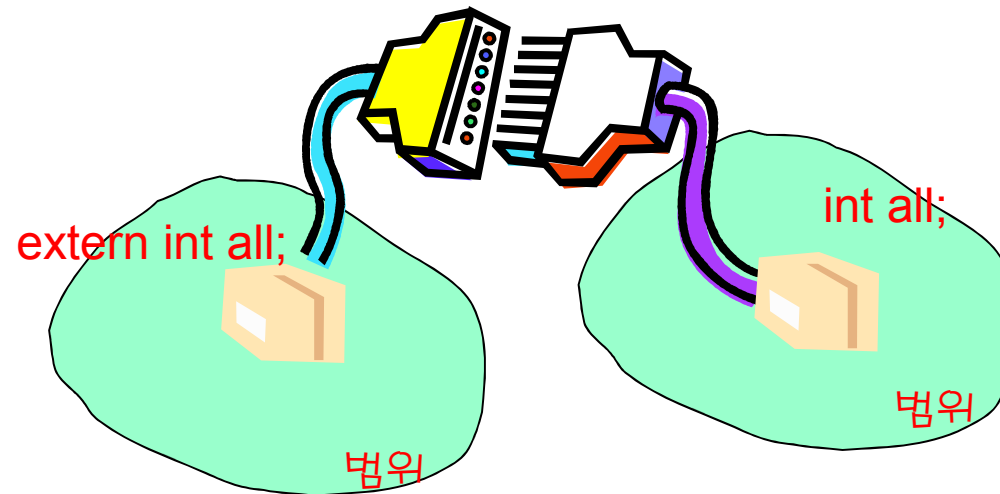


CPU안의 레지스터에
변수가 저장됨



저장 유형 지정자 `extern`

- `extern`은 컴파일러에게 변수가 현재 범위가 아닌 다른 곳에서 선언되었다는 것을 알린다.





extern 예제



linkage1.c

```
#include <stdio.h>
int all_files; // 다른 소스 파일에서도 사용할 수 있는 전역 변수
extern void sub();

int main(void)
{
    sub();
    printf("%d\n", all_files);
    return 0;
}
```

연결



linkage2.c

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



10



저장 유형 정리

- 일반적으로는 **지역 변수** 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**



중간 점검

1. 저장 유형 지정자에는 어떤 것들이 있는가?
2. 지역 변수를 정적 변수로 만들려면 어떤 지정자를 붙여야 하는가?
3. 변수를 **CPU** 내부의 레지스터에 저장시키는 지정자는?
4. 컴파일러에게 변수가 외부에 선언되어 있다고 알리는 지정자는?
5. **extern** 지정자를 변수 앞에 붙이면 무엇을 의미하는가?
6. **static** 지정자를 변수 앞에 붙이면 무엇을 의미하는가?

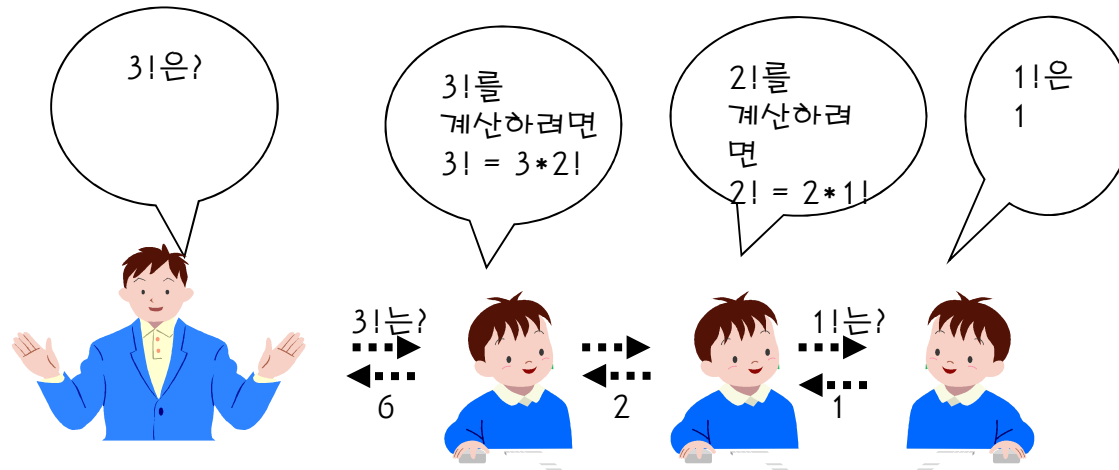




순환(recursion)이란?

