

3-1. 이름

창의적소프트웨어프로그래밍
2022년도 여름학기
Racin

2일차 내용

- 프로그램의 실행을 Code 관점과 Data 관점으로 나누어 구경해 보았어요.
 - Code (**실행**) 흐름
 - 디버그 모드에서 노란 화살표로 표현되는 그것의 여정
 - **순차, 분기, 반복**을 구사할 수 있음
 - 각 개념을 내 프로그램에 도입하기 위해 무슨 **문장**을 적을 수 있는지, 그 **문장**을 어떻게 적을 수 있는지, 그 **문장**의 **실행** 양상은 어떤지 잘 기억해 뒀요
 - Data 흐름
 - 메모리 위의 어떤 **위치**에 **값**을 담거나(write), 담겨 있던 **값**을 가져오는 것(read)
 - '=' **수식**을 통해 내가 원하는 **위치**에 내가 원하는 **값**을 담을 수 있어요
 - 만약 **위치**를 특정할 수 없거나(모르거나), 원하는 **값**이 나오는 **수식**을 적을 수 없는 경우 나 대신 = **수식**을 **계산**해 줄 다른 **함수**를 호출함으로써 목표를 달성할 수 있어요
- 중간에 **함수, 함수 호출식** 이야기도 슬쩍 나왔어요

2일차 내용

- 프로그램의 실행을 Code 관점과 Data 관점으로 나누어 구경해 보았어요.
 - Code 흐름과 Data 흐름 중에 어떤 쪽이 더 어렵게 느껴지나요?
 - 최종 목표 도전하다 잘 안 될 때 보통 어떤 쪽이 말썽이었나요?

2일차 내용

- 프로그램의 실행을 Code 관점과 Data 관점으로 나누어 구경해 보았어요.
 - Code 흐름과 Data 흐름 중에 어떤 쪽이 더 어렵게 느껴지나요?
 - 최종 목표 도전하다 잘 안 될 때 보통 어떤 쪽이 말썽이었나요?
- 여러분이 고생했을만한 시나리오를 몇 개 적어왔어요:
 - 조건식을 적으려고 하는데 뭘 적어야 할 지 잘 안 떠오름
 - 안 겹칠 때만 출력해야 하는데, 겹치는데도 조건식이 nonzero 나와서 출력해버림
 - 5번 **반복**해야 하는데 자꾸 4번이나 6번 **반복**함
- 애네들은 각각, Code쪽 문제로 보이나요, 아니면 Data쪽 문제로 보이나요?

2일차 내용

- 프로그램의 실행을 Code 관점과 Data 관점으로 나누어 구경해 보았어요.
 - Code 흐름과 Data 흐름 중에 어떤 쪽이 더 어렵게 느껴지나요?
 - 최종 목표 도전하다 잘 안 될 때 보통 어떤 쪽이 말썽이었나요?
- 여러분이 고생했을만한 시나리오를 몇 개 적어왔어요:
 - 조건식을 적으려고 하는데 뭘 적어야 할 지 잘 안 떠오름
 - 안 겹칠 때만 출력해야 하는데, 겹치는데도 조건식이 nonzero 나와서 출력해버림
 - 5번 **반복**해야 하는데 자꾸 4번이나 6번 **반복**함
- 애네들은 각각, Code쪽 문제로 보이나요, 아니면 Data쪽 문제로 보이나요?
 - 음... 역시 잘 모르겠어요

2일차 내용

- 프로그램의 실행을 Code 관점과 Data 관점으로 나누어 구경해 보았어요.
 - 사실, Code 흐름과 Data 흐름은 매우 밀접하게 연관되어 있기 때문에 완전히 나누어 생각하는 것은 어려워요
 - = 수식을 잘 적어 놓았더라도
그 수식이 포함된 **문장**을 **실행**하지 않는 한 그 수식은 **계산**되지 않아요(**값**이 담기지 않아요)
 - 모든 Data 흐름은 Code **실행** 도중 발생해요.
디버그 모드에서, 우리가 F10을 누르지 않는 한 그 어떤 Data도 write / read되지 않을 거예요
 - **분기** 구조를 잘 갖추어 두었더라도
조건식 **계산** 결과**값**이 내가 원할 때만 nonzero 나오도록 만들려면
(내용물 **문장**들이 내가 원할 때만 **실행**되도록 만들려면)
그 시점에 적절한 **값**이 적절한 **위치**에 담겨 있어야 해요
 - 조건식 **계산** 결과**값**이 항상 0, 항상 nonzero가 나오는 if문은,
아마 굳이 적을 필요가 없을 거예요

