

8장. 함수

숭실대학교
전자정보공학부 IT융합전공
담당교수: 권민혜
minhae@ssu.ac.kr

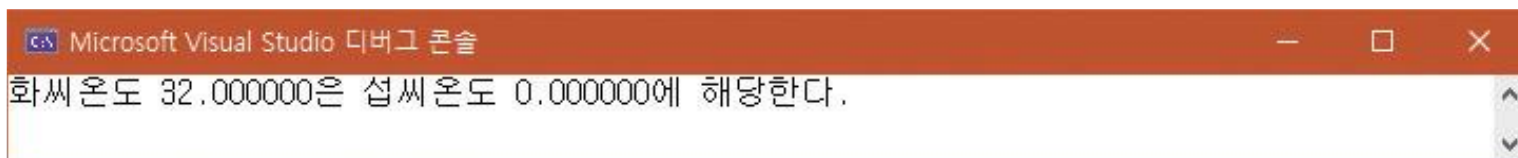
이번 장에서 학습할 내용

- 함수의 개념을 이해한다.
- 함수를 작성할 수 있다.
- 함수의 반환값과 매개 변수를 이해한다.
- 전역 변수와 지역 변수를 이해한다.
- 순환 호출을 이해하고 사용할 수 있다.

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

이번 장에서 만들 프로그램

- 온도를 변환하는 함수를 정의하고 사용해본다.



```
C:\N Microsoft Visual Studio 디버그 콘솔
화씨 온도 32.000000은 섭씨 온도 0.000000에 해당한다.
```

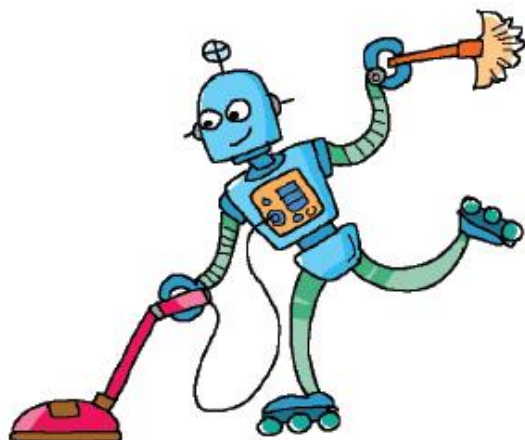
- ATM(현금입출력기)을 구현해본다. 각 기능을 하나의 함수로 구현해본다.



```
C:\N D:\hello\Debug\hello.exe
*****Welcome to 콘서트 ATM*****
****하나를 선택하시오****
<1> 잔고 확인
<2> 입금
<3> 출금
<4> 종료
2
****입금 금액을 입력하시오
10000
새로운 잔고는 20000입니다.
```

되풀이 되는 작업

- 일상생활에서도 되풀이되는 귀찮은 작업이 있듯이 프로그램에서도 되풀이되는 작업이 있다.



함수의 필요성

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

```
int main(void)
```

```
{
```

```
    printf("*****\n");  
    printf(" 송실대학교 전자정보공학부 \n");  
    printf(" 나송실 \n");  
    printf("*****\n");
```

```
    printf("*****\n");  
    printf(" 송실대학교 전자정보공학부 \n");  
    printf(" 나송실 \n");  
    printf("*****\n");
```

```
    return 0;
```

```
}
```

함수의 필요성

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

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

```
void print_name()  
{  
    printf("*****\n");  
    printf(" 송실대학교 전자정보공학부 \n");  
    printf(" 나송실 \n");  
    printf("*****\n");  
}
```

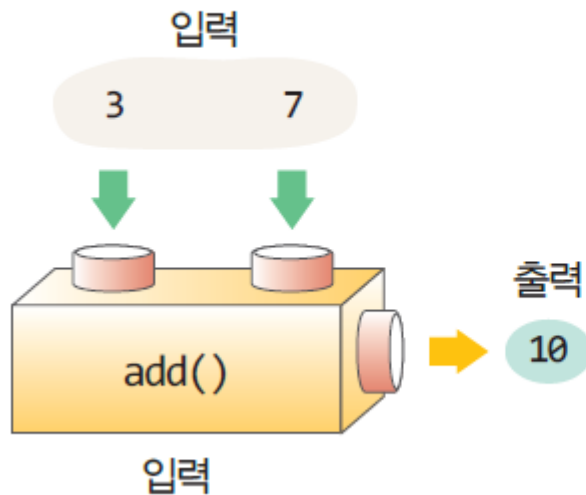
```
int main(void)  
{  
    print_name();  
    print_name();  
  
    return 0;  
}
```

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.

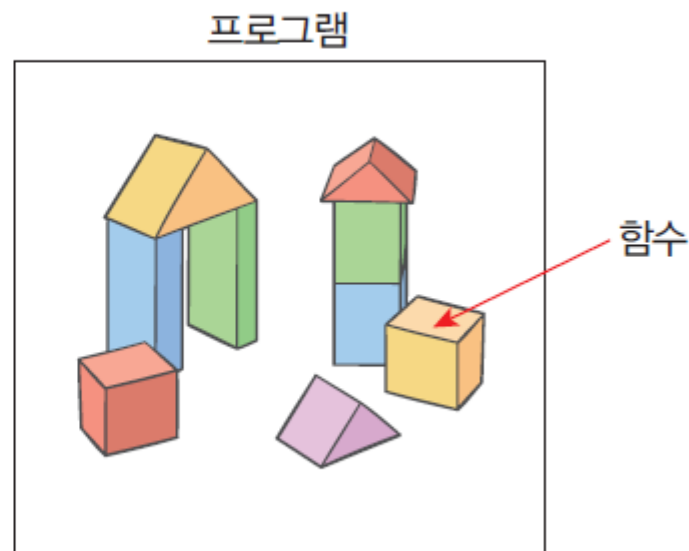
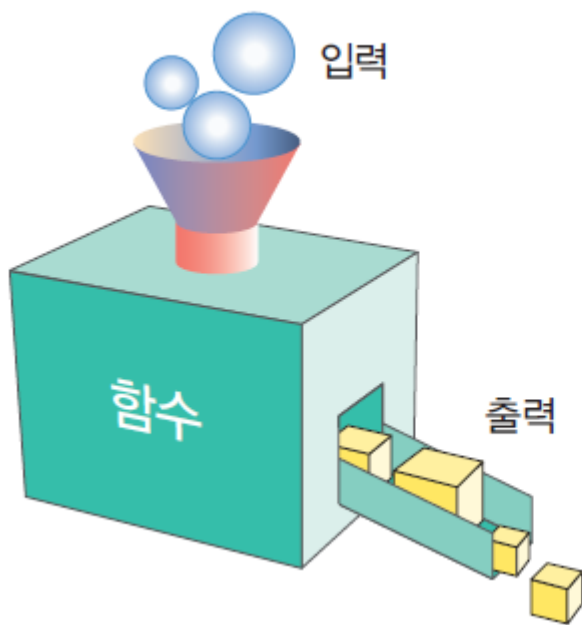
함수의 개념

