

# 실습: *Week 1*

---

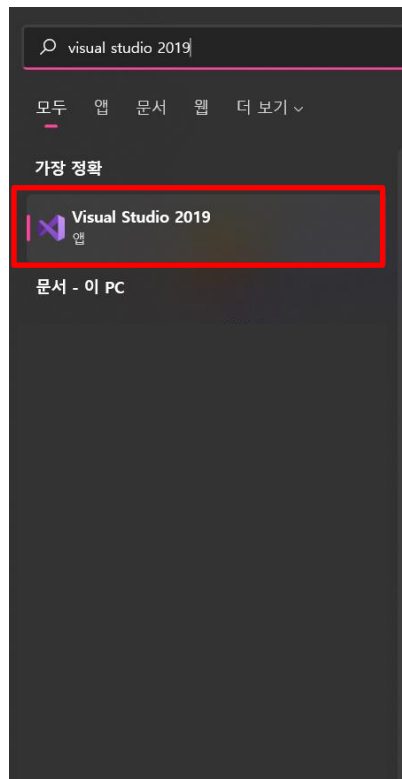
Data Structures

# Contents

- Microsoft Visual Studio 사용 방법
  - 프로그램 작성 (Coding)
  - 컴파일 (Compiling)
  - 실행 및 디버깅 (Debugging)
- 실습
  - 실습 1-1. 1~10 더하기, 디버깅 하기
  - 실습 1-2. 배열, 포인터 값 출력
  - 실습 1-3. 배열에서 **max** 찾기
  - 실습 1-4. **find\_max** 함수 작성

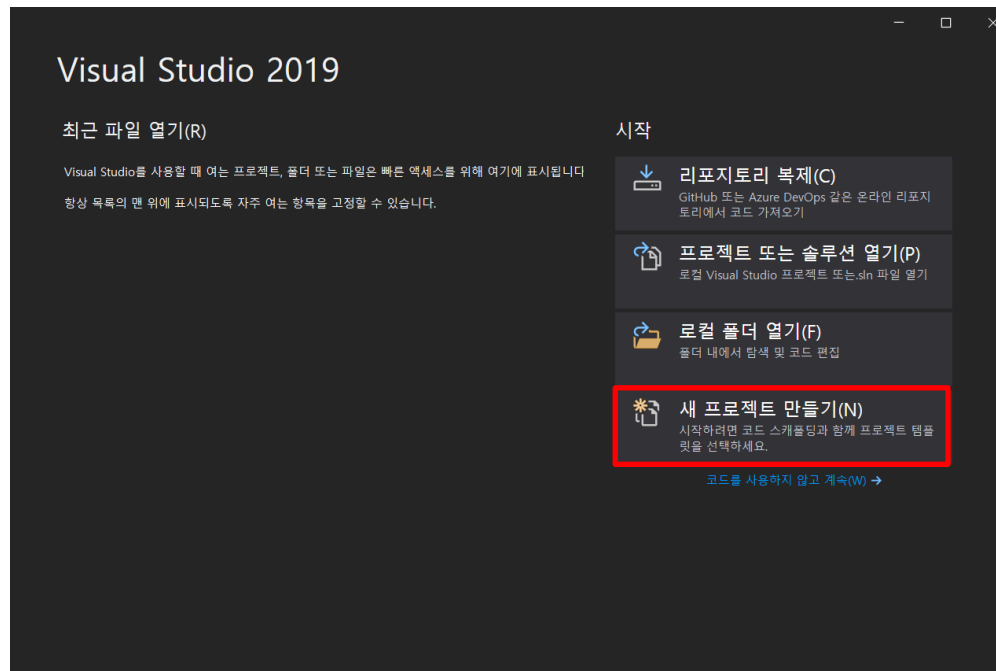
# MS Visual Studio 시작

- 바탕화면 → Microsoft Visual Studio 2019 아이콘 클릭 또는
- 시작 → 프로그램 → Visual Studio 2019 → Visual Studio 2019 클릭