2일차 내용

- 프로그램의 실행을 Code 관점과 Data 관점으로 나누어 구경해 보았어요.
 - 오히려 이 둘을 구분하는 것은, 한 쪽의 난이도가 높을 때 반대쪽을 더 챙겨 가면서 목표에 조금씩 더 가까이 갈 수 있게 해 준다는 점에서 의미가 있어요
 - 복잡한 Code를 다루어야 할 때 추가적인 Data를 도입할 수 있어요
 - 복잡한 Data를 다루어야 할 때 좀 더 정교한 Code **실행** 흐름을 구성해 둘 수 있어요
 - 각종 예시 코드들, 최종 목표 코드들을 구경할 때, 그 코드(가 담긴 프로그램)가 실행될 때의 흐름을 두 관점에서 의식하다 보면 프로그램 실행을 보는 눈이 꽤 성장할 수 있을 거예요
 - 익숙하지 않을테니 당분간은 디버그 모드를 애용해 봐요

2일차 내용

- 그 외 이것저것 등장했던 내용들을 다시 정리해 보는 의미에서...

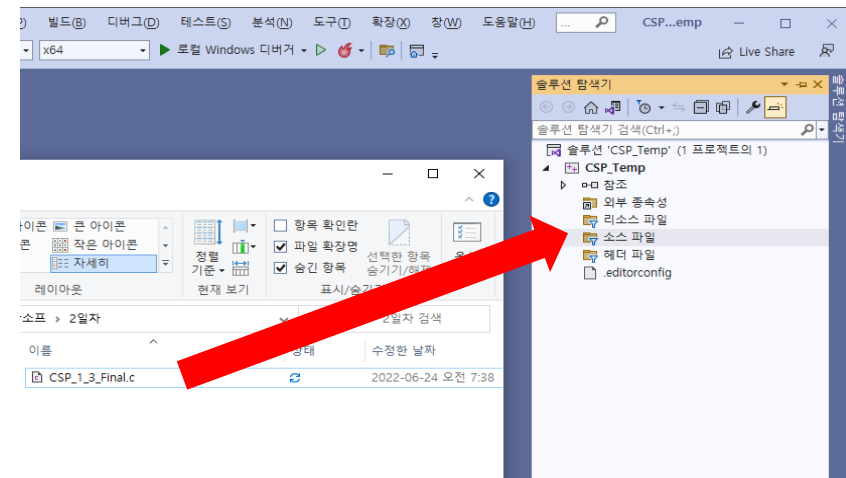
2일차 내용

- 그 외 이것저것 등장했던 내용들을 다시 정리해 보는 의미에서, CSP_2_3_Final.c를 내 Visual Studio로 가져와 구경해 봅시다.
 - 다음 슬라이드에, 가져오는 방법을 복붙해 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



정리

- 지난 시간에 살짝 지나간 Data 흐름 이야기랑 방금 구경한 내용을 조합해서 몇 가지 요소들을 잠시 정리해 볼게요.

정리

- = 연산자

- Data 흐름 중 write를 야기하는 '= 수식'을 적기 위해 사용하는 연산자예요
 - +=, ++ 등 유사품도 있어요
- 우항 수식을 계산한 결과값을 좌항 수식으로 특정되는 (메모리 위) 위치에 담아요
- 프로그래밍 목표가 무엇인지에 따라 다르지만,
모든 프로그램의 실행에서 = 수식 계산의 비중은 꽤 높아요
 - 그 계산 소요를 하나하나 별도의 문장으로 풀어 적을 수도 있고,
반복 흐름을 구성하여 그 = 수식이 포함된 문장을 연거푸 실행하게 만들 수도 있어요
- = 수식을 계산한 결과값은 좌항 자리에 담은 그 값이에요
 - 수식 $x = y = 0$ 은 수식 $x = (y = 0)$ 과 동치이며, x, y 자리에 모두 0을 담는 셈이 돼요

정리

- for문
 - 반복 흐름을 구성하는 세 부분을 한 곳에(조건식 주변에) 몰아 적을 수 있는 **문장**이에요
 - (옛날 C 기준으로) for문은 아래와 같이 생겼어요:

```
for ( [expression] ; [expression] ; [expression] )  
    statement
```