- **함수(function)**: 특정 작업을 수행하여 그 결과를 반환하는 문장들의 집합



함수는 프로그램을 이루는 빌딩블록

- 하나의 프로그램은 여러 함수들이 모여서 이루어진다.
- 함수는 레고 블록과 같은 역할을 하는 것이다. 함수는 프로그램을 이루는 부품으로 생각하면 이해하기 쉽다.



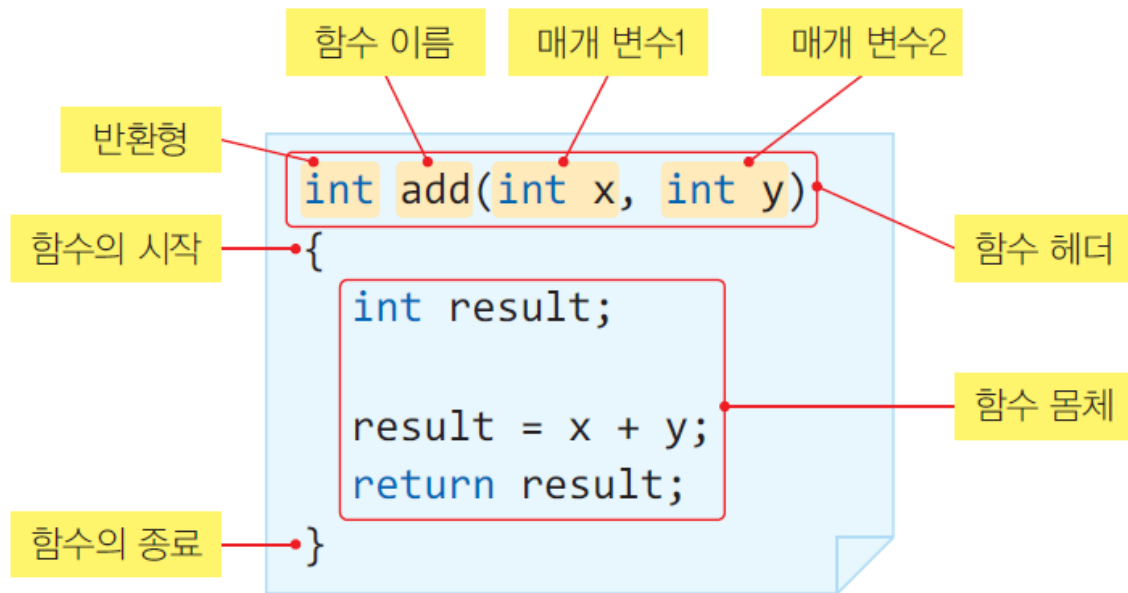
중간점검



중간점검

1. 함수가 필요한 이유는 무엇인가?
2. 함수와 프로그램의 관계는?

함수의 정의



반환형

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

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

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

함수 이름

- 일반적으로 **동사+명사**
- (예)
 - square() // 정수를 제공하는 함수
 - compute_average() // 평균을 구하는 함수
 - get_integer() // 정수를 받아들이는 함수

```
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;
}
```

참고사항



참고

함수에는 얼마든지 많은 문장들을 넣을 수 있지만 함수의 길이가 지나치게 길어지면 좋지 않다. 기본적으로 하나의 함수는 하나의 작업만을 수행하여야 한다. 일반적으로 하나의 함수는 30행을 넘지 않도록 하는 것이 좋다.

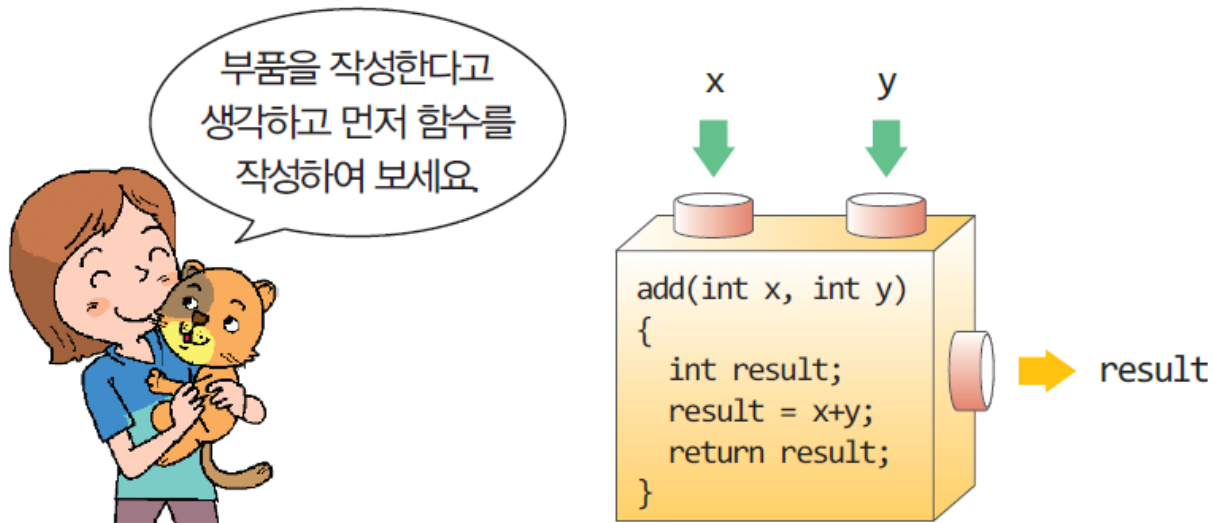


참고

만약 함수의 반환형을 명시하지 않으면 C 컴파일러는 `int` 형을 가정한다. 그러나 특별한 경우가 아니면 반환형을 생략하면 안 된다. 반환형이 `int`일지라도 항상 반환형을 명시하는 것이 좋다. 또한 반환 값이 있는 함수에서 값을 반환하지 않으면 예측 불가능한 값이 전달될 수 있다. 또 반대로 반환 값이 `void`로 지정된 함수에서 값을 반환하면 문법적인 오류가 발생한다.

