

C Programming

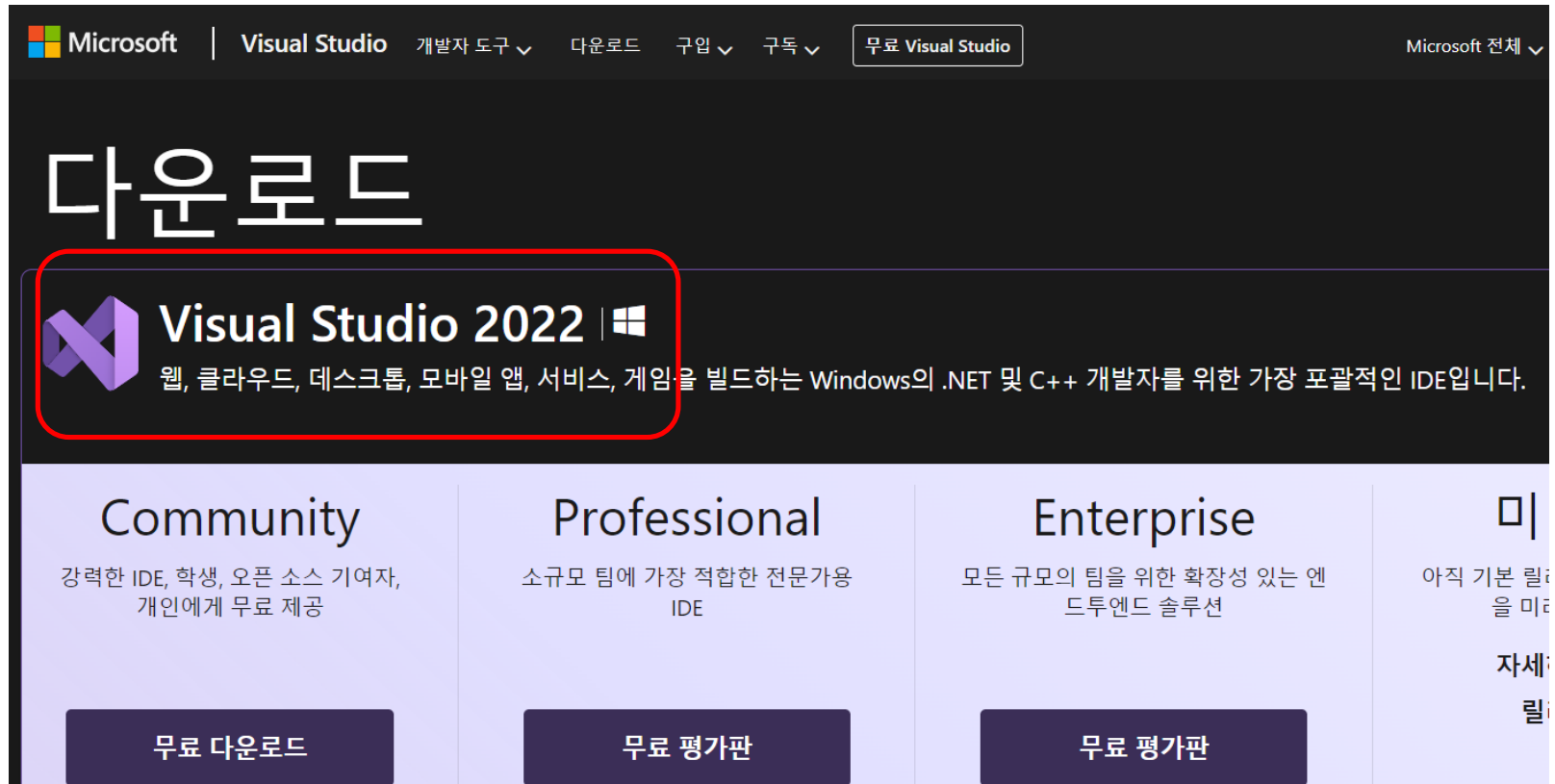
#3/3 ver 0.1

Yongseok Chi

Reference

1. Reference

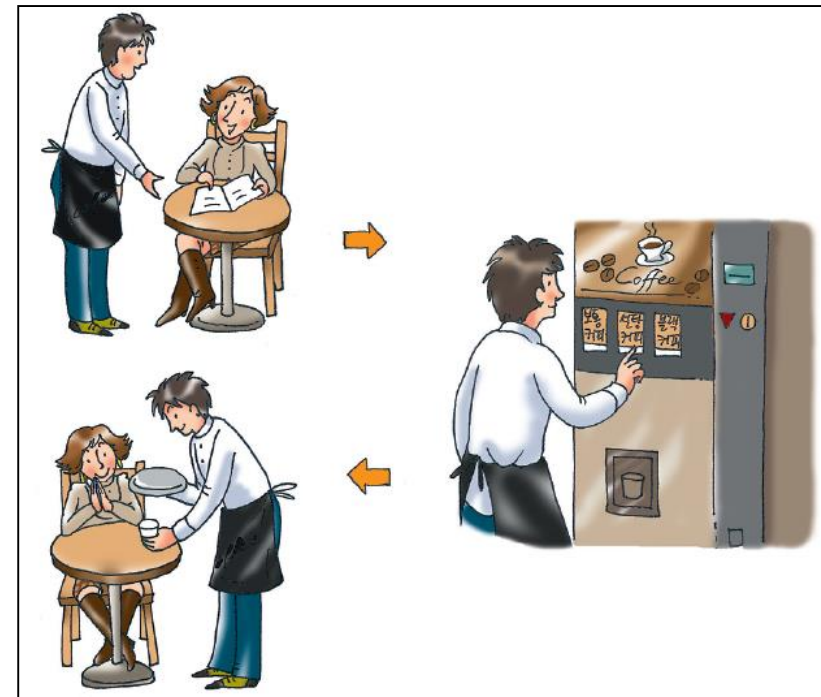
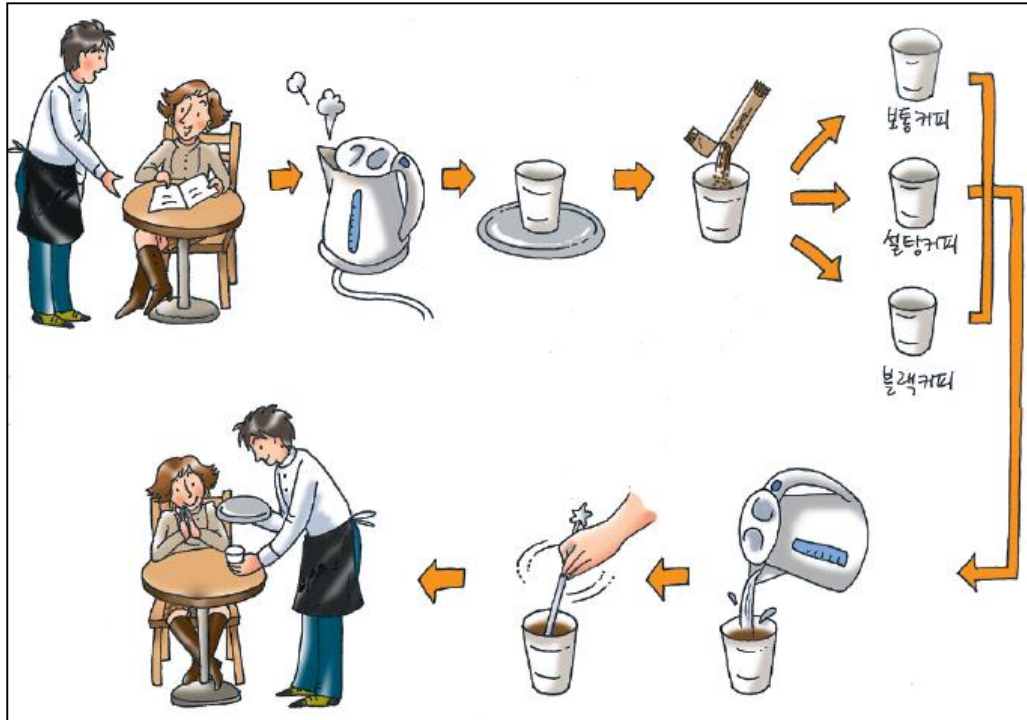
- (1) Preprocessor, msdn.microsoft.com/en-us/library/d9x1s805.aspx
- (2) 비주얼 스튜디오 <https://visualstudio.microsoft.com/ko/downloads/>
- (3) 강의자료 **Visual Studio 2013** version
- (4) Editor : Notepad++ 설치 <https://notepad-plus-plus.org/downloads/>
- (5) code 비교 : beyond compare <https://www.scootersoftware.com/>
- (6) project 분석 : source insight <https://www.sourceinsight.com/>



10. 함수(function)

함수(function)

: 매번 반복되는 부분에 대해 함수화



10. 함수(function)

10.1 라이브러리 함수(library function)

: C 언어에서 미리 정의해서 제공하는 함수

: 함수에 필요한 **전처리기 지시자와 헤더 파일**을 정확하게 명시해야 함

printf와 scanf → **#include <stdio.h>**

• 수학과 관련된 함수 **#include <math.h>**가 필요

함수 호출 예	수학 식
pow(x, n)	x^n
sqrt(3 * x + a)	$\sqrt{3x + a}$
log(x)	$\ln x$
log10(abs(2 * x))	$\log_{10} 2x $
abs(-3)	$ -3 $

함수 호출 예	수학 식
fabs(-7.5)	$ -7.5 $
exp(x)	e^x
ceil(x)	$\lceil x \rceil$
floor(x)	$\lfloor x \rfloor$
sin(2 * 3.141592)	$\sin(360^\circ)$
cos(3.141592)	$\cos(180^\circ)$
tan(3.141592 / 4)	$\tan(45^\circ)$

10. 함수(function)

10.1 라이브러리 함수(library function)

- 문자와 관련된 함수 `#include <ctype.h>`가 필요

함수 호출 예	함수 의미
<code>isdigit(ch)</code>	ch에 저장된 문자가 숫자 문자면 논리값 참을, 그렇지 않으면 거짓을 반환
<code>isalpha(ch)</code>	ch에 저장된 문자가 영문자면 참을, 그렇지 않으면 거짓을 반환
<code>islower(ch)</code>	ch에 저장된 문자가 영문 소문자면 참을, 그렇지 않으면 거짓을 반환
<code>isupper(ch)</code>	ch에 저장된 문자가 영문 대문자면 참을, 그렇지 않으면 거짓을 반환

- 문자열과 관련된 함수 `#include <string.h>`가 필요

함수 호출 예	함수 의미
<code>strlen(str)</code>	str에 저장된 문자열의 길이를 반환
<code>strcmp(str1, str2)</code>	str1과 str2에 저장된 문자열이 같다면 0을, str1이 작으면 -1을, str1이 더 크면 1을 반환
<code>strcpy(str1, str2)</code>	str2의 문자열을 str1의 문자열로 복사

10. 함수(function)

10.1 라이브러리 함수(library function)

- 그 외 범용 함수 `#include <stdlib.h>`가 필요

함수 호출 예	함수 의미
<code>rand()</code>	정수 0~32767 중의 한 개의 난수를 반환
<code>srand(time(NULL))</code>	현재 시간(<code>time(NULL)</code>)을 난수 발생기의 씨드로 설정
<code>exit(0)</code>	프로그램을 종료(인수는 0과 1을 사용, <code>exit(0)</code> 는 프로그램의 정상적인 종료를 <code>exit(1)</code> 은 프로그램의 비정상적인 종료를 의미)
<code>system("cls")</code>	문자열 인수에 해당하는 시스템 명령을 실행, <code>system("cls")</code> 는 화면을 지우는 시스템 명령을 실행

10. 함수(function)

10.1 라이브러리 함수(library function)

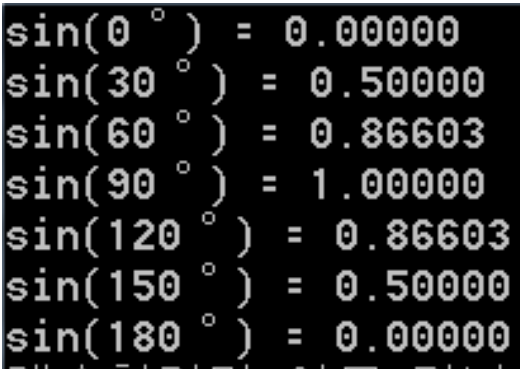
: 예제 0° ~ 180°까지 30° 단위마다 사인 함수의 값 출력하기

: 분석

사인 삼각함수의 값은 라이브러리 함수 **sin**을 이용, sin 함수는 **#include <math.h>** 필요
인수는 **radian 값**, **180°: π = degree°: radian**

```
#include <stdio.h>
#include <math.h>
#define PI 3.141592
```

```
int main() {
    int degree;
    double radian;
    for (degree=0; degree<=180; degree+=30) {
        radian = (PI * degree) / 180;
        printf("sin(%d°) = %.5lf\n", degree, sin(radian));
    }
    return 0;
}
```



Angle (°)	Sine Value
0	0.00000
30	0.50000
60	0.86603
90	1.00000
120	0.86603
150	0.50000
180	0.00000