- 우리 수업에서는 무언가를 '적는 법'을 이렇게 표현하려 해요:
 - 굵은 글씨는 그 글자나 단어 그대로 적어야 함을 의미해요
 - 일반 글씨는 별도로 적는 법이 존재할 다른 무언가를 의미해요
 - []로 둘러싼 부분은 0번 또는 1번 등장할 수 있어요. ()는 그냥 요소들을 묶을 때 사용해요
 - *를 붙인 요소는 0번 또는 여러 번 등장할 수 있어요
 - +를 붙인 요소는 1번 또는 여러 번 등장할 수 있어요

정리

- for문
 - **반복** 흐름을 구성하는 세 부분을 한 곳에(조건식 주변에) 몰아 적을 수 있는 **문장**이에요
 - for문의 **실행**은 while문의 **실행**과 유사해요. 차이점은...
 - 처음 조건식(두 번째 수식)을 **계산**하기 직전에 첫 번째 수식을 **계산**해요
 - 따라서 첫 **반복**을 준비하는 용도로 해당 수식을 적어 둘 수 있어요
 - 첫 번째 수식을 **계산**한 결과값은 사용하지 않아요
 - 조건식 적는 자리를 비워둘 수 있어요. 이 경우 항상 nonzero **값**이 나온다고 간주해요
 - 내용물 **실행**이 끝난 다음 다시 조건식을 **계산**하기 직전에 세 번째 수식을 **계산**해요
 - 따라서 다음 **반복**을 준비하는 용도로 해당 수식을 적어 둘 수 있어요
 - 세 번째 수식을 **계산**한 결과값은 사용하지 않아요

정리

- for문
 - **반복** 흐름을 구성하는 세 부분을 한 곳에(조건식 주변에) 몰아 적을 수 있는 **문장**이에요
 - 이렇다 보니, '원하는 횟수만큼 **반복**'하고 싶을 때 for문을 사용할 수 있어요
 - 다음 **반복** 준비를 수행하는 부분이 '조건식 옆'에 붙어 있다는 점도 특징이에요
 - 시간 관계상 다루기는 어렵지만... continue문을 사용할 때 조금 더 편리해요

정리

- void 형식

정리


- void 형식
 - '값이 없음'을 의미하지 않아요!

정리

- void 형식

- 단어에서 느껴지듯 무언가 '희멀건한' 형식이에요
 - int 형식도 아니고, double 형식도 아니고, ...
 - 조금 더 정밀한 표현으로는 '불완전한' 형식이라 불러요

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

 (지역 변수) void number
온라인 검색
불완전한 형식은 사용할 수 없습니다.
온라인 검색


- 그렇다 보니 우리 중 누구도 void 형식 값을 특정하거나 계산에 사용할 수 없어요
 - 어떤 함수의 return 형식이 void라면
함수 정의 적는 본인 포함 누구도 그 함수 호출식의 계산 결과값이 몇인지 알 수 없어요

정리

- void 형식

- 단어에서 느껴지듯 무언가 '희멀건한' 형식이에요
 - int 형식도 아니고, double 형식도 아니고, ...
 - 조금 더 정밀한 표현으로는 '불완전한' 형식이라 불러요

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

 (지역 변수) void number
온라인 검색
불완전한 형식은 사용할 수 없습니다.
온라인 검색

- 그렇다 보니 우리 중 누구도 void 형식 값을 특정하거나 계산에 사용할 수 없어요
 - 어떤 함수의 return 형식이 void라면
함수 정의 적는 본인 포함 누구도 그 함수 호출식의 계산 결과값이 몇인지 알 수 없어요
- 그래도, 계산 결과값을 사용하지 않는 곳이라면 void 형식 수식을 적는 것이 가능해요
 - 문장 result = PrintThatNumbers_Mk1(); 는 불가능하지만
문장 PrintThatNumbers_Mk1(); 는 충분히 가능해요

정리 마무리

- 대충 이 정도 짚어 두면 충분한 것 같아요.
- 이제 본격적으로 오늘 내용으로 들어가 보도록 합시다.

