

2-1. Code 흐름

창의적소프트웨어프로그래밍
2022년도 여름학기
Racin

1일차 내용

- 프로그래밍 환경을 구성해 보았어요.
- 간단한 C 프로그램을 만들며 몇 가지 스킬들을 확인해 보았어요.
 - 프로그램 밖 검은 창에 Data(양식을 적용한 메시지)를 담는 방법
 - 프로그램 밖 키보드로부터 Data(int **형식 값**)를 가져오는 방법
- Data와 Code의 관점에서 이런저런 단어들을 구경해 보았어요.
- 실습 하면서 이런저런 코드들을 작성해 보았어요.
 - **반복 실행**을 위한 while문도 슬쩍 등장함

오늘 내용

- 이번에는 '프로그램의 실행'에 초점을 맞추어,
다시 한 번 Data와 Code의 두 가지 관점으로 나누어 가며
프로그램이 내 목표에 맞게 실행되도록 구성하기 위해 알아 둘 요소들을
살짝 천천히 구경해 봅니다.

오늘 내용

- 이번에는 '프로그램의 실행'에 초점을 맞추어,
다시 한 번 Data와 Code의 두 가지 관점으로 나누어 가며
프로그램이 내 목표에 맞게 실행되도록 구성하기 위해 알아 둘 요소들을
살짝 천천히 구경해 봅시다.
 - Code 흐름
 - Data 흐름
- 오늘도 실습(이랑 최종 목표)이 있어요.
- (선언 이야기는 3일차로 미룰게요)

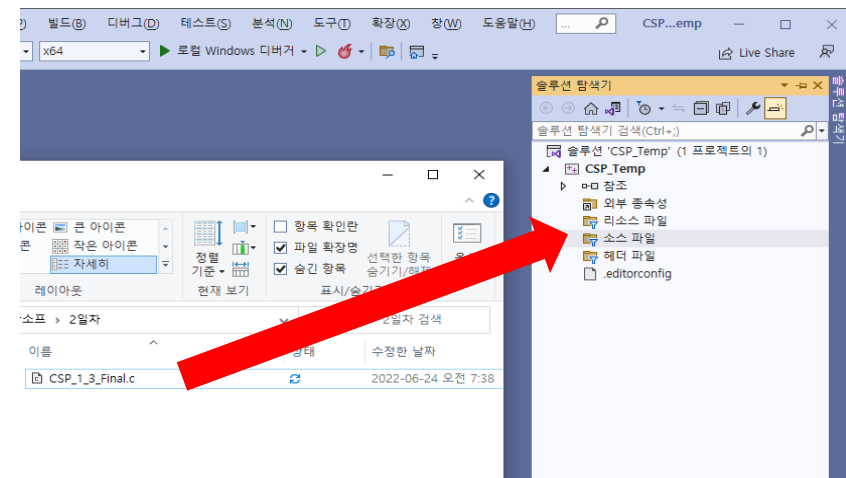
오늘 내용

- 일단 강사와 함께,
CSP_1_3_Final.c를 내 Visual Studio로 가져와 구경해 봅시다.
 - 다음 슬라이드에, 가져오는 방법을 적어 두었어요



예시 코드를 내 VS에 탑재하는 방법

- VS를 켜고, 연습용으로 쓸 프로젝트를 새로 만들거나 엽니다.
 - 이미 main.c 등이 들어 있다면 지워 주세요(이름 클릭 → Delete 키 → 제거 또는 삭제)
- 구경하고 싶은 예시 코드 파일을 드래그해서, 솔루션 탐색기에 보이는 '소스 파일'에 드롭합니다.
- 솔루션 탐색기에 그 파일이 올라가면 성공이에요
 - 이제 그 파일을 더블 클릭해서 연 다음
평소처럼 코드를 수정하거나 실행해 볼 수 있어요
- 다 보았으면 그 파일을 선택하고 Delete 키를 눌러 프로젝트에서 빼 줍니다.
 - 드래그해서 탑재했을 땐 프로젝트에서 빼도 파일은 원위치에 그냥 남아 있어요



오늘 내용

- 음... 저번부터 느낀 건데,
내 VS와 강사 것이 생김새가 좀 다른 것 같습니다.
 - 앞에 보이는 스크롤 바는 마치 '미니맵'처럼 코드 전체 내용을 보여주는데,
내 VS의 스크롤 바는 그냥 막대 하나만 달랑 있는 것 같아요
- 본격적으로 구경하기 전에,
강사가 미리 잡아 둔 버전으로 내 VS를 살짝 튜닝해 보도록 합시다.

VS 설정을 가져오는 방법



- 진행 순서:
 - 이전에 다운로드해서 압축 풀어 둔 ForCSP.vssettings가 어디 있는지 확인
 - VS에서 도구(T) → 설정 가져오기 및 내보내기(I)... 선택
 - 선택한 환경 설정 가져오기(I) 선택
 - (현재 설정을 백업해 두려면 하고, 아니면 스킵하면서) 다음(N) 선택
 - 찾아보기(B)... 선택 후 다운로드한 파일 선택
 - 뭔가 잘 모르겠으니 그냥 엔터 키 강타

VS 설정을 가져오는 방법



- 여기까지 진행했으면 스크롤 바가 미니맵 모드로 바뀌는 등 VS가 크고 작은 성형수술을 하게 될 거예요.
- 이제 Ctrl + F5를 누르면
녹색 화살표를 누른 것처럼 내 프로그램을 실행해 볼 수 있어요.
 - 그 외에 몇몇 단축키 설정이 강사 것과 동일하게 바뀌었어요
 - 이미 자신만의 설정으로 바꾸어 둔 친구들은 미안해요

오늘 내용

- 어느 정도 준비가 된 것 같으니,
CSP_1_3_Final.c를 같이 구경해 보고 옵시다.

구경 마무리

- **문장**들을 하나하나 구경했을 때는 그리 어렵지 않아 보입니다.
 - 그렇긴 하지만, 한국어로 적혀 있는 목표만 보고
백지 상태에서 이 **문장**들을 적어 내려가는 것은 어려울 거예요
 - 그래서, 미리 각을 좀 재어 놓고 시작하는 것이 정시퇴근에 좀 더 유리해요
- 그래서 오늘은 이 '각'을 보기 위해
우리가 알아 두어야 할 것들이 다수 등장할 예정이에요.

이번 시간에는

- Code 흐름
 - 우리 프로그램의 Code 부분을 구성하기 위한 보다 다양한 방법들
 - 조금 더 본격적으로 우리 프로그램의 실행 흐름을 구경하는 방법
 - vscode를 쓰는 친구들은 이 부분은 견학만 해야 할 듯
- 새로 나오는 키워드
 - **순차, 분기, 반복**
 - 시점 - 굵은 글씨는 아니지만 여러 곳에서 자주 쓰일 예정이에요
 - **위치**
 - **함수**

이번 시간에는

- 시작해 봅시다.

Code 흐름

- 우리는 VS에 있는 녹색 화살표를 눌러 프로그램을 실행할 수 있었습니다.
- 실행하면, 우리가 적어 둔 **문장**이 하나하나 **실행**돼요.
 - 이 때, 그 안에 어떤 **수식**을 적어 두었는지에 따라 서로 다른 작업을 수행하게 돼요
- 여기서 코드를 잠시 보면...

Code 흐름

- 지난 시간에 구경해 본 코드예요:

```
#include <stdio.h>

int main()
{
    printf("안녕하세요!\n");
    printf("안녕하지 않으세요?\n");

    return 0;
}
```

Code 흐름

- 지난 시간에 구경해 본 코드예요:

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

```
int main()  
{
```

```
    printf( "안녕하세요!\n" );
```

```
    printf( "안녕하지 않으세요?\n" );
```

```
    return 0;
```

```
}
```

프로그램을 실행하면 이 두 문장이 실행되었어요.

Code 흐름

- 지난 시간에 구경해 본 코드예요:

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

```
int main()  
{
```

```
    printf("안녕하세요!\n");
```

```
    printf("안녕하지 않으세요?\n");
```

```
    return 0;
```

```
}
```

프로그램을 실행하면 이 두 문장이 실행되었어요.
그 당시, 두 문장은 어떤 '순서'로 실행되었나요?

Code 흐름

- 지난 시간에 구경해 본 코드예요:

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

```
int main()  
{
```

```
    printf("안녕하세요!\n");
```

```
    printf("안녕하지 않으세요?\n");
```

```
    return 0;
```

```
}
```

맞아요. 우리가 언제 이 프로그램을 실행하든
이 두 문장들 중에 이 문장이 항상 '먼저' 실행될 거예요.

Code 흐름

- 지난 시간에 구경해 본 코드예요:

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

```
int main()  
{
```

```
    printf("안녕하세요!\n");
```

```
    printf("안녕하지 않으세요?\n");
```

```
    return 0;
```

```
}
```

맞아요. 우리가 언제 이 프로그램을 실행하든
이 두 문장들 중에 이 문장이 항상 '먼저' 실행될 거예요.
그러면, 왜 항상 먼저 실행될까요?

Code 흐름

- 지난 시간에 구경해 본 코드예요:

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

```
int main()  
{
```

```
    printf("안녕하세요!\n");
```

```
    printf("안녕하지 않으세요?\n");
```

```
    return 0;
```

```
}
```

여기서는
'그걸 위에 적어 놓았으니까' 라고 간단하게 말할 수 있을 거예요!
그리고 이게 Code (**실행**) 흐름을 구성하는 가장 중요한 원칙이에요.