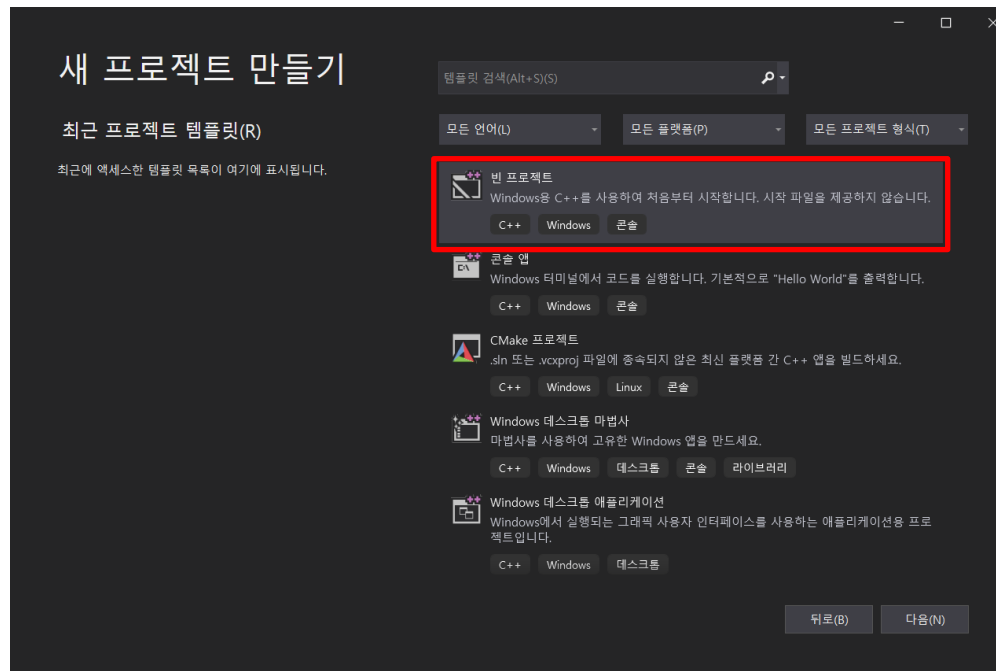
# Project 생성

- 새 프로젝트 만들기(N) 클릭
- 빈 프로젝트 선택
- 프로젝트 이름 지정 (ex\_ project1)



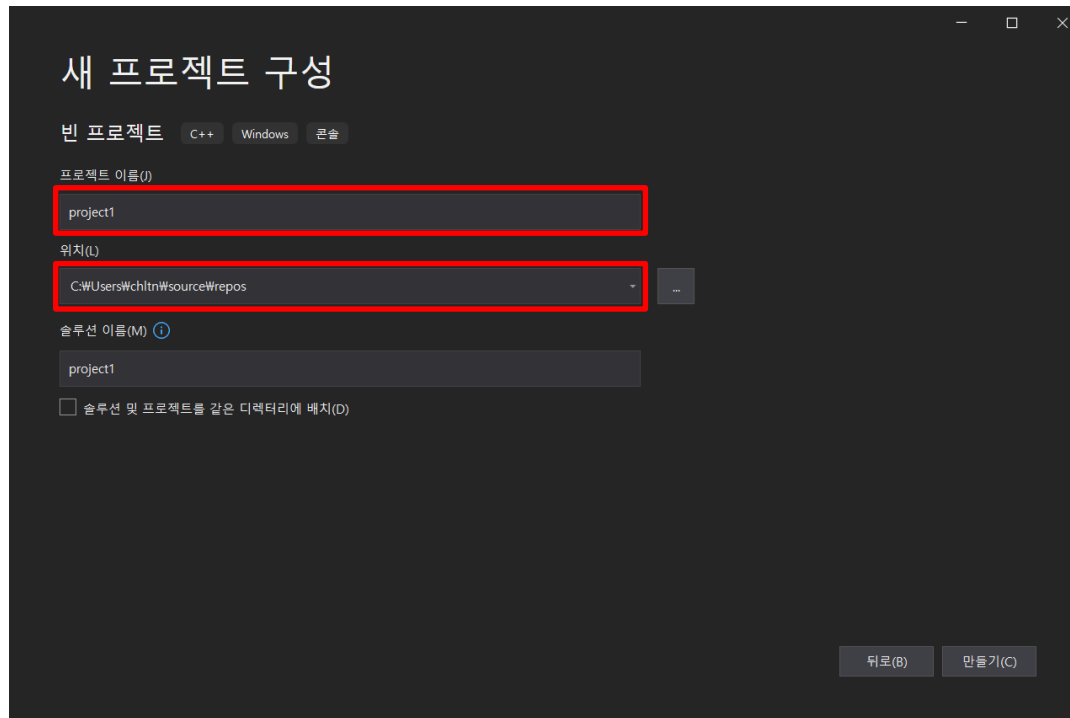
# Project 생성

- 새 프로젝트 만들기(N) 클릭
- 빈 프로젝트 선택
- 프로젝트 이름 지정 (ex\_ project1)