// 각도 저장

// degree의 라디안 값 저장 변수

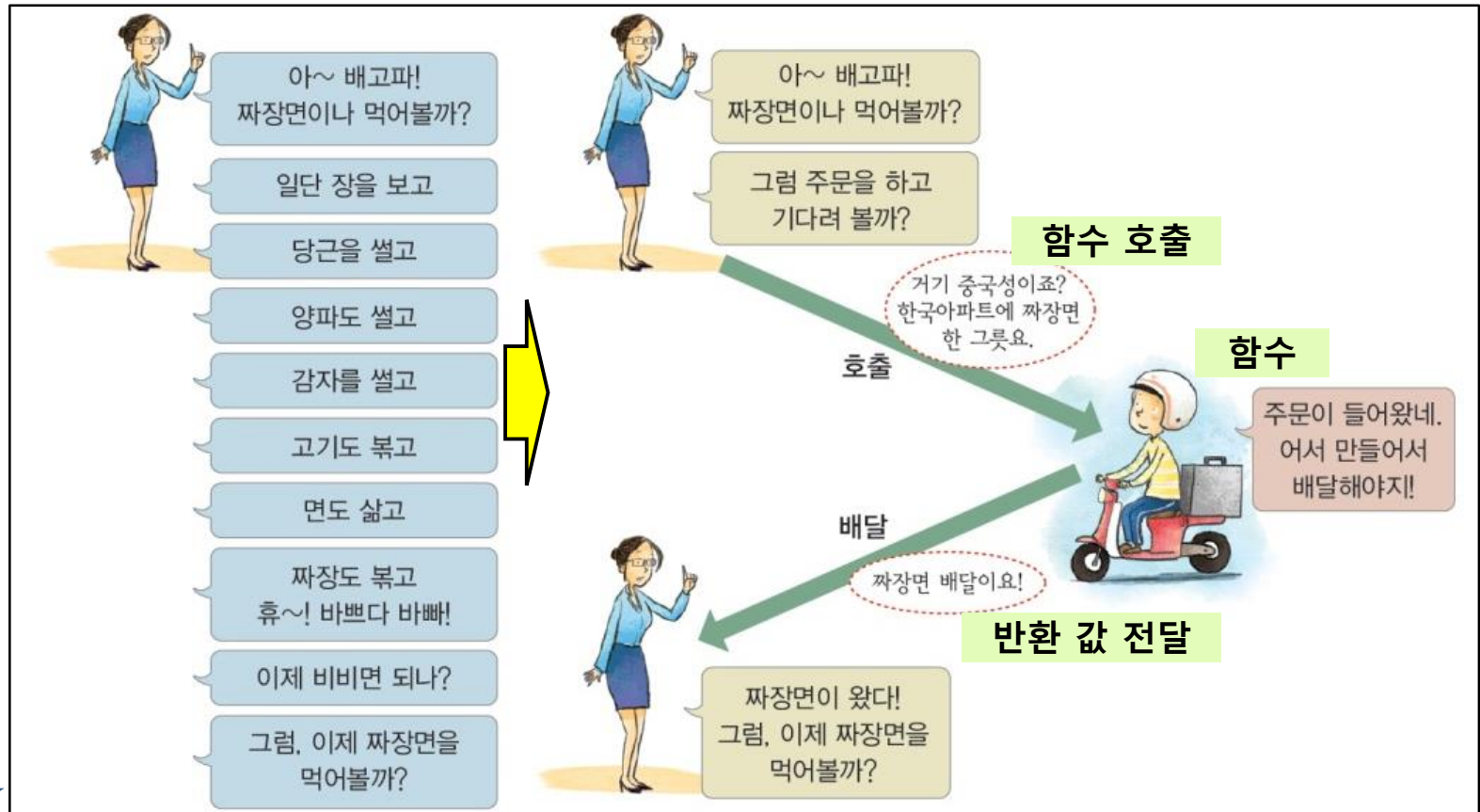
// 각도 → 라디안

디버그 – 디버깅하지 않고 시작

10. 함수(function)

10.2 하향식 프로그래밍(top-down programming) 방법

- 크고 복잡한 문제를 해결하기 쉬운 여러 개의 작은 문제로 나누어 문제를 단순화시킨 후, 각 문제를 함수로 작성하여 해결



- main 프로그램의 길이 단축, 프로그램의 가독성(readability) 향상,
- 프로그램의 수정 및 확장의 용이, 코드의 재활용

10. 함수(function)

10.2 인수전달

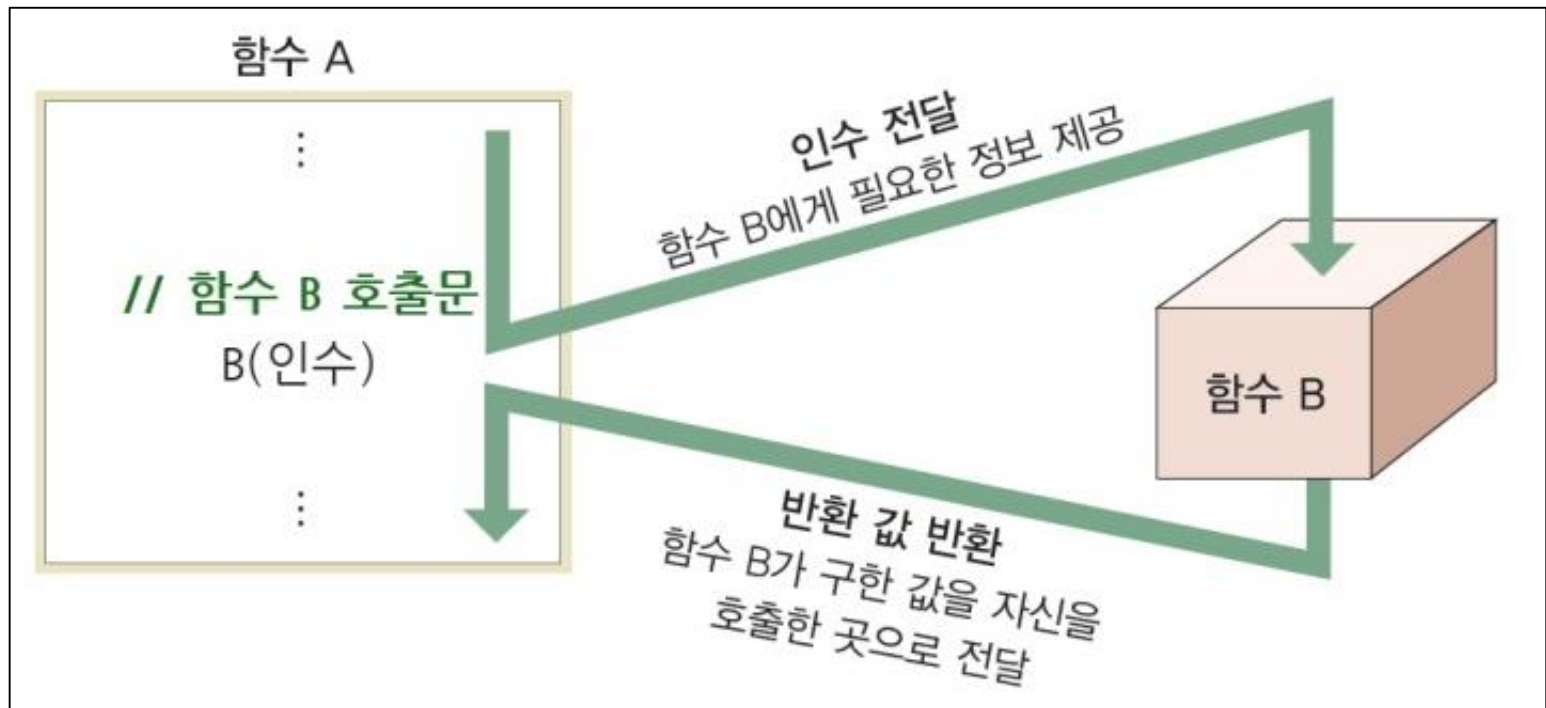
: 함수를 호출할 때는

함수가 일을 하는 데 필요한 최소의 정보(인수 argument, 매개변수)를 전달해야 함

➔ 인수 전달

: 호출된 함수는 자신을 호출한 함수에 결과(반환 값)를 제공해야 함

➔ 반환 값을 전달



10. 함수(function)

10.3 함수 정의와 호출

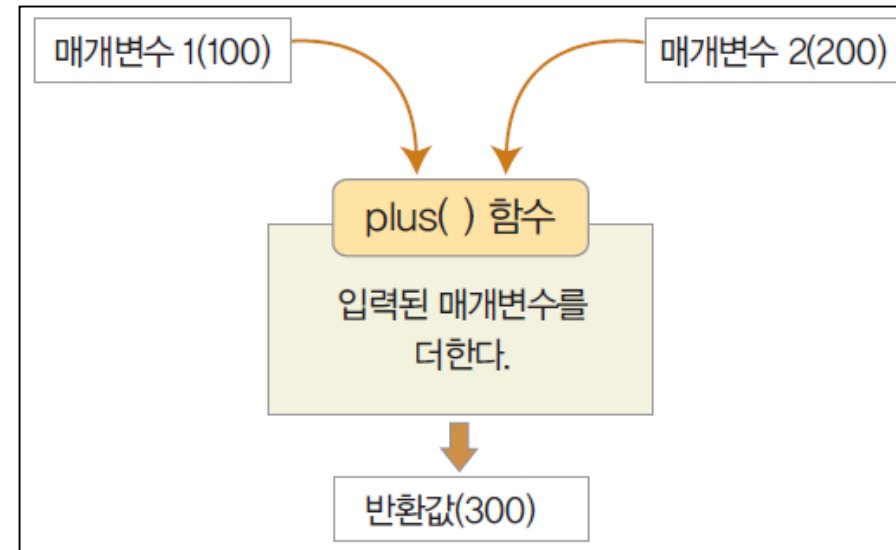
```
#include <stdio.h>
```

```
int plus(int v1, int v2) {  
    int result;  
  
    result = v1 + v2;  
    return result;  
}
```

매개변수를 받고

결과를 반환

함수



```
int main() {  
    int hap;  
  
    hap = plus(100, 200);  
    printf("100과 200의 plus( ) 함수 결과는 : %d\n", hap);  
}
```

매개변수

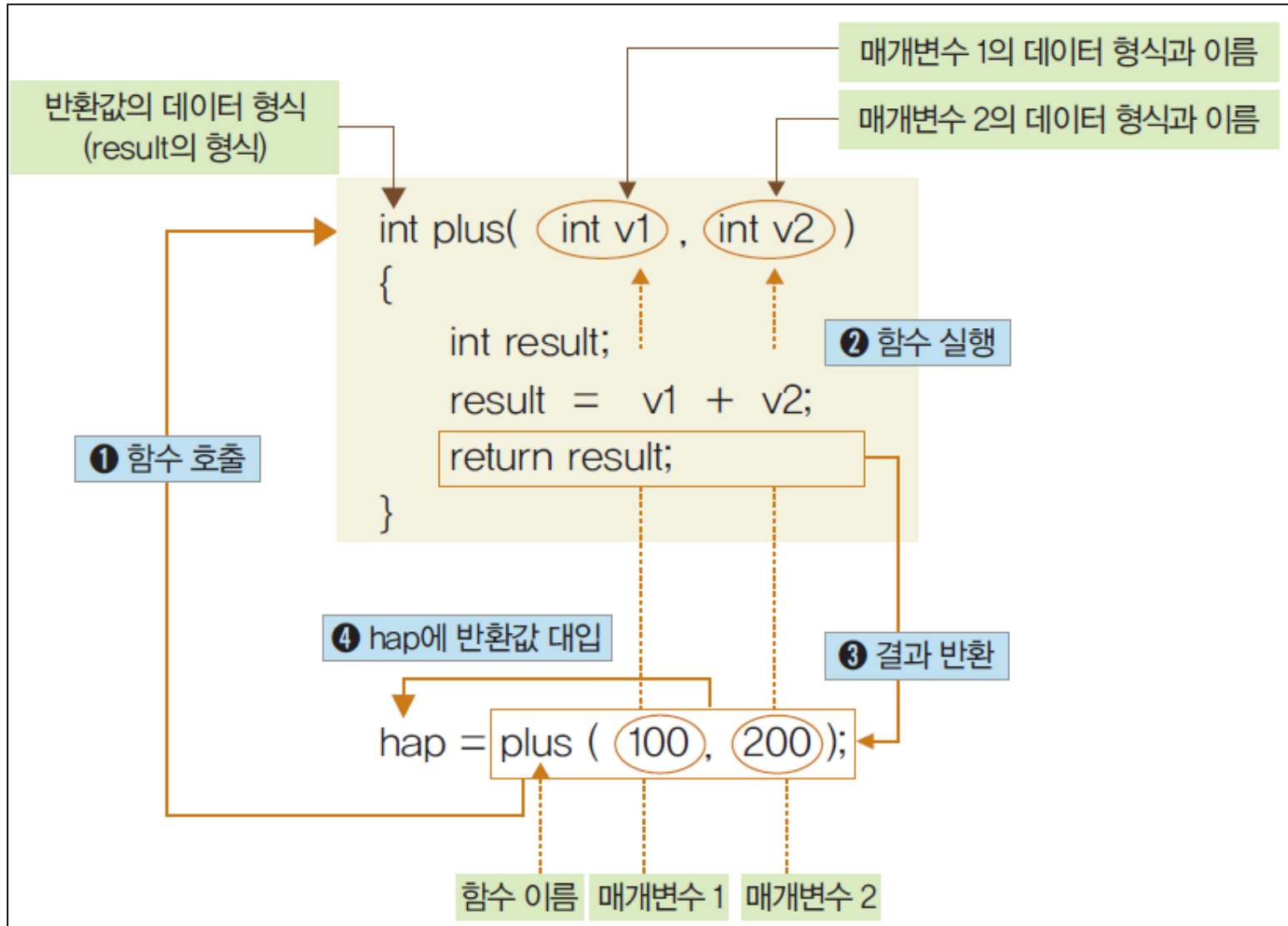
함수 호출

100과 200의 plus() 함수 결과는 : 300
계속하려면 아무 키나 누르십시오 . . .