순차

- **순차**(sequence 정도로 부를 수 있을 듯)
 - Code (**실행**) 흐름을 구성하는 가장 기본적인 방법이자 가장 중요한 원칙이에요

순차

- **순차**(sequence 정도로 부를 수 있을 듯)
 - Code (**실행**) 흐름을 구성하는 가장 기본적인 방법이자 가장 중요한 원칙이에요
- **문장**들은 기본적으로 '적어 둔 순서대로' **실행**돼요
 - `number = 3;`
`printf("%d", number);`
...순으로 적어 두면
일단 `number`에 3을 담아 둔 다음 `number`에 담긴 값 3의 10진수 표현을 출력해요
- VS는 프로그램을 만들 때
우리가 적은 **문장**들이 언제나 **순차**적으로 **실행**되도록 만들어 줘요
 - **순차** 개념이 깨진다는 것은, 달리 말하면
VS가 자기 마음대로 **실행** 흐름을 바꿔버리거나, 더 심하면,
매 프로그램 실행마다 **실행** 흐름이 뒤섞인다는 의미가 될 거예요 → 그러면 안 됨!

Code 흐름

- 반면, 프로그래밍 목표가 무엇인지에 따라 **실행** 흐름에 여러 가능성을 두어야 하는 상황도 나올 수 있을 거예요.
 - 예를 들면, 지난 시간 최종 목표의 경우
실행 도중 사용자가 입력한 **값**이 무엇인지에 따라
몇 줄을 출력해야 하는지가 결정되었어요
 - 프로그래밍을 실행하기 전 시점에는 나중에 사용자가 무얼 입력할 것인지 특정할 수 없어요
 - 그럼에도 불구하고, 어떤 (양수) **값**을 입력하든
그에 맞는 적절한 삼각형/마름모를 출력하는 것이 목표였어요

Code 흐름

- 반면, 프로그래밍 목표가 무엇인지에 따라 **실행** 흐름에 여러 가능성을 두어야 하는 상황도 나올 수 있을 거예요.
 - 예를 들면, 지난 시간 최종 목표의 경우
실행 도중 사용자가 입력한 **값**이 무엇인지에 따라
몇 줄을 출력해야 하는지가 결정되었어요
 - 프로그래밍을 실행하기 전 시점에는 나중에 사용자가 무얼 입력할 것인지 특정할 수 없어요
 - 그럼에도 불구하고, 어떤 (양수) **값**을 입력하든
그에 맞는 적절한 삼각형/마름모를 출력하는 것이 목표였어요
- 추가로, 간단한 프로그램 하나를 함께 만들어 보면서
생각을 살짝 넓혀 봅시다.

입력한 수의 절대값을 출력하는 프로그램

- 목표: 입력한 수의 절대값을 출력하는 프로그램
 - 사용자에게 안내(보통 이럴 때 출력하는 메시지를 prompt라 불러요)
 - int 값 하나 받아 담아 두기
 - 담아 둔 값에 대한 절대값을 계산해 출력하기
- 강사와 함께 살짝 시도해 보고 옵시다.

Code 흐름

- 그럴싸하지요?
 - 프로그램 실행 도중 사용자가 음수를 입력했다면 부호를 바꾸어 출력해야 했어요
 - 근데 녹색 화살표를 누르기 전에는 사용자가 무슨 수를 입력할 것인지 알 수 없어요
 - 따라서, 일단 입력받은 다음 그 값이 음수인지 확인하고,
음수가 맞다면 '부호를 바꾸어 출력하는 문장'을 실행하고,
음수가 아니라면 '부호를 바꾸지 않고 출력하는 문장'을 실행하도록 구성해 보았어요

Code 흐름

- 그럴싸하지요?
 - 프로그램 실행 도중 사용자가 음수를 입력했다면 부호를 바꾸어 출력해야 했어요
 - 근데 녹색 화살표를 누르기 전에는 사용자가 무슨 수를 입력할 것인지 알 수 없어요
 - 따라서, 일단 입력받은 다음 그 값이 음수인지 확인하고,
음수가 맞다면 '부호를 바꾸어 출력하는 문장'을 실행하고,
음수가 아니라면 '부호를 바꾸지 않고 출력하는 문장'을 실행하도록 구성해 보았어요
- 방금 사용한 방법을 프로그래밍에서는 **분기(branch)**라 불러요!

분기

- **분기(branch)**
 - 기본적인 (순차) 실행 흐름에 변화를 주기 위한 방법이에요
 - 분기는 어떤 문장을 실행할 수도, 안 할 수도 있게 해 줘요
 - 문장의 일종인 if문을 적음으로써 **분기** 개념을 내 프로그램에 도입할 수 있어요
 - 다음 슬라이드에서 살짝 정리해 볼게요

분기

- if문
 - **문장**의 일종이에요
 - if문을 적을 때는...
 - 일단 if 키워드를 적고,
(를 적고, 수식을 하나 적고,) 를 적은 다음,
문장 하나를 적어요
 - 원한다면, 위 내용까지 적은 다음 'else부분(clause, 절)'을 덧붙여 적을 수도 있어요
 - else 키워드를 적고, **문장** 하나를 적어요
 - 이에 대비하여 if문의 앞 부분을 'if부분'이라 부를 수도 있어요
 - Python이랑 다르게 C에는 'elif부분'은 없어요

분기

- if문
 - 문장의 일종이에요
 - if문의 실행은...
 - if부분 괄호 안에 적혀 있는 수식을 계산해요
 - 계산한 결과값이 0(C++에서는 false)에 준하는 경우, if부분에 적혀 있는 **문장**(if부분의 내용물)을 **실행**하지 않아요
 - 이 때, 그 if문에 else부분이 있다면 else부분 내용물을 **실행**해요
 - 계산한 결과값이 0에 준하지 않는 경우(C++에서는 true에 준하는 경우), if부분 내용물을 **실행**해요
 - else부분이 있다면 else부분 내용물은 **실행**하지 않아요

분기

- if문

- 문장의 일종이에요

- if문의 실행은...

- if부분 괄호 안에 적혀 있는 수식을 계산해요

- 계산한 결과값이 0(C++ if부분에 적혀 있는 문장)
 - 이 때, 그 if문에 else부

여기서 적는 수식은
내용물을 어떨 때 실행할 것인지 지정하는 셈이 돼요.
그래서 이런 수식을 보통 조건식(condition)이라 불러요.

- 계산한 결과값이 0에 준하지 않는 경우(C++에서는 true에 준하는 경우),
if부분 내용물을 실행해요
 - else부분이 있다면 else부분 내용물은 실행하지 않아요

분기

- if문
 - **문장**의 일종이에요
 - if문을 적어 어떤 **문장**을 **실행**하거나 **실행**하지 않게 만들 수 있어요
 - 그 **문장**을 if문의 내용물로서 적으면 돼요

분기

- if문
 - **문장**의 일종이에요
 - if문을 적어 어떤 **문장**을 **실행**하거나 **실행**하지 않게 만들 수 있어요
 - 그 **문장**을 if문의 내용물로서 적으면 돼요
 - 이 때, 여러 **문장**들을 **실행**하거나 **실행**하지 않게 만들고 싶다면 그 **문장**들을 { }로 둘러 싸 한 덩어리로 취급하면 돼요
 - 좀 더 정밀하게는 { }로 둘러 씌으로써 커다란 **문장** 하나를 적을 수 있어요
 - 이런 **문장**을 정식 명칭으로는 compound statement라 불러요
(우리 수업에선 그냥 '(내용물) **문장**들'이라 부를 예정이에요)

분기

- **분기(branch)**
 - 기본적인 (순차) **실행** 흐름에 변화를 주기 위한 방법이에요
 - **분기**는 어떤 **문장**을 **실행**할 수도, 안 할 수도 있게 해 줘요
 - **문장**의 일종인 if문을 적음으로써 **분기** 개념을 내 프로그램에 도입할 수 있어요
 - '이번에' 내용물 **문장**을 **실행**할지 안 할지는 조건식 **계산 결과값**에 의해 결정돼요

Code 흐름

- 여기까지 본다면 이제 **실행** 흐름을 구성하는 마지막 세 번째 방법인 **반복**(loop, iteration)도 빠르게 정리해볼 수 있을 것 같아요.
 - 지난 시간 실습에서 미리 사용해 봤어요

반복

- **반복**(loop, iteration)
 - 기본적인 **(순차) 실행** 흐름에 변화를 주기 위한 방법이에요
 - **반복**은 어떤 **문장**을 여러 번 **실행**할 수도, 아예 안 할 수도 있게 해 줘요
 - **문장**의 일종인 while문을 적음으로써 **반복** 개념을 내 프로그램에 도입할 수 있어요
 - if문 설명용 슬라이드를 복붙해서 조금만 고쳐 구경해 볼게요

반복

- while문
 - 문장의 일종이에요
 - while문을 적을 때는...
 - 일단 while 키워드를 적고,
(를 적고, 수식을 하나 적고,) 를 적은 다음,
문장 하나를 적어요
 - Python과 다르게 C의 while문에는 else부분을 덧붙일 수 없어요

반복

- while문
 - 문장의 일종이에요
 - while문의 **실행**은...
 - 조건식을 **계산**해요
 - **계산**한 결과값이 0(C++에서는 false)에 준하는 경우, 내용물을 **실행**하지 않아요(여기서 while문 **실행** 끝)
 - **계산**한 결과값이 0에 준하지 않는 경우(C++에서는 true에 준하는 경우), 내용물을 **실행**해요
 - 내용물 **실행**이 끝나면 조건식 **계산** 순서로 '돌아가'요