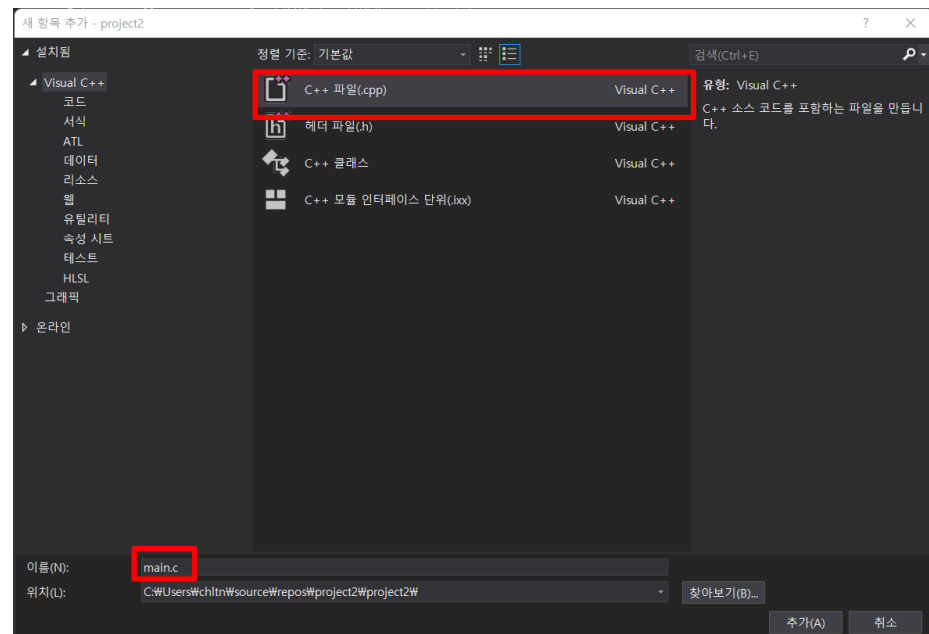
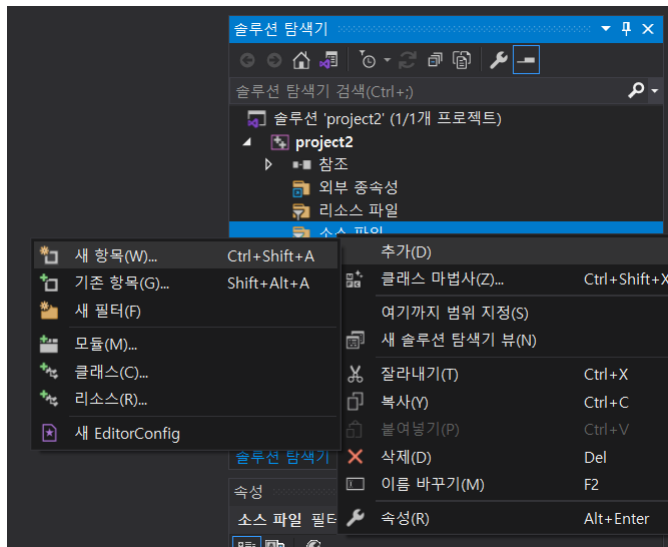
# Project 생성

- 새 프로젝트 만들기(N) 클릭
- 빈 프로젝트 선택
- 프로젝트 이름 지정 (ex\_ project1)



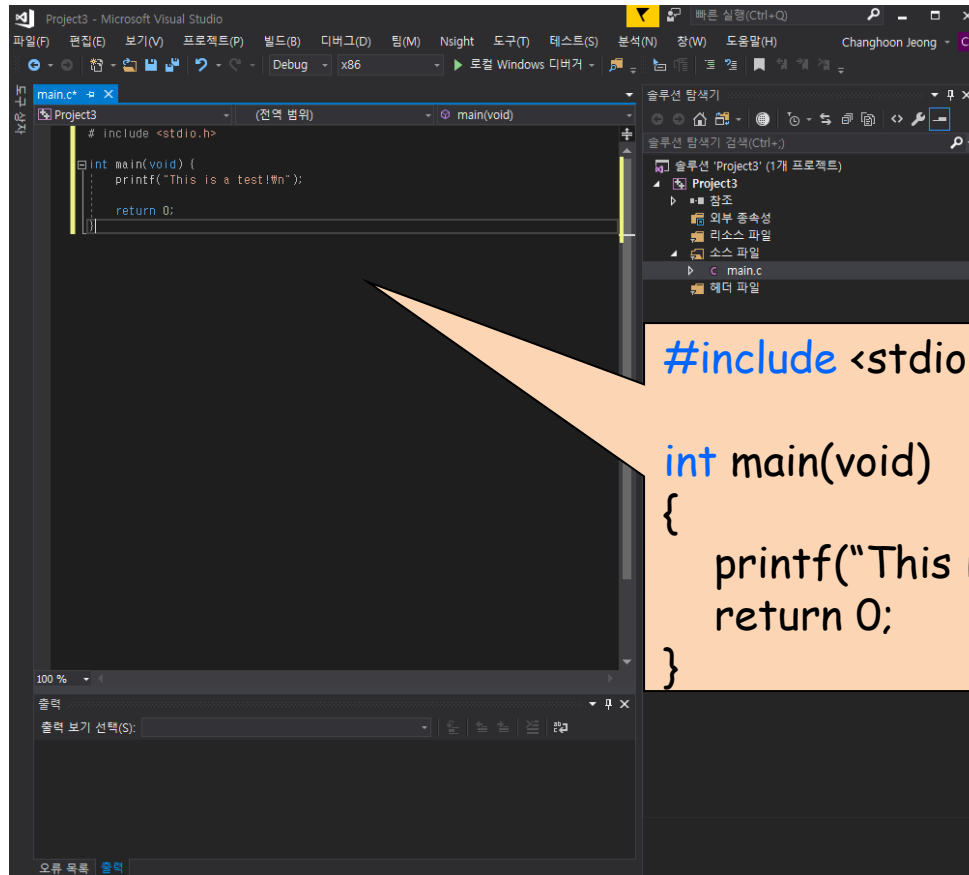
# Source File 생성

- 소스 파일 우클릭 → 추가 → 새 항목
- 코드 탭 선택 → “C++ 파일(.cpp)” 선택(이름 뒤에 반드시 확장자 “.c”를 입력)
- File 이름 지정



