3-3. 포인터, 배열, 함수

창의적소프트웨어프로그래밍
2022년도 여름학기
Racin

이번 시간에는

- **포인터**, **배열**, **함수**를 소개합니다.
  - **포인터**, **배열**, **함수 이름**을 **선언**하고 사용하는 방법
  - 이들 각각의 의의(언제 어떤 **이름**을 **선언**해 사용할 것인지)
  - '칸(object)'이 슬쩍 등장하는데,
    다음 시간에 조금 더 자세하게 정리해 볼께요
    - 일단은 'int 한 칸' 하면 int **값** 하나 담기에 적당한 메모리 공간을 의미해요

포인터, 배열, 함수

- 바로 **선언**을 구경해 봅시다:

```
int number;

int *p_number;

int arr[2];

int Three();
```

포인터, 배열, 함수

- 바로 **선언**을 구경해 봅시다:

```
int number;

int *p_number;

int arr[2];

int Three();
```

얘는 노멀 그 자체에 해당하는 그냥 **변수**입니다.
int 한 칸 짜리예요.

포인터, 배열, 함수

- 바로 **선언**을 구경해 봅시다:

```
int number;

int *p_number;

int arr[2];

int Three();
```

얘는 '* 붙이면 int 한 칸을 사용할 수 있는 **값**'
...을 담는 '**포인터 변수**'입니다.
아무튼 **변수**고, 그래서 일단 한 칸 짜리예요.

포인터, 배열, 함수

- 바로 **선언**을 구경해 봅시다:

```
int number;

int *p_number;

int arr[2];

int Three();
```

얘는 int 두 칸짜리 **배열**입니다.
C에서 **배열**은 **변수**(무조건 한 칸 짜리)가 아니예요!
뭐 [ ] **연산자**를 붙이면 int 한 칸을 쓸 수 있긴 해요.

포인터, 배열, 함수

- 바로 **선언**을 구경해 봅시다:

```
int number;

int *p_number;

int arr[2];

int Three();
```

얘는 int 값을 return하는 **함수**입니다.
뭐 당연하겠지만 **함수**도 **변수**가 아니에요.
얘는 혼자, ( ) **연산자** 붙여 호출하면 '칸'이 아닌 '**값**'이 나와요.

포인터, 배열, 함수

- 오옹... 세 친구가 각각의 개성을 뽐내고 있는 것 같습니다.
  - * 붙여서 int 한 칸을 쓰는 **포인터 변수**(**위치 값** 하나를 담음)
  - [ ] 붙여서 int 한 칸을 쓰는 **배열**(**변수** 아님. 여러 칸이 **정의**됨)
  - ( ) 붙여서 int 값 하나를 쓰는 **함수**(얜 아예 내용물이 **문장**들임)

포인터, 배열, 함수

- 오옹... 세 친구가 각각의 개성을 뽐내고 있는 것 같습니다.
  - * 붙여서 int 한 칸을 쓰는 **포인터 변수**(**위치 값** 하나를 담음)
  - [ ] 붙여서 int 한 칸을 쓰는 **배열**(**변수** 아님. 여러 칸이 **정의**됨)
  - ( ) 붙여서 int 값 하나를 쓰는 **함수**(얜 아예 내용물이 **문장**들임)

- 이 셋을 바라보기 위한 첫 단계는,
  일단 이 셋이 모두 '완전 다른 애들이다'라는 것을 확인하는 것이에요.

포인터, 배열, 함수

- 오옹... 세 친구가 각각의 개성을 뽐내고 있는 것 같습니다.
  - * 붙여서 int 한 칸을 쓰는 **포인터 변수**(**위치 값** 하나를 담음)
  - [ ] 붙여서 int 한 칸을 쓰는 **배열**(**변수** 아님. 여러 칸이 **정의**됨)
  - ( ) 붙여서 int 값 하나를 쓰는 **함수**(얜 아예 내용물이 **문장**들임)

- 이 셋을 바라보기 위한 첫 단계는,
  일단 이 셋이 모두 '완전 다른 애들이다'라는 것을 확인하는 것이에요.
  - 그 다음 단계는, '왜 C는 얘네를 따로 나누어 놓았는지'를 확인하는 게 되겠지요?

포인터, 배열, 함수

- 한 번 직접 시도해 봅시다.
  - 새 프로젝트를 만들거나 원래 있던 거 그대로 쓰고,
    Ctrl + Shift + A 해서 main.c 만들거나 원래 있던 거 그대로 쓰고,
    아마 적게 될 코드를 적어 주세요.