함수 정의 예제

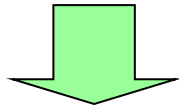
- 함수를 프로그램을 이루는 부품이라고 가정하자.
- 입력을 받아서 작업한 후에 결과를 생성한다.



예제 #1

- 사용자로부터 정수 반환

반환값: `int`
함수 이름: `get_integer`
매개 변수: 없음

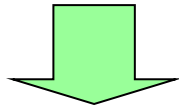


```
int get_integer()
{
    int value;
    printf("정수를 입력하시오 : ");
    scanf("%d", &value);
    return value;
}
```

예제 #2

- 두개의 정수중에서 큰 수를 계산하는 함수

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

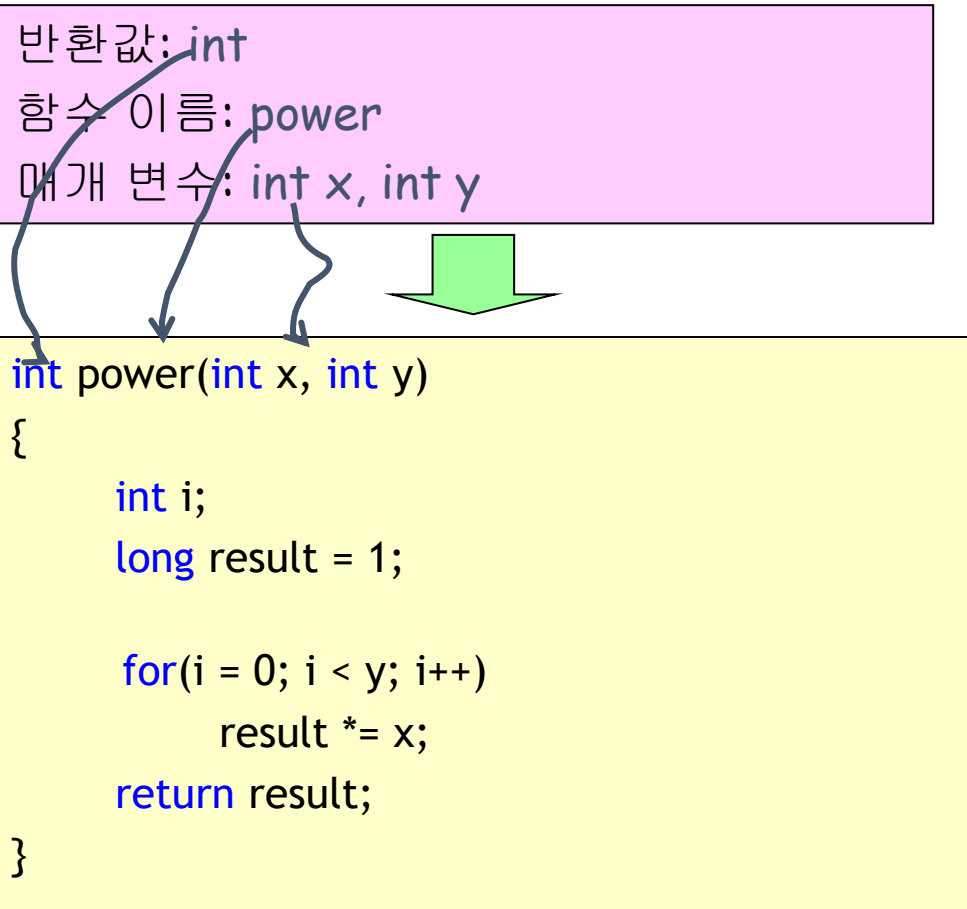


```
int get_max(int x, int y)
{
    if( x > y ) return(x);
    else return(y);
}
```

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



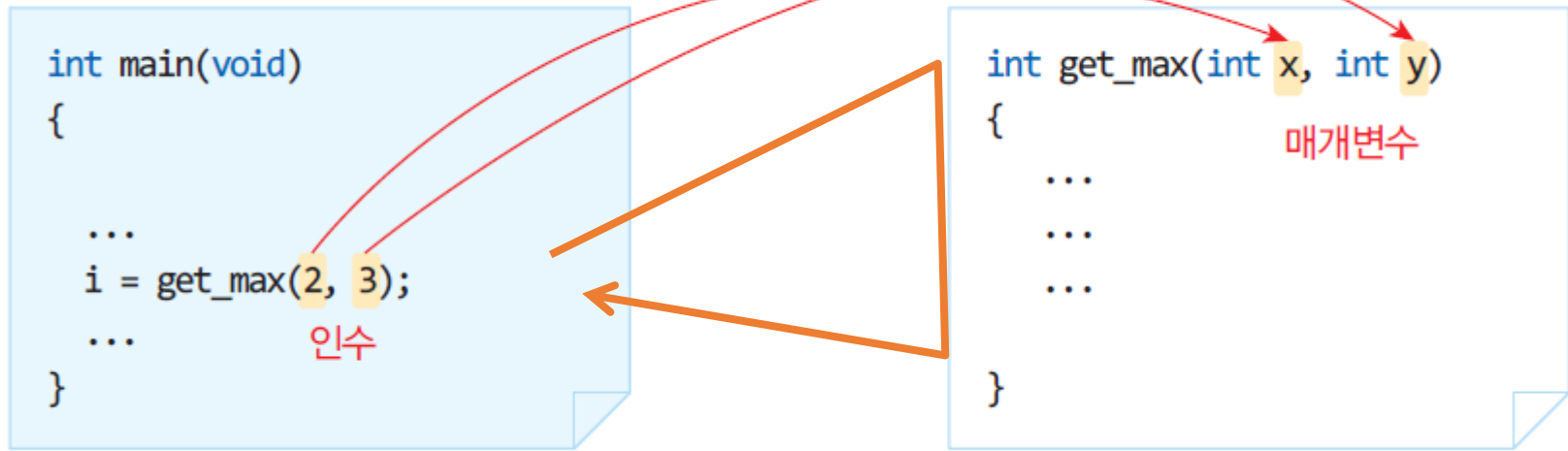
중간점검

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 둘러싸여 있는가?
4. 함수의 몸체 안에서 정의되는 변수를 무엇이라고 하는가?