반복

- while문
 - 문장의 일종이에요
 - while문을 적어 어떤 **문장**을 0번 또는 여러 번 **실행**하게 만들 수 있어요
 - 그 **문장**을 while문의 내용물로서 적으면 돼요
 - 물론 이 때도 { } 써서 여러 **문장**들을 한 덩어리로 취급할 수 있어요

반복

- **반복**(loop, iteration)
 - 기본적인 (순차) **실행** 흐름에 변화를 주기 위한 방법이에요
 - **반복**은 어떤 **문장**을 여러 번 **실행**할 수도, 아예 안 할 수도 있게 해 줘요
 - **문장**의 일종인 while문을 적음으로써 **반복** 개념을 내 프로그램에 도입할 수 있어요
 - '내가 원하는 횟수만큼' **반복 실행**하는 Code 흐름을 만들고 싶을 때, 보편적으로 적용할 수 있는 일종의 패턴이 존재해요
 - 다음 슬라이드에서 요약해 볼게요

반복

- '원하는 횟수만큼 **반복**'하고 싶을 때는 아래 세 부분을 잘 적어 주어야 해요:
 - 1. 첫 **반복** 준비
 - 2. 언제 **반복**을 그만 두어야 하는지 지정
 - (이번 **반복** 수행)
 - 3. 다음 **반복** 준비
- 수학시간에 얼핏 들은 '수학적 귀납법'이 생각났다면 좋고, 그렇지 않더라도 아까 구경한 예시 정도는 무난히 납득할 수 있을 거예요.
 - 각 부분들을 구성하기 위한 **문장**들이 **순차적으로** 잘 **실행**되도록 .c 파일의 적당한 곳에 적어주면 돼요

Code 흐름

- 여기까지, Code 흐름을 구성하기 위한 세 가지 방법을 정리해 봤어요.
 - 듣고 보니 뭐 별로 어렵지 않은듯

실행중인 프로그램 관찰



- 여기까지, Code 흐름을 구성하기 위한 세 가지 방법을 정리해 봤어요.
 - 듣고 보니 뭐 별로 어렵지 않은듯
- 듣기만 해서는 감이 잘 안 올 수 있으니,
VS에 있는 '디버그' 기능을 사용해서
우리 프로그램의 실행을 보다 직관적으로 관찰해 봅시다.

실행중인 프로그램 관찰



- 연습용 프로젝트를 열고, main.c에 아래 내용을 복붙해 보시다:

```
#include <stdio.h>

int main() {
    int number;
    int sum;
    printf("합을 구할 자연수를 입력하세요>");
    scanf("%d", &number);
    sum = 0;
    while ( number > 0 ) {
        sum = sum + number;
        number = number - 1; }
    printf("합은 %d입니다.", sum);
    return 0; }
```

실행중인 프로그램 관찰



- 연습용 프로젝트를 열고, main.c에 아래 내용을 복붙해 봅시다:

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

```
int main()  
{
```

```
    int number;
```

```
    int sum;
```

```
    printf("합을 구할 자연수를 입력하세요>");
```

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

```
    sum = 0;
```

```
    while ( number > 0 )
```

```
{
```

```
        sum = sum + number;
```

```
        number = number - 1;
```

```
}
```

```
    printf("1부터 %d까지의 자연수들을 더한 합은 %d입니다.", sum);
```

```
    return 0;
```

```
}
```

오... 그냥 복붙했을 뿐인데
VS가 적당히 보기 쉬운 형태로 우리 코드를 고쳐 주었을 거예요.
(이전에도 닫는 중괄호, ; 등등을 적을 때도 살짝 고쳐주곤 했어요)

실행중인 프로그램 관찰



- 아무튼, 이 코드를 적어 둔 상태에서 F10을 눌러 봅시다.

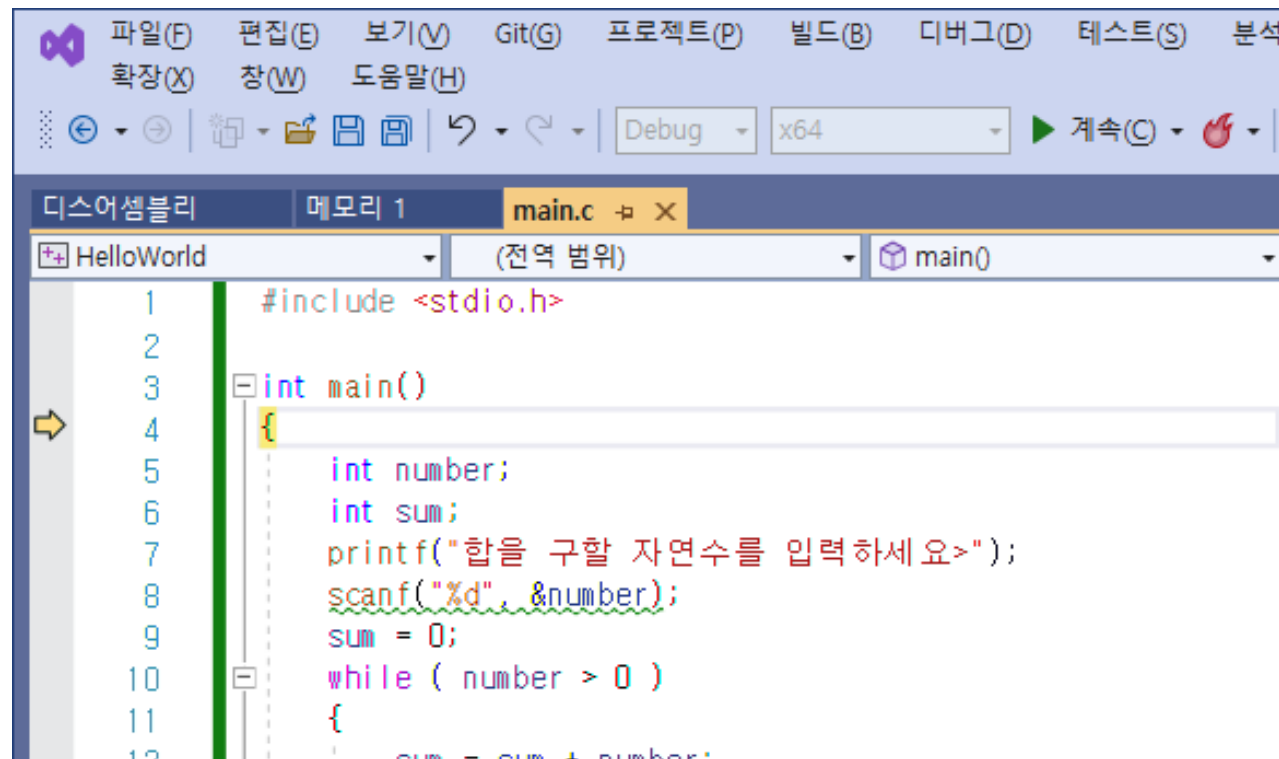
```
#include <stdio.h>

int main()
{
    int number;
    int sum;
    printf("합을 구할 자연수를 입력하세요>");
    scanf("%d", &number);
    sum = 0;
    while ( number > 0 )
    {
        sum = sum + number;
        number = number - 1;
    }
    printf("1부터 %d까지의 자연수들을 더한 합은 %d입니다.", sum);
    return 0;
}
```



실행중인 프로그램 관찰

- 아무튼, 이 코드를 적어 둔 상태에서 F10을 눌러 봅시다.
 - VS 생김새가 이런 느낌으로 바뀌면 성공입니다!



```
#include <stdio.h>

int main()
{
    int number;
    int sum;
    printf("합을 구할 자연수를 입력하세요>");
    scanf("%d", &number);
    sum = 0;
    while ( number > 0 )
    {
        sum = sum + number;
    }
}
```

실행중인 프로그램 관찰



- 아무튼, 이 코드를 적어 둔 상태에서 F10을 눌러 봅시다.
 - VS 생김새가 이런 느낌으로 바뀌면 성공입니다!

프로그램의 의도치 않은 오류를 보통 '버그'라 부르고,
이러한 버그를 찾아내기 위해 뒤적이는 작업을 '디버그'라 불러요.

우리가 방금 누른 F10은
디버그 모드로 내 프로그램의 실행을 시작하는 방법들 중 하나예요.