  - 다 적은 친구들은 아래와 같은 느낌으로 나열되도록 `main()` **정의** 내용물을 적어보세요:
    - `int` 두 칸 짜리 **배열** `numbers` **선언**
    - 두 칸 중 0번째 칸에 3을 담음
    - 두 칸 중 1번째 칸에 5를 담음

    - 두 칸 중 0번째 칸에 담긴 값을 `printf()` 써서 출력

  - 다 적은 친구들은 Ctrl + F5를 눌러 목표대로 잘 실행되나 확인해 보세요!

포인터, 배열, 함수

- 아마도 이런 식으로 적었겠지요?

```c
#include <stdio.h>

int main()
{
    int numbers[2];

    numbers[0] = 3;

    numbers[1] = 5;

    printf("%d", numbers[0]);

    return 0;
}
```

포인터, 배열, 함수

- 아마도 이런 식으로 적었겠지요?

```c
#include <stdio.h>

int main()
{
    int numbers[2];

    numbers[0] = 3;

    numbers[1] = 5;

    printf("%d", numbers[0]);

    return 0;
}
```

배열이고 뭐고를 떠나서 = **수식**도 **수식**이기 때문에,
'int **값** 하나를 int 한 칸에 담는' 느낌으로 적어야 해요.
(= **수식**의 좌항 및 우항이 모두 각각 **수식**이어야 하고
수식은 **계산**하면 결과**값**이 '하나' 나오는 친구예요)

포인터, 배열, 함수

- 아마도 이런 식으로 적었겠지요?

```c
#include <stdio.h>

int main()
{
    int numbers[2];

    numbers[0] = 3;

    numbers[1] = 5;

    printf("%d", numbers[0]);

    return 0;
}
```

물론, **선언**에 적어 둔 대로 **연산자**를 딱딱 맞추어 적으면 돼요.
그러라고 있는 게 **선언**임! (**이름** + 사용 방법)

포인터, 배열, 함수

- 아마도 이런 식으로 적었겠지요?

```c
#include <stdio.h>

int main()
{
    int numbers[2];

    numbers[0] = 3;

    numbers[1] = 5;

    printf("%d", numbers[0]);

    return 0;
}
```

물론, 0번째 칸에 있는 **값**을 사용하고 싶을 때도 그냥 적으면 돼요.
얘가 사실 사용 빈도 1순위에 해당하는 '그 칸에 담긴 **값**'이지요?

포인터, 배열, 함수

- 대충 감이 왔다면, 목표를 다음과 같이 수정해 볼게요:
  - int 두 칸 짜리 **배열** numbers **선언**
  - scanf() 써서 두 칸 중 0번째 칸에 사용자가 입력한 숫자를 담음
  - 두 칸 중 1번째 칸에 5를 담음

  - 두 칸 중 0번째 칸에 담긴 값을 printf() 써서 출력

- 원래 코드에서 한 줄을 scanf() 호출식을 써서 다시 적으면 돼요.
  한 번 직접 도전해 봅시다.
  - 주의: 이번에는 = **연산자**의 좌항 자리가 아님에도 사용 빈도 2위인 '**위치 값**'이 필요해요

포인터, 배열, 함수

- 그렇게 어렵지 않았지요?

```c
#include <stdio.h>

int main()
{
    int numbers[2];

    scanf("%d", &(numbers[0]));

    numbers[1] = 5;

    printf("%d", numbers[0]);

    return 0;
}
```

포인터, 배열, 함수

- 그렇게 어렵지 않았지요?

```c
#include <stdio.h>

int main()
{
    int numbers[2];

    scanf("%d", &(numbers[0]));

    numbers[1] = 5;

    printf("%d", numbers[0]);

    return 0;
}
```

대칭성을 적극적으로 활용해서
아래에 적어 둔 printf() 호출식을 그대로 복붙해 와서 고쳤다면
스스로에게 칭찬을 날려 주어도 좋아요!

포인터, 배열, 함수

- 그렇게 어렵지 않았지요?

```c
#include <stdio.h>

int main()
{
    int numbers[2];

    scanf("%d", &(numbers[0]));

    numbers[1] = 5;

    printf("%d", numbers[0]);

    return 0;
}
```

아까 본 & **연산자**의 유래를 기억한다면
'아무튼 걔의 **위치**!' 같은 느낌으로 이렇게 적을 수 있었을 거예요.

포인터, 배열, 함수

- 그렇게 어렵지 않았지요?

```c
#include <stdio.h>

int main()
{
    int numbers[2];

    scanf("%d", &numbers[0]);

    numbers[1] = 5;

    printf("%d", numbers[0]);

    return 0;
}
```

아까 본 & **연산자**의 유래를 기억한다면
'아무튼 걔의 **위치**!' 같은 느낌으로 이렇게 적을 수 있었을 거예요.