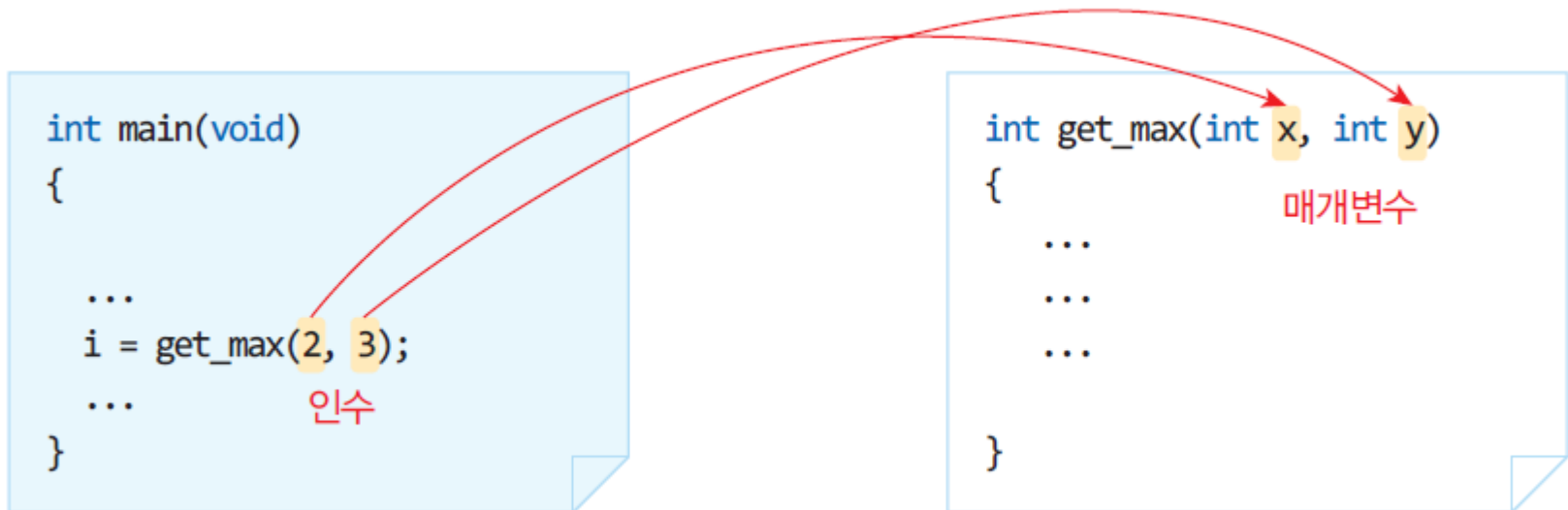
함수 호출과 반환

- 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

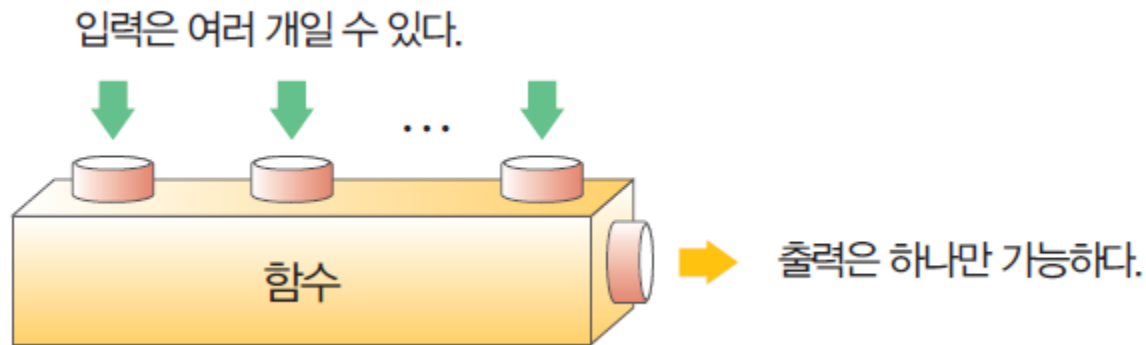
- *인수(argument)*: 실인수, 실매개 변수라고도 한다.
- *매개 변수(parameter)*: 형식 인수, 형식 매개 변수라고도 한다.



`get_max(10);` // 인수가 두 개이어야 한다.
`get_max(0.1, 0.2);` // `get_max()` 인수의 타입은 정수이어야 한다.
`get_max();` // 인수가 두 개이어야 한다.

반환값

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



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


예제 #1

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

// 함수를 정의한다.
int get_integer()
{
    int value;
    printf("정수를 입력하시오 : ");
    scanf("%d", &value);
    return value;
}

int main(void)
{
    int x = get_integer(); // 함수를 호출한다.
    int y = get_integer(); // 함수를 호출한다.
    int result = x + y;
    printf("두수의 합 = %d \n", result);

    return 0;
}
```

실행 결과



The screenshot shows the 'Microsoft Visual Studio 디버그 콘솔' (Debug Console) window. It contains three lines of text: '정수를 입력하시오 : 10', '정수를 입력하시오 : 20', and '두 수의 합 = 30'. The window has a standard Windows title bar with minimize, maximize, and close buttons.