→ 디버그 - 디버깅하지 않고 시작

10. 함수(function)

10.3 함수 정의와 호출 과정



10. 함수(function)

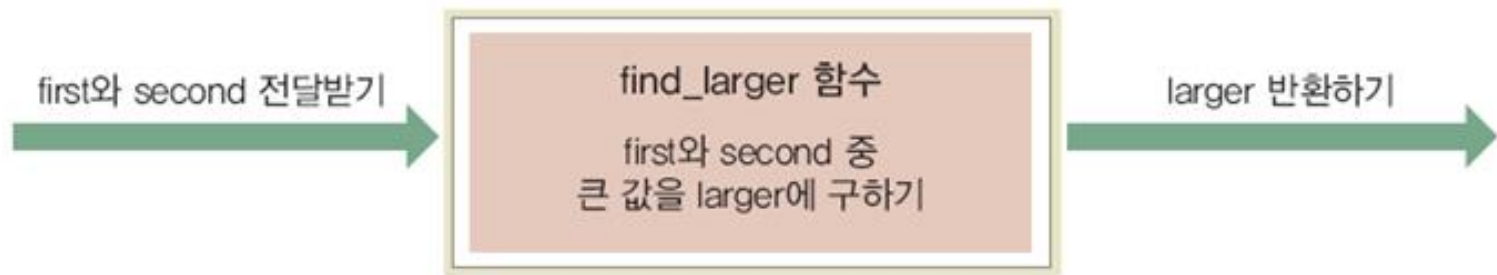
10.3 함수 정의와 호출 예제

문제

입력: 임의의 두 정수 $n1$ 과 $n2$

출력: 두 정수 중 큰 값 \max

함수 만들기: `find_larger`



해결 과정

1. main 함수에서 $n1$ 과 $n2$ 의 값을 **입력받기**
2. main 함수에서 **`find_larger` 함수를 호출하여** $n1$ 과 $n2$ 중 큰 값을 **반환 받아**
 \max 에 저장하기
 $\max = \text{find_larger}(n1, n2);$
3. main 함수에서 \max 출력하기

10. 함수(function)

10.3 함수 정의와 호출 예제

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

```
int find_larger(int first, int second) { // 정수 두 개를 전달받아 큰 값을 반환
    int larger;
    if (first > second) larger = first;
    else larger = second;
    return larger;
}
```

```
int main() {
    int n1, n2, max;
    printf("첫째 정수? "); scanf_s("%d", &n1);
    printf("둘째 정수? "); scanf_s("%d", &n2);
    max = find_larger(n1, n2);
    printf("%d, %d 중 큰 값은 %d Wn", n1, n2, max);
    return 0;
}
```

```
첫째 정수? 5
둘째 정수? 9
5, 9 중 큰 값은 9
계속하려면 아무 키나 누르십시오
```

```
첫째 정수? -1
둘째 정수? -10
-1, -10 중 큰 값은 -1
계속하려면 아무 키나 누르십시오
```

10. 함수(function)

10.3 함수 정의와 호출

주의 (순서)

: 함수 안에 다른 함수를 정의할 수 없다.

→ 한 함수의 정의가 끝난 후

다른 함수를 정의해야 함

: 함수 정의는 함수 호출 전에 위치

→ 함수 정의보다

함수 호출이 앞에 있으면 잘못

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

```
// main 함수의 정의
```

```
int main()
```

```
{
```

```
    int n1, n2, max;
```

```
    printf("첫째 정수? "); scanf("%d", &n1);
```

```
    printf("둘째 정수? "); scanf("%d", &n2);
```

```
// find_larger 함수를 호출 후 반환된 값을 max에 저장
```

```
max = find_larger(n1, n2);
```

```
printf("%d, %d 중 큰 값은 %d \n", n1, n2, max);
```

```
return 0;
```

```
}
```

```
// find_larger 함수의 정의
```

```
int find_larger(int first, int second)
```

```
{
```

```
    : 함수의 본체
```

```
}
```

→ 함수 호출이 앞에 있게 하려면

함수 원형 선언이 필요



10. 함수(function)

10.4 함수 원형 선언

전처리기 지시자

함수1의 원형 선언
함수2의 원형 선언

main 함수 정의
:
함수1 호출문
:

사용자 정의 함수1
:
함수2 호출문
:

사용자 정의 함수2
:

주의 (순서)

함수의 원형(prototype)

: 함수 정의의 헤더 부분에 해당

: 앞 예제의, int **find_larger**(int first, int second);

; 반드시 필요

함수의 원형 선언

: 최소한 맨 처음 나타나는 함수 호출문보다 앞에 있어야 함

: 대부분 프로그램 상단 **main 함수 정의 전에 위치**

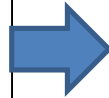
: 사용자 정의 함수는 main 함수 뒤에 위치

10. 함수(function)

```
#include <stdio.h>

int find_larger(int first, int second) {
    int larger;
    if (first > second) larger = first;
    else larger = second;
    return larger;
}

int main() {
    int n1, n2, max;
    printf("첫째 정수? "); scanf_s("%d", &n1);
    printf("둘째 정수? "); scanf_s("%d", &n2);
    max = find_larger(n1, n2);
    printf("%d, %d 중 큰 값은 %d \n", n1, n2, max);
    return 0;
}
```



```
#include <stdio.h>

int find_larger(int first, int second);

int main() {
    int n1, n2, max;
    printf("첫째 정수? "); scanf_s("%d", &n1);
    printf("둘째 정수? "); scanf_s("%d", &n2);
    max = find_larger(n1, n2);
    printf("%d, %d 중 큰 값은 %d \n", n1, n2, max);
    return 0;
}

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


10. 함수(function)

10.5 함수호출, 함수 정의

값에 의한 호출(call-by-value)

- : 함수를 호출하면 인수의 값이 전달됨
- : 인수와 매개변수는 서로 다른 기억장소를 사용
- : 함수 간 독립성 보장(인수, 매개변수)

함수 호출

함수명(인수1, 인수2, ..., 인수n)

함수 정의

반환값형 함수명(매개변수1, 매개변수2, ...)

{
: 함수 본체
}

주의

인수 : 상수, 변수, 식
매개변수 : 변수만 가능

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

```
int find_larger(int first, int second);
```

```
int main() {
```

```
    int n1, n2, max;
```

```
    printf("첫째 정수? "); scanf("%d", &n1);
```

```
    printf("둘째 정수? "); scanf("%d", &n2);
```

```
    max = find_larger(n1, n2);
```

```
    printf("%d, %d 중 큰 값은 %d Wn", n1, n2, max);
```

```
    return 0;
```

```
}
```

```
int find_larger(int first, int second)
```

```
{
```

```
    int larger;
```

```
    if (first > second) larger = first;
```

```
    else larger = second;
```

```
    return larger;
```

```
}
```

반환값

10. 함수(function)

10.5 함수호출, 함수 정의

문제

임금: 시급제와 일급제를 혼용

시급: 10,000원

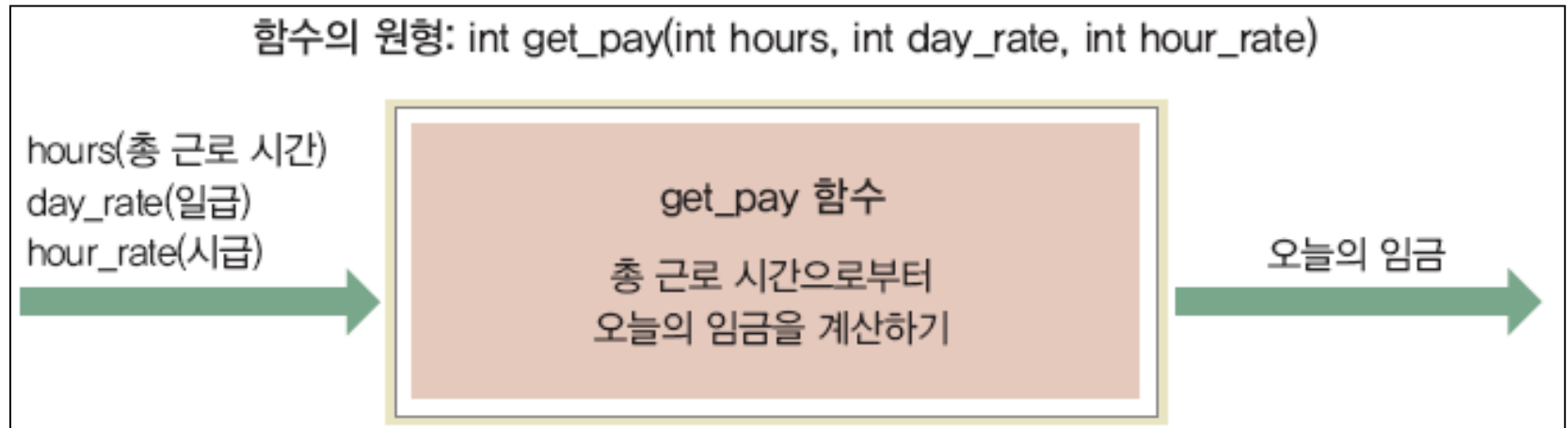
8시간 이상 근로 → 일급으로 100,000원을 적용

일급을 적용할 근로 일 : (총 근로 시간 / 8)

시급을 적용할 나머지 근로 시간 : (총 근로 시간 % 8)

오늘의 임금 = 근로 일*일급 + 나머지 근로 시간*시급

근로 시간은? 39
오늘의 임금은 470000원
계속하려면 아무 키나 누르십시오



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

```
int get_pay(int hours, int day_rate, int hour_rate);           // 함수의 원형 선언
```

```
int main() {
```

```
    int total_hours;                                           // 근로 시간
```

```
    int daily_rate = 100000, hourly_rate = 10000;           // 일급, 시급
```

```
    int pay;                                                  // 임금
```

```
    printf("근로 시간은? ");
```

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

```
    pay = get_pay(total_hours, daily_rate, hourly_rate);      // 함수 호출
```

```
    printf("오늘의 임금은 %d원 \n", pay);
```

```
    return 0;
```

```
}
```

```
int get_pay(int hours, int day_rate, int hour_rate) {         // get_pay 함수의 정의
```

```
    int day = hours / 8;                                       // 근로 일을 계산
```

```
    hours = hours % 8;                                         // 나머지 시간을 계산
```

```
    return (day * day_rate + hours * hour_rate);
```

```
}
```

→ 디버그 - 디버깅하지 않고 시작

```
int get_pay(int hours, int day_rate, int hour_rate);
```

// 함수의 원형 선언

```
int main() {
```

```
    int total_hours;
```

// 근로 시간

```
    int daily_rate = 100000, hourly_rate = 10000;
```

// 일급, 시급

```
    int pay;
```

// 임금

```
    pay = get_pay(total_hours, daily_rate, hourly_rate);
```

// 함수 호출

```
}
```

```
int get_pay(int hours, int day_rate, int hour_rate) {
```

// get_pay 함수의 정의

```
    int day = hours / 8;
```

// 근로 일을 계산

```
    hours = hours % 8;
```

// 나머지 시간을 계산

```
    return (day * day_rate + hours * hour_rate);
```

인수 전달 결과

```
}
```

main 함수의 변수

total_hours	11
daily_rate	100000
hourly_rate	10000
pay	

get_pay 함수의 변수

hours	11
day_rate	100000
hour_rate	10000
day	

10. 함수(function)

10.6 main 함수

: main 함수는 반환 값이 있게도 없게도 정의 가능

: 반환 값이 있는 main 함수

➔ 자신을 호출한 OS에 반환 값을 전달

➔ return 0; 정상 종료를 의미하는 반환 값은 0

➔ return 1; 비정상적인 종료를 의미하는 반환 값은 0 외의 값(일반적으로 1)

: 반환 값이 없는 main 함수

➔ 반환 형 자리에 void를 사용

```
int main()
{
    int i, sum = 0;
    for (i=1; i<=10; i+=2)
        sum += i;
    printf("1~10까지의 홀수의 합:%d", sum);
    return 0;
}
```