# Coding

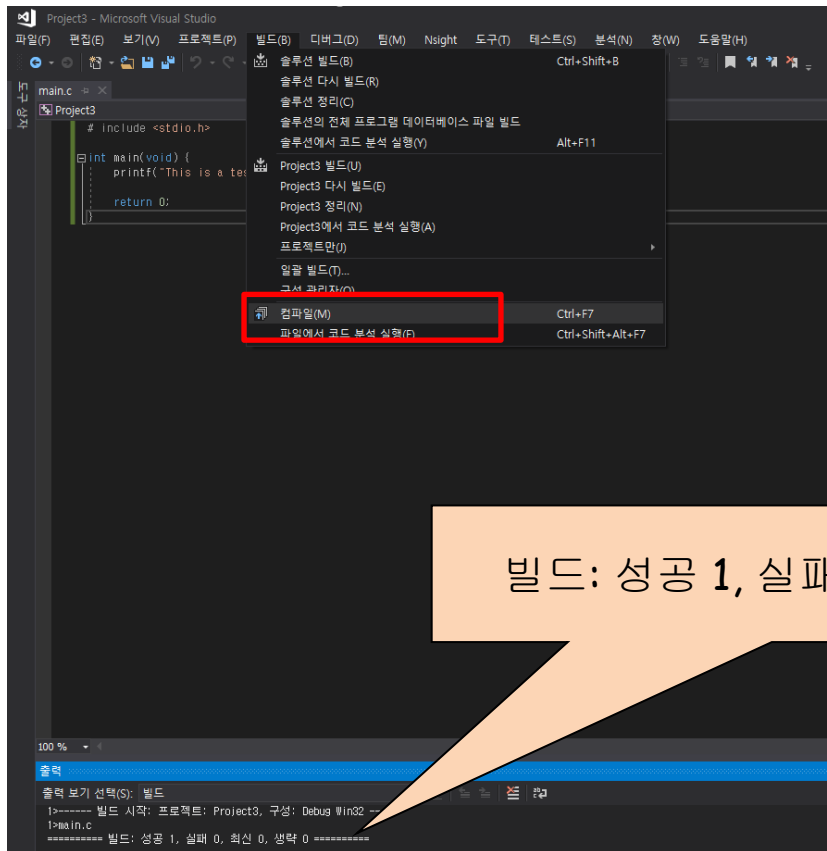
- 빈 Page에 코드를 입력





# Compiling

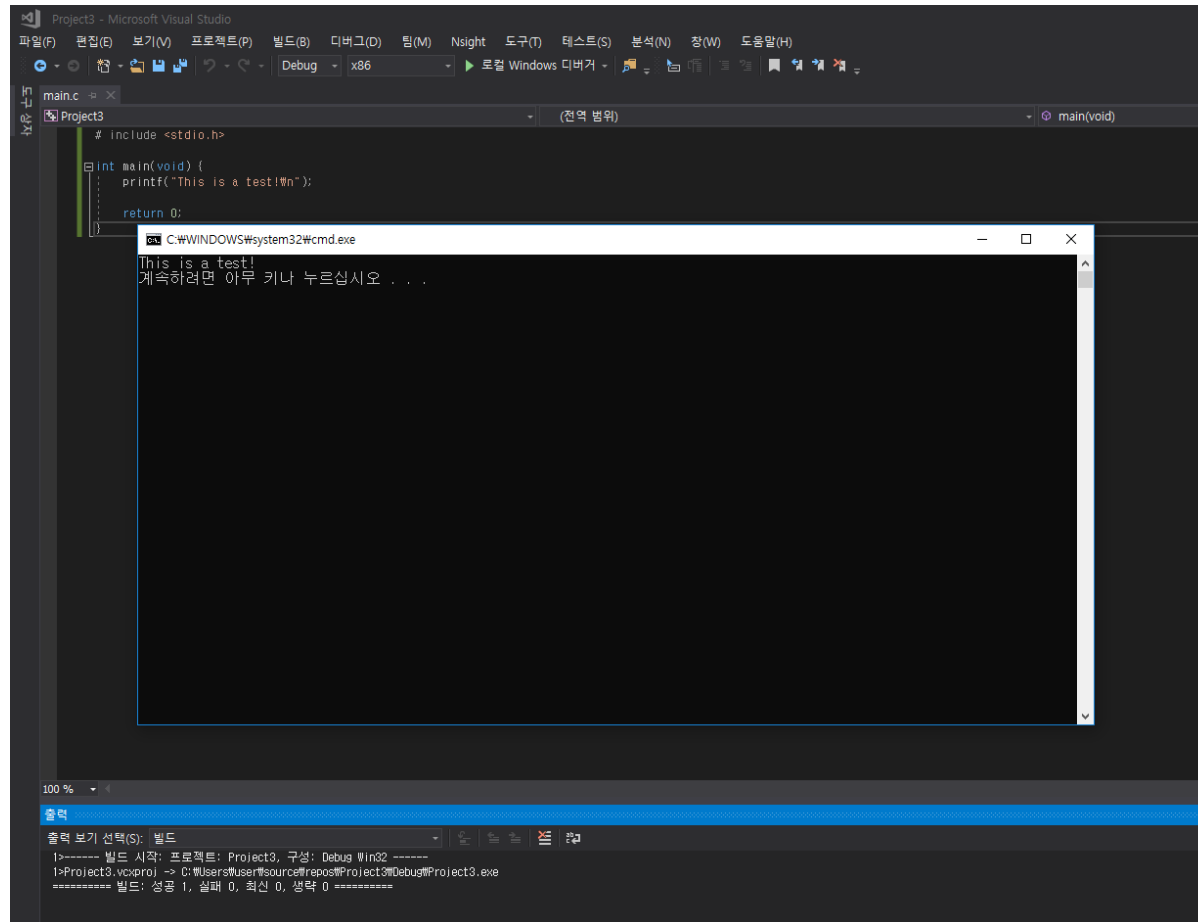
- 빌드 → 컴파일(ctrl+F7)을 사용하여 컴파일 - Error 수정