```
Microsoft Visual Studio 디버그 콘솔
정수를 입력하시오 : 10
정수를 입력하시오 : 20
두 수의 합 = 30
```

예제 #2

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

```
// 함수를 정의한다.
```

```
int get_integer()
{
    int value;
    printf("정수를 입력하시오 : ");
    scanf("%d", &value);
    return value;
}
```

```
// 함수를 정의한다.
```

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

예제 #2

```
int main(void)
{
    int a = get_integer();    // 함수 호출
    int b = get_integer();    // 함수 호출

    printf("두수 중에서 큰 수는 %d입니다.\n", get_max(a, b)); // 함수 호출
    return 0;
}
```

Microsoft Visual Studio 디버그 콘솔

정수를 입력하시오 : 10
정수를 입력하시오 : 20
두수 중에서 큰 수는 20입니다.

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
// 함수를 정의한다.
int get_integer()
{
    int value;
    printf("정수를 입력하시오 : ");
    scanf("%d", &value);
    return value;
}
// 함수를 정의한다.
int power(int x, int y)
{
    int i, result = 1;

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

예제 #3

```
int main(void)
{
    int x = get_integer(); // 함수를 호출한다.
    int y = get_integer(); // 함수를 호출한다.
    int result = power(x, y);
    printf("%d의 %d승 = %d \n", x, y, result);

    return 0;
}
```



The screenshot shows the 'Microsoft Visual Studio 디버그 콘솔' (Debug Console) window. It contains the following text: '정수를 입력하시오 : 10', '정수를 입력하시오 : 2', and '10의 2승 = 100'. The window has a standard Windows title bar with minimize, maximize, and close buttons.

```
C:\> Microsoft Visual Studio 디버그 콘솔
정수를 입력하시오 : 10
정수를 입력하시오 : 2
10의 2승 = 100
```

중간 점검



중간점검

1. 인수와 매개 변수는 어떤 관계가 있는가?
2. 사용자로부터 실수를 받아서 반환하는 함수 `get_real()`을 작성하고 테스트하라.
3. 함수 정의의 첫 번째 줄에는 어떤 정보들이 포함되는가? 이것을 무엇이라고 부르는가?
4. 함수가 반환할 수 있는 값의 개수는?
5. 함수가 값을 반환하지 않는다면 반환형은 어떻게 정의되어야 하는가?



함수 원형

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

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

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

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

```
    int sum;
```

```
    sum = compute_sum(100);
```

```
    printf("1부터 100까지의 합 = %d \n", sum);
```

```
    return 0;
```

```
}
```

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

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

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

```
        result += i;
```

```
    return result;
```

```
}
```

함수 원형

참고와 중간점검



참고

함수 원형을 사용하지 않는 방법도 있다. 함수 원형이란 근본적으로 컴파일러에게 함수에 대한 정보를 주기 위하여 만들어진 것이다. 따라서 사용하려는 함수의 정의가 먼저 등장한다면 구태여 함수 원형을 표시할 필요가 없다. 하지만 함수 호출이 서로 물고 물리는 경우에는 이 방법이 불가능하다. 따라서 대부분의 경우에 먼저 함수 원형을 적어 주는 것이 권장된다.



중간점검

1. 함수 정의와 함수 원형 선언의 차이점은 무엇인가?
2. 함수 원형에 반드시 필요한 것은 아니지만 대개 매개 변수들의 이름을 추가하는 이유는 무엇인가?
3. 다음과 같은 함수 원형을 보고 우리가 알 수 있는 정보는 어떤 것들인가?

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

Lab: 온도 변환 함수

- 섭씨 온도를 화씨 온도로 변환하여 반환하는 함수 `FtoC()`를 작성하고 테스트하라.

```
Microsoft Visual Studio 디버그 콘솔
화씨 온도 32.000000은 섭씨 온도 0.000000에 해당한다.
```



Sol.

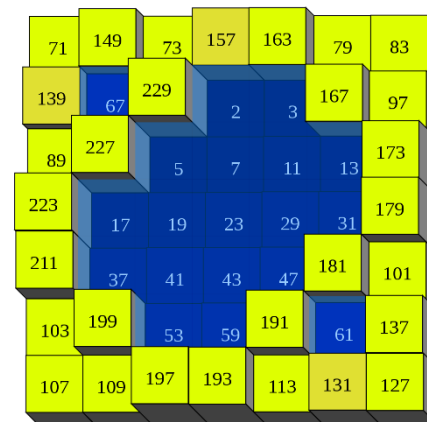
```
#include <stdio.h>
double FtoC(double temp_f);  // 함수 원형 정의

int main(void)
{
    double c, f;
    f = 32.0;
    c = FtoC(f);             // 함수 호출
    printf("화씨온도 %lf은 섭씨온도 %lf에 해당한다. \n", f, c);
    return 0;
}

double FtoC(double temp_f)  // 함수 정의
{
    double temp_c;
    temp_c = (5.0 * (temp_f - 32.0)) / 9.0;
    return temp_c;
}
```

Lab: 소수 검사 함수 작성

- 정수를 전달받아서 소수인지 아닌지를 반환하는 함수를 작성하고 테스트해보자.



Sol.

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
int check_prime(int);

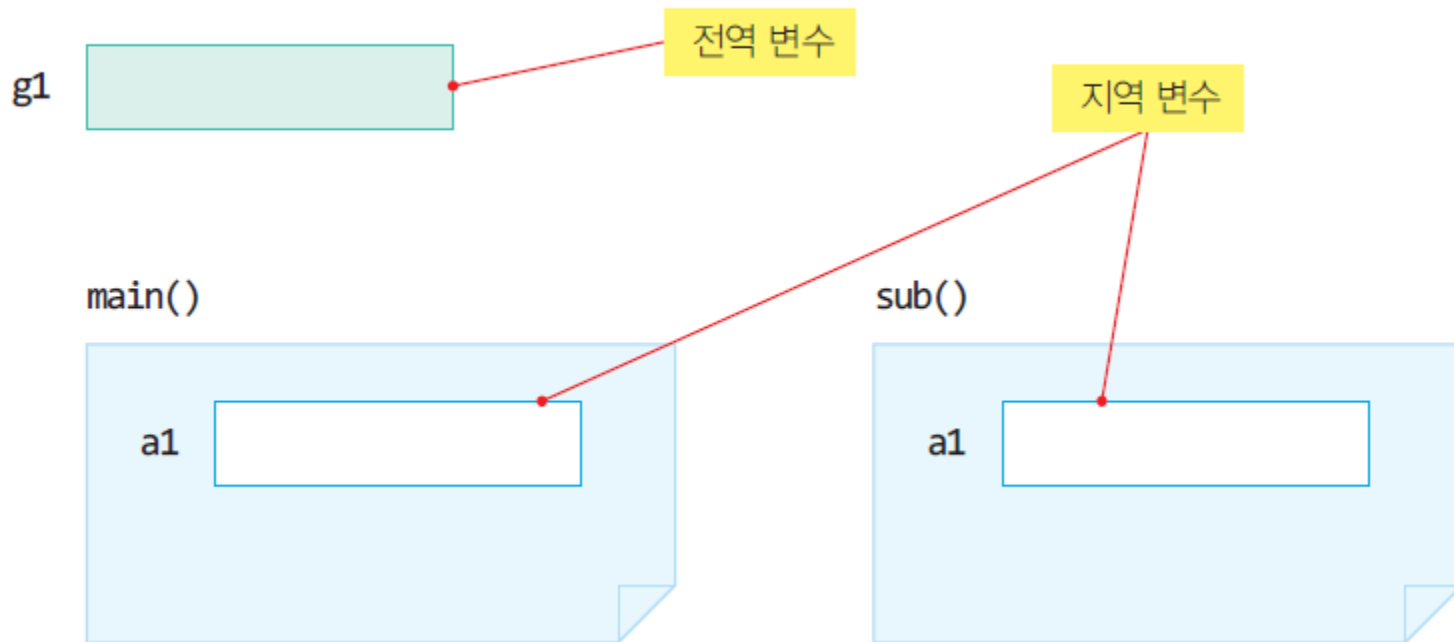
int main(void)
{
    int k;
    printf("정수를 입력하시오: ");
    scanf("%d", &k);
    if (check_prime(k) == 1) printf("소수입니다. \n");
    else printf("소수가 아닙니다. \n");
    return 0;
}
```