오늘 내용

- 이제까지 'Data와 Code' 같은 느낌으로 두 부분을 나누어 다루었지만, 아무래도 Data쪽보다는 Code쪽에 더 많은 시간을 할애했어요.
 - 심지어 지난 수업에서 '2-2. Data 흐름'은 아예 시간 관계상 스킵해버림

오늘 내용

- 이제까지 'Data와 Code' 같은 느낌으로 두 부분을 나누어 다루었지만, 아무래도 Data쪽보다는 Code쪽에 더 많은 시간을 할애했어요.
 - 심지어 지난 수업에서 '2-2. Data 흐름'은 아예 시간 관계상 스킵해버림
- 이를 감안하여, 프로그램의 Data 부분을 본격적으로 챙기기 위한 **이름** 이야기를 먼저 구경해 보고, 이후 수업에서 매우 중요하게 다룰 **선언** 및 **정의** 이야기로 확장해 보려 해요.

이번 시간에는

- **이름**
 - 프로그래밍 동네에서 도입하여 사용하는 다양한 **이름들**
- 새로 나오는 키워드
 - **이름**
 - **선언**
 - Context

이번 시간에는

- 시작해 봅시다.
- 이미 다양한 **이름**들을 이전에 꽤 자주 적어 보았으니 바로 진입할 수 있을 것 같아요.

이름

- 우리는 프로그래밍 과정에서 다양한 **이름**들을...
미래의 나 또는 누군가가 사용할 수 있도록 직접 정해 두거나,
누군가가 미리 정해 둔 것을 사용합니다.
 - 우리는 이미 number, width 등 다양한 **이름**들을 미리 정해 두고 잘 써먹어 보았어요

이름

- 우리는 프로그래밍 과정에서 다양한 **이름**들을...
미래의 나 또는 누군가가 사용할 수 있도록 직접 정해 두거나,
누군가가 미리 정해 둔 것을 사용합니다.
 - 우리는 이미 number, width 등 다양한 **이름**들을 미리 정해 두고 잘 써먹어 보았어요
- Q1) number가 int **변수 이름**일 때,
이 **이름**을 '사용'한다는 것은 구체적으로 무엇을 한다는 걸까요?

이름

- 우리는 프로그래밍 과정에서 다양한 **이름**들을...
미래의 나 또는 누군가가 사용할 수 있도록 직접 정해 두거나,
누군가가 미리 정해 둔 것을 사용합니다.
 - 우리는 이미 number, width 등 다양한 **이름**들을 미리 정해 두고 잘 써먹어 보았어요
- Q1) number가 int **변수 이름**일 때,
이 **이름**을 '사용'한다는 것은 구체적으로 무엇을 한다는 걸까요?
 - 맞아요. 그 **이름**을 '수식으로써 적는' 것을 말해요

이름

- 우리는 프로그래밍 과정에서 다양한 **이름**들을...
미래의 나 또는 누군가가 사용할 수 있도록 직접 정해 두거나,
누군가가 미리 정해 둔 것을 사용합니다.
 - 우리는 이미 number, width 등 다양한 **이름**들을 미리 정해 두고 잘 써먹어 보았어요
- Q2)number는 int **변수 이름**이다...
...라고 미리 정해 두려 할 때 우리는 구체적으로 무엇을 적었었나요?

이름

- 우리는 프로그래밍 과정에서 다양한 **이름**들을...
미래의 나 또는 누군가가 사용할 수 있도록 직접 정해 두거나,
누군가가 미리 정해 둔 것을 사용합니다.
 - 우리는 이미 number, width 등 다양한 **이름**들을 미리 정해 두고 잘 써먹어 보았어요
- Q2)number는 int **변수 이름**이다...
...라고 미리 정해 두려 할 때 우리는 구체적으로 무엇을 적었었나요?
 - **선언** int number; 를 적었어요

이름

- 이 정도만 알고 있으면 일단 **이름** 이야기로 들어갈 준비는 끝나요!
 - 다음 슬라이드부터 슬슬 정리해 볼게요

이름

- 이름
 - 이름은, 말 그대로 이름이에요

이름

- 이름
 - 이름은, 말 그대로 **이름**이에요
 - 우리는 적절한 **선언**(declaration)을 미리 적어 둬으로써 나중에 사용할 **이름**을 미리 정해 둘 수 있어요
 - 예: **선언** `int number;`
 - 여기서의 '사용'은 그 **이름**을 '**수식**으로써 적는' 것을 의미해요
 - 예:
 - **수식** `number = 3.5`의 부분 **수식** `number` (3을 어디에 담을 것인지 지정하기 위해 적음)
 - **수식** `number + 3.5`의 부분 **수식** `number` (지금은 아마 **계산**하면 6.5 나올 듯)