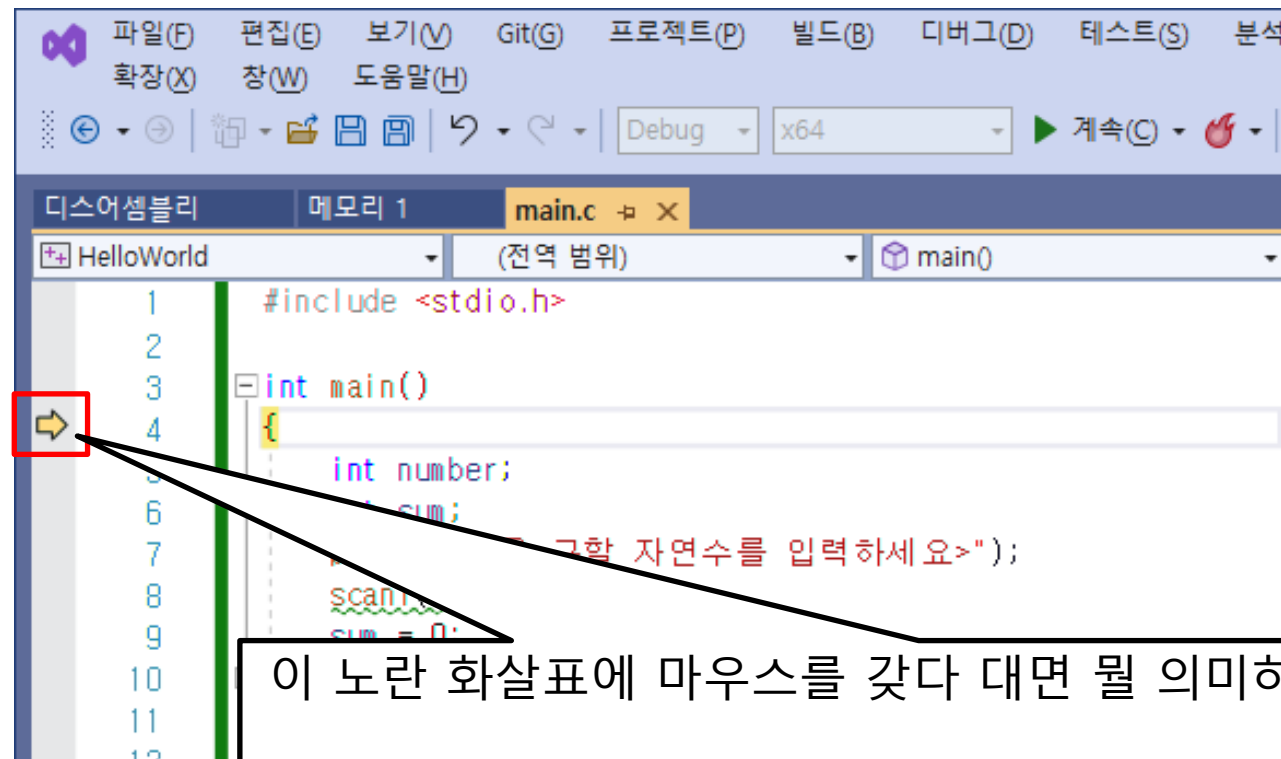
(여러분 컴퓨터에서는 디버깅을 이미 시작한 상태라
지금 이 메뉴를 누르면 살짝 다른 것들이 나올 거예요)





실행중인 프로그램 관찰

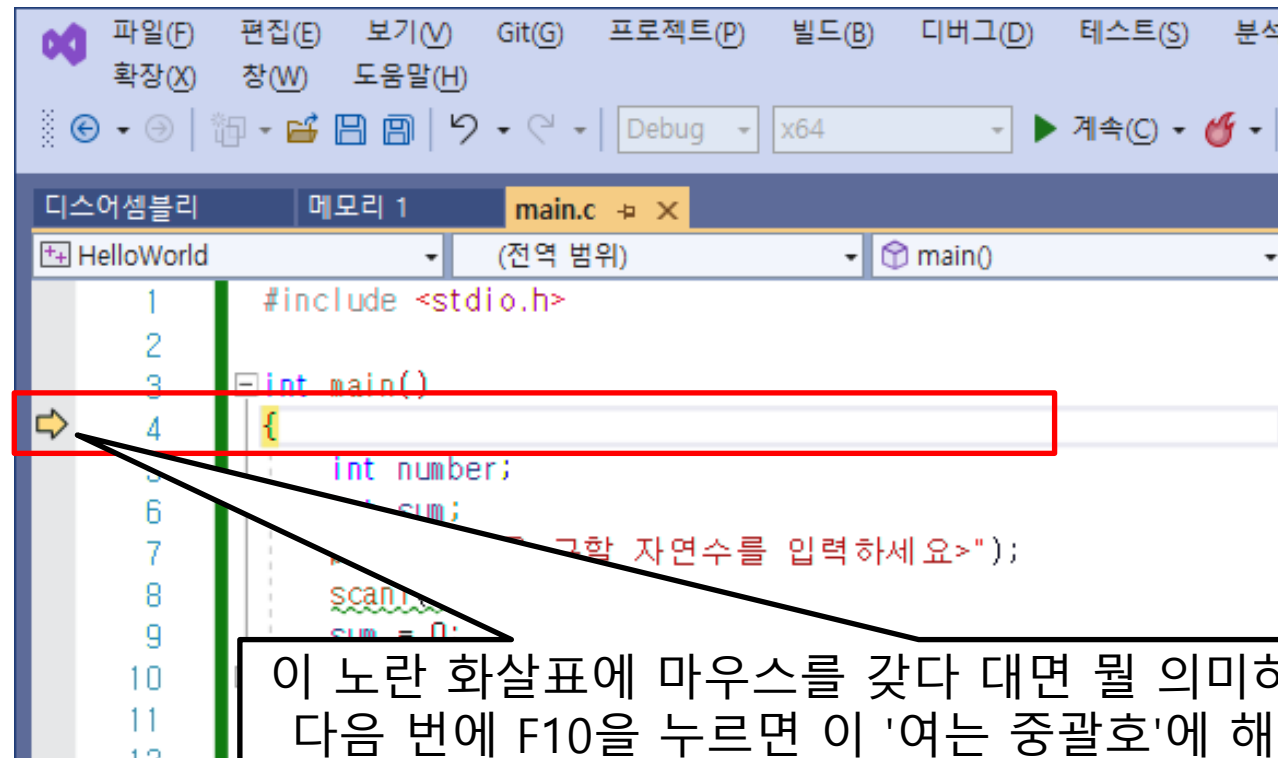
- 아무튼, 이 코드를 적어 둔 상태에서 F10을 눌러 봅시다.
 - VS 생김새가 이런 느낌으로 바뀌면 성공입니다!





실행중인 프로그램 관찰

- 아무튼, 이 코드를 적어 둔 상태에서 F10을 눌러 봅시다.
 - VS 생김새가 이런 느낌으로 바뀌면 성공입니다!



직접 눌러 볼까요?

실행중인 프로그램 관찰



- 아무튼, 이 코드를 적어
- VS 생김새가 이런 느낌

노란 화살표가 '내려갔'어요.
이 위에 적혀 있는 모든 내용들이 다 **실행**된 상태예요.

지금 시점에 Alt + Tab을 누르면 '아무 것도 안 적힌 검은 창'이 보일 것이고,
이제 F10을 누르면 아마 화면에 이 메시지가 출력될 듯 해요.
F10을 한 번 눌러 직접 확인해 봅시다.

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

```
1 #include <stdio.h>
2
3 int main()
4 {
5     int number;
6     int sum;
7     printf("합을 구할 자연수를 입력하세요>");
8     scanf("%d", &number);
9     sum = 0;
10    while ( number > 0 )
11    {
12        sum = sum + number;
```

A yellow arrow points to line 7, which contains the `printf` statement. The line is highlighted in yellow. The right side of the IDE shows the 'main()' function being executed, with a status bar indicating '경과 시간 1ms 이하'.



실행중인 프로그램 관찰

- 아무튼, 이 코드를 적어 둔 상태에서 F10을 눌러 봅시다.
 - VS 생김새가 이런 느낌으로 바뀌면 성공입니다!

화살표는 다음 **문장** 자리로 내려갔고,
실제로 검은 창에는 메시지가 잘 출력되어 있어요.

이제 이 **문장**을 **실행**하면 키보드 입력을 '받기 시작'할 것이고,
검은 창에서 3 치고 엔터 키를 치면 이 **문장**의 **실행**이 끝나게 될 거예요.
F10을 한 번 눌러 직접 확인해 봅시다.

```
1 int main()  
{  
2     int number;  
3     int sum;  
4     printf("한을 구할 자연수를 입력하세요>");  
5     scanf("%d", &number);  
6     sum = 0;  
7     while ( number > 0 )  
8     {  
9         sum = sum + number;  
10    }
```

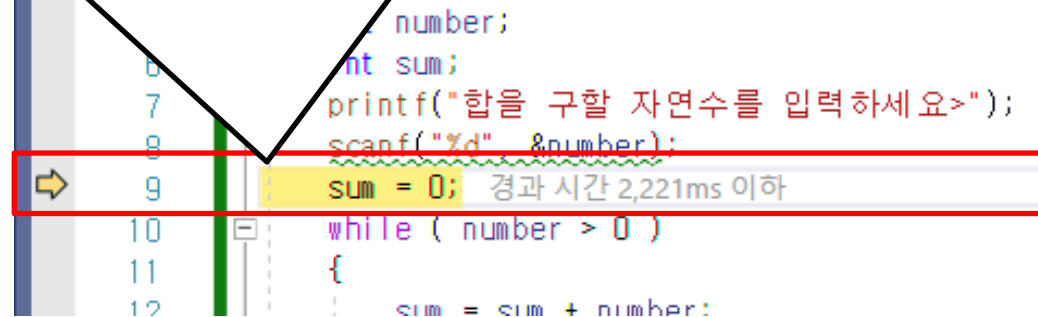


실행중인 프로그램 관찰

- 아무튼, 이 코드를 적어 둔 상태에서 F10을 눌러 봅시다.
 - VS 생김새가 이런 느낌으로 바뀌면 성공입니다!

