<Day 1>

160912 program intro & 과제 #1 intro
doxygen : create the file automatically
과제 : program 돌리고 doxygen 으로 create the file

```
/**
        @file           160912.c
        @brief          First Code
        @author         Minseon Shin ( minseon3@nate.com)
        @date           2016-09-12
*/
```

함수형식
반환형 함수명(인수...)
함수명(동사+명사+부사)

```
#include <stdio.h> // printf scanf 지원
# : 전처리 문법
변수와 상수 (variable, constant variable)
doxygen에 tmp 임시변수 설명하고자 할때///>
///> temp = 임시 변수, 전역(global)변수
int temp = 34;
```

Before the compile -> Pre-processor(전처리기) : e.g. #define #include ....
source Code ->(compiler) Assembly code
Assembly code a=7+2 -> ADD 7 2 (Assembly language)

[프로그램 개발과정 순서]
소스개발 -> 컴파일러 -> 링커 -> 실행 -> 디버깅

```
/** introduction of the bottom Code
        @brief          두 수의 합을 계산
        @param(eter) a 첫 번째 인수
        @param  b       두 번째 인수
        @return         두 수의 합
        @todo           실수 처리
*/
int sum(int a, int b) // int add_two // <Google> int ivel; / float fvel;
{
        int result; // 지역변수(local)
        result =  a + b; // = : 대입연산자
                         // +,-,*,/,% : 산술연산자
                         // 3%2=1
        return result;
}
/**
        @brief  main function
        @return 0 - 정상종료시
*/
int main(void)
{
        함수의 시작&영역 {}
        함수 몸체(Body)
        int a = 3;
        int b = 7;
        int c = a + b;
        printf("%d", c);  // d : decimal(10진수), % : 특수문자
                            // %f : floating point 실수
        return 0; // 0 : normal conclusion 정상적 종료시 0 반환
}
```

<Day 2_1>

```
/**
@brief main    함수
@return 0      정상종료
*/
int main(void)
{
double result;
result = calAvg(70, 71, 72);
printf("Average : %.2lf", result); OR printf("Average : %.2f",calAvg(70, 71,
72);
return 0;

}
```

[프로그램 실행 되는 순서]

1.전처리기(pre-processor) : code -> 전처리기를 통과한 코드

2.컴파일러(compiler) : pre_processor code -> Assembly code

```
ex) int a = 3    c = a+b
            -> Load a
               add c a b
```

3.어셈블러(Assembler) : Assembler code -> 기계어(object) 코드(object file, 목적파일)

링커(Linker) : Object code -> 실행파일

로더(Loader) : 실행파일-> 메모리로 이동

오브젝트 파일, 실행파일: 이진파일, 바이너리파일(binary file) (기계어)

ex)

0.1 => 1bit, 1 byte = 8bits

off, on

false, true (단, true 는 0이 아닌 모든 수.)

ex1)

```
if(1) {printf("Average : %.2f",calAvg(70, 71, 72);}
```

<Day 2_2>

file : calAvg.c

```
/**
@brief          세수의 평균을 구하는 함수
@param a        1st argument
@param b        2nd argument
@param c        3rd argument
@return         세수의 평균
@to do          구현
*/
double calAvg(double a, double b, double c)
{
double avg;
avg = (a + b + c) / 3.0;
return avg; OR return (a+b+c)/3.0;
}
```

file : calAvg.h

```
#pragma once
double calAvg(double a, double b, double c);
```

<Day 3>

```
/**
@file          160921.c
@brief
@author Minseon Shin (minseon3@nate.com)
@date          2016-09-21
*/
```

[자료형 및 scanf]
문자: char(%c)
정수: short, int(%d), long, long long(%ld)
실수: float(%f), double(%lf)
실수: fixed point, floating point
실수를 표현할 때 사용하는 바이트 수에 따라서
single precision, double precision
문자열: string(%s)

```
#include <stdio.h>
int main()
{
   printf("%d\n", (int)3.14); //casting
   printf("%f\n", 3.14);
   printf("%c\n", 'A');
   printf("%s\n", 'SHIN');
   return 0;
}
```

```
#include <stdio.h>
int main()
{
   int input=0;                  // 변수선언: 자료형(초기화!)
   char addre = 0;
   printf("정수를 입력하시오 : ");
   scanf("%d", &input);         // ampersnad(&) : 주소
   printf("%d\n",input);
   printf("%p\n", &input);      // 이상한 값 도출. 잘못된 주소 입력값.
   return 0;
```

```c
#include <stdio.h>
int main()
{
printf("정수를입력하시오: ");
scanf("%d", &input); // ampersnad(&) : 주소
printf("%d\n", input);

printf("주소를입력하시오: ");
scanf("%p", &addre);
printf("%p\n", addre);
return 0 ;
}
```

[실습문제]

```c
@brief   세 개의 정수와 평균값을 출력하는 프로그램

#include <stdio.h>
int main()
{
   short score1, score2, score3;
   printf("세정수를입력하시오: ");
   scanf("%d %d %d", &score1, &score2, &score3);
   printf("세과목평균: %d\n", (score1 + score2 + score3) / 3);
   return 0;
}
```

[실습문제 2]

```c
@brief  달에서의 몸무게(달의중력= 지구의중력의17%)

#include <stdio.h>
int main()
{
   double weight;
   printf("몸무게를입력하시오: ");
   scanf("%lf", &weight);

   printf("달에서의몸무게는%.2lf kg 입니다.", weight*0.17);
   return 0;
}
```

<Day 4>

```
/**
@file   160926.c
@brief 변수와 자료형
@author minseon shin(minseon3@nate.com)
@date   2016-09-26
*/
```

[자료형]
1. 문자: ('A','a',...) char(1 byte) %c
2. 숫자: (1, -2, ...) short (int)(2 byte) int (4 byte)
long (int)(4 byte) %d long long (8 byte) %ld
16bit computer에서는 int 가2byte였는데 32bit computer에서는 int가 4byte 여서 새 컴퓨터 등장을 위해 short 등장함.

3. 실수: floating point, 1.23, -3.45
float (4 byte) %f,%e,%E //%e : 1.4 -> 1.4el
double (8byte) %lf %E,%e
long double (8byte) %lf

unsigned, signed
int a // signed int a;
unsigned char b; (8bit) 0~255 // 0~255 = 0 ~ 2^(n-1)-1
(signed) char c; -128~127 = -2^(n-1) ~2^(n-1)-1

* 음수표현방법: SM(sign & magnitute),
1의 보수(1's complement) = 비트 반전 시킨 수
2의 보수(2's complement) = 1's +1
e,g -7이라는 4bit의 수를 만들고 싶다.
1단계) 7을 2진수로 바꾼다. = 0111
2단계) 1의 보수로 바꾼다 = 0111->1000 (최상위비트가 1이 된다.) = (음수다)
3단계) 2의 보수로 바꾼다 = 1000->1001(2's) = 1000(-8)+0001(+1)=(-7)
최상위비트만 음수고 나머지는 상수로 취급한다.

e.g2. 가장 작은 수: 1000 0000 => (2의 보수에서) - (128)
가장 큰 수: 0111 1111 => 127

unsigned int d; 0~ 2^32-1
signed int e;   -2^31~2^31-1

```c
#include <stdio.h>
int main(void)
{
   printf("size of char : %d\n", sizeof(char));
   printf("size of short : %d\n", sizeof(short));
   printf("size of int : %d\n", sizeof(int));
   printf("size of long : %d\n", sizeof(long));
   printf("size of float : %d\n", sizeof(float));
   printf("size of long long : %d\n", sizeof(long long));
   printf("size of long double : %d\n", sizeof(long double));
   printf("size of double : %d\n", sizeof(double));

   printf("size : %d\n", sizeof(11));    // int 자료형으로 취급
   printf("size : %d\n", sizeof(3.14));  // double형으로 취급
   printf("3.141592 = %e\n", 3.141592);
   printf("3.141592 = %E\n", 3.141592);  // 안 쓰면 자동으로 double로 인식함.
   printf("3.141592 = %E\n", 3.141592f); // f를 쓰면 float의 경우의 수로 들어감.
                                         / 243l(long형) 243d(x)
   printf("%lf\n", 2.43e2); //e2는100 = 10^2
   return 0;
}
```

<Day 4_1>

```
/**
@file          160926_1.c
@brief         변수와 자료형2
@author minseon shin(minseon3@nate.com)
@date          2016-09-26
*/

#include <stdio.h>
int main(void)
{
   unsigned char test = 255;
   test = test + 1;           //overflow
   printf("%d\n", test);
   double r = 3.141592222222222;
   printf("%f\n", 3.1415922222);
   printf("%.16lf\n", r);
   return 0;
}
```

자료형이 표현할 수 있는 한계는 1111 1111 인데 +1을 하면 1 0000 0000 이므로 표현 가능한 곳은 8bit 이기에 0으로 출력됨. // overflow

실수(floating point) : +-가수x2^지수(IEEE754) : 정보 통신 규제를 정하는 곳.IEEE
single precision (단일정밀도, 32bit) : 부호(1) + 가수(23) + 지수(8)
                / 바이어스(127) 유효숫자7자리(cf.범위값에서 바이어스를 항상 빼주어야 함.)
double precision (복수정밀도, 64bit) : 부호(1) + 가수(52) + 지수(11)
                / 바이어스(1023) 유효숫자16자리

[식별자 만드는 규칙]
int a :영어대문자, 영어소문자, 밑줄문자(_), 숫자 (단, 숫자는 제일 앞쪽에 올 수 없다.)

상수(3, 3.141592......)

<Day 4_2>

기호상수 : 1. #define 2. const

```
#include <stdio.h>
#define PI 3.141592   ///> 바퀴반지름
                      // C언어 문법 아님-> 상수만 쓰기 때문에 오류검사가 안돼 실수할 가능성up

const double RADIUS = 100.0;
                           // c언어 문법-> 자료형을 이미 기재했기 때문에 오류검사가 가능하다.
int main(void)
{
   double length = 2. * PI *RADIUS;
   double area = PI *RADIUS *RADIUS;        // RADIUS = 106.0 // compile error

   printf("length = %lf\n", length);
   printf("area = %lf\n", area);

   //"a"를 출력하고자 할 때
   printf("\"A\"\n");
   printf("%c\n", 65);
}
```

제어문자 \n : 줄바꿈. \t : tap(사이즈만큼 띔) \a :알람
       \b : back space(한글자 뒤로) \r : carriage return : 커서가 제일 앞으로 감.
       \", \', \\ : back slush
ASCII : 아스키코드(ASCII : American Standard Code for Information Interchange)
char : 1 byte

48 : 0
65 : A
97 : a

<Day 5>

```
/**
@file          project1.c
@brief         수식과 연산자
@author        MInseon Shin
@date          2016-09-28
*/

[자료형 review]
char(1)
short(2), int(4), long(4), long long(8)
float(4), double(8), long double(8)
IEEE754
single precision - 가수23bits, 지수8bits
double precision - 가수52bits, 지수11bits
float 형의 유효숫자2^23 -> 7자리
double 형의 유효숫자2^52 ->16자리

unsigned char -> U08
char -> S08
int ->S32
unsigned long long -> U64
#define U08 unsigned char
#define S32 signed int


#include <stdio.h>
#include <limits.h>                // 자료형의 한계가 개수로 표현되어있음
int main()
{
    float temp = 0;
    temp = (float)1.0e10 + (float)5.0 - (float)1.0e10;
    // double 연산이 됐기 때문에 정상적으로 5.0 이 나오는 것.
    // float연산으로 하면 유효숫자를 넘어가므로 000000.00000이 나온다.
    printf("%f\n", temp);
    return 0;
}
```

[연산자]
수식은 연산자와 피연산자로 이루어진다.
1. 대입연산자: a=2 / a=b=c(우->좌)
2. 산술연산자: +,-,&,/,%
3. 부호연산자: +.-
4. 증감연산자: ++,--
    e.g) a++ => a= a+1, ++a => a=a+1
5. 관계연산자: >, <, ==, !=, <=, >=
6. 논리연산자: &&, ||, !
7. 비트연산자: &, |, ^(Exculsive or/xor), ~(not), <<(left shift), >>(right shift)
    e.g) temp = 3*2 => temp = 3<<1; because 0011(3)->0110(6) '1'이동.
8. 조건연산자: ?(3항연산자)
    e.g) result = (number % 2 == 0) ? 0 : 1; // (조건) ? 참일때: 거짓일때;
    * 1항연산자++ 2항연산자산술연산자등. 3항연산자: 조건연산자.