Sol.

```
int check_prime(int n) {  
    int is_prime = 1;    // 일단 소수라고 가정한다.  
    for (int i = 2; i < n; ++i) {  
        if (n % i == 0) {  
            is_prime = 0;  
            break;  
        }  
    }  
    return is_prime;  
}
```

지역 변수와 전역 변수

- 함수 안에서 정의되는 변수는 지역 변수라고 불리고 해당 함수 안에서만 사용이 가능하다. 함수의 외부에서 선언되는 변수는 전역 변수라고 불린다.



지역 변수

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

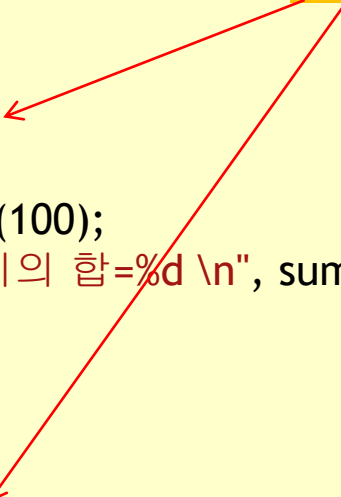
```
#include <stdio.h>
int compute_sum(int n);

int main(void)
{
    int sum;
    sum = compute_sum(100);
    printf("1부터 100까지의 합=%d \n", sum);
    return 0;
}

int compute_sum(int n)
{
    int i;
    int result = 0;

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

지역 변수



지역 변수의 사용 범위

```
int sub1()
```

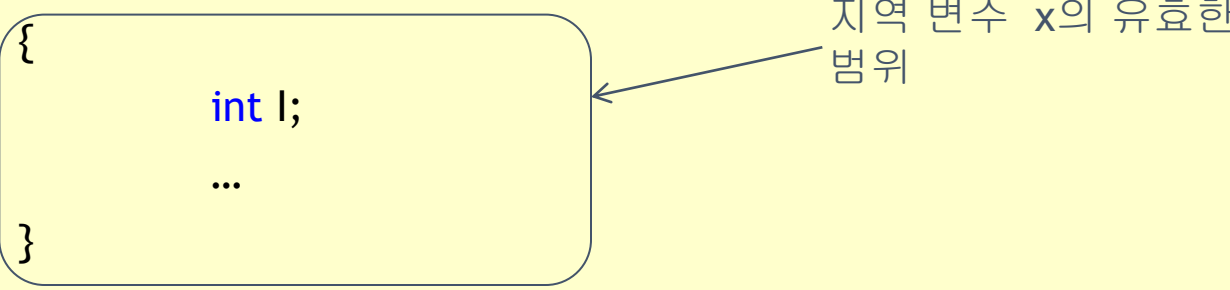
```
{
```

```
    int l;
```

```
    ...
```

```
}
```

지역 변수 x 의 유효한
범위



```
void sub2()
```

```
{
```

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

```
}
```

지역 변수의 초기값 지정 필요

```
int compute_sum(int n)
{
    int i, result;

    for(i=0; i<=n; i++)
        result += i;

    return result;
}
```



Debug Error!

Program:

...15\Projects\ConsoleApplication1\Debug\ConsoleApplication1.exe

Module:

...15\Projects\ConsoleApplication1\Debug\ConsoleApplication1.exe

File:

Run-Time Check Failure #3 - T

(Press Retry to debug the application)



참고



참고: 블록에서도 지역 변수를 선언할 수 있다.

블록은 중괄호 {}을 사용하여 만드는 구간이다. 여기서도 지역 변수를 선언할 수 있다. 블록이 종료되면 지역 변수도 같이 사라진다.

```
while(1) {  
    int x; // 블록 안에서 선언된 지역 변수  
    ...  
} // 변수 x는 여기서 사라진다.
```



참고: 함수의 매개 변수

함수의 헤더 부분에 정의되어 있는 매개 변수도 일종의 지역 변수이다. 즉 지역 변수가 지니는 모든 특징을 가지고 있다. 지역 변수와 다른 점은 함수를 호출할 때 넣어주는 인수 값으로 초기화되어 있다는 점이다.