3 누르고 엔터 키를 쳤더니 실제로 **문장 실행**이 잘 끝났고,
노란 화살표는 저 아래까지 내려가 있어요

지금 시점에, **변수** number 자리에 방금 담은 3이 잘 들어가 있을까요?
다음 슬라이드에서 이를 확인하는 방법을 살펴 볼게요

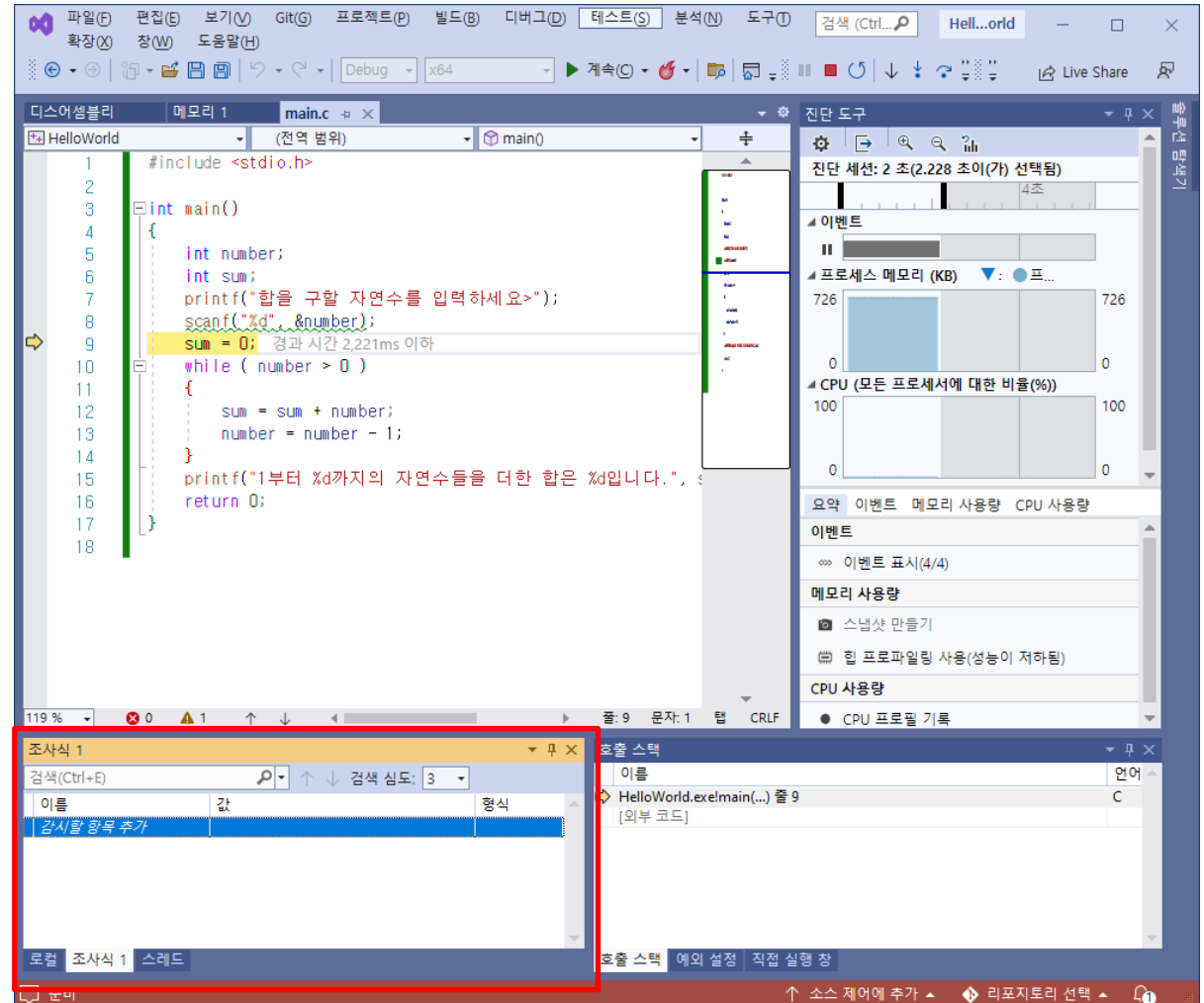


```
1 int number;  
2  
3 int sum;  
4  
5 printf("합을 구할 자연수를 입력하세요>");  
6  
7 scanf("%d", &number);  
8  
9 sum = 0; 경과 시간 2,221ms 이하  
10 while ( number > 0 )  
11 {  
12     sum = sum + number;
```

실행중인 프로그램 관찰



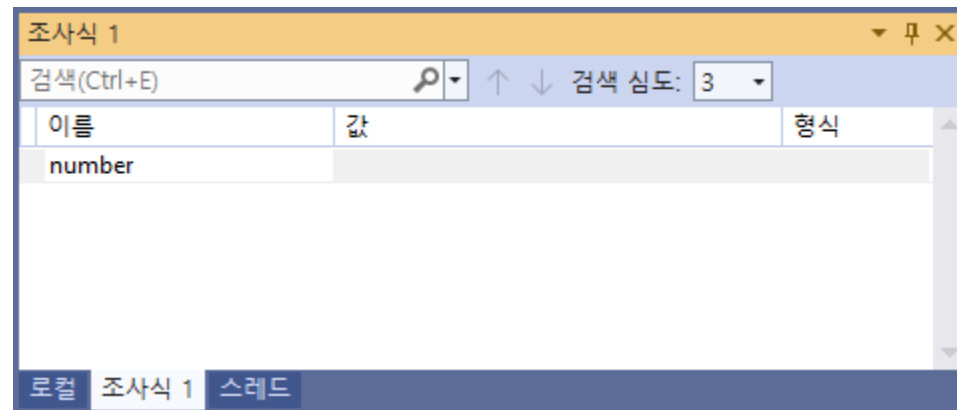
- 코드 창 아래에 있는 '조사식 1' 탭을 봅시다.
 - 만약 안 보인다면
디버그(D)
→ 창(W)
→ 조사식(W)
→ 조사식 1(1)을 눌러 주세요



실행중인 프로그램 관찰



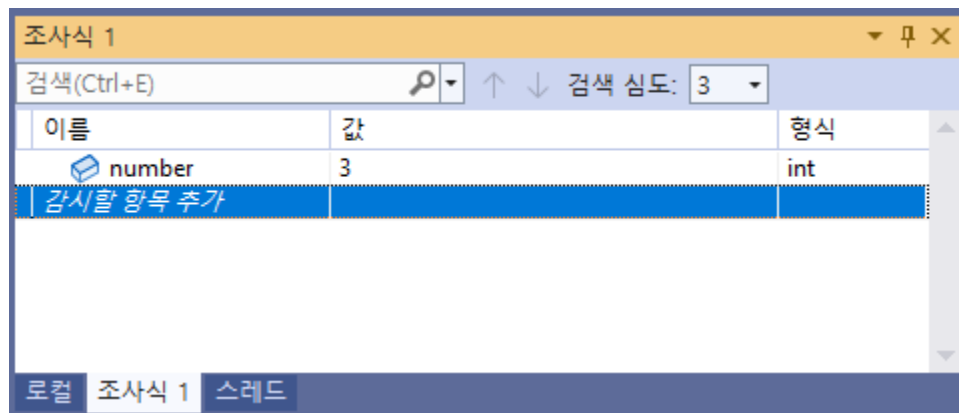
- 조사식 탭의 '이름' 자리에 내가 **계산해** 보고 싶은 수식을 적고 엔터를 치면...



실행중인 프로그램 관찰



- 조사식 탭의 '이름' 자리에 내가 **계산**해 보고 싶은 **수식**을 적고 엔터를 치면...
그 **수식**을 '지금 당장' **계산**했다면 얻을 수 있는 **결과값**이 무엇인지,
그리고 그 **수식**의 **형식**은 무엇인지를 볼 수 있어요!
 - 실제로 '지금 시점에' **변수** number 자리에는 int **형식 값** 3이 잘 들어가 있는 것 같아요
 - 이건 비밀인데, 이 탭을 써서 저 3을 몰래 5로 바꿀 수도 있긴 해요(3을 더블클릭해 봐요)





실행 중인 프로그램 관찰

- 뭔가 엄청 긴 결과가 튀어나오고, 그 옆에 3이 빼꼼 적혀 있습니다.
 - 수식 `&number`는 연산자가 하나 더 붙어 있어서 그런지
수식 `number`와는 다른 형식, 다른 값이 나오는 수식인 것 같습니다
- 오른쪽 형식 자리에는 `int *`이 적혀 있는 것 같기는 한데,
일단 이건 지금은 넘어갑시다

이름	값	형식
number	3	int
&number	0x000000cf8c0ffa94 {3}	int *

감시할 항목 추가

실행중인 프로그램 관찰



- 방금 본 그 긴 **값**은, **변수** number의 자리가 어디인지(int **값** 하나를 메모리 위의 어디에 담을 것인지)를 의미하는 **값**이었어요.
 - 우리 수업에서는 이러한 **값**을 **변수** number에 대한 '**위치 값**'이라 부르려 해요
 - number 자리에 3을 담는다는 것은, 메모리 위의 딱 저 위치에 3을 담는 것을 의미해요