[우선순위]
단항연산자> 이항연산자> 삼항연산자 (피연산자의 수가 적을수록 우선순위가 높다)

Casting (형변환)

[산술연산자 실행예제]
```c
#include <stdio.h>

int main()
{
   int number = 0;
   printf("숫자를 입력하시오: ");
   scanf("%d", &number);
   if (number % 2 == 0) printf("This number is even number\n");
   else printf("This number is odd number\n");
   OR if((number & 1)==1) printf("odd number");
      else printf("even number");
   // '&' 가'%'보다 좋은 이유: 속도가 훨씬 더 빠르다.
   // 비트연산자는 대입연산자보다 우선순위가 낮기 때문에 ()가 필수!

   return 0;
}
```

[증감연산자 실행예제]

```c
#include <stdio.h>

int main()
{
   int temp = 3;
   printf("%d\n", temp++); //= 3 <후위연산자> : 출력 후에++해준다.
   printf("%d\n", ++temp); //= 5 <전위연산자> : ++해준 후에 출력한다.
   return 0;
}
```

[조건연산자 실행예제]

```c
#include <stdio.h>

int main()
{
   int number = 0;
   int result = 0;
   scanf("%d", &number);
   result = (number % 2 == 0) ? 0 : 1; // (조건) ? 참일때: 거짓일때;
   printf("%d",result);
   return 0;
}
```

<Day 5_1>

실습문제: 초를 입력받아 시, 분, 초를 출력하시오

[My Way]

```
#include <stdio.h>
int main()
{
   int sec, min, hour, sec_limit;
   printf("계산할 초를 입력하시오: ");
   scanf("%d", &sec);
   hour = sec / 3600;
   min = (sec - hour * 3600) / 60;
   sec_limit = sec - hour * 3600 - min * 60;

   printf("입력하신 시간은 %d시간 %d분 %d초입니다.", hour, min, sec_limit);

   return 0;
}
```

[professor correct]

```
#include <stdio.h>
int main()
{
   int hour=0, minute=0, sec=0;
   printf("초: ");
   scanf("%d", &sec);
   hour = sec / 3600;
   minute = (sec%3600)/60;
   sec = sec % 60;

   printf("%d시%d분%d초\n", hour, minute, sec);

   return 0;
}
```

<Day 6>

```
/**
@file          161005
@brief         변수와 자료형, 수식과 연산자 #2
@author        MINSEON SHIN (minseon3@nate.com)
@date          2016-10-05
*/

// 증감연산자 review
#include <stdio.h>
int main()
{
   int temp = 5, tmp = 5;

   if (temp++ > 5) printf("bigger than 5\n");
   else printf("not bigger than 5\n");            // -> not bigger than 5
   if (++tmp > 5) printf("bigger than 5\n");
   else printf("not bigger than 5\n");            // -> bigger than 5

   return 0;
}

// 복합대입연산자
#include <stdio.h>
int main()
{
   int temp = 5;
   temp = temp + 1;
   temp += 1;

   int a, b, c;
   c *= a;// c= c * a

   return 0;
}
```

```c
// 진수설명
#include <stdio.h>
int main()
{
   printf("%o") // 08진수표현
   printf("%x") // 16진수표현
}

// #define 설명
#include <stdio.h>
#define unsigned int U32
#define mySum(a,b) (a+b)
// #define으로 function이 생성되기도 함. 단, 전처리기를 위해 () 필수! OR ((a)+(b))(정석)
int main()
{
   U32 temp; // = unsigned int temp;
   printf("%d\n", mySum(3, 4)*5);
   //-> printf("%d\n",(a+b)*5);
         전처리기가 이렇게 바꾸면c 컴파일러가 이 코드를 실행한다. 단, 전처리기를 위해() 필수!
   return 0;
}

// 참고1 (강의자료3강)
#include <stdio.h>
int main()
{
   float temp = 1234.5678;
   float temp2 = 123.;
   printf("%f\n", temp);      // 1234.56717
   printf("%f\n", temp2);     // 123.00000

   printf("%30.5f\n", temp);
   // 소숫점은 반올림해서 표현함, 없으면 비우고 표현함. 자리 위치 정할 때 사용

   return 0;
}
   // '\0' == NULL (문자열의 끝)
```

```c
// 삼항연산자 쓰임
#include <stdio.h>
#define MAX(a,b) ((a),(b))?(a):(b);
int main()
{
   int a = 5;
   if (a == 5) printf("true\n");
   else printf("false\n");

   (a == 5) ? printf("true\n") : printf("false\n");
   return 0;
}

// 우선연산자
(a & b > c | d )  : 순서가 우리가 생각하는 것과 많이 다름.(암기)
```

[실습문제]

$ax^2 + bx + c$ 이차방정식의 근 구하기. (단, a, b, c를 입력 받아 해를 출력)

```c
// 루트는 sqrt 함수이용. <math.h> 안에 있음
#include <stdio.h>
#include <math.h>
int main(void)
{
    int a, b, c;
    double x1, x2;

    printf("이차방정식 a*x^2 + b*x + c에서 a, b, c를 입력하시오. : ");
    scanf("%d %d %d", &a, &b, &c);

    x1 = (-b + sqrt(b*b - 4 * a*c)) / (2 * a);
    x2 = (-b - sqrt(b*b - 4 * a*c)) / (2 * a);

    printf("입력한 이차방정식의 두 근은 %lf %lf 입니다.", x1, x2);
    return 0;
}
```

<Day 7>

```
/**
@file          example.c
@brief         조건문과 반복문 #1
@author        MInseon shin
@date          2016-10-10
*/

// 조건문
조건문: if, switch
if (조건)
    문장A;              // 조건이 참일 때 수행되는 문장
    문장B;
// 조건문의 할당은 한 문장. 즉, 문장 A까지만 할당됨. 더 이으려면 {} 필요
--------------------------------------------------------------------------------
if (조건)
    문장A;
else           //else 는 생략가능..
    문장B;
--------------------------------------------------------------------------------
if (조건A)
    문장A;
else if (조건B)
    문장B;                      // 조건A는 거짓이고, 조건B가 참일 때
else if (조건C)
    문장C;                      // 조건A, B 둘 다 거짓이고, 조건c가 참일 때.
else
    문장E;                      // A,B,C 다 아니다.
```

```
//switch 문
switch (변수)           //switch (menu)
{
    case A(정수):               // case 1 :
                                // case 'A': (가능하다)  ASCII CODE
        문장A;                  // 변수가 A인 경우 수행
        break;                  // 만약 break가 없으면 문장B로 내려감.(필수)
    case B():
        문장B;                  /// 변수가 B인 경우 수행
        break;
    case C() :
        문장c;                  // 변수가 B인 경우 수행
        break;
    default:
        문장C;                  // 변수가 A,B,C가 아닌 경우 수행
        break;                  //(안 써도 무방함, 컴파일 에러 안 나지만 문법상 씀.)
}
                                //예외처리  A,B,C가 아닌 경우 default 처리함.

// break를 안걸었을 때 결과를 의도하기도 함.(나쁜 예)
switch(score)
{
case 100:
case 99:
case 98:
case 97:
case 96:
case 95:
    printf("A+\n");
    break;
case 94:
case 93:
case 92:
case 91:
case 90:
    printf("A\n");
    break;
}
                                // 일반적으로 switch가 if보다는 더 빠르지만 위의 예는 나쁜 예
```

```
// if는 한 문장만 가지고 간다는 예
if (조건A)
문장A;
문장B;

// else는 가까운 if를 모심.
if (조건A)
    if (조건B); 문장A;
    else 문장B;
else 문장C;
문장D;
}
```

[실습문제] C언어콘서트 P.186 예제4번
이차방정식 aX ^ 2 + bX + c 의 근을 계산하는 프로그램을 작성하여 보자.
1. 사용자에게 이차방정식의 계수 a,b,c를 입력하도록 한다.
2. 만약 a가 0이면 근은 -a/c이다.
3. 만약 판별식이 음수면 실근은 존재하지 않는다.
4. 위의 조건에 해당되지 않으면 다음과 같은 공식을 이용하여 실근을 구한다.

```
#include <stdio.h>
#include <math.h>
int main(void)
{
    double a, b, c;
    double x1, x2;

    printf("이차방정식 aX^2+bX+c에서 a를 입력하시오 : ");
    scanf("%lf", &a);
    printf("이차방정식 aX^2+bX+c에서 b를 입력하시오 : ");
    scanf("%lf", &b);
    printf("이차방정식 aX^2+bX+c에서 c를 입력하시오 : ");
    scanf("%lf", &c);

    if (a == 0.)              // 실수니까 실수형으로!!!
    {
       x1 = x2 = (-c) / b;
       printf("방정식의 근은 %lf 입니다.", x1);
    }
    else if (((b*b) - (4. * a*c)) < 0)
       printf("실근이 존재하지 않습니다.");
    else
    {
       x1 = ((-b) + (sqrt((b*b) - (4. * a*c)))) / (2. * a);
       x2 = ((-b) - (sqrt((b*b) - (4. * a*c)))) / (2. * a);
       printf("방정식의 근은 %lf %lf 입니다.", x1, x2);
    }
    return 0;
    }

// 주의
int a;
if (a = 0)
a++ (수행안됨)
```

<Day 8>

```
/**
@file          example1.c
@brief         퀴즈#2 review, 조건문
@author Minseon Shin (minseon3@nate.com)
@date          2016-10-12
*/
```

```
// 퀴즈review
```

관계연산자: >=, <=, ==, !=

(unsigned, signed) char(%c), short(%d), int(%d), long(%ld), long long(%lld)
<-> float(%f), double(%lf), long double(%lf) (sigle precision, double precision)

Float, Double, Int (x) 대문자 안됨

변수명은 일반적으로 소문자 씀. : speed, light_speed, ligntSpeed,....

대문자는 기호상수 쓸 때 씀. #define PI

```
// goto문
#include <stdio.h>
int main()
{
   int menu ;
   printf("menu(1~3) : ");
   scanf("%d", &menu);

   if ((menu != 1) && (menu != 2) && (menu != 3))
   goto error;
   else printf("MENU : %d", menu);

   return 0;

   error:
   printf("ERROR : invalid command\n");
   return -1;
}
```

goto는 프로그래머가 가장 피해야 하는 코드. 하지만 쓸데가 딱 한번! error 표시할 때 씀.(오류처리)

<Day 8-1>

```
/**
@file          example2.c
@brief         실습문제
@author        Minseon shin (minseon3@nate.com)
@date          2016-10-12
*/

// 점수를 입력 받아 학점을 출력(if-else문)
#include <stdio.h>
int main()
{
   int score;
   printf("학점을 입력하시오: ");
   scanf("%d", &score);

   if ((score > 100) || (score < 0))
   {
    printf("ERROR");
    return -1;
   }
   else if (score >= 90)
   printf("당신의 학점은 A입니다.");
   else if (score >= 80)
   printf("당신의 학점은 B입니다.");
   else if (score >= 70)
   printf("당신의 학점은 C입니다.");
   else if (score >= 60)
   printf("당신의 학점은 D입니다.");
   else
   printf("당신의 학점은 E입니다.");

   return 0;
   }
```

```c
// 점수를 입력 받아 학점을 출력 (switch문)
#include <stdio.h>
int main()
{
   int score;
   printf("학점을 입력하시오: ");
   scanf("%d", &score);

   switch (score/10)
   {
     case 10;
     case 9;
     case 8;
     case 7;
     case 6;
     case 5;
     case 4;
     case 3;
     case 2;
     case 1;
     break;
   }
```

<Day 8-2>

```
/**
@file          example4.c
@brief         C언어콘서트p.186 6번
@author        Minseon shin (minseon3@nate.com)
@date          2016-10-12
*/

#include <stdio.h>
int main()
{
   int price, get_price, give_price;
   int thousand, five_thousand, million;

   printf("물건의 가격을 입력하시오: ");
   scanf("%d", &price);

   printf("받으신 돈을 입력하시오: ");
   scanf("%d", &get_price);

   give_price = get_price - price;

   if (give_price < 0)
   {
     printf("돈이 부족합니다.");
     return -1;
   }
   /*
   million = give_price/ 10000;
   five_thousand = (give_price - 10000*million) / 5000;
   thousand = (give_price - 10000*million - 5000*five_thousand) / 1000;
   */

   million = give_price/ 10000;
   five_thousand = (give_price % 10000) / 5000;
   thousand = ((give_price % 10000) %5000) / 1000;// (change %5000)/1000

   printf("거스름돈은 천원짜리%d장 오천원짜리%d장 만원짜리%d장입니다.",thousand,
          five_thousand, million);
   return 0;
   }
```

<Day 9>

```
/**
@file   example1.c
@brief 반복문
@author Minseon Shin (minseon3@nate.com)
@date   2016-10-24
*/
```