- 알고리즘이나 함수가 수행 도중에 자기 자신을 다시 호출하여 문제를 해결하는 기법
- (예제) 팩토리얼의 정의

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





팩토리얼 구하기

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

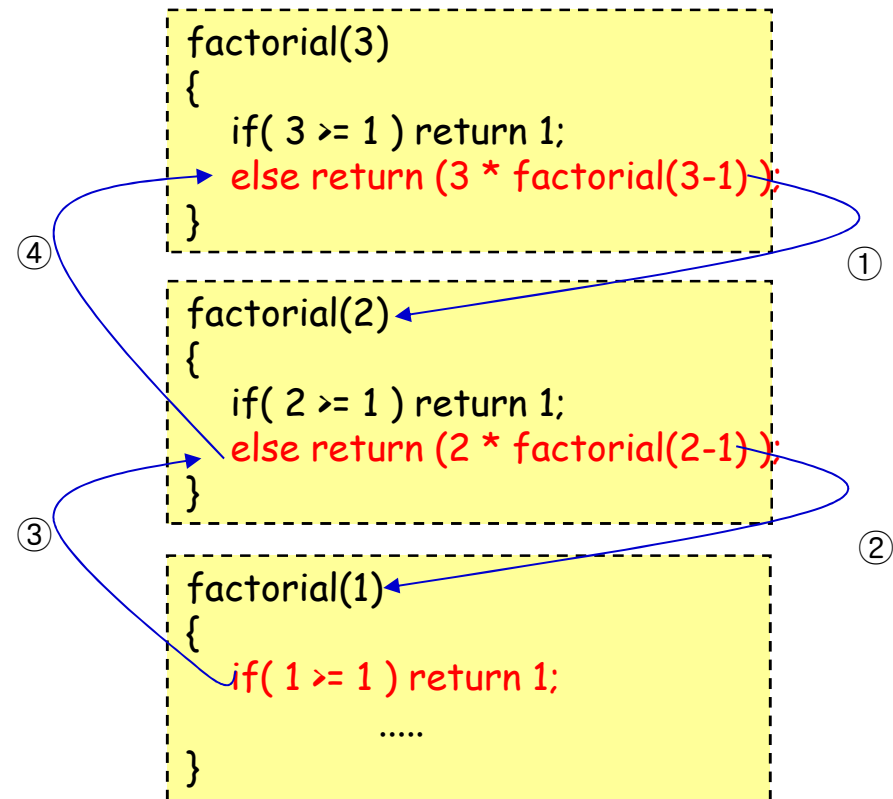
```
int factorial(int n)
{
    if( n >= 1 ) return(1);
    else return (n * factorial(n-1) );
}
```




팩토리얼 구하기 #2

- 팩토리얼의 호출 순서

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





팩토리얼 예제



// 재귀적인 팩토리얼 함수 계산

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

```
long factorial(int n);
```

```
int main(void)
```

```
{
```

```
    int x = 0;
```

```
    long f;
```

```
    printf("정수를 입력하시오:");
```

```
    scanf("%d", &x);
```

```
    f = factorial(x);
```

```
    printf("%d!은 %d입니다.\n", x, f);
```

```
    return 0;
```

```
}
```



팩토리얼 예제



```
long factorial(int n)
```

```
{
```

```
    printf("factorial(%d)\n", n);
```

```
    if(n <= 1) return 1;
```

```
    else return n * factorial(n - 1);
```

```
}
```

순환호출



정수를 입력하시오:5

factorial(5)

factorial(4)

factorial(3)

factorial(2)

factorial(1)

5!은 120입니다.



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

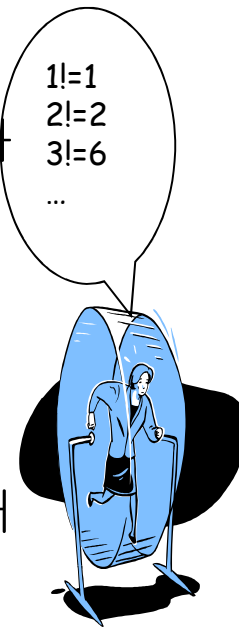
- 컴퓨터에서의 되풀이
 - 순환(recursion): 순환 호출 이용
 - 반복(iteration): for나 while을 이용한 반복
- 대부분의 순환은 반복으로 바꾸어 작성할 수 있다.

- 순환

- 순환적인 문제에서는 자연스러운 방법
- 함수 호출의 오버헤드

- 반복

- 수행속도가 빠르다.
- 순환적인 문제에 대해서는 프로그램 작성이 아주 어려울 수도 있다.





팩토리얼의 반복적 구현

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

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



중간 점검

1. `factorial()` 함수를 재귀를 사용하지 않고 반복문으로 다시 작성하여 보자.
2. `factorial()` 함수 안에 `if(n <= 1) return;`이라는 문장이 없으면 어떻게 될까?