```
void main()
{
    int i, sum=0;
    for (i=1; i<=10; i++)
        sum += i;
    printf("1~10까지의 합:%d", sum);
    return;
}
```

함수 실행 끝내고 호출한 곳으로 돌아감
맨 마지막 문장이라면 생략 가능

10. 함수(function)

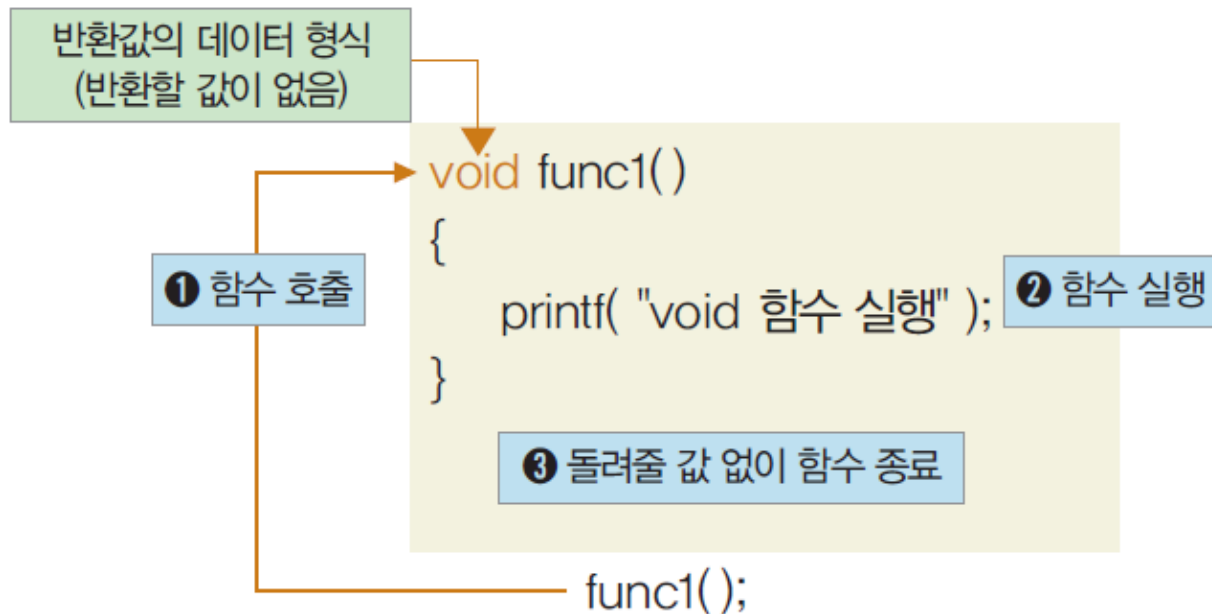
10.7 void (뜻: 빈, 비어있는)

함수 정의時, 특정 자리에 명시할 내용이 없음을 나타내는 데 사용

: 함수가 반환할 값이 없다면 → 반환 형 자리에 void

: 함수가 매개변수가 없다면 → 매개변수 목록에 void 또는 비워두기

➔ 특정 일만 수행하고 반환 값을 전달하지 않는 함수, void 함수라고 함



10. 함수(function)

10.7 void (뜻: 빈, 비어있는)

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

void 생략 가능

```
void func1(void) ← void형 함수므로 반환 값이 없다.
```

```
{  
    printf("void 형 함수는 돌려줄게 없음.\n");  
}
```

```
int func2() ← int형 함수므로 반환 값이 있다.
```

```
{  
    return 100; ← 반환 값  
}
```

```
int main()
```

```
{  
    int a;  
    func1(); ← void형 함수를 호출한다.  
    a = func2(); ← int형 함수를 호출한다.  
    printf("int 형 함수에서 돌려준 값 = = > %d\n", a);  
}
```

10. 함수(function)

10.7 예제(void 함수 호출)

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

```
void print_title();
```

```
void print_information(void);
```

```
int main() {
```

```
    print_title();  
    print_information();
```

```
    return 0;
```

```
}
```

```
void print_title() {
```

```
    printf("=====Wn");  
    printf("==  C 프로그래밍 과제  ==Wn");  
    printf("== 사인 함수 그래프 그리기 ==Wn");  
    printf("=====Wn");
```

```
}
```

← 함수 원형 선언

```
void print_information(void)
```

```
{
```

```
    printf("WnWn");  
    printf("%30s Wn", " 동서대학교");  
    printf("%30s Wn", " 전자공학 ");  
    printf(" %30s Wn ", " 지용석");
```

```
}
```

→ 디버그 - 디버깅하지 않고 시작

10. 함수(function)

10.8 예제(달력 출력하기- 프로그래밍과 이해하기)

2018년 5월 달력						
화	수	목	금	토	일	월
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

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

```
void print_header();
```

```
void print_numbers(int end);
```

```
int main() {
```

```
    int year = 2018, month = 5, days = 31;
```

```
    printf("WnWtWt%d년 %d월 달력 Wn", year, month);
```

```
    print_header();
```

```
    print_numbers(days);
```

```
    return 0;
```

```
}
```

```

void print_header() {
    printf("WnWt=====Wn");
    printf("Wt화Wt수Wt목Wt금Wt토Wt일Wt월");
    printf("WnWt=====Wn");
}

```

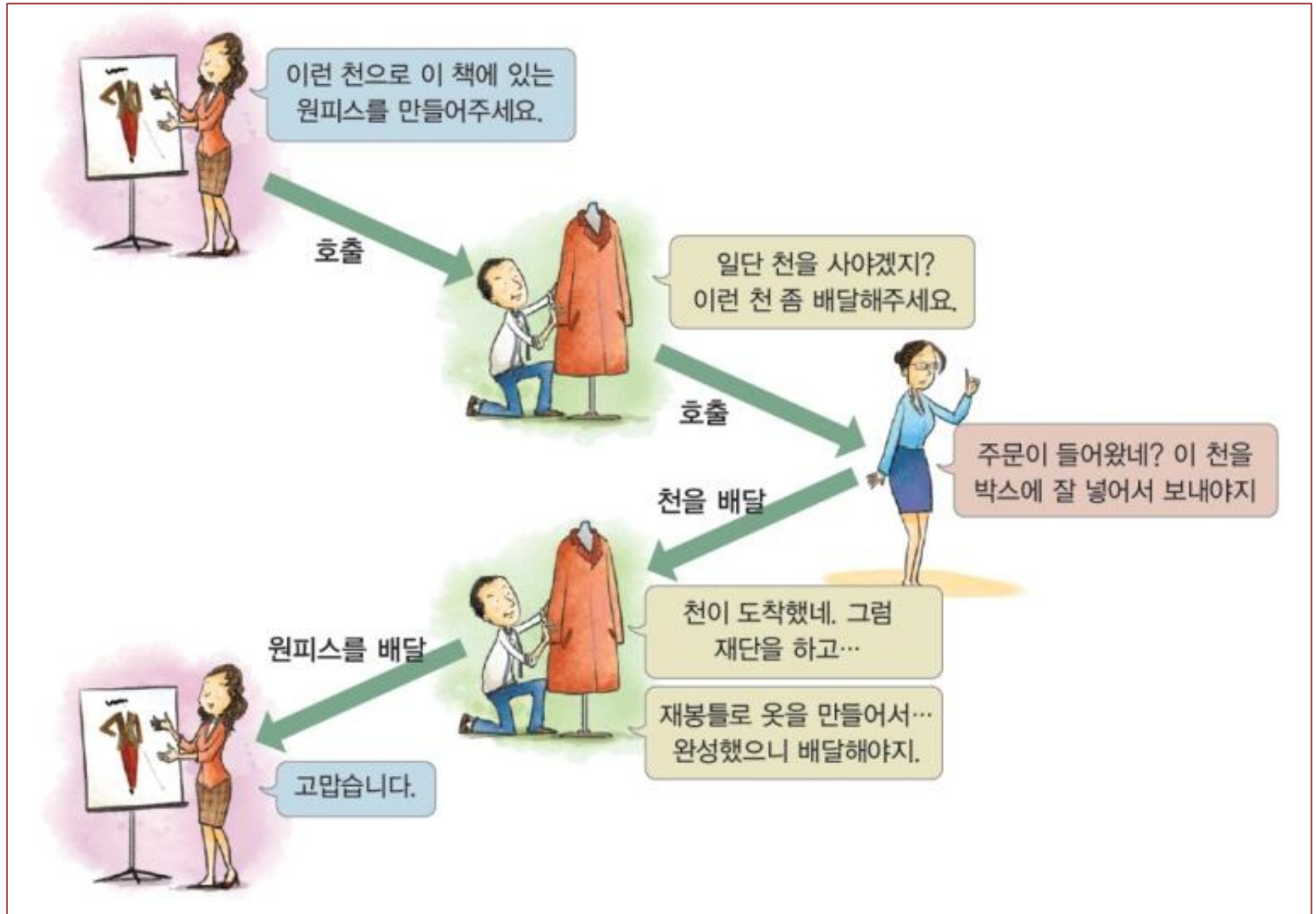
```

void print_numbers(int end) {
    int day_num;
    for (day_num=1; day_num<=end; day_num++)
    {
        printf("Wt%2d", day_num);
        if (day_num % 7 == 0)        // 7일이 지날 때마다 줄을 바꾸기
            printf("Wn");
    }
    printf("Wn");
}

```

10. 함수(function)

10.9 호출된 함수가 또 다른 함수 호출하기



10. 함수(function)

10.9 호출된 함수가 또 다른 함수 호출하기

문제

입력: 연도(year)

출력: **입력 연도의 총일** 즉 365 또는 366

분석

함수 `int leap_year(int y)` : y년도의 윤년 여부 반환

→ y가 윤년이면 1(TRUE), 평년이면 0(FALSE)을 반환

윤년의 조건

→ 연도가 400의 배수면 무조건 윤년

또는 연도가 4의 배수지만 100의 배수가 아니면 윤년

함수 `int days(int yy)` : yy년의 총일 구하기

→ yy가 윤년이면 366, 평년이면 365 반환

→ yy의 윤년 여부를 `leap_year` 함수를 이용해 구하기

```
총일을 구하고 싶은 연도는? 2016
2016년은 366일까지 있습니다.
계속하려면 아무 키나 누르십시오 . . .
```

```
총일을 구하고 싶은 연도는? 2018
2018년은 365일까지 있습니다.
계속하려면 아무 키나 누르십시오 . . .
```

10. 함수(function)

10.9 호출된 함수가 또 다른 함수 호출하기 : 사용자가 입력한 해의 총 일 출력하기

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

```
#define TRUE 1
```

```
#define FALSE 0
```

```
int days(int yy);
```

→ 함수 원형 선언

```
int leap_year(int y);
```

```
int main() {
```

```
    int year;
```

```
    printf("총 일을 구하고 싶은 연도는? ");
```

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

```
    /* days 함수를 호출해 year년의 총 일을 구하기 */
```

```
    printf("%d년은 %d일까지 있습니다. \n", year, days(year));
```

```
    return 0;
```

```
}
```

→ days 함수 호출

int days(int yy)

// yy년의 총일(365 또는 366)을 반환하는 함수

{

if (**leap_year(yy)** == TRUE) // **leap_year** 함수 호출

return 366;

else

return 365;

}

윤년의 조건

→ 연도가 400의 배수면 무조건 윤년

또는 연도가 4의 배수지만 100의 배수가 아니면 윤년

int leap_year(int y)

// y년도의 윤년 여부를 반환하는 함수

{

if (y%400 == 0 || ((y%4==0) && (y%100 != 0)))

return TRUE;

// 윤년이므로 return 1;

else

return FALSE;

// 윤년이 아니므로 return 0;

}

→ 디버그 - 디버깅하지 않고 시작

10. 함수(function)

10.10 배열을 함수로 전달하기

: 배열 원소를 함수로 전달하기

: 배열을 함수로 전달하기

10. 함수(function)

10.10 배열을 함수로 전달하기(1) 배열 원소 두 개 중 큰 값을 함수를 이용해 구하기

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

```
#define N 10
```

```
int find_larger(int first, int second);           // 함수의 원형 선언
```

```
int main() {
```

```
    int max, score[5] = {10, 8, 9, 7, 8};
```

```
    max = find_larger(score[3], score[4]); // 4째, 5째 배열 원소를 함수로 전달
```

```
    printf(" score[3]=%d과 score[4]=%d 중 큰 값은 %d \n", score[3], score[4], max);
```

```
    return 0;
```

```
}
```

```
int find_larger(int first, int second) {
```

```
    if (first > second) return first;
```

```
    else return second;
```

```
}
```

→ 디버그 - 디버깅하지 않고 시작

score[3]=7과 score[4]=8 중 큰 값은 8
계속하려면 아무 키나 누르십시오 . . .

10.10 배열을 함수로 전달하기(2) 최대값 구하기

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

```
#define N 10
```

```
int find_larger(int first, int second);
```

// 함수의 원형 선언

```
int main() {
```

```
    int freeze[N] = {15, 0, -20, -30, 50, -5, -120, -5, 10, -12};
```

```
    int i, max;
```

/* max의 초기값을 첫째 원소로 지정한 후, 현재의 max와 나머지 배열 원소 중 큰 값을
find_larger 함수를 이용하여 구한 후 다시 max에 저장한다. */

```
    max = freeze[0];
```

```
    for (i=1; i< N; i++) {
```

```
        max = find_larger(max, freeze[i]);
```

```
    }
```

```
    printf("어는 점 목록 : ");
```

```
    for (i=0; i< N; i++) {
```

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

```
    }
```

```
    printf("\n 가장 높은 어는 점 : %d \n\n", max);
```

```
    return 0;
```

```
}
```

배열을 함수로 전달

```
int find_larger(int first, int second) {  
    if (first > second) return first;  
    else return second;  
}
```

→ 빌드 - 솔루션빌드

→ 디버그 - 디버깅하지 않고 시작