Review

조건문: if-else, switch (break, goto)

반복문: while, do-while, for (break, countinue)

```
#include <stdio.h>
int main(void)
 {
   int a;
   printf("A");
   int b;
 }
// 변수선언은 왠만하면 코드 맨 앞에 위치하게 한다.
```

≫ while

```
while (조건)
문장A;
문장B;
```

: while도 한 문장만 carry 하기 때문에 대괄호{}가 필요하다

```
while (조건A)
{
   문장A;
   while (조건B)
   {
     문장E;
     //break;
     continue     -> continue면 문장F로 간다고 생각하는데X
                     다시 조건문으로 comeback한다. 결국 문장F는 건들지 않음.
     문장F;
    }
  문장B;
  //break;
  continue     -> continue면 문장C로 간다고 생각하는데X
                  다시 조건문으로 comeback한다. 결국 문장C는 건들지않음.
  문장C;
}
```

```
// while 예제 1
#include <stdio.h>
int main(void)
{
  int temp = 0;

  while (temp < 10)
  {
    printf("A");
    temp++;
  }

  printf("\n temp = %d\n", temp);
  return 0;
}
```

```c
// while 예제 2
#include <stdio.h>
int main(void)
{
  int temp = 0;

  while (temp++ < 10)
  {
    printf("temp = %d\n", temp);
    temp++;
  }

  printf("\n temp = %d\n", temp);
  return 0;
}
```

```c
// while 메뉴선택예제
#include <stdio.h>
int main(void)
{
  int menu = 0;

  printf("메뉴를 입력하시오. (1~2)\n");
  printf("1. coke\n");
  printf("2. sprite\n");
  while (menu != 1 && menu != 2)
  {
    printf("메뉴를 입력하시오. (1-2)");
    scanf("%d", &menu);
  }
  scanf("%d", &menu);
  return 0;
}
```

```c
// while문은 맨 처음 한번은 돌아가더라도 다음에는 무조건 조건이 만족해야 돌아감.
#include <stdio.h>
int main(void)
{
  int menu = 0;

  printf("메뉴를 입력하시오. (1~2)\n");
  printf("1. coke\n");
  printf("2. sprite\n");
  do
  {
    printf("메뉴를 입력하시오. (1-2)");
    //break;
    //continue;
    scanf("%d", &menu);
  } while (menu != 1 && menu != 2);

  return 0;
}
```

```c
//do-while 문   : 입력을 처리하는 부분에서 많이 사용. 입력을 받은 후에 while을 처리.
do
{
  문장A;
  문장B;
} while (조건);
```

```c
//for문
for(초기식;조건식;증감식) // 초기식, 조건식, 증감식 문장이 들어감 printf 가능.
                        // 세 개 다 없어도 작동 가능.
```

```c
//for문 예제: 1부터 10까지 더하기
#include <stdio.h>
int main(void)
{
  int number = -1;
  int sum = 0;

  for (number = 1; number <= 10; number++)  //compile ERROR 날수도 있지만 가능은 함.
  for (int number = 1; number <= 10; number++)
  {
    number += sum;
    //break;
    continue;
    printf("A\n");
   }
}
```

```c
//p.218 6번 예제
```

사용자로부터 일련의 데이터를 받아서 최솟값을 찾아보자. 일반적으로 최솟값은 반복을 이용하여 찾는다. 각 반복에서 현재의 최솟값과 사용자로부터 읽은 값을 비교한다. 만약 새로운 값이 현재의 최솟값보다 작으면 새로운 수가 최솟값이 된다. 여기서 데이터의 값은 음수가 될 수 없다고 가정한다. 만약 사용자가 음수를 입력하면 데이터의 입력을 끝내고 그때까지의 최솟값을 출력한다. 데이터는 10000을 넘을수 없다고 가정한다.

```
데이터를 입력하시오.(음수를 입력하면 종료) : 60
데이터를 입력하시오.(음수를 입력하면 종료) : 5
데이터를 입력하시오.(음수를 입력하면 종료) : 30
데이터를 입력하시오.(음수를 입력하면 종료) : 10
데이터를 입력하시오.(음수를 입력하면 종료) : -1
최솟값은 5입니다.
```

```c
// while 문 이용
#include <stdio.h>
#include <limits.h>
int main(void)
{
  int get_Min = 0;
  int print_Min = INT_MAX
  do
  {
    printf("데이터를 입력하시오(음수를 입력하면 종료) : ");
    scanf("%d", &get_Min);
    if (get_Min >= 0 && get_Min < print_Min) print_Min = get_Min;
  } while (get_Min >= 0);
  printf("최솟값: %d", print_Min);
}
```

```c
// if문 이용
#include <stdio.h>
#include <limits.h>
int main(void)
{
  int input = 0;
  int min;
  for (min = INT_MAX input >= 0;)
  {
    printf("데이터를 입력하시오(음수를 입력하면 종료) : ");
    scanf("%d", &min);
    if (input >= 0 && input < min) min = input;
  }
  printf("최솟값은 %d입니다.", min);
  return 0;
}
```

```
// P.218 7번
컴퓨터는 막대그래프를 그리는 데도 사용된다. 사용자로부터 1부터 50 사이의 숫자 10개를 입력받아서 숫자만큼의 별표를 출력하는 프로그램을 작성하여보자. 막대는 세로로 그려지게 된다.
막대 #1높이 : 5
*****
막대 #2높이 : 10
**********
막대 #10높이 : 20
********************
```

```c
#include <stdio.h>
int main(void)
{
  int number = 1; int input = 0;
  //for (int i = 0; i < 10; i++)
  for (; number <= 10; number++)
  {
    printf("막대 #%d의 높이:",number);
    scanf("%d", &input);
    for (; input > 0; input--)
      printf("*");
    printf("\n");
  }
  return 0;
}
```

```
// p. 218 9번
1. 연산선택(+, -, / , *) (1. + , 2. - , 3. *, 4. / , 5. Quit)
1~5 사이의 숫자가 아니면 재입력
2. 두 정수 입력받기
3. 연산 수행해서 출력
앞장에서 간단한 정수계산기를 만들어본 적이 있다. 이 계산기 프로그램에 메뉴를 추가하도록 한다. 다음과 같은 메뉴를 화면에 출력하고 사용자가 메뉴 중에서 하나를 선택할 때까지 반복을 계속한다. do...while 반복문을 사용하여 사용자가 적절한 선택을 했는지를 검사하도록 하여보자. 만약 사용자가 1부터 5사이의 정수가 아닌 다른 숫자를 입력하면 “연산을 선택하시오:” 메시지를 계속하여 출력한다. 하나의 메뉴가 선택되면 해당되는 연산을 실행하고 다시 메뉴를 선택할 수 있도록 하여보자. 반복을 종료하는 5번 메뉴는 break문을 이용하여 구현하도록 한다.
**********
1.---Add
2.---Subject
3.---Multiply
4.---Divide
5.---Quit
**********
연산을 선택하시오: 1
두 수를 공백으로 분리하여 입력하시오 : 10 20
30
```

```c
#include <stdio.h>
int main(void)
{
  int choice;
  int a, b;
  int Max=0, Min=0;
  printf("**********\n");
  printf("1.---Add\n");
  printf("2.---Subtract\n");
  printf("3.---Multiply\n");
  printf("4.---Divide\n");
  printf("5.---Quit\n");
  printf("**********\n");
  do
  {
    printf("연산을 선택하시오: ");
    scanf("%d", &choice);
  } while (choice < 1 || choice >5);


  // if (choice == 5) break;
  printf("두수를 공백으로 분리하여 입력하시오 :");
  scanf("%d %d", &a, &b);
```

```
  switch (choice)
  {
    case 1:
    printf("두 수를 공백으로 분리하여 입력하시오: ");
    scanf("%d %d", &a, &b);
    printf("%d", a + b);
    break;
    case 2:
    printf("두수를 공백으로 분리하여 입력하시오: ");
    scanf("%d %d", &a, &b);
    printf("%d", a - b);
    break;
    case 3:
    printf("두수를 공백으로 분리하여 입력하시오: ");
    scanf("%d %d", &a, &b);
    printf("%d", a * b);
    break;
    case 4:
    printf("두수를 공백으로 분리하여 입력하시오: ");
    scanf("%d %d", &a, &b);
    printf("%d", a / b);
    break;
    case 5:
    printf("끝내기");
    break;
  }
  return 0;
}
```

<Day 10>

```
/**
@file          example1_20161426.c
@brief         함수
@author        MINSEON SHIN
@date          2016-10-26
*/


fflush(stdin);  // 입력 버퍼 지움._7+enter를 누르면 enter도 입력이기에 두 번 입력된다.
                // 따라서 버퍼가 필요함. 하지만 fflush가 2016부터 지원이 안됨.

getchar(); : enter 입력 안됨. but 1byte까지는 무시하지만 그 이상은 무시할 수 없다.
// while (getchar() != '\n'); : enter가 입력이 마무리 될 때까지 인식하지 않는다.



// 함수의 기본형
반환형 함수이름(인수1, 인수2, ....)
{
  코드;
}

// 함수정의하기 예제
#include <stdio.h>
int get_max(int n1, int n2)
{
  int result;
  result = (n1 > n2) ? n1 : n2
  return result;
}

int main(void)
{
  printf("%d\n", get_max(3,4));  // get_max = 4
  return 0;
}


// main함수 뒤에 함수를 정의하고 싶다면 함수원형을 먼저 main함수 앞에 정의해야 한다.
```

```c
// 함수정의하기 예제2
#include <stdio.h>
int get_max(int, int)   // 함수원형

int main(void)
{
  printf("%d\n", get_max(3, 4));  // get_max = 4
  return 0;
}

int get_max(int n1, int n2)
{
  int result;
  result = (n1 > n2) ? n1 : n2
  return result;
}
// 이렇게 쓰지만 보통은 헤더파일을 추가한다.
```

```c
// 함수정의하기 예제3 _ 헤더파일 추가하기
#include <stdio.h>
#include "example_1.h"

int main(void)
{
  printf("%d\n", get_max(3, 4));  // get_max = 4
  return 0;
}
```

example_1.h

```c
#pragma once
static int get_max(int n1, int n2)
{
  int result;
  result = (n1 > n2) ? n1 : n2
  return result;
}
```

만약 c코드소스에 get_max를 함수를 구현해놓고 헤더파일을 만들어 함수원형을 정의해 놓아본 Main 코드에 ".h" 헤더 참조하면 실행 가능.

```c
// 함수정의하기 예제4_1부터 합계
#include <stdio.h>

int computer_sum(int n)
{
  int result = 0;
  for (; n > 0; n--)
  result += n
  return result;
}

int main(void)
{
  printf("%d\n", computer_sum(10));
  return 0;
}
```

```c
// 변수_전역변수, 지역변수
#include <stdio.h>

int result = 10;       // 쓸 수 있을까? 가능하다.
                         main함수 밖에서 정의한 변수를 전역변수(global variable)라고 한다.
                       // 즉, 저 computer_sum 안에 있는 지역변수는 저 함수 속에서만 적용.
                       // 전역변수는 1. 메모리를 쓸데없이 잡아먹음. 2. 가독성이 좋지않다.
                       // 그래서 쓸 때에는 전역변수와 지역변수와 이름을 다르게 한다.
                          ex. gi_result or _result
                       // 지역변수> 전역변수(우선 순위 높음)
int computer_sum(int n)
{
  int result = 0;           // 지역변수(local variable)
  for (; n > 0; n--)
  result += n
  return result;
}
int main(void)
{
  printf("%d\n", computer_sum(10));
  return 0;
}
```

```c
// factorial function (팩토리얼 함수)
#include <stdio.h>
int factorial(int n)
{
  int result=1;
  for (; n>0; n++) result *= n
  return result;
}
int main()
{
  printf("%d\n", factorial(3));
  return 0;
}
```