실행중인 프로그램 관찰

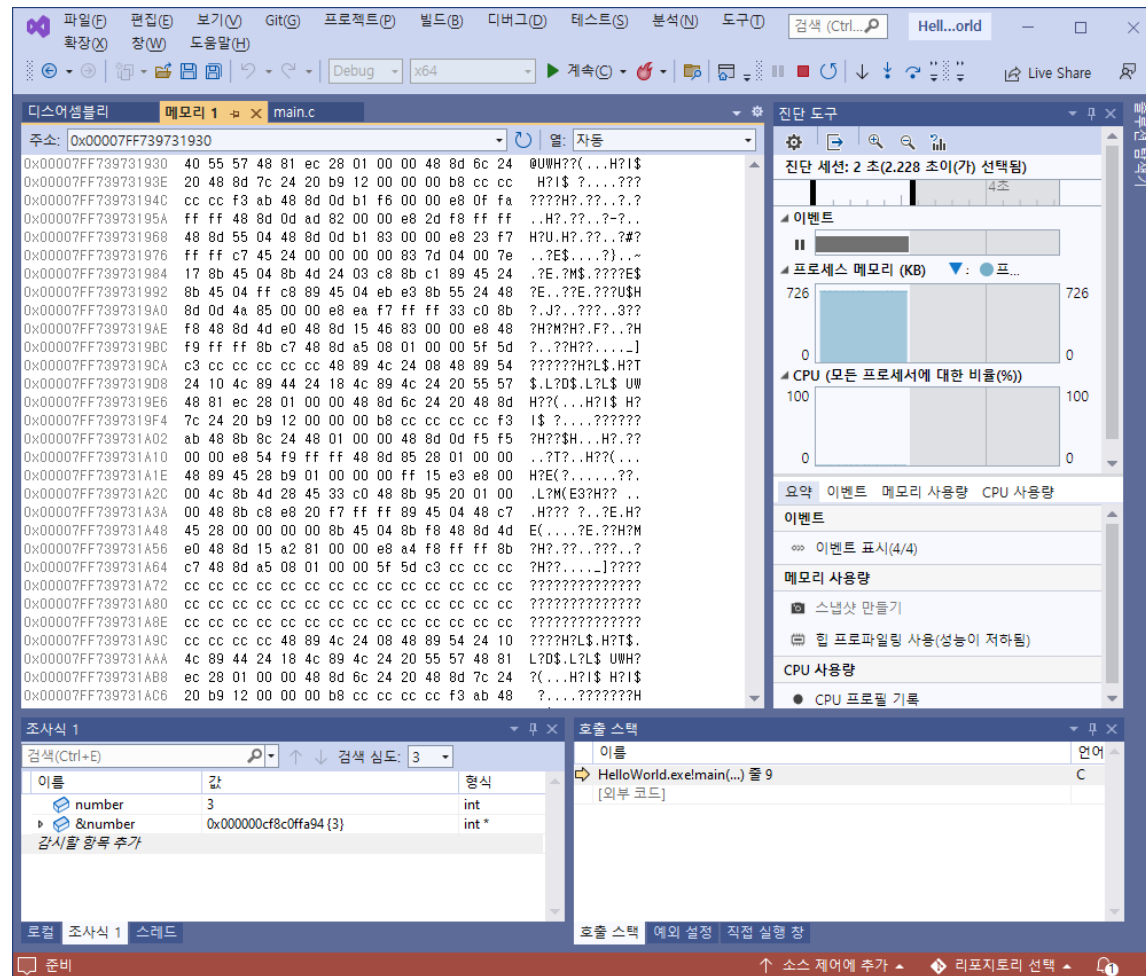


- 방금 본 그 긴 값은, 변수 number의 자리가 어디인지(int 값을 하나를 메모리 위의 어디에 담을 것인지)를 의미하는 값이었어요.
 - 우리 수업에서는 이러한 값을 변수 number에 대한 '위치 값'이라 부르려 해요
 - number 자리에 3을 담는다는 것은, 메모리 위의 딱 저 위치에 3을 담는 것을 의미해요
- 내친 김에, '메모리'도 직접 한 번 구경해 보도록 합시다!
 - 코드 창의 내 코드 파일 이름 옆에 '메모리 1' 탭이 있다면 눌러 보세요
 - 만약 안 보인다면
 - 디버그(D)
 - 창(W)
 - 메모리(M)
 - 메모리 1(1)을 눌러 주세요

실행중인 프로그램 관찰



- 이런 식으로 뭔가 엄청나게 많은 숫자들이 보이면 성공입니다!
 - 프로그램이 현재 실행중일 땐 안 보여요
- 가끔 정지중에도 안 보일 수 있어요
 - 이 경우에는 조사식 몇 개를 지우면 보이기 시작할 거예요



실행중인 프로그램 관찰

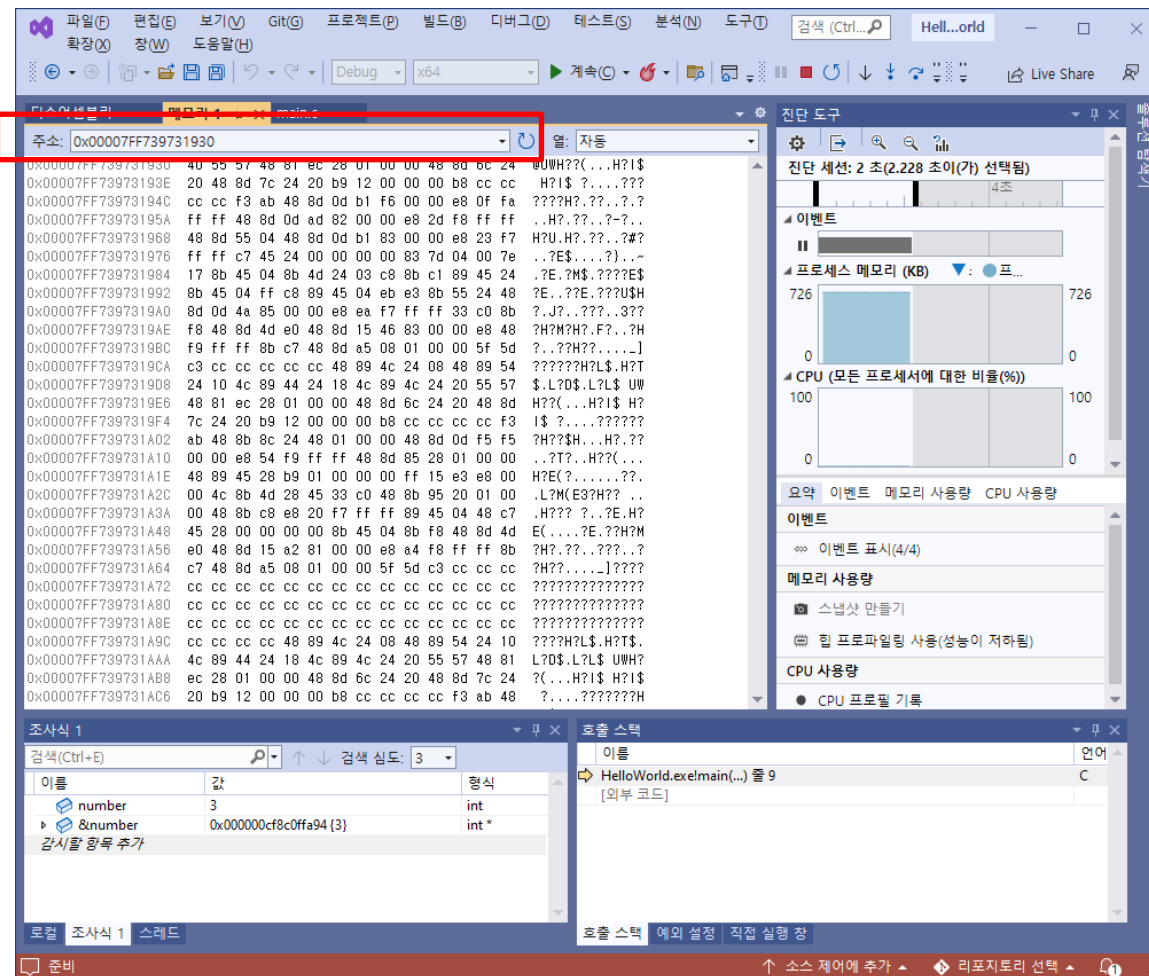


- 이런 식으로 뭔가 엄청나게 많은 숫자들이 보이면 성공입니다!