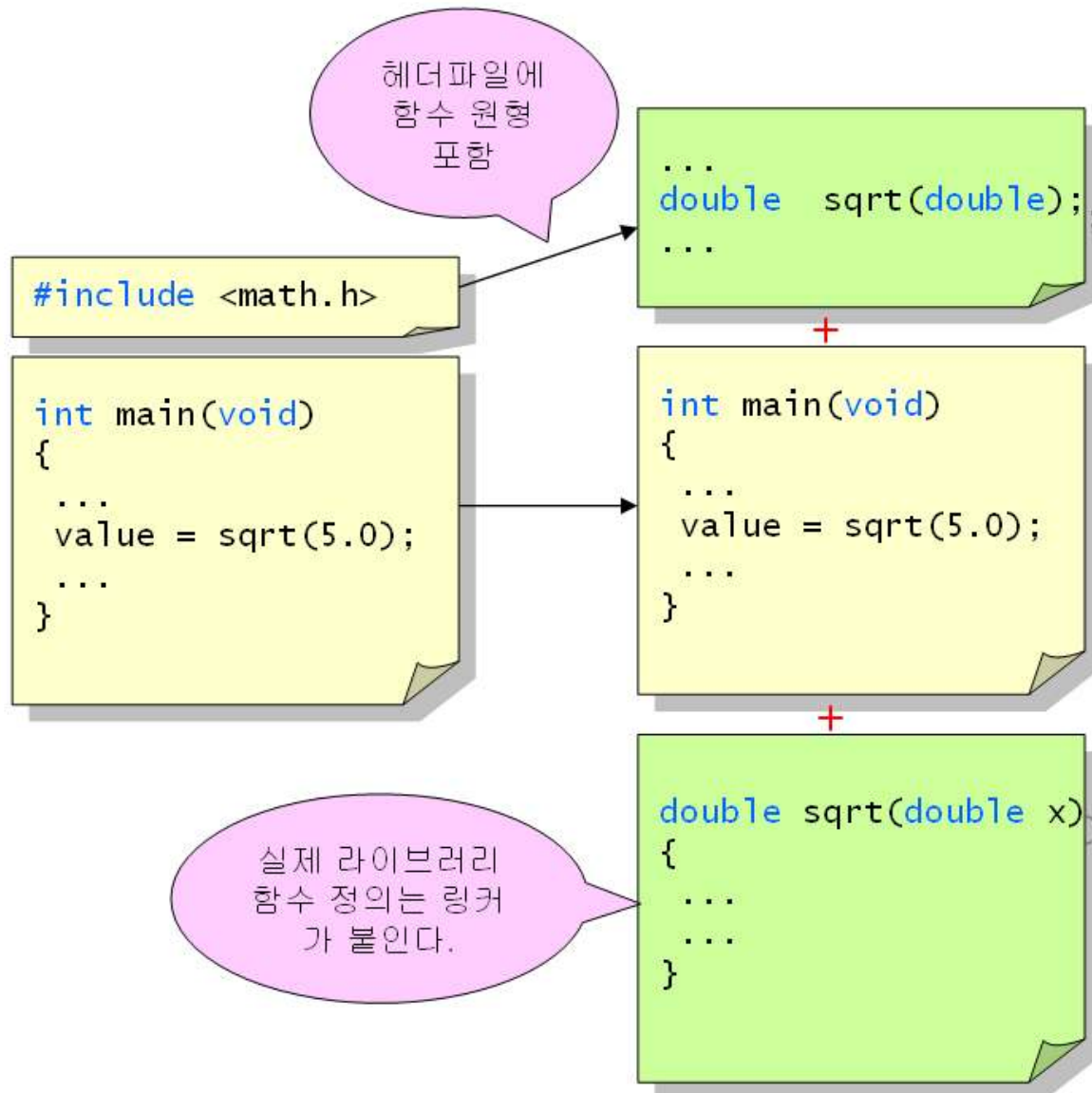


라이브러리 함수

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



라이브러리 사용





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}



예제

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

int main()
{
    double pi = 3.1415926535;

    printf("sin()= %f\n", sin(pi/2.0));
    printf("cos()= %f\n", cos(pi/2.0));
    printf("tan()= %f\n", tan(0.5));
    printf("log()= %f\n", log(10.0));
    printf("log10()= %f\n", log10(100.0));
    printf("exp()= %f\n", exp(10.0));
    return 0;
}
```



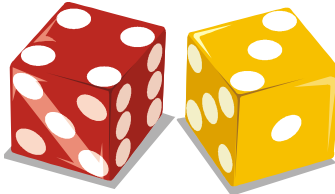
```
sin()= 1.000000
cos()= 0.000000
tan()= 0.546302
log()= 2.302585
log10()= 2.000000
exp()= 22026.465795
```

계속하려면 아무 키나 누르십시오 ...



난수 생성 라이브러리 함수

- rand()
 - 난수를 생성하는 함수
 - 0부터 **RAND_MAX**까지의 난수를 생성





난수 생성 라이브러리 함수



// 난수 생성 프로그램

#include <stdlib.h>

#include <stdio.h>

#include <time.h>

// n개의 난수를 화면에 출력한다.

```
void get_random( int n )
{
    int i;
    for( i = 0; i < n; i++ )
        printf( " %6d\n", rand() );
}
```

```
int main( void )
```

```
{
    // 일반적으로 난수 발생기의 시드(seed)를 현재 시간으로 설정한다.
    // 현재 시간은 수행할 때마다 달라지기 때문이다.
    srand( (unsigned)time( NULL ) );
    get_random( 10 );
    return 0;
}
```



```
16154
18011
20719
15002
25104
31802
587
8161
28527
8385
```



중간 점검

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





Q & A