이렇게 적는 빈도가 많기 때문에 괄호 안 쳐도 되게 설계되어 있음!

포인터, 배열, 함수

- 이번에는 조금 다른 **함수** 하나를 직접 **정의**해 볼게요:
  - `fill_with_three()`

포인터, 배열, 함수

- 이번에는 조금 다른 **함수** 하나를 직접 **정의**해 볼게요:
  - fill_with_three()
    - 'int 한 칸에 대한 **위치 값**(int **포인터 값**)'을 인수 **값**으로써 받아요
      - 인수 **이름**은 기본적으로 **변수 이름**이에요.
        함수 정의의 첫 줄 괄호 안에 'int **포인터 변수 선언**'을 ; 빼고 적으면 인수 **선언**이 끝나요
    - 인수로 받은 **위치 값**을 가지고, '그 **위치**'에 3을 담아요
    - 뭐 그거만 하면 돼요!
  - 다 작성했다면,
    이제 그 **함수**를 호출해서 **배열** numbers의 0번째 칸을 3으로 채우도록
    main() 내용물을 수정해 보세요!

포인터, 배열, 함수

스포일 방지를 위한 여백

포인터, 배열, 함수

- 이런 식으로 적었다면 성공이에요!

```
int fill_with_three(int *p_number)
{
    *p_number = 3;
}
```

포인터, 배열, 함수

- 이런 식으로 적었다면 성공이에요!

```
int fill_with_three(int *p_number)
{
    *p_number = 3;
}
```

VS에서는 인수 **이름**을 이렇게 다른 색으로 보여주는 것 같아요.
이런 색으로 **이름**이 적혀 있다면
'그 칸에 처음 담겨 있을 **값**은 호출자가 정했다'가 떠오르면 될 듯?

포인터, 배열, 함수

- 이런 식으로 적었다면 성공이에요!

```
int fill_with_three(int *p_number)
{
    *p_number = 3;
}
```

지금의 경우에는, 이 안에는,
'3을 담고 싶은 호출자가 갖다 바친 int **포인터 값**'이 들어 있어요.

포인터, 배열, 함수

- 이런 식으로 적었다면 성공이에요!

```
int fill_with_three(int *p_number)
{
    *p_number = 3;
}
```

인수 **선언**을 보니, 이렇게 **수식**을 적으면 int 한 칸이 나오나봐요.

포인터, 배열, 함수

- 이런 식으로 적었다면 성공이에요!

```
int fill_with_three(int *p_number)
{
    *p_number = 3;
}
```

인수 **선언**을 보니, 이렇게 수식을 적으면 int 한 칸이 나오나봐요.

그 사실에 착안하면 = **수식**은 이렇게 적을 수 있을 거예요!

포인터, 배열, 함수

- 나쁘지 않지요?
  서로 다른 세 친구들이지만, 그들의 특성을 어느 정도 구분할 수 있다면
  이들을 서로 섞어 가며 프로그램을 작성하는 것은 그리 어렵지 않아요.

포인터, 배열, 함수

- 나쁘지 않지요?
  서로 다른 세 친구들이지만, 그들의 특성을 어느 정도 구분할 수 있다면 이들을 서로 섞어 가며 프로그램을 작성하는 것은 그리 어렵지 않아요.
  - **포인터** 변수는 '특정 **형식** 한 칸에 대한 **위치 값**'을 담아 두기 위해 사용해요
    - 미리 골라 두었다 나중에 쓰는 게 가능해요
    - 특히, 호출자가 담아 준 **위치 값**을 **함수 정의** 중괄호 안에서 사용하는 게 가능해요
      - 호출자가 지정한 칸에 어떤 **값**을 담도록 = **수식**을 적는 게 가능해요!

포인터, 배열, 함수

- 나쁘지 않지요?
  서로 다른 세 친구들이지만, 그들의 특성을 어느 정도 구분할 수 있다면 이들을 서로 섞어 가며 프로그램을 작성하는 것은 그리 어렵지 않아요.
  - **배열**은, 나중에 골라 쓰기 위한 여러 칸을 일렬로 **정의**하고 싶을 때 사용해요
    - **정의**에서, 몇 칸을 쓸 것인가(내용)은 여러분이 정해요
      정의에서, 각 칸을 메모리의 어느 자리에 둘 것인가(**위치**)는 컴파일러가 해 줘요
      - **배열**의 경우 무조건 0번째 칸이 맨 앞이고 그 뒤로 줄줄이 일렬로 나열됨이 보장돼요!
  - 실제로 **배열**을 사용할 때는 보통 **배열**의 여러 칸들 중 하나를 골라서 쓰게 돼요
    - 특정 한 **칸**에 대한 **포인터 값**이 나오는 **수식**을 써서 **함수** 호출식을 구성할 수도 있음!