- 프로그램이 현재 실행중일 땐 안 보여요

- 가끔 정지중에도 안 보일 수 있어요
 - 이 경우에는 조사식 몇 개를 지우면 보이기 시작할 거예요

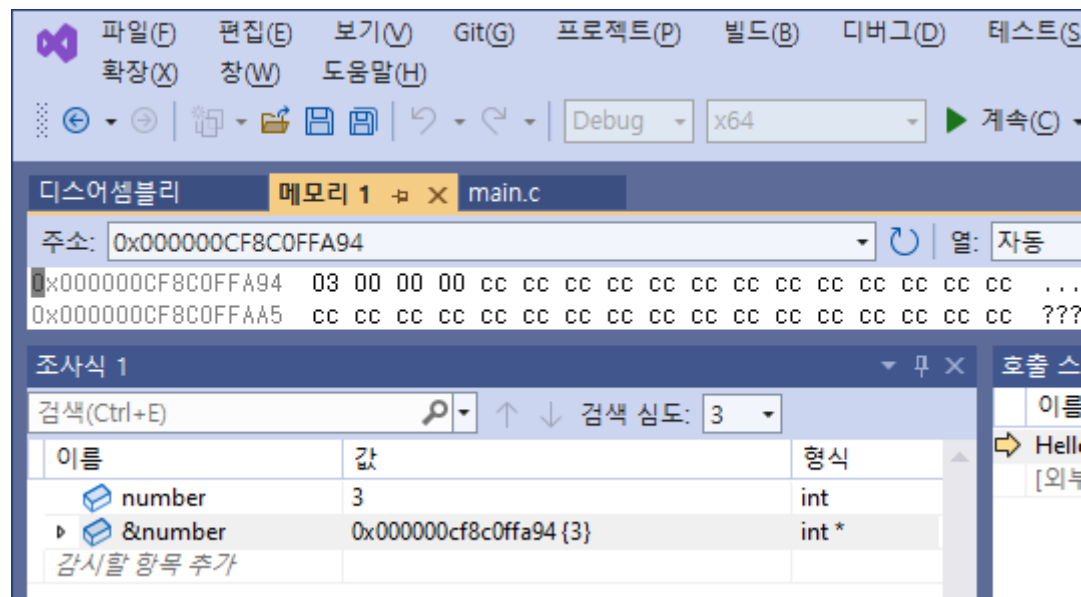
- 여기 있는 '주소' 칸에,
아래 보이는 &number 값을 적어 봅시다





실행중인 프로그램 관찰

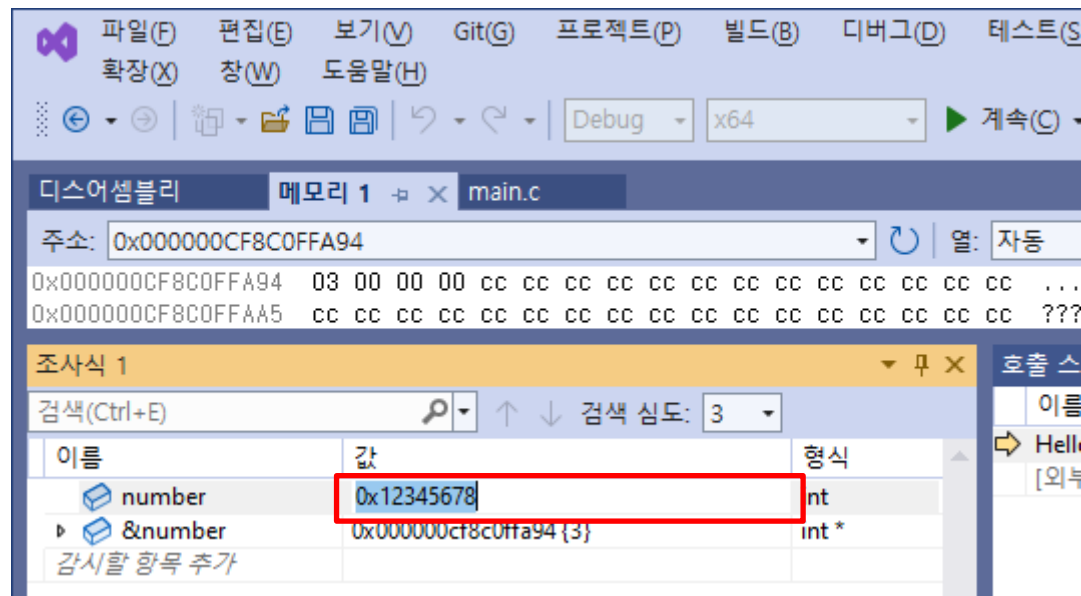
- 오오... 정말로 03이 들어 있습니다!





실행중인 프로그램 관찰

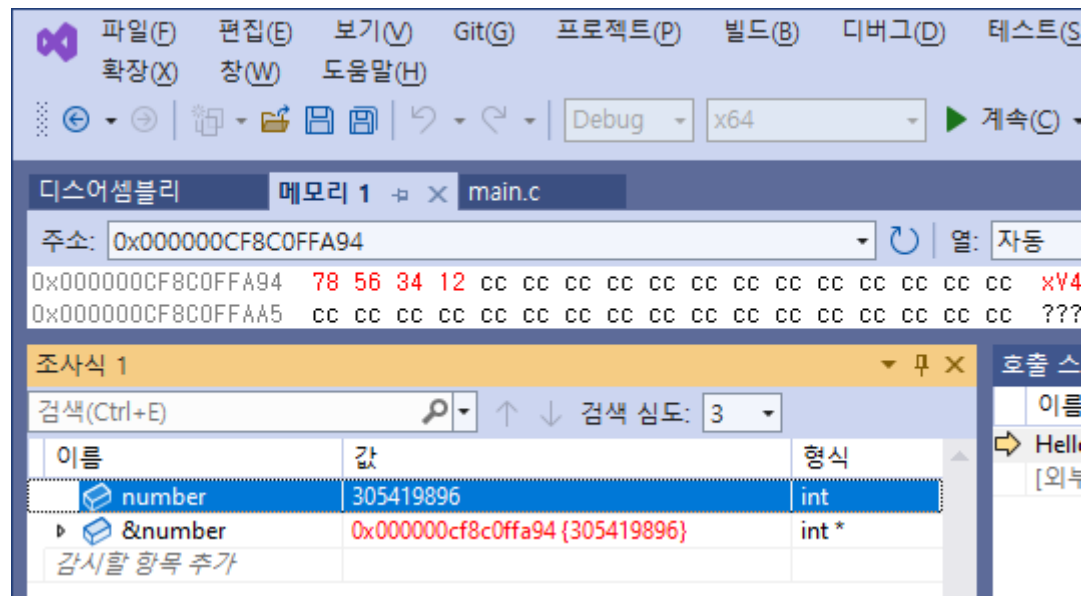
- 오오... 정말로 03이 들어 있습니다!
 - 조사식 탭 써서 3을 0x12345678로 바꾸고 엔터 키를 친다면?





실행중인 프로그램 관찰

- 오오오오..... 78 56 34 12로 바뀌었습니다?



실행중인 프로그램 관찰



- 일단 이 썸에서 잠시 정리하면...
 - F10을 눌러 가면서 Code를 한 단계씩 **실행** 가능
 - 주로 **문장** 단위로 움직일 거예요(이 코드의 '선언'에는 노란 화살표가 걸리지 않았어요)

실행중인 프로그램 관찰



- 일단 이 썸에서 잠시 정리하면...
 - 조사식 탭을 쓰면 어떤 수식을 '여기에 적혀 있었다 치고' 즉석에서 **계산**해 볼 수 있음
 - 주로 Data를 관찰하는 데 쓰여요
 - 뭔가 변화가 느껴지면 VS가 빨간 색으로 강조해 주는 듯 해요
 - 사실 $3 + 5$ 같은 것도 충분히 가능해용
 - **변수** 이름 등을 적어 둔 경우, 그 **변수** 자리에 담긴 **값**을 바꿔치기할 수도 있는 것 같아요
 - &number **값**은 더블 클릭해도 바꿔치기가 되지 않아요
 - 메모리 탭을 쓰면 Data들을 있는 그대로 관람 가능
 - 다만 적는 순서가 사람이랑 반대인 듯 해요
 - 사실 우클릭해서 '4바이트 정수' 누르면 사람이 보기 편하게 줄을 다시 맞춰주긴 해요

실행중인 프로그램 관찰



- 이제 잠시 시간을 들여서,
코드 창에서 F10을 조금씩 눌러 가면서,
노란 화살표 아래에 기다리고 있을 **문장**을, while문의 내용물 **문장**들을
조금씩 잘근잘근 **실행**해 봅시다.
 - **실행**해 보면서 조사식 및 메모리 창이 우리가 예상하는 대로 변하고 있는지,
반복 흐름이 잘 돌고 있는지 직접 확인해 봅시다