빌드: 성공 1, 실패 0, 최신 0, 생략 0

# 실행

- 디버그 → 디버깅 시작(F5)을 사용하여 실행



# 변수의 개념

## ■ 변수 (Variables)

- 프로그램 실행 중에 임의의 값(데이터)을 저장하는 기억장소
- 변수의 형식에 따라 데이터가 저장되는 방식이 달라짐

C basic data types	
char	문자 (1 바이트)
int	정수 (4 바이트)
float	실수 (4 바이트)
double	Double precision 실수 (8 바이트)

```
int count;    // int type으로 count라는 이름의 변수를 선언
double x, y;  // double type의 변수 x, y 를 선언
```

# 함수 (1/3)

- 함수
  - 특정한 작업을 수행하는 독립된 프로그램 단위
- C 프로그램은 함수의 집합
  - `main()` 도 이름이 이미 정의된 특수한 하나의 함수
  - 응용 프로그램은 하나의 메인 함수와 여러 개의 함수로 구성
  - 필요에 따라 여러 소스 파일에 나누어 코딩 할 수 있음
- 프로그램 실행
  - C 프로그램은 `main()` 함수의 첫 줄에서 시작
  - `main()` 이외의 모든 다른 함수들은 `main()` 함수로부터 호출
- 라이브러리 함수
  - 이미 개발 도구에 구현되어 있는 프로그램 부품 함수
  - 예> 출력함수 : `printf()`

# 함수 (2/3)

- 함수 정의

```
[type] function_name(parameter_list)
{
    declaration
    statements;
    [return expression;]
}
```

- 함수의 머리(header)

- 함수 반환값 자료 유형, 함수 이름, 인자 목록(parameter\_list)

- 함수의 몸체(body)

- 중괄호로 시작하여 중괄호로 종료
- 수행하고자 하는 적절한 문장으로 구성

# 함수 (3/3)

```
#include <stdio.h>
int max(int, int);

void main()
{
    int a, b, k;

    printf("input1: ");
    scanf("%d", &a);
    printf("input2: ");
    scanf("%d", &b);

    k = max(a, b);
    printf("MAX = %d\n", k);
}
```

```
int max(int i, int j)
{
    int s;
    if(i >= j)
        s = i;
    else
        s = j;

    return s;
}
```

```
input1: 10
input2: 20

MAX = 20
```

# 배열 (1/4)

## ■ C 언어의 배열 특성

- 같은 타입의 변수들로 이루어진 유한 집합
- 배열명은 첫 번째 원소의 주소
- 배열의 크기는 고정 - 실행 중 크기 변경 불가
- 배열의 원소는 연속적인 메모리에 할당
- 배열의 첨자는 0부터 시작
  - $a[5] == a[0], a[1], a[2], a[3], a[4]$
- 배열 참조 시의 원소를 나타내는 첨자
  - 정수형 상수나 정수형 변수로 구성된 식
  - ex>  $a[2] = 5;$  ,  $\text{for}(j=0; j \leq 5; j++) a[j] = 2;$

# 배열 (2/4)

- 배열의 선언
  - `data_type array_name[size];`
    - `data_type` : 배열의 데이터 형
      - 기본 데이터형(`int`, `float`, `char` 등)
      - 사용자 정의 데이터형(구조체, 공용체 등)
    - `array_name` : 배열명(변수 정의 규칙에 따름)
    - `size` : 배열의 요소의 수. 생략 가능
  - `ex>`
    - `int a[10]; /* index는 0~9까지 사용*/`