≫저장유형자 : auto, static, register, extern

· auto : 변수를 자동으로 메모리에 할당./삭제
· static : 변수를 메모리에 남김, 초기화는 한번만 수행
　　함수는 다른 코드에서 extern으로 사용할 수 없게 함.
· register : 변수를 레지스터에 할당/삭제
· extern : 해당변수/함수가 다른 코드에 있음

```
// 저장유형자 예제
#include <stdio.h>

extern int get_max(int, int);   // 변수도 할 수 있고, 함수도 할 수 있다.
extern int link;
int result = 10;                // 전역변수 : 별도로 초기화하지 않으면 자동 0

int computer_sum(int n)
{
  auto int result = 0;
  static int enc = 0;   // 지역변수지만 영역을 벗어나도 메모리(stack)에서 사라지지 않는다.
                           하지만 함수 밖에서 쓸 수 있다는 말은 아니다.
                        // 계속 업데이트하게 하려면 씀.
                        // static 변수: 메모리에서 사라지지 않음. 변수선언
                        // 한번 읽으면 다시 읽지 않는다.
  register int abc = 0;  // register 할당하는 변수.
                          // 접근이 빠른 register에 저장하시게~ (빈번사용)
  레지스터는 CPU에 존재하는 매우 작은 메모리, 하지만 cpu에 있어 연산이 매우 빠름. 하지만 판단은 컴파일러가...

  printf("enc = %d\n", ++enc);  // 만약에 한번 거쳐셔 enc가 1이 되면 두 번째 불렀을 때
                                   메모리에 이미 enc가 있기에 선언을 생략하고 ++enc한다.
  for (; n > 0; n--) result += n
  return result;
}
int main(void)
{
  printf("%d\n", computer_sum(10));    //1
  printf("%d\n", computer_sum(5));     //2
  printf("link = %d\n", link);          //2
  return 0;
}
```

```c
//static example
#include <stdio.h>

void simpleFunc(void)
{
  static int num1 = 0;
  int num2=0
  num1++, num2++;
  printf("static: %d, local : %d \n", num1, num2);
}

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

// result
static : 1, local : 1
static : 2, local : 1
static : 3, local : 1
```

≫ 재귀함수(recursive function) : 순환호출(recursive call)
: 자기 자신을 호출할 수 있음(알고리즘에 사용)

1. 종료조건
2. recursive call(순환호출)

```
// factorial 재귀함수 예제
#include <stdio.h>
int factorial(int n)
{
  if (n == 1) return 1;
  return n * factorial(n - 1);
}
int main(void)
{
  printf("%d\n", factorial(3));
  return 0;
}
```

≫피보나치수열

$a_0 = 0,\ a_1 = 1,\ a_n = a_{n-1} + a_{n-2}$

e.g = 0, 1, 1, 2, 3, 5, 8, 13.....

```
// 항의 수를 입력 받아서 피보나치 수열 출력
#include <stdio.h>
int Fibonachi(int n)
{
  if (n == 0) return 0;
  if (n == 1) return 1;                          // 종료조건
  return Fibonachi(n - 1) + Fibonachi(n - 2);    // reculsive call(순환호출)
}

int main(void)
{
  int Hang = 0;
  printf("항의 수를 입력하시오: ");
  scanf("%d", &Hang);
  for (; Hang >= 0; Hang--)
  printf("%d ,", Fibonachi(Hang));

  return 0;
}
```

```c
// 항의 수를 입력 받아서 피보나치 수열 출력_열혈강의
#include <stdio.h>
void Fibonachi(int num)
{
  int f1=0, f2=1, f3, i;
  if (num==1)
      printf("%d ", f1);
  else
      printf("%d %d ", f1, f2);

  for (i=0; i<num-2; i++)
      {
         f3=f1+f2;
         printf("%d ",f3);
         f1=f2;
         f2=f3;
      }
}

int main(void)
{
  int n;
  printf("출력하고자 하는 피보나치 수열의 개수 : ");
  scanf("%d", &n);
  if (n<1)
  {
    printf("1 이상의 값을 입력하시오. \n");
    return -1;
  }
  Fibonachi(n);
  return 0;
}
```

<Day 11>

```c
/**
@file   example1.c
@brief 복습
@author Minseon Shin (minseon3@nate.com)
@date   2016-10-31
*/

//Fibonachi 수열
#include <stdio.h>
int Fib(int n)
{
  if (n == 0) return 0;
  if (n == 1) return 1;
  return Fib(n - 1) + Fib(n - 2);
}
int main(void)
{
  int n;
  printf("피보나치 수열의 개수를 입력하시오: ");
  scanf("%d", &n);
  for (int i = 0; i<n; i++) printf("%d, ", Fib(i));
  return 0;
}

// 반복문
#include <stdio.h>
int main(void)
{
  int n;
  int a = 0, b = 1, c;
  printf("피보나치 수열의 개수를 입력하시오: ");
  scanf("%d", &n);
  for (int i = 0; i < n; i++)
  { if (i <= 1) printf("%d, ", i);
    c = a + b;
    printf("%d, ", c);
    a = b,  b = c;
  }
  return 0;
}
```

```c
// 십진수로 입력 받아 이진수로 표현하시오._ 재귀함수
#include <stdio.h>
int dec2bin(const int n)    // const 함수 : n이 변하지 않는다.  예) n=n+1
{                             int c;
  if (n / 2 == 0)             c = n+1;
  {                           c = n+2; // c가 두 번 덧입혀져 결국 n+2가 되므로.
  printf("%d", n % 2);                 => 컴파일러의 최적화
  return;
  }
  dec2bin(n / 2);
  printf("%d", n % 2);
  return;
}

int main(void)
{
  int n = 0;
  scanf("%d", &n);
  dec2bin(n);
}


// 십진수로 입력 받아 이진수로 표현하시오._ 반복문
#include <stdio.h>
int main(void)
{
  int n = 0;
  int result = 0;
  int digit = 10;

  scanf("%d", &n);
  do
  {
    result += digit*(n % 2);
    n = n / 2;
    digit *= 10;
  } while (n);
  printf("result = %d\n", result);

  return 0;
}
```

```c
//랜덤 rand()
#include <stdio.h>
#include <stdlib.h>    //rand : 0~32767
#include <time.h>
int main(void)
{
  srand((unsigned)time(NULL));
  for (int i = 0; i < 10; i++)  printf("%d ",rand());

  return 0;
}
```

rand()%3 : 이렇게 하면 경우의 수가0~2밖에 안 나온다.
따라서 가위바위보에서 랜덤 난수 생성하려면 이렇게 변환한다.
10~15까지로하고싶으면: (rand()%6)+10

```c
// rand() size 검사
#include <stdio.h>
#include <stdlib.h>    //rand : 0~32767
#include <time.h>
int main(void)
{
  printf("%d", RAND_MAX);
}
```

```c
// p. 226 12번문제
   1^3 + 2^3 + ... + n^3
#include <stdio.h>
#include <math.h>
int main(void)
{
  int n = 0;
  int sum = 0;
  scanf("%d", &n);
  for (int i = 0; i <= n; i++) sum += pow(i, 3);        // pow 지수함수
  return 0;
}
```

```c
// p. 226 12번문제_ 재귀함수
// 근데 굳이 재귀함수 쓸 필요가 없다.
#include <stdio.h>
#include <math.h>
int func(int n)
{
  if (n == 1) return pow(1,3);
  return pow(n, 3) + func(n - 1);
}
int main(void)
{
  int n = 0;
  scanf("%d", &n);
  printf("%d\n", func(n));
  return 0;
}
```

<Day 12>

```
/*
@file          20161102.c
@brief         배열(array)
@author        minseon shin
@date          2016-11-02
*/
```

≫ array, index

자료형 배열이름[개수]; //!= abc()

메모리할당(memory allocation)

≫ 초기화

1. int grade[3] = {80, 75, 99};

   int grade[3] = {80, 75};

   // grade[0]=80, grade[1]=75, grade[2]=? (초기화 하지 않으면 0이 된다.)

   int grade[3]={0} // grade[0]=0, grade[1] & grade[2] = 0

   int grade[3] = {0, }; 한칸 띄어 놓기도 함. compile error 안남.

   int grade[STUDENT_NUMBER] = {0, }; 물론 #define STUDENT_NUMBER 정의해야함.

```
// 초기화_1 예
#include <stdio.h>
int main()
{
  int a = 0;          // memory 상에 들어감 loader가 할당함.

  //double score 1, score2, score3,....score38;
  double score[38];    // 반환형 배열 [배열의 개수(index)]
                       // score[0],....score[37] : 0부터시작
  return 0;
}
```

2. int grade[]={80, 75}; // 안될 것 같지만 compiler 가알아들음.

   // grade[0]=80, grade[1]=75

```
   3. memset(memory set)을 이용함.(<memory.h> include 해야함.) <메모리 조작>
                                  // string.h 에 원래 memset이 있었는데 바뀜.(둘다 가능)
      memset(시작주소, 초기화 값(바꾸고자 하는 수), 크기);
      memset(grade, 0, sizeof(grade));
                                 //grade를 0으로 초기화하고 사이즈는 sizeof(grade)만큼!
      // but. 1로 초기화가 안됨-> 4byte : 16bit니까
         0x01 0x01 0x01 0x01 => 0x01010101 =1? No
      // memset(grade, -1, sizeof(grade));
          => 1111 = -8+4+2+1 = -1 (2의 보수로 컴퓨터는 인식함)
      // memset은 결국0, -1밖에 초기화가 안됨.


// 배열을 출력하자.
#include <stdio.h>
int main()
{
  double score[38];
  printf("%p\n", score);              // 원소전체출력? No 배열의 시작주소가 출력
                                        예) 0xFFA0 F004
  scanf("%lf", &score[1]);            // score 배열의 2번째에 입력받기.
  scanf("%lf", score);                // 가능 Yes! score는 주소이기 때문에 &없는 상태로
                                         가능하다. = score[0]에 입력받기
                                      // = scanf("%lf", &score[0]);
  return 0;
}

// 배열이름 = 첫 번째 원소의 주소
// 예: int array[10];     => array 와&array[0]와 같다.


// 배열이 같다
#include <stdio.h>
int main()
{
  int array1[3] = { 0,1,2 };    // array는 이미 함수명으로 있기에 적절하지 않은 변수명이다.
  int array2[3] = { 0,1,2 };
  if (array1 == array2) printf("같다!\n");
  return 0;
}
// 배열의 주소가 같다? = 할당 받는 주소가 같을 수가 없기에 같다고 나올 수가 없다.
// 굳이 배열주소를 따지려면 일일이 주소를 대응해야 함
```

```c
#include <stdio.h>
int main()
{
  int array1[3] = { 0,1,2 };
  int array2[3] = { 0,1,2 };

  for(int i=0;i<3;i++)
    if (array1 == array2) printf("같다!\n");

  return 0;
}

// 배열의 원소의 개수
#include <stdio.h>
int main()
{
  int temp[] = { 90, 85, 95, 45, 50 };
  for (int i = 0; i < sizeof(temp) / sizeof(temp[0]); i++)
    printf("%d ", temp[i]);
  return 0;
}
```

```c
// 예제
// 평균, 분산, 표준편차구하기(배열이용)_ 5개의 정수를 입력 받은 후
#include <stdio.h>
#include <math.h>
int main()
{
  double avg = 0, var = 0, standard = 0;
  double number[5] = { 0, };
  printf("점수를 5개 입력하시오");
   scanf("%lf %lf %lf %lf %lf", &number[0], &number[1],
                                          &number[2], &number[3], &number[4]);

  for (int i = 0; i < 5; i++)
  avg += number[i];
  avg = avg / 5;

  for (int j = 0; j < 5; j++)
  var = var + (number[j] - avg)*(number[j] - avg);
  var = var / 5;

  standard = sqrt(var);

  printf("평균:%lf 분산:%lf  표준편차: %lf", avg, var, standard);

  return 0;
}
```

```c
// 함수 버전
// 평균, 분산, 표준편차구하기(배열이용)_ 5개의 정수를 입력받은 후
#include <stdio.h>
#include <math.h>
void input_number(double number[], int size);
double cal_avg(double number[], int size);
double cal_var(double number[], double avg, int size);

int main()
{
  double avg = 0, var = 0, standard = 0;
  double number[5] = { 0, };
  input_number(number, 5);

  printf("평균:%lf 분산:%lf  표준편차: %lf",
    avg = cal_avg(number, 5), var = cal_var(number, avg, 5), sqrt(var));

  return 0;
}
void input_number(double number[], int size)
{
  printf("점수를 5개 입력하시오 : ");
  for (int i = 0; i < 5; i++)
  scanf("%lf", &number[i]);
}
double cal_avg(double number[], int size)
{
  double avg = 0;
  for (int i = 0; i < 5; i++)
  avg += number[i];
  avg = avg / 5;
  return avg;
}
double cal_var(double number[], double avg, int size)
{
  double var = 0;
  for (int j = 0; j < 5; j++)
  var = var + (number[j] - avg)*(number[j] - avg);
  var = var / 5;
  return var;
}
```

c.f) Quiz 5.(a)