실행중인 프로그램 관찰

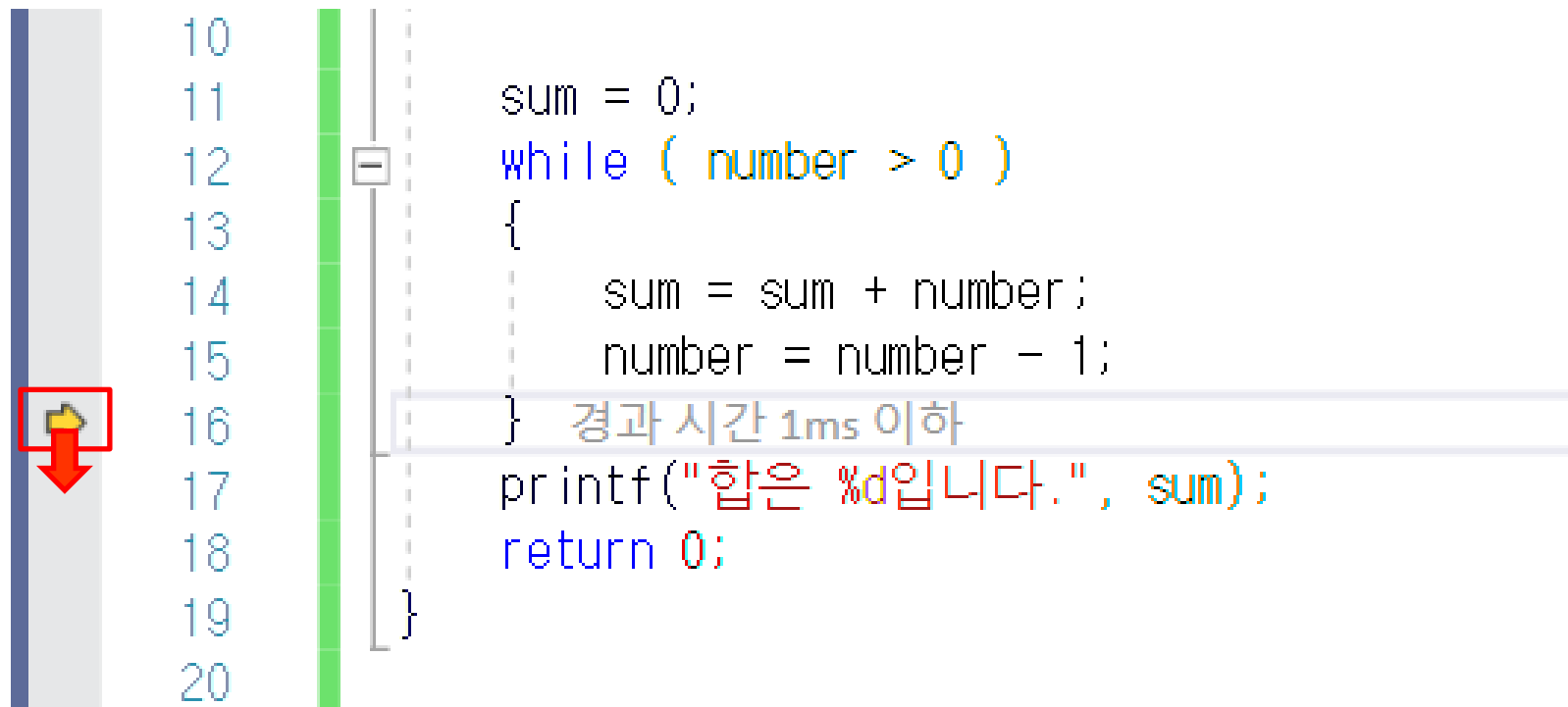


- 아마 우리 수업 흐름을 잘 따라 왔다면
F10을 약 9억 번 정도 눌러야 반복이 끝날 거예요.



실행중인 프로그램 관찰

- 아마 우리 수업 흐름을 잘 따라 왔다면
F10을 약 9억 번 정도 눌러야 반복이 끝날 거예요.
- 그렇다간 집에 갈 수 없게 될 테니,
노란 화살표를 직접 누른 채로 중괄호 아래 **문장** 자리에 드래그 & 드롭해 줘시다





실행중인 프로그램 관찰

- 맞아요. 실행중인 프로그램의 Data를 '조작'할 수 있었던 것처럼, Code **실행** 흐름 또한 우리 마음대로 직접 조작할 수 있어요!
 - 이 상태에서 F10을 누르면 '현재까지 합산된 (매우 이상한) sum 값'이 출력될 거예요!
 - 직접 확인해 봅시다!

```
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20
```

```
sum = 0;  
while ( number > 0 )  
{  
    sum = sum + number;  
    number = number - 1;  
} 경과 시간 1ms 이하  
printf("합은 %d입니다.", sum);  
return 0;  
}
```



실행중인 프로그램 관찰

- 형언하기 힘든 값이 포함된 메시지가 보이고,
어느덧 우리는 마지막 'return문'을 남겨두고 있는 상태가 되었어요.

```
14 sum = sum + number;  
15 number = number - 1;  
16 }  
17 printf("합은 %d입니다.", sum);  
18 return 0;  
19 }  
20
```

경과 시간 3ms 이하

C:\Users\Wracin\Documents\Visual Studio 2019\Projects\...
합을 구할 자연수를 입력하세요>3
합은 1527099470입니다.



실행중인 프로그램 관찰

- 형언하기 힘든 값이 포함된 메시지가 보이고,
어느덧 우리는 마지막 'return문'을 남겨두고 있는 상태가 되었어요.
 - 이 시점에서 F10을 두 번만 눌러 남은 Code들을 모두 **실행**해 봅시다

```
14         sum = sum + number;  
15         number = number - 1;  
16     }  
17     printf("합은 %d입니다.", sum);  
18     return 0; 경과 시간 3ms 이하  
19 }  
20
```

C:\Users\Wracin\Documents\Visual Studio 2019\Projects\2-1. Code 흐름\2-1. Code 흐름\2-1. Code 흐름.exe
합을 구할 자연수를 입력하세요>3
합은 1527099470입니다.

실행중인 프로그램 관찰



- 형언하기 힘든 값이 포함된 메시지가 보이고,
어느덧 우리는 마지막 'return문'을 남겨두고 있는 상태가 되었어요.
 - 이 시점에서 F10을 두 번만 눌러 남은 Code들을 모두 **실행**해 봅시다
- 마지막 닫는 중괄호 부분까지 **실행**하면
비로소 우리 프로그램 전체 실행이 끝나는 것을 볼 수 있을 거예요.
 - 달리 말하면 현재 '우리 프로그램의 실행'은,
계속 구경해 왔던,
int main() 아래에 적혀 있던 Code 덩어리의 **실행**과 동치인 것 같아요

함수



- 형언하기 힘든 값이 포함된 메시지가 보이고,
어느덧 우리는 마지막 'return문'을 남겨두고 있는 상태가 되었어요.
 - 이 시점에서 F10을 두 번만 눌러 남은 Code들을 모두 **실행**해 봅시다
- 마지막 닫는 중괄호 부분까지 **실행**하면
비로소 우리 프로그램 전체 실행이 끝나는 것을 볼 수 있을 거예요.
 - 달리 말하면 현재 '우리 프로그램의 실행'은,
계속 구경해 왔던,
int main() 아래에 적혀 있던 Code 덩어리의 **실행**과 동치인 것 같아요
- 그리고 이 main()같은 친구, Code 덩어리 하나에 이름 붙여서 다루는 친구들이 바로
우리가 아마 이미 들어본 적 있을 **함수**들이에요

함수

- 함수(function)

함수

- **함수(function)**
 - 변수, 수식과 다르게,
프로그래밍의 **함수** 개념은 지금 당장 한 줄로 설명하기 쉽지 않아요
 - 일단은, 아래의 세 가지만 살짝 터치해 둥시다:
 - 이름이 있고, 어떤 **형식** 인수 **값**(들)을 받는지 정해져 있어요
 - **함수**를 호출(call)한다는 것은 그 **함수**의 내용물 **문장**들을 **실행**하는 것과 같아요
 - 우리는 수식의 일종인 **함수** 호출식을 적어 어떤 **함수**를 호출하게 만들 수 있어요
 - (**함수** 호출식도 수식이므로) 그 수식의 **형식**이 정해져 있어요
 - 애네들은 지금 당장 납득할 수 있을 거예요!

함수

- **함수(function)**
 - 내 프로그램에 새 **함수**를 도입할 때는,
새 **변수**를 도입할 때 선언을 적은 것과 비슷한 느낌으로 **함수 정의(definition)**를 적을 수 있어요
 - 실제로 C++에서 **함수** 정의는 선언의 일종으로 간주돼요
 - **함수** 정의를 적을 때 우리는,
새 **함수**의 이름, 인수의 **형식**과 이름, 그리고 'return **형식**'을 지정하게 돼요
 - 지금 main.c에는 main()에 대한 **함수** 정의가 적혀 있을 거예요.
이거랑 비슷한 느낌으로 새 **함수**에 대한 **함수** 정의를 적을 수 있어요

함수

- **함수(function)**
 - (중요)사실 우리가 이전에 사용해 본 `printf()`, `scanf()`도 모두 **함수**였어요
 - 이 두 **함수**도, 우리가 직접 적지는 않았지만 어딘가에 그 정의가 들어 있어요
 - 우리가 적어 둔 `printf()`에 대한 **함수** 호출식을 계산할 때, `printf()` 정의 내용물 **문장**들을 **실행**하게 돼요

함수

- **함수(function)**

- (중요)사실 우리가 이전에 사용해 본 printf(), scanf()도 모두 **함수**였어요
 - 이 두 **함수**도, 우리가 직접 적지는 않았지만 어딘가에 그 정의가 들어 있어요
 - 우리가 적어 둔 printf()에 대한 **함수** 호출식을 계산할 때, printf() 정의 내용물 **문장**들을 **실행**하게 돼요
 - 그 내용물 안에는 우리가 지정한 Data를 검은 창에 담은 Code들이 잔뜩 들어 있을 거예요
- (중요)달리 말하면,
우리는 짝막한 **수식**(**함수** 호출식)을 우리 **문장** 안에 적음으로써
어딘가에 적혀 있을 큼직한 Code 덩어리(**문장**들) **실행**을 야기할 수 있어요
 - 우리가 아직 목표를 달성할 수 있을 만한 Code를 직접 적을 수 없다 하더라도, 그것들을 미리 담아 둔 **함수**를 호출함으로써 전체 프로그램을 구성할 수 있어요
 - 아무튼, 이렇게 놓고 보면 **수식**이 **문장**보다 더 규모가 크다고 말할 수 있을 것 같아요

마무리

- 이 정도 해 두면, 프로그램의 실행 흐름 중 Code 흐름을 다루기 위한 기본적인 요소들은 다 다룬 셈이 될 것 같아요.
 - 나만의 **함수** 정의 적는거는 잠시 뒤에 같이 해 봅시다
- 중간에 갑자기 이상한 것이 튀어나오기는 했지만...
순차, 분기, 반복 이야기와 **함수** 이야기는 그럭저럭 납득할 수 있었을 거예요.
- 잠시 쉬었다가 이번에는 Data 흐름쪽을 한 번 구경해 보도록 합시다.