```
어는 점 목록 : 15 0 -20 -30 50 -5 -120 -5 10 -12  
가장 높은 어는 점 : 50
```

10. 함수(function)

10.11 함수 만들기 실습

: 분기별 판매 수로부터
연평균 판매 수 구하기

배열명만 다를 뿐

평균을 구하는 동일한 코드가

두 번 반복해서 나타난다

→ 함수로 작성하기

```
int main()
{
    int notebook[N] = {2507, 2232, 2009, 2890};    // 노트 판매수
    int i, sum, pen[N] = {4527, 5370, 4923, 6097};  // 펜 판매수
    double average;

    // 노트의 평균 판매수 구하기
    sum = 0;
    for (i=0; i<N; i++)
        sum = sum + notebook[i];
    average = (double)sum / N;

    printf("노트 평균 판매수: %.1lf \n", average);

    // 펜의 평균 판매수 구하기
    sum = 0;
    for (i=0; i<N; i++)
        sum = sum + pen[i];
    average = (double)sum / N;

    printf("펜 평균 판매수: %.1lf \n", average);

    return 0;
}
```

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

```
#define N 4
```

```
double compute_avg(int arr[]);
```

// 함수의 원형 선언

```
int main()
```

```
{
```

```
    int notebook[N] = {2507, 2232, 2009, 2890};
```

```
    int pen[N] = {4527, 5370, 4923, 6097};
```

```
    double average;
```

```
    average = compute_avg(notebook);
```

// 노트의 평균 판매수 구하기

```
    printf("노트 평균 판매수: %.1lf Wn", average);
```

```
    average = compute_avg(pen);
```

// 펜의 평균 판매수 구하기

```
    printf("펜 평균 판매수: %.1lf Wn", average);
```

```
    return 0;
```

```
}
```

```
double compute_avg(int arr[])
```

// 전달된 배열의 평균을 구하는 함수

```
{
```

```
    int i, sum = 0;
```

```
    for (i=0; i<N; i++) sum = sum + arr[i];
```

```
    return (double)sum / N;
```

```
}
```

10. 함수(function)

10.12 함수 만들기 실습

: 앞의 예제는 배열의 크기가 동일하였으나, 배열의 크기가 다를 경우의 함수는 어떻게 만들까

```
int main()
{
    int notebook[N] = {2507, 2232, 2009, 2890};    // 노트 판매수
    int i, sum, pen[N] = {4527, 5370, 4923, 6097};  // 펜 판매수
    double average;
```

// 노트의 평균 판매수 구하기

```
sum = 0;
for (i=0; i<N; i++)
    sum = sum + notebook[i];
average = (double)sum / N;
```

```
printf("노트 평균 판매수: %.1lf \n", average);
```

// 펜의 평균 판매수 구하기

```
sum = 0;
for (i=0; i<N; i++)
    sum = sum + pen[i];
average = (double)sum / N;
```

```
int pen[4] = {4500, 5370, 4920, 6090};
int monthly_stock[12] =
    {505, 409, 389, 257, 450, 501, 500, 621,
     480, 350, 389, 250};
```

: 배열의 크기와 상관없이

평균을 구하는 함수 만들기

→ 배열의 원소수 n을 매개변수로 전달받기

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

```
double array_avg(int arr[], int n);
```

// 함수의 원형 선언

```
int main()
```

```
{
```

```
int pen[4] = {4500, 5370, 4920, 6090};
```

```
int monthly_stock[12] = {505, 409, 389, 257, 450, 501, 500, 621, 480, 350, 389, 250};
```

```
double average;
```

```
average = array_avg(pen, 4);
```

// 연 평균 판매수 구하기

```
printf("펜 평균 판매수: %.1lf \n", average);
```

```
average = array_avg(monthly_stock, 12);
```

// 연 평균 재고량 구하기

```
printf("평균 재고량: %.1lf \n", average);
```

```
return 0;
```

```
}
```

→ 배열의 원소수 n을 매개변수로 전달받기

```
double array_avg(int arr[], int n)
```

```
{
```

```
int i, sum=0;
```

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

```
    sum = sum + arr[i];
```

```
return (double)sum / n;
```

```
}
```

→ 디버그 - 디버깅하지 않고 시작

```
펜 평균 판매수: 5220.0
평균 재고량: 425.1
```

10. 함수(function)

10.13 난수(random number) 생성 함수 : rand

: 컴퓨터와 사람이 가위바위보 게임을 하는 프로그램

➔ 컴퓨터가 무엇을 낼지?

: 주사위를 두 개 던져 나오는 값의 합을 맞추는 게임 프로그램

➔ 게임 때마다 두 주사위의 값을 어떻게 다르게 지정할까?

: 초등학생의 덧셈과 뺄셈 연습 프로그램

➔ 초등학생에게 문제를 낼 때마다 덧셈이나 뺄셈하는 두 수가 달라져야 한다.

이 두 값을 어떻게 지정할까?

: 난수 생성기(random number generator) : 무작위로 선택된 임의의 값을 한 개 반환하는 함수

➔ C 언어의 난수 생성기는 rand 함수, #include <stdlib.h> 필요

: rand 함수의 사용

➔ 함수 원형 : int rand(void)

➔ rand 함수의 호출 : rand()

- 호출 결과 값은 0~32767 범위 안의 임의의 정수

10. 함수(function)

10.13 난수(random number) 생성 함수 : rand

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

```
#include <stdlib.h>           // rand 함수를 위한 헤더 파일
```

```
int main() {
```

```
    int i, random;
```

```
    for (i=1; i<=5; i++) {
```

```
        random = rand(); // 난수를 얻어 random 변수에 저장
```

```
        printf("%d번째 난수: %5d \n", i, random);
```

```
    }
```

```
    return 0;
```

```
}
```

```
1번째 난수:    41
2번째 난수:  18467
3번째 난수:   6334
4번째 난수:  26500
5번째 난수:  19169
계속하려면 아무 키나 누르십시오
```

→ 임의의 값 추출이지만,
세 번 반복하여 실행한다 하여도
동일한 값이 추출됨

10. 함수(function)

10.14 씨드(seed) 설정 함수 : **srand**

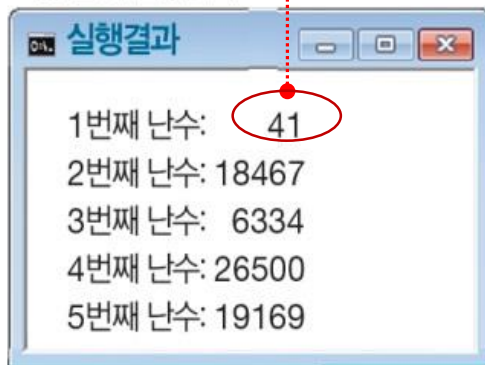
씨드(seed)

: 난수 생성기가 생성하는 **난수 값과 순서를 결정**

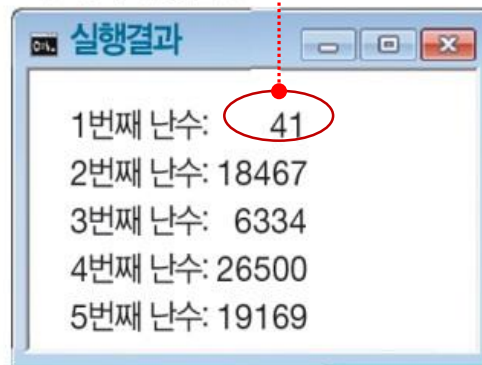
➔ 앞 예제는 **세 번 실행 결과 모두 동일한 순서의 난수가 발생**

그림 8-18 프로그램을 실행할 때마다 동일한 순서의 난수가 발생하는 예

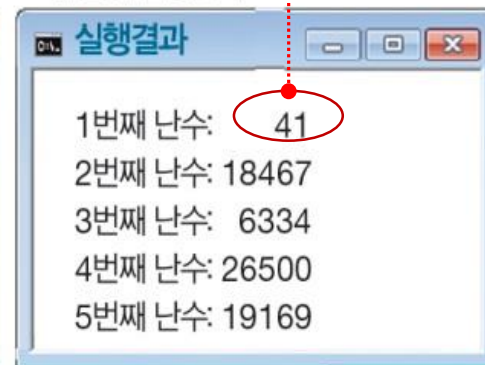
• 첫 번째 실행결과



• 두 번째 실행결과



• 세 번째 실행결과



기본 씨드가 설정된 상태에서 발생한 난수이므로 언제나 동일 순서

씨드 설정 (프로그래머가 직접 씨드를 지정 가능)

➔ **지정한 씨드에 따라 rand의 난수가 순서대로 발생**

➔ **srand(씨드값);**

- **인수에 해당하는 값이 rand 함수의 씨드로 설정됨**
- 이후로 이 씨드에 따른 난수가 발생함, **#include <stdlib.h>**가 필요

10. 함수(function)

10.14 씨드(seed) 설정 함수 : **srand**

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

```
#include <stdlib.h>           // rand, srand 함수를 위한 헤더 파일
```

```
int main() {
```

```
    int i, random;
```

```
    srand(1); 또는 srand(2); 또는 srand(3);을 넣게 되면 아래결과와 같다
```

```
    for (i=1; i<=5; i++) {
```

```
        random = rand();
```

```
        printf("%d번째 난수: %5d \n", i, random);
```

```
    }
```

```
    return 0;
```

```
}
```

씨드가 달라짐에 따라 발생하는
난수 순서도 달라진다.

srand(1);을 추가한 결과	srand(2);을 추가한 결과	srand(3);을 추가한 결과
실행결과	실행결과	실행결과
1번째 난수: 41	1번째 난수: 45	1번째 난수: 48
2번째 난수: 18467	2번째 난수: 29216	2번째 난수: 7196
3번째 난수: 6334	3번째 난수: 24198	3번째 난수: 9294
4번째 난수: 26500	4번째 난수: 17795	4번째 난수: 9091
5번째 난수: 19169	5번째 난수: 29484	5번째 난수: 7031

10. 함수(function)

10.15 씨드(seed) 설정 함수 : **srand(time(NULL))**

➔ 그러나 매번 임의값을 얻기 위하여 **srand**의 값을 변경해야 함

```
int main() {  
    int i, random;  
    srand(1);  
    for (i=1; i<=5; i++) {  
        random = rand();  
    }  
}
```



현재 시간을 씨드로 설정하기

srand(time(NULL)) ← time 함수를 실행할 때
컴퓨터의 현재 시간을 rand 함수의 씨드로 설정하기

함수 호출문 **time(NULL)** 실행 시 컴퓨터의 시간이 반환됨
time 함수는 **#include <time.h>**가 필요

10. 함수(function)

10.15 씨드(seed) 설정 함수 : **srand(time(NULL))**

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

```
#include <stdlib.h>           // rand, srand 함수를 위한 헤더 파일
```

```
#include <time.h>           // time 함수를 위한 헤더 파일
```

```
int main() {
```

```
    int i, random;
```

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

```
    for (i=1; i<=5; i++) {
```

```
        random = rand();
```

```
        printf("%d번째 난수: %5d \n", i, random);
```

```
    }
```

```
    return 0;
```

```
}
```



프로그램을 실행할 때마다 time 함수의 반환 값이 달라지므로 서로 다른 순서의 난수 5개를 얻는다.

1번째 난수: 10809	1번째 난수: 10829	1번째 난수: 10845	1번째 난수: 10855
2번째 난수: 16570	2번째 난수: 15524	2번째 난수: 3730	2번째 난수: 3208
3번째 난수: 21255	3번째 난수: 30136	3번째 난수: 21153	3번째 난수: 9209
4번째 난수: 13156	4번째 난수: 26464	4번째 난수: 15709	4번째 난수: 22363
5번째 난수: 17007	5번째 난수: 13358	5번째 난수: 32163	5번째 난수: 30339
계속하려면 아무 키나 누르십시오...	계속하려면 아무 키나 누르십시오...	계속하려면 아무 키나 누르십시오...	계속하려면 아무 키나 누르십시오...

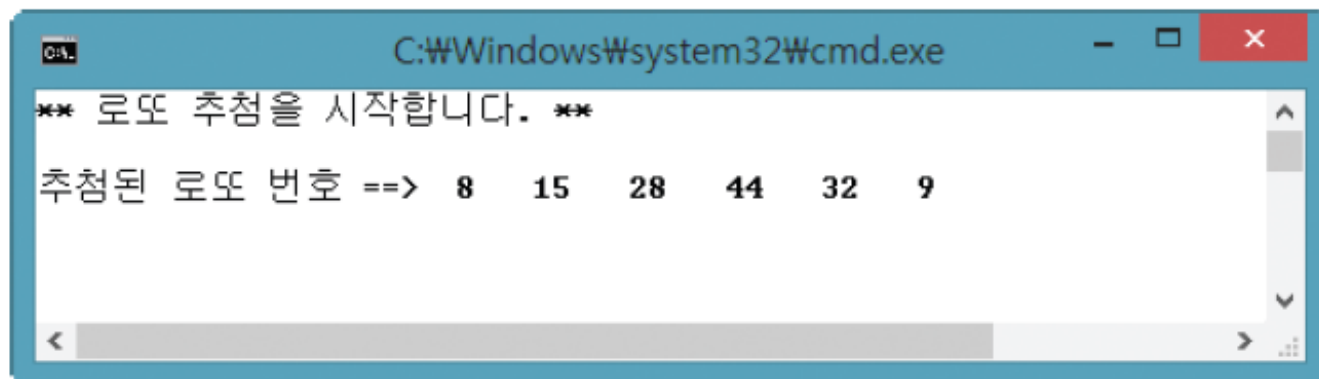
10. 함수(function)

10.16 응용 프로그램 (로또 숫자 자동 추첨 프로그램 만들기)

: 1 ~ 45 번호 중에서 6개를 추출하기

예제설명 1~45까지 숫자 중에서 숫자 6개를 자동으로 뽑는 프로그램이다.

실행결과