이름

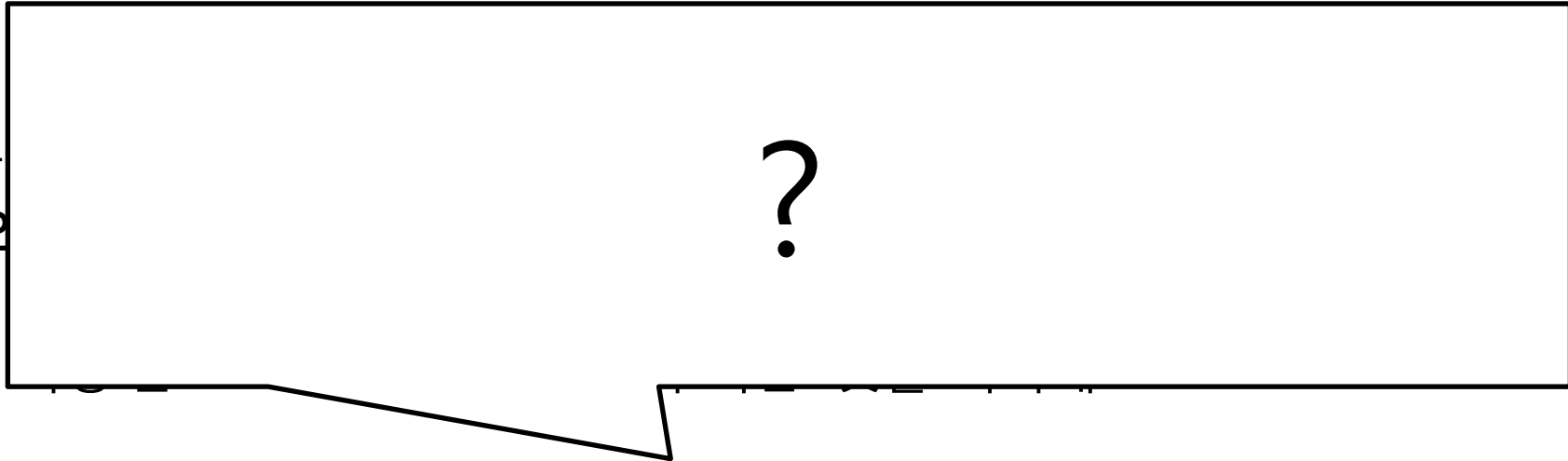
- 이름

- 이름은, 말 그대로 이름이에요

- 우리는 적
나중에 사
 - 예: 선언

- 여기서의
 - 예:

- 수식 `number = 3.5`의 부분 수식 `number` (3을 어디에 담을 것인지 지정하기 위해 적음)
 - 수식 `number + 3.5`의 부분 수식 `number` (지금은 아마 계산하면 6.5 나올 듯)



이름

- 이름

- 이름은, 말 그대로 이름이에요

- 우리는 적
나중에 사

- 예: 선언

이 = 수식의 우항 자리에는
'계산하면 항상 double 형식 값 3.5가 나오는 수식'이 적혀 있지만,
좌항 자리 수식의 형식이 int(라 가정하고 있)기 때문에
결과적으로 이 수식은 int 형식 값 3 나오는 수식으로 간주(deduce)돼요.

- 여기서의

- 예:

- 수식 `number = 3.5`의 부분 수식 `number` (3을 어디에 담을 것인지 지정하기 위해 적음)
 - 수식 `number + 3.5`의 부분 수식 `number` (지금은 아마 계산하면 6.5 나올 듯)

이름

- 이름

- 이름은, 말 그대로 **이름**이에요

- 우리는 적
나중에 사

- 예: 선언

비슷한 느낌으로,
이 덧셈식의 좌항 자리에는 int(라 가정하고 있는) **형식 수식**이 있지만,
우항 자리 **수식의 형식**이 double이기 때문에
결과적으로 이 **수식**은 double **형식 수식**으로 간주돼요.
(int **값** 3에 대한 double **값**인 3.0이 **계산 결과값**으로 나올 거예요)

- 여기서의

- 예:

- 수식 number 3.5의 부분 수식 number (3을 어디에 담을 것인지 지정하기 위해 적음)
- 수식 number + 3.5의 부분 수식 number (지금은 아마 **계산**하면 6.5 나올 듯)

이름

- 이름

- 이름은, 말 그대로 **이름**이에요

- 우리는 적
나중에 사
▪ 예: 선언

이런 식으로, 동일한 **이름**(이 경우에는 사실 **수식**)을 적는다 하더라도
그 주변에 무엇을 어떻게 적어 두었는지에 따라
실질적인 의미(**형식** 등)가 그 때 그 때 달라질 수 있어요.