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
------	------	------	------	------	------	------	------	------	------



# 배열 (3/4)

- 배열의 매개변수 전달
  - 배열의 이름은 첫 번째 원소의 주소
    - 예) `int a[];` ➡ `a == &a[0]`
  - 함수 호출과 함수 정의
    - 실 인자 : `a, b, c`
    - 형식 인자 : `int a[], int b[], int c[]`로 배열 크기 생략

```
ex) int dim1[10];          void transfer(int d1[ ] )
    ... transfer(dim1);    {
    ...                    {
                           ...
                           }
```

# 배열 (4/4)

- 예: 배열에서 가장 큰 값 찾기

```
int score[10] = {10, 20, 33, 35, 13, 22, 88, 45, 67, 77};  
  
max = score[0];  
for (i=1; i<10; i++) {  
    if (max < score[i]) max = score[i];  
};
```

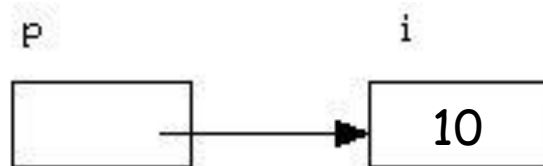
# 포인터 (1/3)

## ■ 포인터

- 자료 객체나 함수의 **메모리 번지**를 저장할 수 있는 변수
- 프로그램에서 메모리에 접근하고 주소를 다루기 위해 사용
- 예) `int i, *p;`

`i = 10`

`p = &i;`



# 포인터 (2/3)

- 포인터의 장점
  - call by reference
  - 동적 기억장소 할당 가능
  - 메모리에 직접 접근 가능
- 포인터의 단점
  - 오류 범하기 쉬움
  - 프로그램의 이해와 버그 찾기가 어려움

# 포인터 (3/3)

## ■ 포인터 변수의 연산

- `int a, b[10];`
- `int *ptr;`
- 변수의 주소값 획득: `&` 연산자 사용
  - `ptr = &a; ptr = b (&b[0]과 같은 의미)`
- 포인터 주소의 원소 값 획득: `*` 연산자 사용
  - `x = *ptr;`
- 포인터 값의 증가와 감소: 그 수만큼 떨어져 있는 데이터의 주소
  - `ptr + 3, ptr - 1`
  - `ptr1 - ptr2`
- 포인터에 대한 곱, 나눗셈 불가

# 실습 1-1. 1~10 더하기, 디버깅

// 1에서 10까지 더하기

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

```
void main()
```

```
{
```

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

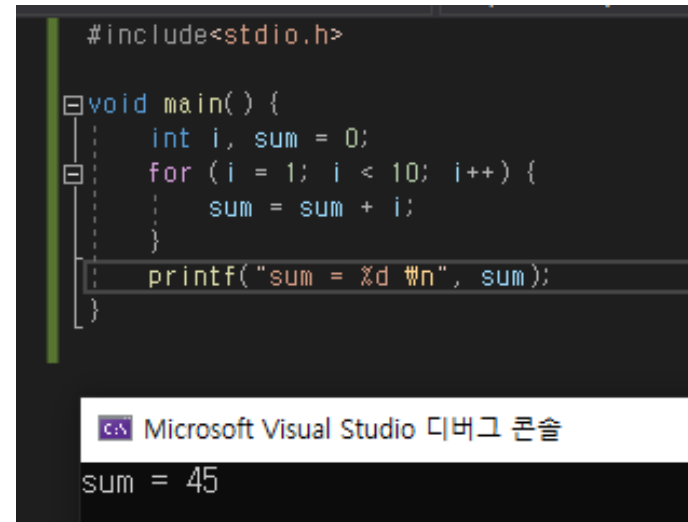
```
    for(i=1; i<10; i++){
```

```
        sum = sum + i;
```

```
    }
```