```
** 로또 추첨을 시작합니다. **  
추첨된 로또 번호 = = > 9 37 26 35 30 40  
계속하려면 아무 키나 누르십시오 . . .
```

```
** 로또 추첨을 시작합니다. **  
추첨된 로또 번호 = = > 22 3 44 9 20 24  
계속하려면 아무 키나 누르십시오 . . .
```

```

#include <stdio.h>
#include <stdlib.h> // rand, srand 함수를 위한 헤더 파일
#include <time.h> // time 함수를 위한 헤더 파일

int getNumber() {
    return rand() % 45 + 1; // 0 ~ 44 + 1    1 ~ 45
}

int main() {
    int lotto[6] = {};
    int i, k, num;
    char dup = 'N'; // 중복성 확인을 위함

    printf("** 로또 추첨을 시작합니다. ** \n\n");
    srand((unsigned)time(NULL));
    for (i = 0; i < 6; i++) { // 6개의 숫자 뽑힐 때까지
        num = getNumber();
        for (k = 0; k < 6; k++) {
            if (lotto[k] == num) dup = 'Y'; // 중복성 확인
        }
        if (dup == 'N') lotto[i++] = num; // 중복이 안된 경우만 증가
        else dup = 'N';
    }

    printf("추첨된 로또 번호 = = > ");
    for (i = 0; i < 6; i++) {
        printf("%d ", lotto[i]);
    }
    printf("\n\n");
}

```

➔ 디버그 – 디버깅하지 않고 시작

11. 변수

11.1 지역변수, 블록의 지역 변수(권장하지 않음)

```
4 int main()
5 {
6     int i, sum;
7
8     sum = 0;
9     for (i=0; i<N; i++)
10    {
11        int height; // for 블록의 지역 변수 선언
12
13        printf("%d번의 키는? ", i+1);
14        scanf("%d", &height);
15        sum += height;
16    } // for 블록의 끝
17
18    printf("평균 키: %.1lf cm \n", (double)sum/N);
19
20    return 0;
21 } // main 블록의 끝
```

main 함수의 지역 변수
→ 5~21행에서만 참조 가능

for 블록의 지역 변수
→ 10~16행에서만 참조 가능

height 참조 불가

11. 변수

11.2 변수명은 같지만 완전히 다른 변수

```
6 int main()
7 {
8     int i;
9     int pass = 0, sum = 0; // 합격자 수, 합격자 점수의 합
10    int score[N] = {93, 82, 49, 55, 75}; // N명의 점수
13    printf("전체 평균: %.11f \n", compute_ave(score));
16    for (i=0; i<N; i++)
17    {
18        if (score[i] >= 60)
19        {
20            sum += score[i];
21            pass++;
22        }
23    }
24    printf("합격자의 평균: %.11f \n", (double) sum / pass);
26    return 0;
27 }
```

9행 sum의 참조 영역

서로 다른 지역 변수

```
30 double compute_ave(int ary[N])
31 {
32     int i, sum = 0;
33
34     for (i=0; i<N; i++)
35         sum += ary[i];
36
37     return (double)sum / N;
38 }
```

32행 sum의 참조 영역

11. 변수

11.3 전역 변수는 모든 함수에서 참조 가능 → 인수 전달이 필요 없음

```
int find_larger(); // 함수 원형 선언
int n1, n2, max;   // 전역 변수 선언
```

```
int main()
{
    printf("첫째 정수? "); scanf("%d", &n1);
    printf("둘째 정수? "); scanf("%d", &n2);
```

```
    max = find_larger();
```

find_larger 함수에서도 n1, n2 변수를
참조할 수 있으므로 인수로 전달할
필요가 없다.

```
    printf("n1=%d, n2=%d 중 큰 값은 %d \n", n1, n2, max);
```

```
    return 0;
```

```
}
```

```
// 전역 변수 n1, n2 중 큰 값을 반환하는 find_larger 함수
```

```
int find_larger()
```

```
{
```

```
    if (n1 > n2)
```

```
        return n1;
```

```
    else
```

```
        return n2;
```

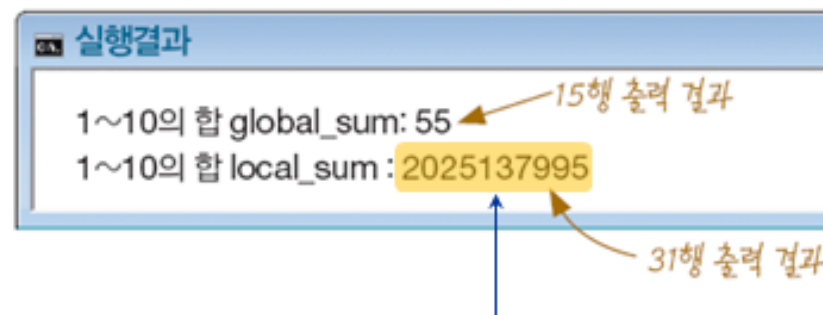
```
}
```

← 전역 변수

11. 변수

11.4 변수 초기화

```
5 int global_sum; // 전역 변수 선언 ← 초기화 필요
6
7 int main()
8 {
9     auto int i; // 자동 (지역) 변수 선언
12    for (i=1; i<=10; i++) ← 지역 변수 초기화
13        global_sum += i;
15    printf("1~10의 합 global_sum: %d\n", global_sum);
16
17    sum_to_10();
19    return 0;
20 }
22
23 void sum_to_10() 초기화 필요
24 {
25     auto int i, local_sum; // 자동 (지역) 변수
26
27     for (i=1; i<=10; i++)
28         local_sum += i;
29
30     printf("1~10의 합 local_sum : %d\n",
31           local_sum);
32 }
```



쓰레기 값 2025137940에 1부터 10까지 누적한 결과 값
(⊗) 이 결과는 'release' 모드에서 실행한 것이며, 'debug' 모드에서는 29행에서 local_sum이 초기화되지 않고 사용되므로 실행이 중단됨

12. 포인터 (pointer)

12.1 개요

: 일반 변수

프로그램에서 사용하는 데이터를 저장

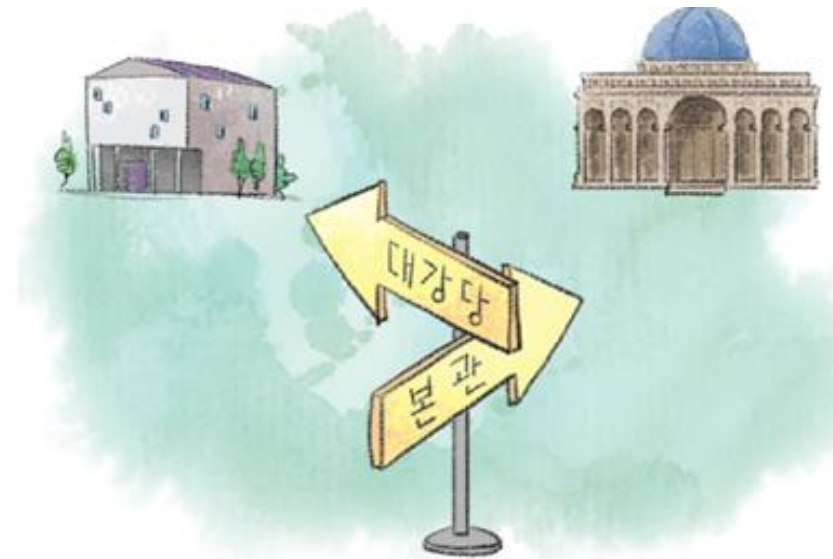
: 포인터 변수

데이터가 저장된 주기억장치의 주소만 저장
(간단히 포인터(pointer)라고도 함)

: 포인터의 장점

직접 참조할 수 없는 변수를 포인터를 이용하여
간접적으로 참조 가능

기억 공간의 효율적 사용, 프로그램 성능 개선



12. 포인터 (pointer)

12.1 개요

: 주소 부여 단위

설치된 운영체제에 의해 결정되는데 대부분은 1바이트마다 1개 주소 부여

: 2GB 주기억장치에 char ch = 'Z';로

선언된 변수 ch에 할당된 기억장소

- 용량이 2G Byte 라면 주소의 범위는?

$$1\text{Byte} = 8\text{bits} = 0 \sim 255 : 2^8$$

$$9\text{bits} = 0 \sim 511 : 2^9$$

$$1\text{KByte} = 1024 \text{ Byte} : 0 \sim 1023 : 2^{10} \text{ Byte}$$

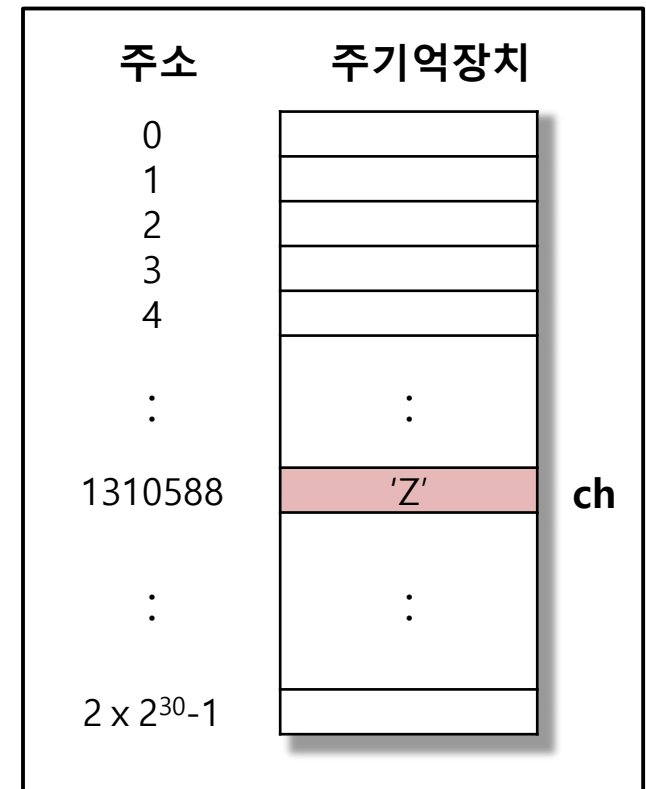
$$1\text{MByte} = 1\text{KByte} \times 1\text{KByte} = 2^{20} \text{ Byte}$$

$$\begin{aligned} 1\text{GByte} &= 1024\text{MByte} = 2^{10} \times 1\text{KB} \times 1\text{KB} \\ &= 2^{10} \times 2^{10} \times 2^{10} = 2^{30} \end{aligned}$$

$$2\text{G(Giga)} = 2 \times 2^{30}$$

➔ $0 \sim 2 \times 2^{30} - 1$ 번지를 사용

$$1\text{TByte} = 1024\text{GByte} = 2^{10} \times 1\text{GByte} = 2^{40}$$

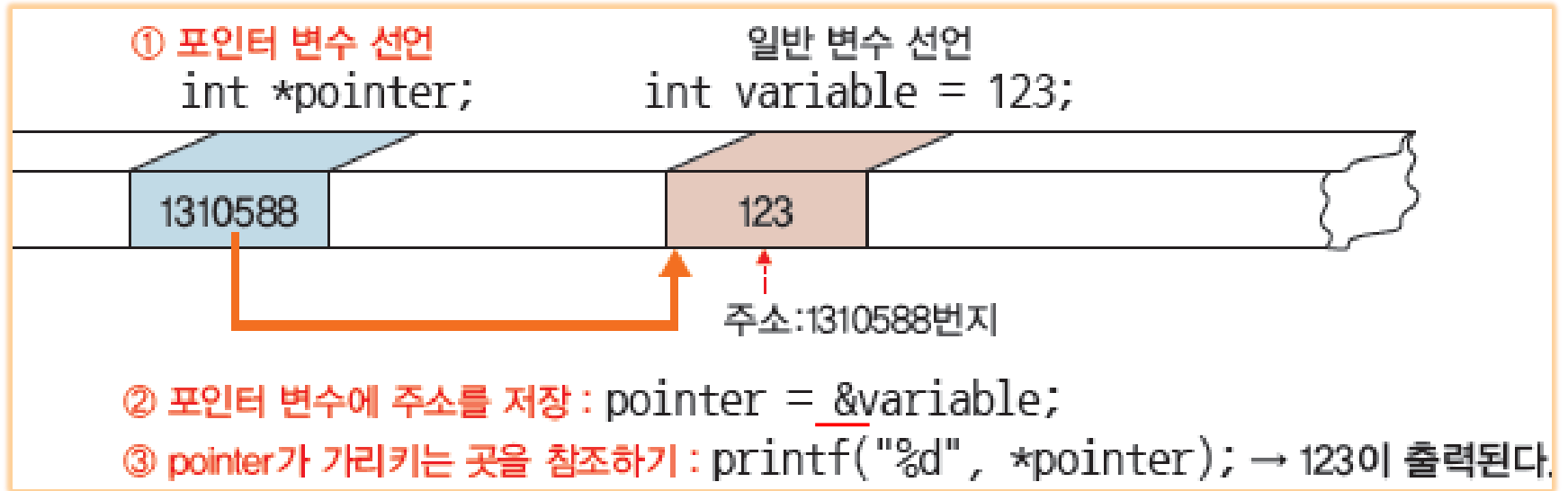


12. 포인터 (pointer)

12.1 개요

: 포인터 사용 3단계

- ① 포인터 변수 선언
- ② 포인터 변수가 특정 기억장소를 가리키기
→ 가리키고 싶은 기억장소의 주소를 포인터 변수에 대입
- ③ 포인터를 사용한 간접 참조
→ 특별 연산자인 간접연산자 '*'를 이용



12. 포인터 (pointer)

12.1 개요

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