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

int main()
{
  printf("%d", sum(5));
  return 0;
}
int sum(int n)
{
  printf("%d\n", n);
  if (n < 1) return 1;
  else return (n + sum(n - 1));
}
```

출력 값 : 5, 4, 3, 2, 1, 16

c.f) Quiz 3.

(a) 반복문을 이용하여 다음의 값을 출력하는 코드를 작성하여라. 9, 6, 3, 0

(1) #include <stdio.h> => cpu가 훨씬 덜 들어가 빨리 재생됨. 시간 절약가능. (좋은 코드)

```
int main()
{
  int i = 9;
  while (i >= 0)
  {
    printf("%d, \n", i);
    i -= 3;
  }  return 0;
}
```

(2) #include <stdio.h>

```
int main()
{
  int x = 9;
  while (x >= 0)
  {
    if (x % 3 == 0)
      printf("%d, ", x);
    x--;
  }  return 0;
}
```

<Day 13>

```
/*
@file   20161107.c
@brief  배열 review, 다차원배열
@author minseon shin (minseon3@nate.com)
@date   2016-11-07
*/


// 배열 review_1
#include <stdio.h>
int main()
{
        int temp[5] = { 1,2,3,4,5 };
        // temp[0] ={1}, temp[1]={2},.....
        // temp [5] 불가능
                -> 공간을 할당하지 않았는데 값을 넣으면 bluescreen 뜰 수도 잇음
           (만약 저장되있는 메모리에 중요한게 들어가잇엇을때)

        printf("%d\n",temp);  // temp= index. 즉 주소값이 출력됨. 7600996
        printf("%d\n", temp + 1); // 7601000 즉 +1당 4byte 추가(자료형 : int)

        int temp1[] = { 1,2,3,4,5 }; // 배열의 크기 비워놀 수 있다.
}

// 배열 review_2
#include <stdio.h>
int main()
{
        int number = 5;
        int temp[number] = { 1,2,3,4,5 };
        // -> 불가능.(변수를 넣으면 값이 바뀔수 있기 때문에...(동적 할당))

        int a; char b; double c; float d;
        // 정적 할당(static allocation) <-> 동적 할당(dynamic allocation)
}
```

```c
// 배열 review_3
#include <stdio.h>
#define NUMBER 5
const int NUMBER 5

int main()
{
	int temp[NUMBER] = { 1,2,3,4,5 };	// -> 가능.(const, #define 가능)
}

// 배열 review_4
#include <stdio.h>
int main()
{
	int temp[5] = { 1,2,3,4,5 };

	printf("%d\n", temp);
	printf("%d\n", &temp[0]);
	printf("%d\n", temp+1);
	printf("%d\n", &temp[1]);	// 주소 (&) <-> 주소가 가리키는 값(*)
	printf("%d\n", temp[0]);	//1
	printf("%d\n", *temp);		// temp는 주소인데 주소가 가리키는 값 *temp
	//1

	printf("%d\n", temp[2]);	//3
	printf("%d\n", *(temp + 2));	//3
	// if *temp+2 (x) : *는 단항연산자여서 제일 세다.
}

#include <stdio.h>
int main()
{
	int A[3][3] = { {1,2,3},
			{4,5,6},
			{7,8,9} };
	int B[3][3] = { 9,8,7,6,5,4,3,2,1 };
	// int A와B 처럼 해도 됨. 순서대로 빈 공간에 들어감.
	int C[][3] = { 9,8,7,6,5,4,3,2,1 }; // 3개씩 들어가면 열이 3임을 아므로 .
	int D[3][]; // 불가능
	int F[4][3][5]; // 60개가 들어감.
	return 0;}
```

```c
// A와 B 두 행렬을 더하자
#include <stdio.h>
#define ROW (3)
#define COL (3)
int main()
{
	int A[ROW][COL] = { { 1,2,3 },
	{ 4,5,6 },
	{ 7,8,9 } };

	int B[ROW][COL] = { 9,8,7,6,5,4,3,2,1 };
	int C[ROW][COL] = { 0, };		// 전부 0으로 초기화

	for (int i = 0; i < ROW; i++)
	for (int j = 0; j < COL; j++)
	C[i][j] = A[i][j] + B[i][j];

	// 삼차원 배열시 k를 이용한 for문을 더 넣자.
	for (int i = 0; i < ROW; i++)
	{
	for (int j = 0; j < COL; j++)
	printf("%d ", C[i][j]);
	printf("\n");		// i 반복문에만 걸려야 행이 끝날 때 마다 띄어씀.
	}
	//예제
	printf("\n");
	printf("%d\n", *A);		// 출력값 : 1
	printf("%d\n", *(A + 3));	// 출력값 : 4

	printf("%d\n", *(&A[0][0]));
	printf("%d\n", *(&A[1][0]));  // 이렇게도 쓸수 있음. 결과는 위와 동일.
	return 0;
}
```

≫ 기본적인 알고리즘 (Algorithm)
정렬(sorting): 선택정렬(selection sort),
7,11,2,8,1,11 -> 1,2,7,8,11,11 (선택정렬 예)
bubble sort, insert sort, merge sort,
quick sort, heap sort
탐색(search)

```c
// bubble sort 예제(버블정렬)
#include <stdio.h>
#define NUMBER 8
int main()
{
	int A[NUMBER] = { 2,7,11,9,1,13,4,8 };
	int temp;
	for (int j = NUMBER; j >1; j--)// j>1? 밑에 ifor 조건문에서 한번은 돔 1>1
	for (int i = 0; i <j - 1; i++)
	// 8이면 7번만 비교하면 되므로... for(int i=1;i<NUMBER;i++)
	if (A[i] > A[i + 1])  //swap(자리 바꿈)
	{
	temp = A[i];
	A[i] = A[i + 1];
	A[i + 1] = temp;
	}
	for (int i = 0; i < NUMBER; i++)
	printf("%d ", A[i]);
}
```

<Day 13>

```
/*
@file          20161109.c
@brief         배열 review
@author        minseon shin (minseon3@nate.com)
@date          2016-11-09
*/

// 다차원 배열 - 주소개념
#include <stdio.h>
int main()
{
        int A[3][3] = { { 1,2,3 },
        { 4,5,6 },
        { 7,8,9 } };

        int B[3] = { 11,12,13 };
        printf("%d\n", B);    // &B[0]
        printf("%d\n", *B);   // B[0]
        printf("%d\n", A);    // &A[0], A[0] ={1,2,3}
        printf("%d\n", *A);   // &A[0][0]
        printf("%d\n", **A);  // A[0][0] = {1}
        printf("%d\n", **(A + 1));          // A[1][0] = {4}
        printf("%d\n", *(*(A + 1) + 1));    // A[1][1] = {5}
        return 0;
}
```

≫ 정렬 알고리즘

선택 정렬(selection sort), 버블정렬(bubble sort)

삽입 정렬(insertion sort), 퀵 정렬(quick sort)

```c
// Examination of selection sort(정답)
#include <stdio.h>
#define SIZE 7
int main()
{
	int A[SIZE] = { 7,4,2,8,1,13,6 };
	int least;	//최솟값이 들어있는 곳의 index
	int temp;
	for (int i = 0; i < SIZE - 1; i++)	//0~5 : 뒤지는 시작 위치
	{
	least = i;	// 초기 최솟값이 들어 있는 곳의 index 초기화

	//최솟값이 들어 있는 index(least) 찾기
	for (int j = i; j < SIZE; j++)
	if (A[j] < A[least]) least = j;
	//swap
	temp = A[i];
	A[i] = A[least];
	A[least] = temp;

	for (int k = 0; k < 7; k++)
	printf("%d ", A[k]);
	printf("\n");
	}

	return 0;
}

//탐색 알고리즘 : 순차탐색(sequential search), 이진탐색(binary search)
```

```c
// Examination of selection sort_mine(수정해야함.) 오류뜸.
#include <stdio.h>
#define SIZE   7
int main()
{
        int A[SIZE] = { 7,4,2,8,1,13,6 };
        int min = 0, temp;
        for (int i = 0; i < 5; i++)
        {
        for (int j = i; j < 6; j++)
        {
        if (A[i] > A[j])
        min = A[j];
        temp = A[i];
        A[j] = temp;
        A[i] = min;
        }

        for (int k = 0; k < 7; k++)
        printf("%d ", A[k]);
        printf("\n");
        }

        return 0;
}
```

<DAY 14>

```
/**
@file   20161114.c
@brief  포인터
@author minseon shin
@date   2016-11-14
*/
```

포인터
정의 : 변수의 주소가 가지고 있는 변수
e.g. int *p;   //*: asterisk, 역참조연산자, 포인터
int형 포인터 p
char *q char 형 포인터 q

```
int a = 7;   정수형 변수 a
int b[7] = {0, };  배열 b
int c();     함수 c
&a
```

```
// 예1
#include <stdio.h>
int main()
{
  int a = 7;
  int *p = NULL        // point 초기화는 NULL]
  double *q = NULL
  p = &a;
  a = 3;
  *p = 3;              // 둘 다 똑같은 얘기 a=3

  printf("a의 주소 : %p\n", &a);
  printf("p가 가르키는 곳의 값 : %d\n", *p);
  printf("p : %p\n", p);
  printf("p의 주소 : %p\n", &p);
  printf("포인터 p의 크기 : %d byte\n", sizeof(p));            // 4 byte
  printf("포인터 q의 크기 : %d byte\n", sizeof(q));            // 4 byte
            = 가르키는 포인터의 자료형과는 달리 주소가 들어있기 때문에 자료형과는 별개로 크기는 동일.
  return 0;
}
```

```
// 포인터를 선언할 때 2가지.
1)
int *p = NULL
p = &a;

2)
int *p = &a;            // 하지만 예외적인 상황이므로 1번을 이용하자.



d = a*b*c;  왼쪽->오른쪽
a = b = c;  오른쪽 ->왼쪽 ( 대입/단항 연산자 )
*p++;       p 주소 값을 ++
++ *p;      p가 가리키는 곳의 값을 ++
*++p;       p 주소값을 ++ 후 가리킴.....        // p+1이 가리키는 곳의 값.



// 예2
#include <stdio.h>
int main()
{
  int *p = NULL
  int a[3] = { 1,3,5 };
  p = a;                                  // 배열에서 a는 a라는 배열의 주소이므로... & 안씀

  printf("*p++ = %d\n", *p++);            // ++이 후위 연산자여서 *p만 실행.
  printf("*p = %d\n", *p);
  printf("++*p = %d\n", ++*p);
  return 0;
}



int* p = NULL 실제로 읽는 건 이렇게지만 사용 NO
int *p = NULL 표준형

int *p;
char b[7];
double add();
```

<포인터, 배열, 함수 읽는 법>

1. * 역참조연산자.
- 지금까지 읽은 것(의 타입)은 주소다.
- 앞으로 읽을 것은 그 주소가 가리키는 대상의 타입

2. [] 배열연산자
- 지금까지 읽은 것은 배열이다.
- 앞으로 읽을 것은 배열의 원소 하나의 타입이다.

3. () 함수연산자
- 지금까지 읽은 것은 함수구나.
- 앞으로 읽을 것은 함수의 반환형이다.