```
int inc(int counter) // counter도 지역 변수의 일종이라고 생각할 수 있다.  
{  
    counter++; // 지역 변수처럼 사용할 수 있다.  
    return counter;  
}
```

전역 변수

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

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

```
int global = 123;
```



```
void sub1()
```

```
{
```

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

```
}
```

```
void sub2()
```

```
{
```

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

```
}
```

```
int main(void)
```

```
{
```

```
    sub1();
```

```
    sub2();
```

```
    return 0;
```

```
}
```

Microsoft Visual Studio 디버그 콘솔

```
global=123
```

```
global=123
```

전역 변수의 특징

- 전역 변수는 프로그램 시작과 동시에 생성되어 프로그램이 종료되기전까지 메모리에 존재한다.
- 전역 변수는 상당히 편리할 것처럼 생각되지만 전문가들은 사용을 권하지 않는다. 그 이유는 어디서나 접근이 가능하다는 장점이 도리어 단점이 될 수 있기 때문이다. 프로그램이 복잡해지다 보면 전역 변수를 도대체 어떤 부분에서 변경하고 있는지를 잘 모르는 경우가 허다하다.



같은 이름의 전역 변수와 지역 변수

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

```
int sum = 123;
```

```
int main(void)
```

```
{
```

```
    int sum = 321;
```

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

```
    return 0;
```

```
}
```

지역 변수가
전역 변수를
가린다.

C:\ Microsoft Visual Studio 디버그 콘솔

sum=321

중간 점검



중간점검

1. 변수의 범위에는 몇 가지의 종류가 있는가?
2. 블록 범위를 가지는 변수를 무엇이라고 하는가?
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. 전역 변수는 어디에 선언되는가?
8. 전역 변수의 생존 기간과 초기값은?
9. 똑같은 이름의 전역 변수와 지역 변수가 동시에 존재하면 어떻게 되는가?



Lab: 소수의 합 찾기

- 어떤 정수가 소수 2개의 합으로 표시될 수 있는지를 검사하는 프로그램을 작성해보자.

```
Microsoft Visual Studio 디버그 콘솔
양의 정수를 입력하시오: 33
33 = 2 + 31
33 = 31 + 2
```

$$33 = 2 + 31$$

함수를 사용하지
않으면 아주 작성하기
힘든 문제입니다.




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

```
int check_prime(int n);
```

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

```
    int n, flag = 0;
```

```
    printf("양의 정수를 입력하시오: ");
```

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

```
    for (int i = 2; i < n; i++) {
```

```
        if (check_prime(i) == 1) {
```

```
            if (check_prime(n - i) == 1) {
```

```
                printf("%d = %d + %d\n", n, i, n - i);
```

```
                flag = 1;
```

```
            }
```

```
        }
```

```
    }
```

```
    if (flag == 0)
```

```
        printf("%d은 소수들의 합으로 표시될 수 없습니다.\n", n);
```

```
    return 0;
```

```
}
```

Sol.

```
int check_prime(int n) {  
    int is_prime = 1;    // 일단 소수라고 가정한다.  
    for (int i = 2; i < n; ++i) {  
        if (n % i == 0) {  
            is_prime = 0;  
            break;  
        }  
    }  
    return is_prime;  
}
```

정적 지역 변수

- 정적 변수: 블록에서만 사용되지만 블록을 벗어나도 자동으로 삭제되지 않는 변수
- 앞에 static을 붙인다.

```
void sub()
{
    static int count;
    ....

    return;
}
```

정적 변수

저장 유형 지정자 static

```
#include <stdio.h>
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);
}

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

Sub()가 몇 번 호출되었는지 카운터로 사용 가능!

실행 결과



```
Microsoft Visual Studio 디버그 콘솔
auto_count=1
static_count=1
auto_count=1
static_count=2
auto_count=1
static_count=3
```

순환호출

- 순환은 함수가 자기 자신을 호출하여 문제를 해결하는 프로그래밍 기법이다

$$n! = \begin{cases} 1 & n=0 \\ n*(n-1)! & n \geq 1 \end{cases}$$



순환호출

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
int factorial(int n);

int main(void)
{
    int x = 0, result;

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

    result = factorial(x);
    printf("%d!은 %d입니다.\n", x, result);

    return 0;
}

int factorial(int n)
{
    printf("factorial(%d)\n", n);

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

실행 결과

Microsoft Visual Studio 디버그 콘솔

```
정수를 입력하시오:5  
factorial(5)  
factorial(4)  
factorial(3)  
factorial(2)  
factorial(1)  
5!은 120입니다.
```


팩토리얼 함수의 계산

$$\begin{aligned}\text{factorial}(5) &= 5 * \text{factorial}(4) \\ &= 5 * 4 * \text{factorial}(3) \\ &= 5 * 4 * 3 * \text{factorial}(2) \\ &= 5 * 4 * 3 * 2 * \text{factorial}(1) \\ &= 5 * 4 * 3 * 2 * 1 \\ &= 120\end{aligned}$$

중간점검



중간점검

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

Lab: 피보나치 수열 계산(순환 버전)

- 순환 호출을 사용하여 피보나치 수열을 출력하는 프로그램을 작성해보자.
피보나치 수열이란 다음과 같이 정의되는 수열이다.

$$fib(n) \begin{cases} 0 & n=0 \\ 1 & n=1 \\ fib(n-2) + fib(n-1) & otherwise \end{cases}$$

Sol:

```
#include <stdio.h>

int fibonacci(int n) {
    if (n == 0) {
        return 0;
    }
    else if (n == 1) {
        return 1;
    }
    else {
        return (fibonacci(n - 1) + fibonacci(n - 2));
    }
}

int main(void)
{
    for (int i = 0; i < 10; i++) {
        printf("%d ", fibonacci(i));
    }
}
```

라이브러리 함수

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

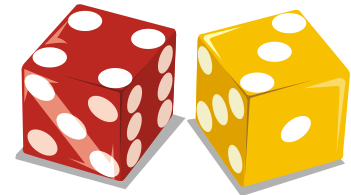


라이브러리 함수

함수	설명	사용예	반환 값
double sin(double x)	사인값 계산	sin(3.14/2.0)	1.0
double cos(double x)	코사인값 계산	cos(3.14/2.0)	0.0
double tan(double x)	탄젠트값 계산	tan(0.5)	0.546302
double exp(double x)	e^x	exp(10.0)	22026.5
double log(double x)	$\log_e x$	log(10.0)	2.30259
double log10(double x)	$\log_{10} x$	log10(100.0)	2.0
double ceil(double x)	x보다 작지 않은 가장 작은 정수	ceil(3.8)	4.0
double floor(double x)	x보다 크지 않은 가장 큰 정수	floor(3.8)	3.0
double fabs(double x)	x의 절대값	fabs(-3.67)	3.67
double pow(double x, double y)	x^y	pow(3.0,2.0)	9.0
double sqrt(double x)	\sqrt{x}	sqrt(4.0)	2.0