```
int main() {
```

```
    int var = 100;    // int형 변수 var을 선언하면서 값을 100으로 초기화
```

```
    int *ptr;         // int형 자료, 기억 장소의 주소를 저장할 포인터 변수 ptr 선언
```

```
    ptr = &var;       // ptr이 변수 var(주소)을 가리키게 함
```

```
    printf("변수 var의 값: %d \n", var);
```

```
    printf("var의 간접 참조 (*ptr) 결과값: %d \n\n", *ptr); // 변수 var 주소에 저장된 값을 출력하기
```

```
    printf("변수 var의 주소:%u(%p) \n", &var, &var); // 변수 var의 주소 출력
```

```
    printf("변수 ptr에 저장된 주소:%u(%p) \n", ptr, ptr); // 포인터 변수 ptr에 저장된 주소 출력하기
```

```
    return 0;
```

```
}
```

%u : 양의 정수

%p : 16진수 포인터

→ 디버그 - 디버깅하지 않고 시작

```
변수 var의 값: 100
var의 간접 참조 (*ptr) 결과값: 100

변수 var의 주소:3136747060(00000001BAF6F634)
변수 ptr에 저장된 주소:3136747060(00000001BAF6F634)
```

12. 포인터 (pointer)

12.2 포인터 변수 선언

: 포인터 변수의 선언

포인터변수가 가리키는 곳에 저장될 자료형

자료형 *포인터 변수명;

char *p

주소



'Y'

int *pi

주소



1024

나중에 포인터가 가리키는 곳에
자료가 저장된 경우의 그림

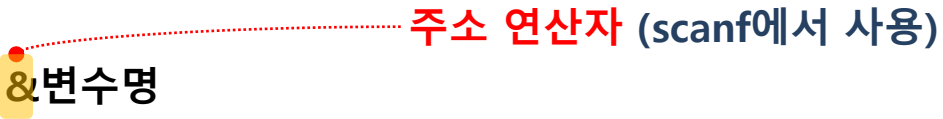
int *ptr1, ptr2; → ptr1만 포인터 변수

int *ptr1, *ptr2; → 둘 다 포인터 변수

12. 포인터 (pointer)

12.3 주소 연산자 &와 주소 대입

: 변수의 주소 구하기

 주소 연산자 (scanf에서 사용)
&변수명

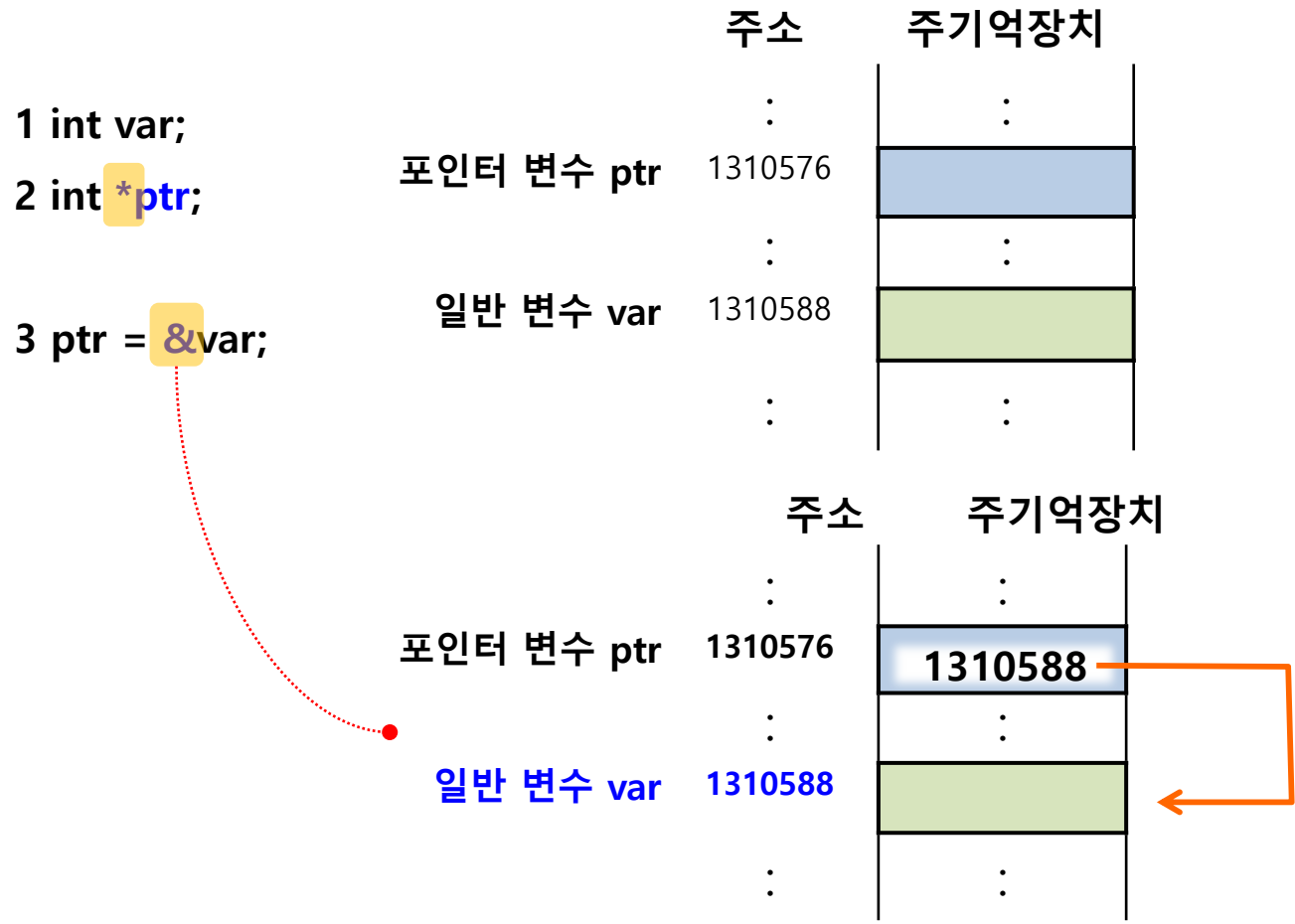
: 주소의 대입

- 포인터변수가 해당 주소의 기억장소를 가리키게 됨

포인터 변수명 = &변수명;

12. 포인터 (pointer)

12.3 주소 연산자 &와 주소 대입



12. 포인터 (pointer)

12.3 주소 연산자 &와 주소 대입

[주의]

: `int *ptr = &var;` 의미

```
int *ptr;  
ptr = &var;
```

```
int *ptr;  
*ptr = &var;
```

: 주소 출력 변환 명세

`%u`: 양의 10진수로 출력, `%p`: 16진수로 출력

[예]

```
printf("변수 var의 주소: %u (16진수: %p) \n", &var, &var);
```

→ 변수 var의 주소: 1310588 (16진수: 0013FF7C)

```
printf("포인터 변수 ptr의 주소: %u (16진수: %p) \n", &ptr, &ptr);
```

→ 포인터 변수 ptr의 주소: 1310576 (16진수: 0013FF70)

12. 포인터 (pointer)

12.3 주소 연산자 &와 주소 대입

일반 변수의 직접 참조

```
1 var = 100;  
2 printf("변수 var에 저장된 값: %d", var);
```

변수 var
100
주소: 1310588번지

포인터 변수의 직접 참조

```
1 int var = 100;  
2 int *ptr;  
3 ptr = &var;  
4 printf("포인터 변수 ptr의 주소:%u Wn", ptr);
```

포인터 변수
1310588

ptr = &var

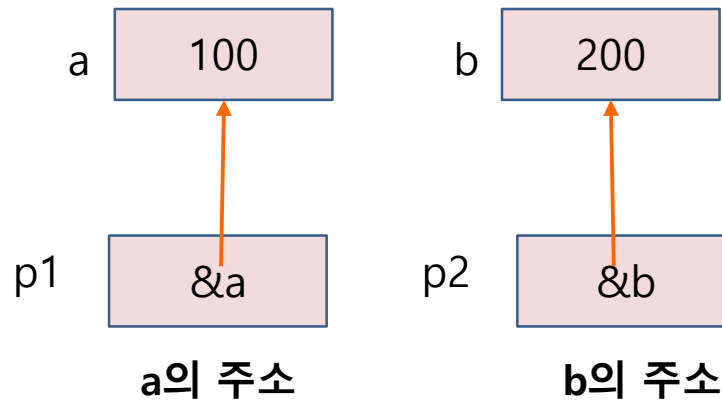
변수 var
100
주소: 1310588번지

12. 포인터 (pointer)

12.4 변수간의 교환과 포인터 간의 교환 비교

두 변수의 값 교환 방법 두 가지

: a와 b 두 값을 직접 교환 vs. 두 변수를 가리키는 포인터 p1과 p2 교환



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

```
int main() {
```

```
    int a = 100, b = 200, temp;
```

```
    int *p1, *p2, *p_temp;
```

```
    p1 = &a;           // p1은 a의 주소를 가리키게 함
```

```
    p2 = &b;           // p2는 b의 주소를 가리키게 함
```

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

```
    printf("*p1 = %d, *p2 = %d\n", *p1, *p2);    // p1, p2에 있는 값을 나타냄
```

```
a = 100,    b = 200
*p1 = 100,  *p2 = 200
```

```
    printf("\n>> p1과 p2가 가리키는 곳에 저장된 값을 직접 바꾸기 \n");
```

```
    temp = *p1;        // temp에 data 100을 넣음
```

```
    *p1 = *p2;         // p2 주소에 값 200을 p1 주소의 값에 넣음
```

```
    *p2 = temp;        // data 100을 p2 주소의 값에 넣음
```

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

```
    printf("*p1 = %d, *p2 = %d\n", *p1, *p2);
```

```
>> p1과 p2가 가리키는 곳에 저장된 값을 직접 바꾸기
a = 200,    b = 100
*p1 = 200,  *p2 = 100
```

```
    printf("\n>> 포인터 p1과 p2에 저장된 주소를 바꾸기 \n");
```

```
    a = 100, b = 200;    // 원래 a와 b의 값으로 초기화
```

```
    p_temp = p1;         // p_temp에 p1 주소를 넣음
```

```
    p1 = p2;             // p2주소를 p1 주소에 넣음
```

```
    p2 = p_temp;         // p1 주소를 p2 주소에 넣음
```

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

```
    printf("*p1 = %d, *p2 = %d\n", *p1, *p2);
```

```
    return 0;
```

```
}
```

```
>> 포인터 p1과 p2에 저장된 주소를 바꾸기
a = 100,    b = 200
*p1 = 200,  *p2 = 100
```

12. 포인터 (pointer)

12.5 포인터의 덧셈과 뺄셈

포인터에 대한 덧셈과 뺄셈이 가능 → 주소의 증가와 감소를 의미함

→ 곱셈과 나눗셈은 불가능, 실수와의 연산은 불가능

→ 포인터 +1

주소가 1 증가함

자료형	ptr + 1의 실제 연산	ptr - 1의 실제 연산
char *ptr;	ptr + 1	ptr - 1
int *ptr;	ptr + 4	ptr - 4
float *ptr;	ptr + 4	ptr - 4
double *ptr;	ptr + 8	ptr - 8

ptr+1 의 실제 연산은

주소 1개 증가 : char 형이 값(data) 8bit, 즉 1byte이므로

주소 4개 증가 : int 형이 값(data) 32bit, 즉 4byte이므로

→ 1byte 가 주소 1개

주소 8개 증가 : double 형이 값 64bit, 즉 8byte이므로

12. 포인터 (pointer)

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

```
int main() {
```

```
    char vc = 'A', *pc;
```

```
    int vi = 123, *pi;
```

```
    double vd = 12.345, *pd;
```

```
    pc = &vc;           // pc는 vc의 주소를
```

```
    pi = &vi;           // pi는 vi의 주소를
```

```
    pd = &vd;           // pd는 vd의 주소를
```

```
    printf("Wn pc-1 = %u, pc = %u, pc+1 = %u", pc-1, pc, pc+1);
```

```
    printf("Wn pi-1 = %u, pi = %u, pi+1 = %u", pi-1, pi, pi+1);
```

```
    printf("Wn pd-1 = %u, pd = %u, pd+1 = %u", pd-1, pd, pd+1);
```

```
    return 0;
```

```
}
```

→ 디버그 – 디버깅하지 않고 시작

주소 1개 가변

주소 4개 가변

주소 8개 가변

```
pc-1 = 2424138, pc = 2424139, pc+1 = 2424140
pi-1 = 2424108, pi = 2424112, pi+1 = 2424116
pd-1 = 2424076, pd = 2424084, pd+1 = 2424092
```

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