```
// 포인터 읽는 법

char *(*a)[10];         // 이름은 a
                        // a의 타입은 주소
                        // 그 주소가 가리키는 것은 배열
                        // 배열의 원소는 주소다.
                        // 배열의 원소인 주소가 가리키는 것은 char형이다.


int (*b[5])(int c);     // 이름은 b
                        // b는 배열이다
                        // 배열의 원소는 주소다
                        // 그 주소가 가리키는 것은 함수다
                        // 그 함수의 반환형은 int다.


char(*encrypt(char(*)[10]))[10];     // 이름은 encrypt다
                                     // encrypt는 함수다
                                     // 이 함수의 반환형은 주소이다.
                                     // 그 주소가 가리키는 것은 배열이다.
                                     // 배열의 원소는 char이다.
```

```c
// 예3
#include <stdio.h>
int main()
{
  char *p = NULL
  *p = 1000;
                        // 주소를 정확하게 넣어준 다음 값을 넣어야 한다.
                        // 할당하지 않은 값에다가 넣으면 compile ERROR 가 난다.
  return 0;
}
```

```c
// 예4
#include <stdio.h>
int main()
{
  double *p = NULL
  double a = 'A'
  p = &a;

  printf("%p\n",p);
  printf("%p\n",++p); // 값은 ++ 하면 1이 증가하지만 결국 8만큼 주소 증가함.
  return 0;
}
```

```c
// 예5
#include <stdio.h>
int main()
{
  const int *p;// int가 constant
  int* const p;// 변하지 않는 값.. 주소. p가 constant

  return 0;
}
```

```c
// 예5_평균
#include <stdio.h>
int getAverage(int arr[], int cnt)
                          // 배열을 다 copy하는 것처럼 보이지만 주소만 넘겨주는 것이다.
                             그러므로 포인터랑 별반 다를게 없음.
{
  int sum = 0;
  for (int i = 0; i < cnt i++) sum += arr[i];
  return sum / cnt
}
int main()
{
  int score[5] = { 70, 80, 90, 85, 75 };

  printf("average : %d\n", getAverage(score, 5));
  return 0;
}
```

```c
// 예5_평균_point
#include <stdio.h>
int getAverage(int* arr, int cnt)//arr에 주소를 넣어야함.
{
  int sum = 0;
  for (int i = 0; i < cnt i++) sum += *(arr+i);
  return sum / cnt;
}
int main()
{
  int score[5] = { 70, 80, 90, 85, 75 };

  printf("average : %d\n", getAverage(score, 5));
  return 0;
}
```

```c
// 예6_포인터를 매개변수로 받는 getMin, getMax 함수 구현
#include <stdio.h>
int getMin(int* arr, int cnt)
{
  int Min = *arr
  for (int i = 0; i < cnt i++)
    if (Min > *(arr + i))
      Min = *(arr + i);
  return Min;
}
int getMax(int* arr, int cnt)
{
  int max = *arr
  for (int i = 0; i < cnt i++)
    if (max < *(arr + i))
      max = *(arr + i);
  return max;
}
int main()
{
  int score[5] = { 1,3,5,7,10 };

  printf("average : %d\n", getMin(score, 5));
  printf("average : %d\n", getMax(score, 5));
  return 0;
}
```

<DAY 15>

```
/*
@file          161116.c
@brief         pointer 활용, 기본 file 입출력
@author        Minseon shin
@date          2016-11-16
*/


// 지난 시간 복습
#include <stdio.h>
int getMin(int arr[], int cnt);

int main(void)
{
  int temp[5] = { 1, 3, 5, 7, 4 };
  printf("min = %d\n", getMin(temp, 5));
  return 0;
}
int getMin(int arr[], int cnt)
{
  int min = arr[0];
  for (int i = 0; i < cnt i++)
    if (min > arr[i])
      min = arr[i];
  return min;
}




// call by value (값에 의한 호출)

int sum(int a, int b)
{}
int main(void)
{
  sum(5, 7);
  return 0;
}
```

```
// 값을 넣어서 호출함.

//예2
#include <stdio.h>
int sum(int a, int b)
{
  a = 7;
  return 0;
}
int main(void)
{
  int a = 5, b = 7;
  sum(a, b);
  printf("a=%d\n", a);// 출력 : 5
  return 0;
}

// call by refernece (참조에 의한 호출) :1. 여러 개의 함수 수행 결과를 받을 때
                                         2. 많은 데이터를 주고 받을때, 메모리/연산 시간을 절약
// 포인터를 이용함.
```

```
#include <stdio.h>
int sum(int *a, int b)
{
  *a = 7;
  return 0;
}
int main(void)
{
  int a = 5, b = 7;
  sum(&a, b);
  printf("a=%d\n", a);        // 출력 : 7
                              // *a a가 가리키는 곳의 값 = a 주소가 가리키는 값을  =7
  return 0;
}
```

```c
//ex_call by value
#include <stdio.h>
int swap(int n1, int n2)
{
  int temp = 0;
  temp = n1
  n1 = n2
  n2 = temp;
  return 0;
}
int main(void)
{
  int a = 5, b = 7;
  swap(a, b);
  printf("a= %d, b=%d", a, b);
  return 0;
}
// compile 결과 swap 되어야 하지만 되지 않는다. 따라서 call by reference 를 써야함.
```

```c
//ex_call by reference
#include <stdio.h>
int swap(int* n1, int* n2)
{
  int temp = 0;
  temp = *n1
  *n1 = *n2
  *n2 = temp;
  return 0;
}
int main(void)
{
  int a = 5, b = 7;
  swap(&a, &b);
  printf("a= %d, b=%d", a, b);// a:7, b:5
  return 0;
}
```

```c
// 이용하는 곳 :
#include <stdio.h>
int sum(int a, int b)
  {return a + b}
int calAddSub(int n1, int n2, int *a_Result, int *s_Result);
int main(void)
{
  int a = 7, b = 5;
  int addResult = 0, subResult = 0;
  sum(a, b);
  calAddSub(a, b, &addResult, &subResult);
              // 함수 하나당 return 값을 여러개 갖고 싶을때. call by reference 를 쓴다.
  printf("add = %d, sub = %d\n", addResult, subResult);
  return 0;
}
int calAddSub(int n1, int n2, int *a_Result, int *s_Result)
{
  *a_Result = n1 + n2
  *s_Result = n1 - n2
  return 0;
}
```

```c
// 과제
// file operation : fopen, fclose, fscanf, fprintf
#include <stdio.h>
int main(void)
{
  FILE* fp;
  fp = fopen("sample_input.txt", "r");// :r": read

  if (fp == NULL)
  {
    printf("fopen error\n");
    return -1;
  }

  char temp;
  fscanf(fp, "%c", &temp);
  printf("%c\n", temp);
  fscanf(fp, "%c", &temp);
  printf("%c\n", temp);
  fscanf(fp, "%c", &temp);
  printf("%c\n", temp);

  fclose(fp);
  return 0;
}
```

```c
// 함수 포인터
#include <stdio.h>
int add(int a, int b)
{
  return a + b
}
int sub(int a, int b)
{
  return a - b
}
int mul(int a, int b)
{
  return a*b
}

int main(void)
{
  int(*fp)(int, int);         //  함수 포인터
                              //*fp(); 하면 fp() 하고 *하기에 함수포인터가 아니다.
  fp = add;
  printf("%p\n", add);                // 주소가 찍힌다.
  printf("%d\n", (*fp)(7, 5));                // 7+5
  printf("%d\n", fp(7, 5));           // 예외적이지만 위에가 제일 정석!

  return 0;
}



// 포인터 배열
#include <stdio.h>

int main(void)
{
  double a = 0.3, b = 0.7, c = -0.1;
  double *arr[] = { &a, &b, &c };

  printf("%lf\n", *arr[0]);

  return 0;
}
```

```c
// 함수 포인터 배열
#include <stdio.h>
int add(int a, int b)
{
  return a + b
}
int sub(int a, int b)
{
  return a - b
}
int mul(int a, int b)
{
  return a*b
}

int main(void)
{
  double a = 0.3, b = 0.7, c = -0.1;
  int (*arr[])(int , int ) = { add, sub, mul };

  printf("%d\n", (*arr[0])(1,3));

  return 0;
}
```

<Day 16>

```c
/**
	@file   20161121.c
	@brief 이중포인터, void 포인터
	@author Minseon
	@date   2016-11-21
*/

// 이중포인터 : int **p, char** p(p는 주소다, char포인터를 가리키는 주소다.

// 예제1
#include <stdio.h>
int main(void)
{
	int a = 100;
	int *p = NULL; // 항상 초기화 해야함.
	p = &a;
	// int *p = &a;

	*p = 200;	// a값을 200으로 바꾸고자 함. 단, a를 직접 건들이지 않고!
	printf("a= %d\n", a);
	return 0;
}

// 예제2 _ 이중포인터 **q
#include <stdio.h>
int main(void)
{
	int a = 100;
	int *p = NULL;
	int **q = NULL;		// (int*) *q  ==  int **q
	p = &a;
	// int *p = &a;
	q = &p;
	// int **q = &p;
	*p = 200;
	printf("a= %d\n", a);
	**q = 300;
	printf("a= %d\n", a); // a값을 300으로 바꾸고자 함. 단, a,p를 직접 건들이지 않고!
	return 0;
}
```

```c
// void 포인터
#include <stdio.h>
int add(int, int);
int main(void)
{
	double a = 0.1;
	int *p = NULL;
	//p = &a;	// ERROR
	(double*)p = &a;

	int b[5] = { 1,3,5,7,9 };
	void *vp = NULL;	// 뭘 가르킬지 지정X == 뭐든지 가리킬 수 있음.
	vp = b;
	vp = &b[2];

	*vp = 3;	// ERROR : 뭘 넣을지 지정하지 않은 void이기에.
	*(int*)vp = 3;

	vp++;	// ERROR : ++는 자료형에 따라 크기를 결정해 +byte만큼 커지므로.
	((int*)vp)++;

	// int (*vp)(int, int) = add;
	void *vp = NULL;
	void *vpf(int, int);	// 함수 콜 가능.

	return 0;
}
int add(int n1, int n2)
{
	return n1 + n2;
}
```

```c
// p.343, 4번
#include <stdio.h>
int add_ISBN(int ar[])
{
	int sum = 0;
	for (int k = 0; k < 13; k++)
	sum += ar[k];
	return sum;
}
int ISBN_check(int ar[])
{
	int ISBN_sum = 0;
	for (int j = 0; j < 13; j++)
	if (j % 2 != 0)
	*(ar + j) = (*(ar + j)) * 3;

	ISBN_sum = add_ISBN(ar);
	if (ISBN_sum % 10 == 0) return 1;
	else return 0;
}

int main(void)
{
	int ISBN[13] = { 0, };
	printf("ISBN 번호를 입력하시오: ");
	for (int i = 0; i < 13; i++)
	scanf("%d", &ISBN[i]);
	if (ISBN_check(ISBN)) printf("유효한 ISBN 번호입니다.\n");
	else printf("유효하지 않은 ISBN 번호입니다.\n");
	return 0;
}
```

<Day 17>

```
/**
        @file   20161123.c
        @brief  문자(character)와 문자열(string)
        @author minseon shin
        @date   2016-11-23
*/

// 문자
#include <stdio.h>
int main()
{
        char temp = NULL;
        temp = 'A';
        temp = 65;      // 위와 동일.
        char *p = NULL;
        p = &temp;

        printf("%c\n", temp);
        printf("%c\n", *p);
        printf("%d\n", *p);     // 문자를 d로 읽으면.. 숫자 65.
        return 0;
}

// 문자열_DJU를 만들자.
#include <stdio.h>
int main()
{
        // 4byte를 넣어야 하는 이유 : 뒤에 \0 값도 넣어줘야 하므로 +1해야함.

        char ar[4] = { 'D','J','U' };  // { 'D','J','U',\0 };
        //or char ar[4] = "DJU" (단, 문자열만 가능)

        for (int i = 0; i<4; i++)
        printf("%c", ar[i]);    // 뒤에 변수가 옴.
        printf("\n%s\n", ar);  // 문자열을 출력할때는(%s) 뒤에 (,배열을 출력하므로),
                               배열의 시작주소를 넣는다.
        // 예외.
        printf("%s\n", &ar[0]);
        // 만약 JU만 출력하고 싶으면 ar+1을 넣으면 된다. 원하는 부분의 시작주소를 넣자.
        printf("%s\n", ar + 1);
        return 0;
}
```

```c
// 문자열_DJU를 만들자.(파일저장)
#include <stdio.h>
int main()
{
	char fname[80] = "a.txt";
	FILE *fp = NULL;
	fp = fopen(fname, "w"); // 읽을때는 "r"
	if (fp == NULL)
	{
	printf("file open error\n");
	return -1;

	}
	// 4byte를 넣어야 하는 이유 : 뒤에 \0 값도 넣어줘야 하므로 +1해야함.

	char ar[4] = { 'D','J','U' };  // { 'D','J','U',\0 };
	//or char ar[4] = "DJU" (단, 문자열만 가능)

	for (int i = 0; i<4; i++)
	fprintf(fp, "%c", ar[i]);		// 뒤에 변수가 옴.
	fprintf(fp, "\n%s\n", ar);		// 문자열을 출력할때는(%s) 뒤에 (,배열을 출력하므로),
					배열의 시작주소를 넣는다.
	// 예외.
	fprintf(fp, "%s\n", &ar[0]);
	// 만약 JU만 출력하고 싶으면 ar+1을 넣으면 된다. 원하는 부분의 시작주소를 넣자.
	fprintf(fp, "%s\n", ar + 1);

	fclose(fp);	// 안해서 오류가 나면, 다시 열어도 컴파일 오류가 이상하게 난다.
	return 0;
}
```

```c
// 과제 파일을 불러오자.
#include <stdio.h>
int main()
{
	FILE *fp = NULL;
	fp = fopen("sample_input.txt", "r");	// 읽을때는 "r"
	if (fp == NULL)
	{
	printf("file open error\n");
	return -1;
	}
	char c;
	int d;
	int result = 0;
	fscanf(fp, "%c", &c);  // 문자
	printf("%c\n", c);
	fscanf(fp, "%d", &d);  // 숫자
	printf("%d\n", d);