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

```
}
```



The screenshot shows a C program in Visual Studio. The code calculates the sum of integers from 1 to 10. The variable 'sum' is initialized to 0, and a for loop iterates from i=1 to i=9, adding each value to 'sum'. The final result is printed using printf. The debug console at the bottom shows the output 'sum = 45'.

```
#include<stdio.h>

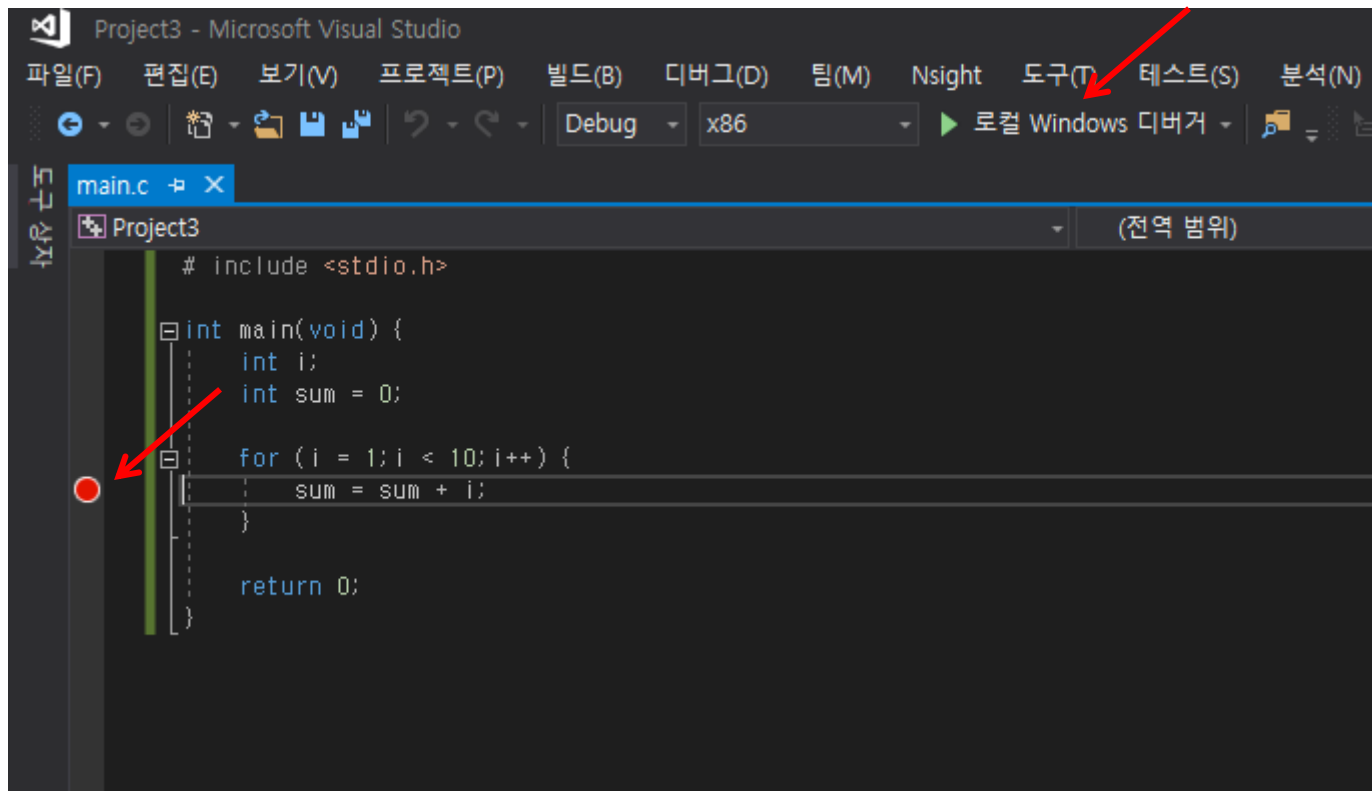
void main() {
    int i, sum = 0;
    for (i = 1; i < 10; i++) {
        sum = sum + i;
    }
    printf("sum = %d \n", sum);
}
```