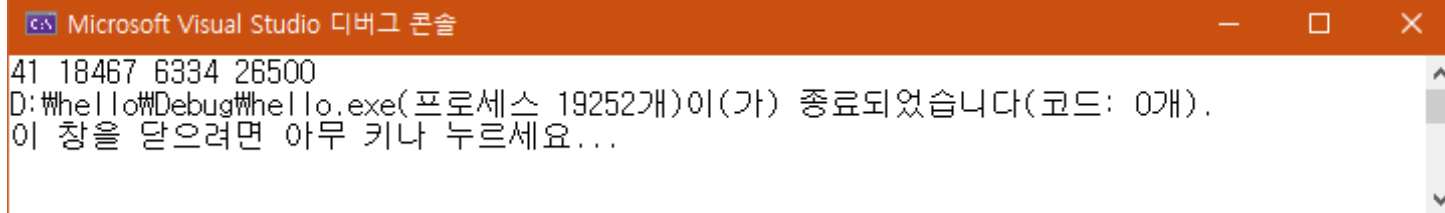
난수 함수

- 난수(**random number**)는 규칙성이 없이 임의로 생성되는 수이다.
- 난수는 암호학이나 시뮬레이션, 게임 등에서 필수적이다.
- rand()
 - 난수를 생성하는 함수
 - 0부터 RAND_MAX까지의 난수를 생성
- `number = rand();`



예제

```
int main(void)
{
    printf("%d ", rand());
    printf("%d ", rand());
    printf("%d ", rand());
    printf("%d ", rand());
}
```



The screenshot shows the 'Microsoft Visual Studio 디버그 콘솔' (Debug Console) window. The title bar is orange with the Visual Studio icon and text. The console area has a white background and shows the output of the program: '41 18467 6334 26500'. Below this, a message in Korean states: 'D:\hello\Debug\hello.exe(프로세스 19252개)이(가) 종료되었습니다(코드: 0개). 이 창을 닫으려면 아무 키나 누르세요...' (D:\hello\Debug\hello.exe (Process 19252) has terminated (code: 0). Press any key to close this window...). The window has standard Windows window controls (minimize, maximize, close) in the top right corner.

```
Microsoft Visual Studio 디버그 콘솔
41 18467 6334 26500
D:\hello\Debug\hello.exe( 프로세스 19252개)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```


난수의 값을 어떤 범위로 한정하려면

전체적으로는 1에서 45까지의 값이 생성된다.

$1 + (\text{rand()} \% 45)$

0에서 44까지의 값이 생성된다.

난수 시드

- 프로그램을 실행할 때마다 다른 난수가 생성되게 하려면 어떻게 해야 할까?
매번 난수를 다르게 생성하려면 시드(seed)라는 개념을 사용한다.
- 시드란 "씨앗"이라는 의미로, 시드는 난수 생성시에 씨앗값이 된다.

난수의 시드를 설정한다.

```
srand( time(NULL) );
```

현재 시간을 얻는다.

실행할 때마다 다르게 하려면

- 매번 난수를 다르게 생성하려면 시드(seed)를 다르게 하여야 한다.

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

```
#define MAX 45
```

```
int main( void )
```

```
{
```

```
    int i;
```

```
    srand( time( NULL ) );
```

```
    for( i = 0; i < 6; i++ )
```

```
        printf("%d ", 1+rand()%MAX );
```

```
    return 0;
```

```
}
```

시드를 설정하는 가장 일반적인 방법은 현재의 시각을 시드로 사용하는 것이다. 현재 시각은 실행할 때마다 달라지기 때문이다.

Lab: 로또 번호 생성하기

- 1부터 45번 사이의 난수 발생
- 중복을 방지해보자.



4 21 22 34 37 38 + 보너스번호 33 내 번호 당첨조회

Sol.

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

```
#define MAX 45
```

```
int main(void)
```

```
{
```

```
    int i, k, lotto[6] = { 0 };
```

```
    int dup_check[MAX + 1] = { 0 };
```



```
    srand(time(NULL));
```

```
    for (i = 0; i < 6; i++)
```

```
    {
```

```
        k = 1 + (rand() % MAX);
```

```
        while (dup_check[k] == 1)
```

```
            k = 1 + (rand() % MAX);
```

```
        lotto[i] = k;
```

```
        dup_check[k] = 1;
```

```
        printf("%d ", lotto[i]);
```

```
    }
```


```
    return 0;
```

```
}
```

Lab: 테일러 급수 계산하기

지수 함수 e^x 에 대한 테일러 급수는 다음과 같다.

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = \frac{x^0}{0!} + \frac{x^1}{1!} + \frac{x^2}{2!} + \dots$$

 Microsoft Visual Studio 디버그 콘솔

```
x와 n의 값을 입력하시오: 2 100  
e^2 = 7.389
```

Sol:

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdlib.h>
#include <stdio.h>

double power(int x, int y) {
    double result = 1.0;
    for (int i = 0; i < y; i++)
        result *= x;
    return result;
}

double factorial(int n) {
    double result = 1.0;
    if (n <= 1) return 1;
    for (int i = 1; i <= n; i++)
        result *= i;
    return result;
}
```

Sol:

```
int main(void) {  
    double sum = 0.0;  
    int x, n;  
    printf("x와 n의 값을 입력하시오: ");  
    scanf("%d %d", &x, &n);  
    for (int i = 0; i <= n; i++)  
        sum += power(x, i) / factorial(i);  
  
    printf("e^%d = %.3lf\n", x, sum);  
    return 0;  
}
```


Mini Project: ATM 만들기

- 은행에 설치되어 있는 ATM을 프로그램으로 구현해보자.

```
D:\hello\Debug\hello.exe
*****Welcome to 콘서트 ATM*****
****하나를 선택하시오****
<1> 잔고 확인
<2> 입금
<3> 인출
<4> 종료
2
****입금 금액을 입력하시오
10000
새로운 잔고는 20000입니다.
```