달리 말하면, '문맥(context)'이 존재한다고 볼 수 있어요!

- 여기서의
▪ 예:

- **수식** `number = 3.5`의 부분 **수식** `number` (3을 어디에 담을 것인지 지정하기 위해 적음)
 - **수식** `number + 3.5`의 부분 **수식** `number` (지금은 아마 **계산**하면 6.5 나올 듯)

이름

- 이름

- 이름은, 말 그대로 **이름**이에요

- 우리는 적
나중에 사

- 예: 선언

Context 이야기는 우리 수업 곳곳에서 다시 등장할 예정이고,
당장 내일 수업에서 보다 자세한 예시들이 나올 예정이에요.

지금은 일단 본론인 **이름** 이야기로 돌아가 봅시다.

- 여기서의

- 예:

- 수식 `number = 3.5`의 부분 수식 `number` (3을 어디에 담을 것인지 지정하기 위해 적음)
 - 수식 `number + 3.5`의 부분 수식 `number` (지금은 아마 **계산**하면 6.5 나올 듯)

이름

- 이름

- 이름은, 말 그대로 **이름**이에요

- 우리는 적절한 **선언**(declaration)을 미리 적어 둬으로써 나중에 사용할 **이름**을 미리 정해 둘 수 있어요

- 예: **선언** `int number;`

- 여기서의 '사용'은 그 **이름**을 '수식으로써 적는' 것을 의미해요

- 예:

- 수식 `number = 3.5`의 부분 수식 `number` (3을 어디에 담을 것인지 지정하기 위해 적음)
- 수식 `number + 3.5`의 부분 수식 `number` (지금은 아마 **계산**하면 6.5 나올 듯)

이름

- **이름**

- 프로그래밍에서 **이름**은 크게 네 가지 의도로 쓰여요:

이름

- **이름**

- 프로그래밍에서 **이름**은 크게 네 가지 의도로 쓰여요:
 - Data **이름**
 - Code **이름**
 - Typename(**형식 이름**)
 - 무언가 다른 **이름**들이 소속?되어 있는 **이름**

이름

- 이름

- 프로그래밍에서 **이름**은 크게 네 가지 의도로 쓰여요:

- Data **이름**

- number

- Code **이름**

- printf

- Typename(**형식 이름**)

- int

- 무언가 다른 **이름**들이 소속?되어 있는 **이름**

- 이걸 일단 Python의 `time` 같은 것을 생각해 볼 수 있을 듯

이름

- 이름

- 프로그래밍에서 **이름**은 크게 네 가지 의도로 쓰여요:

- Data **이름**

- number

- Code **이름**

- printf

- Typename(**형식 이름**)

- int

수식 printf = 3이나 수식 int = 3은
대충 봐도 뭔가 좀 이상한 것 같아요.

Python이라면
문장 print = 3 엔터 나 문장 int = 3 엔터
...가 가능하긴 하지만
뭐 굳이 그럴 필요는 없을 것 같아요.

- 무언가 다른 **이름**들이 소속?되어 있는 **이름**

- 이걸 일단 Python의 time 같은 것을 생각해 볼 수 있을 듯

이름

- 이름

- 우리는 옛날 C를 다루고 있으니,
Data **이름**과 Code **이름**은 살짝만 구분해 두고,
이 둘과 **형식 이름**은 매우 명백하게 구분지어 두면 좋을 것 같아요
 - 소속 어찌구 하는거는 C++ 다룰 때 구경해 볼게요

마무리

- 이름
 - 우리는 옛날 C를 다루고 있으니,
Data **이름**과 Code **이름**은 살짝만 구분해 두고,
이 둘과 **형식 이름**은 매우 명백하게 구분지어 두면 좋을 것 같아요
 - 소속 어찌구 하는거는 C++ 다룰 때 구경해 볼게요
- 일단 이 정도로 해 두고, 잠시 쉽시다.