Microsoft Visual Studio 디버그 콘솔

sum = 45

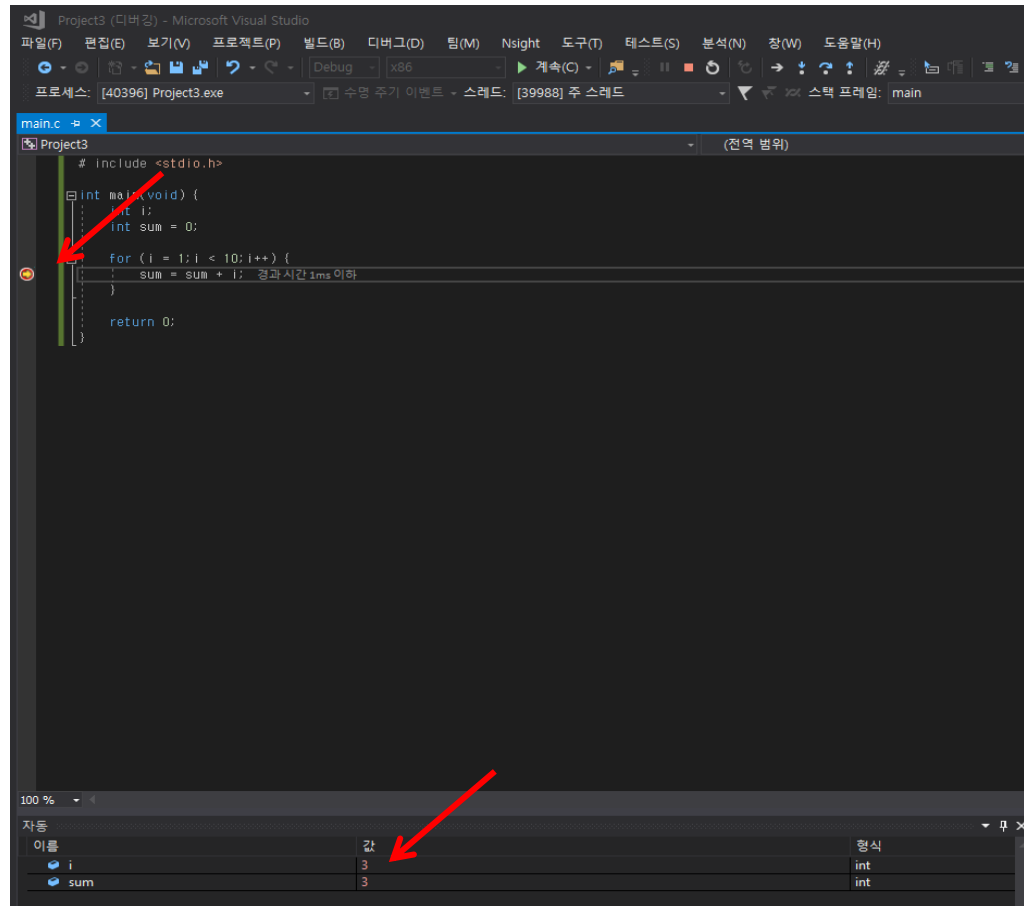
# 실습 1-1 - break point 잡기

- 코드 왼쪽 **break**하고 싶은 지점 클릭
- 로컬 **Windows** 디버거 클릭 또는 **F5** 입력 → 디버깅 진행



# 실습 1-1 - 디버깅

- F5 2번 입력 결과, 1과 2가 더해져 변수 **i**와 **sum**에 각각 3이 들어감



The screenshot shows the Visual Studio IDE with a C program named 'Project3'. The code is as follows:

```
#include <stdio.h>

int main(void) {
    int i;
    int sum = 0;

    for (i = 1; i < 10; i++) {
        sum = sum + i; // 결과 시간 1ms 이하
    }

    return 0;
}
```

The debugger is set to the 'Debug' mode, and the execution is paused at the first iteration of the for loop. The 'Locals' window at the bottom shows the current values of the variables:

이름	값	형식
i	3	int
sum	3	int



# 실습 1-2. 배열, 포인터 값 출력

- 다음과 같은 정수, 정수 배열, 포인터의 값을 출력하시오

```
int a = 10;
```

```
int b[5] = {10, 20, 30, 40, 50}
```

```
int *p;
```

**p = &a** → a, a의 주소, p, \*p 출력 (주소 출력 format은 %p 사용)

**p = b** → b, b[0], b[1], b[2], p, \*p, \*(p+1), \*(p+2) 출력

```
Microsoft Visual Studio 디버그 콘솔
a = 10, address of a = 00EFFF44
p = 00EFFF44, *p = 10
b = 00EFFF28, b[0] = 10, b[1] = 20, b[2] = 30
p = 00EFFF28, *p = 10, *(p+1) = 20, *(p+2) = 30
```

# 실습 1-2. 배열, 포인터 값 출력

```
#include <stdio.h>

void main()
{
    int a=10, b[5]={10,20,30,40,50}, *p;

    p = &a;
    printf(    ...    );
    printf(    ...    );

    p = b;
    printf(    ...    );
    printf(    ...    );
}
```

# 실습 1-3. 배열에서 max 찾기

- 다음과 같은 정수 배열에서 최대 원소의 위치(index)와 값을 찾아 출력하는 프로그램을 작성하시오

```
int score[10] = {55, 20, 33, 85, 13, 22, 45, 98, 67, 77}
```

```
0 : 55
1 : 20
2 : 33
3 : 85
4 : 13
5 : 22
6 : 45
7 : 98
8 : 67
9 : 77
1등은 7 번, 성적은 98 입니다.
```

# 실습 1-3. 배열에서 max 찾기

```
#include <stdio.h>

void main( ) {
    int i, max_index;
    int score[10] = {55, 20, 33, 85, 13, 22, 45, 98, 67, 77};

    // 각 원소 값 출력

    // 가장 큰 원소의 위치(max_index) 구하기

    printf("1등은 %d 번, 성적은 %d 입니다. \n\n",
           max_index, score[max_index]);
}
```

# 실습 1-4. find\_max 함수 작성

- 실습 1-3과 같은 프로그램을 다음과 같은 함수로 작성하시오

```
void main( ) {  
    int score[10] = {55, 20, 33, 85, 13, 22, 45, 98, 67, 77};  
    ...  
    max_index = find_max(score, 10); //find_max(배열,배열사이즈)  
    ...  
}  
  
// a[size]에서 최대 원소 위치(index)를 구하는 함수  
int find_max(...)  
{  
    ...  
}
```

# 구현 완료 후 손들어 주세요.

- 구현 및 테스트 완료 후 손 들고 게시면 자리로 가서 검사할 예정입니다.
- 실습 1은 **break point** 지정 후 실행하는 모습 보여주시고,
- 실습 2~4는 차례대로 실행 결과 보여주세요.