```
int main() {
```

```
    int a = 100, b = 200;
```

```
    int *p1, *p2;           // int형 포인터 변수 선언(int 이므로 주소 4개)
```

```
    p1 = &a;                // p1에 a의 주소를
```

```
    printf("p1 = &a 후: a = %d, *p1 = %d \n", a, *p1);
```

```
p1 = &a 후: a = 100, *p1 = 100
```

```
    *p1 = *p1 + 1;          // p1이 가리키는 주소에 있는 값에 1 더하기
```

```
    printf("( *p1 )++ 후: a = %d, *p1 = %d \n", a, *p1);
```

```
( *p1 )++ 후: a = 101, *p1 = 101
```

```
    p2 = p1;                // p2도 p1 주소를 넣기
```

```
    printf("p2 = p1 후: a = %d, *p1 = %d, *p2 = %d \n", a, *p1, *p2);
```

```
p2 = p1 후: a = 101, *p1 = 101, *p2 = 101
```

```
    (*p2)++;                // 위로 인해 *p2 값과 *p1 값이 동일, 1증가
```

```
    printf("( *p2 )++ 후: a = %d, *p1 = %d \n\n", a, *p1);
```

```
    printf("&a = %u, &b = %u, b = %d \n", &a, &b, b);
```

```
    printf("p1 = %u, p1-1 = %u, *(p1-3) = %d \n", p1, p1-1, *(p1-3));
```

```
    // 주소, p1주소-1, p1주소-3에 있는 값
```

```
    return 0;
```

```
( *p2 )++ 후: a = 102, *p1 = 102
```

```
&a = 3930496, &b = 3930484, b = 200  
p1 = 3930496, p1-1 = 3930492, *(p1-3) = 200
```

```
}
```

12. 포인터 (pointer) & 배열

12.6 포인터의 가감 연산은 어디에서 사용될까?

: 배열명은 배열의 시작 주소인 포인터 상수

`int arr[5]={1, 2, 3, 4, 5};` 에서

배열명 `arr` == 배열 시작 주소 == 첫 원소 시작 주소 == `&arr[0]`

`int arr[5]={1, 2, 3, 4, 5};` 에서

`*(arr + 0)` 또는 `*arr` → `arr[0]`

`*(arr + 1)` : `arr`이 가리키는 곳 보다 하나 뒤 → `arr[1]`

`*(arr + 2)` : `arr`이 가리키는 곳 보다 두개 뒤 → `arr[2]`

`*(arr + i)` : `arr`이 가리키는 곳 보다 `i`개 뒤 → `arr[i]`

12. 포인터 (pointer) & 배열

12.6 메모리와 관련된 포인터의 기본 내용

- ① 컴퓨터의 모든 메모리에는 주소(Address)가 지정되어 있음.
- ② `int aa[3];` 과 같이 배열을 선언하면
배열 `aa`는 변수가 아닌 메모리의 주소값 그 자체를 의미
→ 이를 '포인터 상수'라고도 함.
- ③ 포인터 변수란 "주소를 담는 그릇(변수)"이고, 포인터 변수를 선언할 때에는 `int *p;` 또는 `char *p;`와 같이 '*'를 붙여서 선언한다
- ④ 포인터 변수에는 주소만 대입해야 하는데, 이는 변수 앞에 '&'를 붙이면 된다.

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

```
int main() {
```

```
    int aa[3];
```

```
    int *p;    // int 이므로 주소 4개(1개주소-1byte, 총 4byte)
```

```
    int i, hap = 0;
```

```
    for (i = 0; i < 3; i++) {
```

```
        printf(" %d 번째 숫자 : ", i + 1);
```

```
        scanf_s("%d", &aa[i]);
```

```
    }
```

```
    p = aa;    ← 포인터 변수에 배열 aa의 주소를 대입한다.
```

```
    for (i = 0; i < 3; i++) {
```

```
        hap = hap + *(p + i);
```

```
        printf("주소값 = %dWn", p+i);
```

```
    }
```

```
    printf("입력 숫자의 합 = > %dWn", hap);
```

```
    return 0;
```

```
}
```

← aa[0]~aa[2]까지의 합계를 구한다
← p의 주소에 i값 증가만큼 주소를 증가

```
1 번째 숫자 : 40
2 번째 숫자 : 50
3 번째 숫자 : 10
주소값 = 3144612
주소값 = 3144616
주소값 = 3144620
입력 숫자의 합 = > 100
```

12. 포인터 (pointer) & 배열

```
for (i = 0; i < 3; i++) {  
    hap = hap + *(p + i);  
    printf("주소값 = %d\n", p+i);  
}
```

```
1 번째 숫자 : 40  
2 번째 숫자 : 50  
3 번째 숫자 : 10  
주소값 = 3144612  
주소값 = 3144616  
주소값 = 3144620  
입력 숫자의 합 = > 100
```

➔ hap에 *(p+i)를 누적하는 과정을 세 번 반복.

i값이 0일 때는 *(p+0)을 의미하는데

: 이는 (p+0)번지가 가리키는 곳의 실제값, 즉 3144612번지의 실제 값인 정수 40이 된다.

: 따라서 *(p+1)은 (p+1)번지가 가리키는 실제값 이므로 정수 50 됨

➔ 정수형 aa[3] 배열을 선언하면

3144612 ~ 3144616 ~ 3144620 번지까지 4byte × 3 = 12byte의 메모리를 확보

배열 aa는 3144612 번지 자체를 의미하는 포인터 상수

3144612 3144613 3144614 3144615 3144616 --- 3144620

aa[0]

aa[1] ---

aa[2]

p ➔ 3144612

p+1 ➔ 3144616

p+2 ➔ 3144620

12.7 함수 간 인수 전달과 포인터

```
#include <stdio.h>

void swap_value(int x, int y);           // 값에 의한 호출 방식
void swap_address(int *x, int *y);       // 주소에 의한 호출 방식

int main() {
    int x = 100, y = 200;  // local 변수
    printf("In main: x=%d, y=%d \n\n", x, y);

    swap_value(x, y);        // 값에 의한 호출: x와 y의 값을 전달
    printf("In main: x=%d, y=%d (swap_value(x, y) 호출 후)\n\n", x, y);

    swap_address(&x, &y);    // 주소에 의한 호출: x와 y의 주소를 전달
    printf("In main: x=%d, y=%d (swap_address(&x, &y) 호출 후)\n\n", x, y);

    return 0;
}
```

```
In main: x=100, y=200
In swap_value: x=200, y=100
In main: x=100, y=200 (swap_value(x, y) 호출 후)

In swap_address: *x=200, *y=100
In main: x=200, y=100 (swap_address(&x, &y) 호출 후)
```

```

void swap_value(int x, int y) {
    int temp;
    temp = x;
    x = y;
    y = temp;
    printf("In swap_value: x=%d, y=%d \n", x, y);
}

```

```

void swap_address(int *x, int *y) {           // x, y는 주소를 저장하는 포인터 변수로 선언
    int temp;
    temp = *x;                               // x가 가리키는 곳의 값을 temp에 대입
    *x = *y;                                 // y가 가리키는 곳의 값을 x가 가리키는 곳에 대입
    *y = temp;                               // temp의 값을 y가 가리키는 곳에 대입
    printf("In swap_address: *x=%d, *y=%d \n", *x, *y);
}

```

→ 디버그 – 디버깅하지 않고 시작

```

In main: x=100, y=200
In swap_value: x=200, y=100
In main: x=100, y=200 (swap_value(x, y) 호출 후)
In swap_address: *x=200, *y=100
In main: x=200, y=100 (swap_address(&x, &y) 호출 후)

```

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

함께 생각하기

```
void swap_value(int x, int y);
```

// 값에 의한 호출 방식

```
void swap_address(int *x, int *y);
```

// 주소에 의한 호출 방식

```
int main() {
```

```
    int x = 100, y = 200;
```

```
    printf("In main: x=%d, y=%d \n\n", x, y);
```

```
In main: x=100, y=200
```

```
    swap_value(x, y);
```

// 값에 의한 호출: x와 y의 값을 전달

```
    printf("In main: x=%d, y=%d (swap_value(x, y) 호출 후)\n\n", x, y);
```

```
In main: x=100, y=200 (swap_value(x, y) 호출 후)
```



: swap_value(x, y); 에서 x=200, y=100 이었으나,
함수 호출 후에는, x와 y 값은 변경되지 않음

```
    swap_address(&x, &y);
```

// 주소에 의한 호출: x와 y의 주소를 전달

```
    printf("In main: x=%d, y=%d (swap_address(&x, &y) 호출 후)\n\n", x, y);
```

```
    return 0;
```



: swap_address(&x, &y); 에서 x=200, y=100 변경되고,
함수 호출 후에도, x와 y 값은 변경됨

```
}
```

```
void swap_value(int x, int y) {
```

```
    int temp;
```

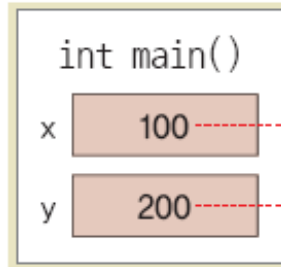
```
    temp = x;
```

```
    x = y;
```

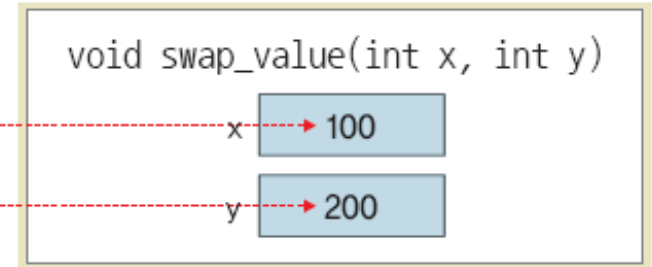
```
    y = temp;
```

```
    printf("In swap_value: x=%d, y=%d \n", x, y);
```

```
}
```



인수의 값이 전달된다.



```
In swap_value: x=200, y=100
```

```
void swap_address(int *x, int *y) {
```

```
    int temp;
```

```
    temp = *x;
```

```
    *x = *y;
```

```
    *y = temp;
```

```
    printf("In swap_address: *x=%d, *y=%d \n", *x, *y);
```

```
}
```

// x, y는 주소를 저장하는 포인터 변수로 선언

// x가 가리키는 곳의 값을 temp에 대입

// y가 가리키는 곳의 값을 x가 가리키는 곳에 대입

// temp의 값을 y가 가리키는 곳에 대입

```
In swap_address: *x=200, *y=100
```

```
In main: x=200, y=100 (swap_address(&x, &y) 호출 후)
```

12.8 배열을 함수로 전달하기 : 배열을 매개변수로 갖는 함수 - 백분율 구하기

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

```
#define N 4
```

```
void print_arr(int arr[N]);
```

```
void percentage(int arr[N]);
```

```
int main() {
```

```
    int count[N] = {42, 37, 83, 33};
```

```
    printf("인원수: ");
```

```
    print_arr(count);
```

// 함수 호출 전 배열 출력

```
    percentage(count);
```

// 인원수를 백분율로 변경

```
    printf("\n백분율: ");
```

```
    print_arr(count);
```

// 함수 호출 후 배열 출력

```
    return 0;
```

```
}
```

```
void print_arr(int arr[N]) {    // int arr[]도 가능
```

```
    int i;
```

```
    for (i=0; i<N; i++) printf("%3d", arr[i]);
```

```
}
```

```
void percentage(int arr[N]) {
```

```
    int i, total = 0;
```

```
    for (i=0; i<N; i++) total += arr[i];
```

// total에 구하기

```
    for (i=0; i<N; i++) arr[i] = (int)((double)arr[i] / total * 100);
```

// 백분율 구하기

```
}
```

인원수:	42	37	83	33
백분율:	21	18	42	16

12.8 배열을 함수로 전달하기 ➔ 포인터를 사용하여 변경하기

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

```
#define N 4
```

```
void print_arr(int *arr);           // 비교 10-10 void print_arr(int arr[N]);  
void percentage(int *arr);         // 비교 10-10 void percentage(int arr[N]);
```

```
int main() {  
    int count[N] = {42, 37, 83, 33};  
    printf("인원수: ");  
    print_arr(count);              // count 배열의 주소를 전달해 출력하기  
    percentage(count);            // count 배열의 주소를 전달해 백분율로 변환하기  
  
    printf("\n백분율: ");  
    print_arr(count);              // count 배열의 주소를 전달해 전달해 출력하기  
  
    return 0;  
}
```

```

void print_arr(int *arr)
{
    int i;

    for (i=0; i<N; i++)
        printf("%3d", *(arr + i));
}

```

```
// void print_arr(int arr[N])
```

```
// 주소를 증가시키며, 주소의 값을 출력
// printf("%3d", arr[i]);
```

```

void percentage(int *arr)
{
    int i, total = 0;

    for(i=0; i<N; i++)
        total += *(arr + i);

    for (i=0 ;i<N; i++)
        *(arr + i) = (int) ((double) *(arr + i) / total * 100);

    // arr[i] = (int) ((double) arr[i] / total * 100);
}

```

```
// void percentage(int arr[N])
```

```
// 주소에 있는 값을 더하기
// total += arr[i];
```

```
*(arr + i) = (int) ((double) *(arr + i) / total * 100);
```

```
// arr[i] = (int) ((double) arr[i] / total * 100);
```

인원수: 42 37 83 33
 백분율: 21 18 42 16 계속하려면 아무 키나 누르십시오