	//int test_case;
	//fscanf(fp, "%d", &test_case);


	for (int i = 1; i < 10; i++)
	{
	fscanf(fp, "%c", &c);
	//printf("%c\n", c);
	}
	result = 1;
	printf("%d\n", result); // 결과값만 출력되면 과제 ok

	fclose(fp);
	return 0;
}
```

<DAY 18>

```
/**
	@file   20161128.c
	@brief 문자와 문자열 cont'd
	@author Minseon Shin
	@date   2016-11-28
*/

// 문자와 문자열 라이브러리
#include <stdio.h>
int main()
{
	char name[] = { 'K','I','M','\0' };  //'\0' or 'NULL'= char name[4]
	char *p;
	*p = "text";
	printf("%s\n", &name);
	printf("%s\n", *p);
	return 0;

}

// 문자 입출력 함수 : getchar, getche, getch,
#include <stdio.h>
int main()
{
	char name[4] = { 'K','I','M','\0' };
	//입출력함수
	scanf("%c", &name[0]);
	name[0] = getchar();
	printf("%s\n", name);

}


// 문자 입출력 함수 : getchar, getche, getch, putchar, putch
//           eco    0        0      X
//         buffer   0        X      X      0       X


//buffer -> full buffer (바이트 수가 다 차야 출력)
//       -> line buffer (줄바꿈)
```

```c
#include <stdio.h>
#include <conio.h>
int main()
{
	char a, b, c;  // getchar = line buffer
	a = getchar(); // enter를 눌러야 저장. => line buffer 사용.
	b = getchar();
	c = getchar();
	printf("%c %c %c", a, b, c);
	return 0;

}
#include <stdio.h>
#include <conio.h>
int main()
{
	char a, b, c;  // getch : no buffer eco off
	a = getch();   // 엔터치기 전에 바로 들어감. 대신 화면에 안뜸(eco기능)
	b = getch();
	c = getch();
	printf("%c %c %c", a, b, c);
	putchar(a);
	putchar(b);
	putchar(c);    // buffer on
	return 0;
}
#include <stdio.h>
#include <conio.h>
int main()
{
	char a, b, c;  // getche : no buffer eco on
	a = getche();  // 엔터치기 전에 바로 들어감.
	b = getche();
	c = getche();
	printf("%c %c %c", a, b, c);

	putch(a);
	putch(b);      // buffer off
	putch(c);
	return 0;
}
```

```c
// 문자열 입력함수 : gets, puts
#include <stdio.h>

int main()
{
	char temp[80] = {0, };
	gets(temp);
	puts(temp);

	return 0;
}

// 문자처리 함수 (ctype.h) : isalpha, isdigit, isupper, toupper, toascii, ...
#include <stdio.h>
#include <ctype.h>
int main()
{
	char temp = 'a';
	temp = getch();
	if (isalpha(temp)) printf("%c is alphabet\n", temp);
	else printf("%c is not alphabet\n", temp);
}


// 문자열 처리함수 (string.h) : strlen, str(n)cpy, str(n)cat, strcmp(compare),
strchr(character)

// 문자열 처리함수_strlen()

char temp[80] = "ABC";
int number = strlen(temp);

// 문자열 처리함수_strcpy()
char temp1[80] = "ABC";
char temp2[80] = "123";
strcpy(temp2, temp1);

// 문자열 처리함수_strcat()
char temp1[80] = "ABC";
char temp2[80] = "123";
strcat(temp2, temp1);
```

```
// 문자열 처리함수_strcmp()
char temp1[80] = "ABC";
char temp2[80] = "ABD";
strcmp(temp1, temp2); // 같으면 0을 출력, 같지 않으면 양수 혹은 음수가 뜸.


// 문자열 처리함수_strchr() : string 에서 특정 문자를 찾아서 그 특정문자의 주소를 출력.

// 예제
#include <stdio.h>
#include <string.h>   // strcpy, strstr, .....
int main(void)
{
        char str1[80] = { 0, };
        char str2[80] = { 0, };
        gets(str1);
        printf("strlen: %d\n", strlen(str1));
        strcpy(str2, str1);
        puts(str1);
        puts(str2);
        return 0;
}

#include <stdio.h>
#include <string.h>   // strcpy, strstr, .....
int main(void)
{
        char str1[80] = { 0, };
        char str2[80] = " everyone";
        gets(str1);
        printf("strlen: %d\n", strlen(str1));
        strcat(str1, str2);
        puts(str1);
        return 0;
}
```

```c
#include <stdio.h>
#include <string.h>   // strcpy, strstr, .....
int main(void)
{
	char str1[80] = { 0, };
	char str2[80] = { 0, };
	gets(str1);
	gets(str2);

	printf("strcmp: %d\n", strcmp(str1, str2));
	return 0;
}
```

```c
#include <stdio.h>
#include <string.h>   // strcpy, strstr, .....
int main(void)
{
	char str1[80] = { 0, };
	char *p = NULL;
	char *q = NULL;
	gets(str1);

	p = strchr(str1, 'h');
	printf("p : %c\n", *p);
	printf("p : %s\n", p);

	q = strchr(str1, 'll');
	printf("p : %s\n", q);
	return 0;
}
```

<DAY 19>

```
/**
        @file   20161130.c
        @brief 지난시간마무리, 구조체, 공용체, enum
        @author Minseon Shin
        @date   2016-11-30
*/

//      p. 384 9번
//      '찾아바꾸기':1. 사용자로부터 문자열 입력받기
//      2. 찾을 문자열을 입력받기
//      3. 바꿀 문자열을 입력받기
//      4. 결과 문자열을 화면에 출력

#include <stdio.h>
#include <string.h>
int main()
{
        char input_str[80], key_str[80], replace_str[80];
        char *str1 = NULL;
        char *str2 = NULL;
        char temp[80];

        printf("문자열을 입력하시오: ");
        gets(input_str);
        printf("찾을 문자열: ");
        gets(key_str);
        printf("바꿀 문자열: ");
        gets(replace_str);

        str1 = strstr(input_str, key_str);
        str2 = str1;
        while (*str2 != ' ') str2++;
        strcpy(temp, str2);
        strcpy(str1, replace_str);
        strcat(input_str, temp);

        puts(input_str);
        return 0;
}
```

```c
// 구조체(Struct), 공용체(union), enum
#include <stdio.h>
// 구조체 정의, 할당
// struct tag = int, double,...

// example
//정의 : 메모리에 공간 차지 안함. (초기화)
struct pos {
        double x;
        double y;
};
struct student {
        int id;
        char name[20];
        double grade;
        struct pos pose;        // 구조체 정의 안에 구조체 올 수 있다.
};
int a;
struct student s1 = { 20075051, "Kim", 4.3, {1.0,2.0} };
// 선언/할당 : 선언으로 하여 메모리에 차지하고자 하면 할당.
// 선언
struct pos {
        double x;
        double y;
};
int main()
        struct pos s1;

// 정의와 선언/할당을 동시에 할 수도 있음.(위에 선언 + 할당)과 동일
struct student {
        int id;
        char name[20];
        double grade;
}s1 = {20081051, "Lee", 3.5};

// tag 없어도 가능? ok
struct {        // 한번만 쓸거면 ok
        int id;
        char name[20];
        double grade;
}s2;
```

Structure Tag(이름)

Structure member

구조체 변수 선언

```
// 상용화

#define S32 (int)      // 전처리문법
#define S08 (char)
#define U08 (unsigned char)

int a;
S32 a;
unsigned char a;
U08 a;


typedef int S32;         // 컴파일문법 : 전처리문법보다는 컴파일 오류도 잡을수 잇어서 더
좋지만... 상용코드는 #define
typedef unsigned char U08;

typedef struct {
        int id;
        char name[20];
        double grade;
}str_stu;


struct student s1;     // 위의 코드처럼 구조화하면 이렇게 길게 선언해야함.
str_stu s1;    // str_stu 가 int나 double 같은 자료형이 된 셈.
```

```c
// 구조체 끼리 복사는 가능하지만 비교는 불가능하다.
#include <stdio.h>
typedef struct {
	int id;
	char name[20];
	double grade;
}str_stu;

int main(void)
{
	str_stu s1 = { 20161430, "SHIN MINSEON", 4.0 };
	str_stu s2;
	str_stu AClass[10];   // 구조체가 10개 있다.
	s2 = s1;
	s1.grade = 4.3;

	printf("%d %s %0.2lf", s1.id, s1.name, s1.grade);
	printf("%d %s %0.2lf", s2.id, s2.name, s2.grade);
	if (s1 == s2); // 불가능!
	if (s1.id == s2.id); // 이건 가능

	if (s1.name == s2.name); // 이건 불가능. 배열이라서 strcmp 써야함.
	if (strcmp(s1.name, s2.name)) printf("same name\n");

	AClass[4].id = 20153015;

	str_stu* sp = NULL;   // 구조체 포인터
	sp = &s1;
	prinft("%d\n", (*sp).id);     // 단항연산자 주의할 것.
	prinft("%d\n", *sp->id);      // -> 간접멤버 연산자

	return 0;
}
```

```c
// 구조체 속 문자열 배열을 입력할 때 주의할 점!
#include <stdio.h>
struct student {
	int number;
	char name[10];
	double grade;
};
int main()
{
	struct student s;
	s.number = 20070001;
	s.name = "홍길동";                    // ERROR : 배열의 경우 strcpy()
	strcpy(s.name, "홍길동");
	s.grade = 4.3;

	struct student list[SIZE];          // 구조체 배열의 선언
	struct student list[3] = {
                                    {1, "Park", 3.42},
                                    {2, "Kim", 4.31},
                                    {3, "Lee", 2.98}
                                    };
	return 0;
}

// Tip
구조체 배열의 원소 개수 자동 계산
n = sizeof(list)/sizeof(list[0]);
or   n = sizeof(list)/sizeof(struct student)

// 원래는 16 바이트지만 24인 이유는 8 단위로 읽기 때문.
#include <stdio.h>
struct temp {
	int a;
	double b;
	int c;
};
int main(void)
{
	int a;
	printf("sizeof : %d\n", sizeof(struct temp));
	return 0;}
```

```c
#include <stdio.h>
struct temp {
	int a;
	char b : 3;	// bit field :  bit를 쪼개서 설정.
	char c : 4;
	char resolved : 1;
};

int main(void)
{
	int a;
	printf("sizeof : %d\n", sizeof(struct temp));
	return 0;
}
```

```c
#include <stdio.h>
// UNION(공용체) : 같은 메모리 공간을 여러개의 변수들이 공유할 수 있게 하는 기능.
union ip_addr {
	char a;
	int b;
	short c;
};
int main()
{
	//printf("sizeof : %d\n", sizeof(union ip_addr));

	union ip_addr a1;
	printf("sizeof : %d\n", sizeof(a1));
	return 0;
}// 다 덮어써버려서 int 값만 남음.. 제일 멈.
```

```c
// 상용화
#include <stdio.h>
 // UNION(공용체)
struct sensor {
        unsigned char type;
        union power {
        int reset;
        unsigned char laser;
        unsigned char ir;
        unsigned char sonar;
        };
};
int main()
{
        struct sensor s1;
        s1.reset = 0;
        s1.laser = 0;
        s1.ir = 0;
        s1.sonar = 0;
        return 0;            }


// p. 414 union 예제
#include <stdio.h>
union ip_address {
        unsigned long laddr;
        unsigned char saddr[4];
};
int main()
{
        union ip_address addr;

        addr.saddr[0] = 1;
        addr.saddr[1] = 0;
        addr.saddr[2] = 0;
        addr.saddr[3] = 127;
        //ip.address : 1.0.0.27 : 컴퓨터 내부적 ip주소
        printf("%x\n", addr.laddr);  // /x : 16진수

        return 0;
}               // 실행 결과 : 7f000001
```

<DAY 20>

```
/*
        @file   20161205.c
        @brief 구조체, cont'd, 파일 입출력
        @author minseon shin
        @date   2016-12-05
*/

// 문자, 문자열 -> 숫자
// 아스키코드를..
          atoi(int로 ), atof(실수형으로), fscanf(파일 불러옴), sscanf(string에서 불러옴)
// atoi, atof 의 라이브러리. stdlib.h

#include <stdio.h>
#include <stdlib.h>
int main()
{
        char str[80];
        char menu;
        float temp;

        gets(str);
        menu = getchar();

        printf("%s\n", str);
        printf("%d\n", menu);


        printf("%f\n", atof(str));
        printf("%d\n", atoi(&menu));
                  // 문법 :  int_cdecl atoi(const char*_string)(주소 써야함)
        sscanf(str, "%f", &temp);
        printf("%f\n", temp);
        return 0;
}
```

```
//tip
//한바이트씩 주고받는다. char 디바이스 _ 키보드(아스키코드로 받음)
//수천키로바이트씩 주고받는다 . block 디바이스 _ usb


// string concatenation (문자열 붙이기) : strcat
// string concatenation (문자열 붙이기) pattern == sstring copy 이용(참고)
// e.g
// s1 = "c:"; s2= "/data/"; s3 = "test.c";

// c++
// string s1, s2, s3;
// string result = s1 + s2 + s3;

// c_1
#include <stdio.h>
#include <string.h>   // strcat, strcpy, strlen
int main()
{
        char *s1 = "c:";
        char *s2 = "/data/";
        char *s3 = "test.c";
        char result[80];

        strcpy(result, s1);   // result = "c:"
        strcat(result, s2);   // result = "c:/data/"
        strcat(result, s3);   // result = "c:/data/test.c"

        printf("%s\n", result);
        printf("%s\n", strcat(strcat(strcpy(result, s1), s2), s3));
        // 4줄을 1줄로!
        return 0;
}
```

```c
// c_2(sscanf 이용)_____가독성이 휠씬 뛰어남.
#include <stdio.h>
#include <string.h>   // strcat, strcpy, strlen
int main()
{
        char *s1 = "c:";
        char *s2 = "/data/";
        char *s3 = "test.c";
        char result[80];

        sscanf(result, "%s%s%s", s1, s2, s3);
        sprintf("%s\n", result);
        return 0;
}



//tip
char *s1 = "c:";       // 읽기전용 메모리.(**)
char *s2 = "/data/";
char *s3 = "test.c";
char result[80];
// 원래는
int *p;
*p = 7;
// 이렇게 초기화 해야하는데 string은 가능. 대신 읽기메모리.

strcpy(s1, s2); //불가능
strcpy(result, s1); // 읽는건 가능.
```

```
// 구조체 : struct, union, enum(enumeration)

#include <stdio.h>
struct student {      // struct 대신 union ok
        int id; // member 변수
        char name[80];
        double grade;
};
typedef struct {
        int id;
        char name[80];
        double grade;
}st_student;
int main()
{
        struct student s1;
        st_student s2;

        return 0;
}
```

>> typedef : 새로운 자료형(type)을 정의하는 것.

```
typedef int INT32;
typedef unsigned char BYTE;

BYTE index;   // unsigned char index; 와 같다.
INT32 i;      // int i; 와 같다.


// 구조체_ 선언 속 typedef(구조체를 정의하면서 동시에 point 타입으로 재 정비)
typedef struct point POINT;

typedef struct point{
        int x;
        int y;
} POINT;
```

장점 : 이식성 높음, #define과 차이점(#define은 전처리기가, typedef는 컴파일러가), 문서화의 역할

```c
// emum(enumeration): 변수가 가질 수 있는 값들을 미리 열거해 놓은 자료.
#include <stdio.h>
#define FL1 0                       #define FL2 1
#define FL3 2                       #define FR1 3
#define FR2 4                       #define FR3 5
#define RL1 6                       #define RL2 7
#define RL3 8                       #define RR1 9
#define RR2 10
#define RR3 11
#define SS     12
#define NUMBER_OF_SENSER 13

struct sensor sonar_sensor[NUMBER_OF_SENSER];      // 위랑 동일함.
enum {
        FL1 = -1,
        FL2,    //0
        FL3,    //1
        FR1,
        FR2 = 100,
        FR3,    //101
        RL1,
        RL2,
        RL3,
        RR1,
        RR2,
        RR3,
        NUMBER_OF_SENSORS     // 12개가 들어감.
};
enum type {
        SONAR,
        LASER,
        IR
};
int main()
{
        // temp = sonar_sensor[FL2].data;
        enum type t1;
        t1 = SONAR;

        return 0;
}
```

```c
// call by value, call by reference
// func(int a); func(int* a);

#include <stdio.h>
char **src = "src";
void func(char** str)
{
	*str = *src;
}
int main()
{
	char dst[80] = "dst";
	func(&dst);
	printf("%s\n", dst);
	return 0;
}
```

12장 : 표준입출력과 파일입출력

```
// 파일입출력 : fopen, fclose, fprintf, fscanf, fgetc,
                fgets, fputc, fputs, fread, fwrite
// 파일 임의접근 : fseek (커서를 옮겨줌)

#include <stdio.h>
char fname[80] = "test.txt";
int main(void)
{
        FILE* fp = NULL;
        if ((fp = fopen(fname, "w")) == NULL)
        // mode : r, w, a, r+, w+, a+
        // r+ w+ 차이?  둘다 읽고 쓸수 있다. 차이는 w+는 쓰고 읽을거라서 열림. r+ 읽고 쓸거라서
열리지도 않음.
        {
        perror("%s file is not opened.", fname);
        return -1;
        }
        fputs("Hello, ", fp);
        fputc('A', fp);
        fwrite((char*)"\n12", 4, 1, fp);
        fseek(fp, 1, SEEK_CUR);
        // SEEK_SET : 처음
        // SEEK_CUR : 현재
        // SEEK_END : 끝
        char temp = fgetc(fp);
        printf("%c\n", temp); // 빈자리를 읽음.
        fclose(fp);
}
```

```c
// stream : 모든 입력과 출력을 바이트들의 흐름으로 생각하는 것.
// 기본 입출력 스트림 : stdin, stdout(출력_모니터), stderr(에러_모니터)
// 실행하면 코드로 포인터를 만들지 않아도 기본적으로 생기는 입출력 포인터들!

#include <stdio.h>
int main(void)
{
	printf("abc\n");
	// printf("  ",....)에서 “”를 첫 번째 매개변수 = 형식제어문자열
	// fprintf(stdout, "abc\n");

	int temp = 0;
	scanf("%d", temp);
	// fscanf(stdin, "%d", &temp);
	// stdout, stderr의 차이 : buffer의 유무. (줄버퍼(\n입력됐을때 표시),
	   풀버퍼(버퍼속에 다 찼을때 표시))
	fprintf(stdout, "1");
	fprintf(stderr, "1"); // 버퍼안씀. 따라서 바로 출력해줌.(디버깅할때 사용)
	return 0;
}



// 형식지정자 : %c %d. %ld, %lld, %f, %lf, %s, %u(unsigned 부호없는 정수), %o(16진수),
		%x(16진수_소문자), %X(16진수_대문자), %e(지수 12=1.2e1000)
		%E(지수 E), %p(pointer)


// p. 438
#include <stdio.h>
int main(void)
{
	printf("%10.3f\n", 1.23456789);
	// ______1.235(반올림 함) _공간띔.  (-): 좌측정렬
	printf("%010.4f\n", 1.23456789);     // 0을 앞에 붙이면 빈공간은 0으로 출력
	printf("%10.4f\n", 1.23456789);
	printf("%.3f\n", 1.23456789);
	printf("%.4f\n", 1.23456789);
	printf("%.5f\n", 1.23456789);
	return 0;
}
```

<DAY 21>

```
/*
	@file	20161207.c
	@brief	13장 14장
	@author	minseon shin
	@date	2016-12-07
*/

// 전처리기(pre-processor) : 컴파일하기에 앞서서 소스파일을 처리하는 컴파일러의 한 부분
// #define(매크로 정의), #include(파일 포함),
// #if, #else, #endif : DEBUG(조건에 따른 컴파일), 모델
// #ifdef, #endif(매크로가 정의되어 있는 경우의 컴파일)- 헤더파일에 자주 씀
// #ifndef (매크로가 정의되어 있지 않은 경우의 컴파일)

#include <stdio.h>
// #define DEBUG
int main()
{
	int temp[] = { 1,3,5,7,9 };
#if DEBUG
	printf("%d\n", temp[2]);
#endif

	return 0;
}

// #if, #else, #endif :  모델이 다른 것을 하나의 소프트웨어로 제어 하고 싶다!
#include <stdio.h>
#define KOREA
#define WORLD
int main()
{
#if KOREA
	한글메세지 출력;
#else WORLD
	각국 메세지 출력;
	return 0;
}
```

```
// #ifdef, #ifndef, #endif
#ifdef DEBUG // 디버그가 정의되어 있으면?!
#ifndef DEBUG // 디버그가 정의되어 있지 않으면?!


// 헤더파일로 불러올때...
헤더파일에서 '20161207.h'

#pragma once
#ifndef 20161207.h
#define 20161207.h
..
..
..
..
#endif

100개의 c코드가 동시실행된다(include)하더라도 첫번째 코드는 실행안되고
두번쨰부터 중복없이 h파일이 실행된다.h / #define

// sample_input.g
// #ifndef _SAMPLE__INPUT_H_ (헤더파일앞에 _를 붙이는게 회사마다다르지만 보통이렇게...)


// #define 매크로함수 : fuction like macro : 매크로가 함수처럼 매개변수를 가지는 것.
기호상수일때
#include <stdio.h>
#define PI 3.141592
const int PI = 3.141592

#define MAX(x,y)      ((x)>(y))?x:y  // x, y에 숫자 뿐이아니라 수식이 들어갈수 있으므로
()필요
// 장점 : 코드의 양이 적어짐. 자료형에 관계가 없음.
// 단점 : 자료형을 써주지 않기대문에 주의, 항상 변수는 ()로 묶어줄 것.


#define HALFOF(y,x) ((x)/2)      // 매개변수는 다 써야함. ERROR
#define ADD (x, y) ((x)+(y))      // 오류! 괄호사이에 공백 있으면 안됨.ADD ()
```

```
int main()
{
	MAX(a*b + c, b >> 8);
	return 0;
}
```

만약에 매크로함수 안 쓰면 이렇게 다 써야함.

```
int max(int a, int b)
{
	return (a > b) ? a : b;
}
float max(float a, float b)
{
	return (a > b) ? a : b;
}
```

```
// main함수의 인수
#include <stdio.h>
int main(int argc, char* argv[])	// dos 파일로 입력 후 확인했음..
{
	// 161207.exe a b c
	// argc : 4
	// argv[0] = 161207.c
	// argv[1] = a
	// argv[2] = b
	// argv[3] = c

	printf("argc: %d\n", argc);
	return 0;
}
```

```
// 동적할당(Dynamic allocation)
//      <-> 정적할당(static allocation)
// malloc(메모리 할당이 안빠짐), free(malloc할당(동적할당)후에 메모리 리셋)
// calloc(malloc + clear)= 0으로 초기화, realloc(메모리 블록의 크기를 변경)

int a; // 4byte
double temp[4];         // 32 bytes
char name[var];        // compile error
struct student {
        int id;
        char name[20];
        double c_grade;
        double eng_grade;
};

int main(int argc, char* argv[])
// 161207.exe 30
// argc = 2
// argv[0] = '161207.exe'
// argv[1] = 30
{
        struct student male[40];
        struct student male[argv[1]]; // error
        return 0;
}
```

```c
#include <stdio.h>    // printf, scanf,.....
#include <stdlib.h>   // malloc, free, calloc, relloc....
#include <string.h>

//void* malloc(size_t size)  // bytes
int a;
size_t a;
typedef size_t int;
int main(void)
{
	void* p = NULL;
	int a = 7;
	double b = 9.0;
	p = &a;
	(int*)p = 9;   // void 포인터는 값을 바꾸거나 넣을때 형지정을 해야함. casting

	malloc()
	return 0;
}

int main(void)
{
	double grade[30];     // 정적할당.
	grade[1] = 3.0;
	double* dp = NULL;
	dp = (double*)malloc(30 * sizeof(double));  //240 byte 예약!
	memset(dp, 0, 30 * sizeof(double));
	// memset :  reset (<string.h>)
	*(dp + 1) = 3.0;

	만약에 memset 하기싫으면 malloc 대신에 calloc 써라.
	double* dp2 = NULL;
	dp2 = (double*)calloc(30, sizeof(double));

	realloc(dp, 60);
	dp = (double*)realloc(dp, 60);

	free(dp);
	free(dp2);
	return 0;
}
```

```c
// 구조체로도 동적할당 받을 수 있다.
#include <stdio.h>
#include <sting.h>
typedef struct {
	int id;
	char name[20];
	double grade;
}st_stu;

int main()
{
	st_stu male[30];
	st_stu* sp = NULL;
	sp = (struct*)malloc(30 * sizeof(st_stu));
	sp = (struct student*)malloc(30 * sizeof(struct student));
	// 안쓰면 이렇게 다...써야함. 길어짐!
}


//	연결리스트(Linked list) : 자료구조
enum type {
	ENTERTAINMENT,
	EDUCATION,
	ENGLISH,
	MATH
};
struct book{
	char name[20];
	char isbn[13];
	enum type b_type;
	struct book* new_book;
}
```