포인터, 배열, 함수

- 나쁘지 않지요?
  서로 다른 세 친구들이지만, 그들의 특성을 어느 정도 구분할 수 있다면
  이들을 서로 섞어 가며 프로그램을 작성하는 것은 그리 어렵지 않아요.
  - 이 과정에서 **함수**까지 끼어 들어가면 조금 어려워지긴 한데,
    지금은 일단 다음 슬라이드 내용 정도는 구경할 수 있을 것 같아요:

포인터, 배열, 함수

- 지금 아마 완성되어 있을 코드예요:

```c
#include <stdio.h>

int fill_with_three(int *p_number)
{
    *p_number = 3;
}

int main()
{
    int numbers[2];

    fill_with_three(&numbers[0]);

    numbers[1] = 5;

    printf("%d", numbers[0]);

    return 0;
}
```

포인터, 배열, 함수

- 지금 아마 완성되어 있을 코드예요:

```c
#include <stdio.h>

int fill_with_three(int *p_number)
{
    *p_number = 3;
}

int main()
{
    int numbers[2];

    fill_with_three(&numbers[0]);

    numbers[1] = 5;

    printf("%d", numbers[0]);

    return 0;
}
```

이 두 친구들은 모두 = **수식**이에요.

'이 칸에 이 **값**을 담는다' 를 의미하지요.

포인터, 배열, 함수

- 지금 아마 완성되어 있을 코드예요:

```c
#include <stdio.h>

int fill_with_three(int *p_number)
{
    *p_number = 3;
}

int main()
{
    int numbers[2];

    fill_with_three(&numbers[0]);

    numbers[1] = 5;

    printf("%d", numbers[0]);

    return 0;
}
```

따라서, = **수식**을 성공적으로 적으려면
그 두 정보를 모두 다 알고 있어야 해요.

'어디 담을 지'와 '무얼 담을 지' 말이에요.

포인터, 배열, 함수

- 지금 아마 완성되어 있을 코드예요:

```c
#include <stdio.h>

int fill_with_three(int *p_number)
{
    *p_number = 3;
}

int main()
{
    int numbers[2];

    fill_with_three(&numbers[0]);

    numbers[1] = 5;

    printf("%d", numbers[0]);

    return 0;
}
```

배열의 각 칸에 적절한 값을 담는 시점에서...
main() 짜는 입장에서는
'어디 담을 지'는 명백히 알고 있어요.

본인이 직접 **선언**을 적고 **정의**도 끝내 놨으니까요!

포인터, 배열, 함수

- 지금 아마 완성되어 있을 코드예요:

```c
#include <stdio.h>

int fill_with_three(int *p_number)
{
    *p_number = 3;
}

int main()
{
    int numbers[2];

    fill_with_three(&numbers[0]);

    numbers[1] = 5;

    printf("%d", numbers[0]);

    return 0;
}
```

'무얼 담을 지'도 본인이 알고 있다면,
그 **값**이 나오는 **수식**을 적어서
목표하는 = **수식**을 직접 완성할 수 있어요!

포인터, 배열, 함수

- 지금 아마 완성되어 있을 코드예요:

```c
#include <stdio.h>

int fill_with_three(int *p_number)
{
    *p_number = 3;
}

int main()
{
    int numbers[2];

    fill_with_three(&numbers[0]);

    numbers[1] = Five();

    printf("%d", numbers[0]);

    return 0;
}
```

본인이 그 **값**을 모른다 하더라도,
'그 **값**을 얻을 수 있는 **수식**'을 적을 수만 있으면 돼요.

지금 예시에서는 안 만들었지만,
이런 경우도 충분히 가능했겠지요?

포인터, 배열, 함수

- 지금 아마 완성되어 있을 코드예요:

```c
#include <stdio.h>

int fill_with_three(int *p_number)
{
    *p_number = 3;
}

int main()
{
    int numbers[2];

    fill_with_three(&numbers[0]);

    numbers[1] = Five();

    printf("%d", numbers[0]);

    return 0;
}
```

반면, 어떤 경우에는
그 **값**을 절대로 내가 직접 얻을 수 없는 경우도 있어요.
내가 원하는 **값**을 return해 주는 **함수**가 하나도 없다면
이제는 내가 직접 = **수식**을 적는 게 불가능해요.

포인터, 배열, 함수

- 지금 아마 완성되어 있을 코드예요:

```c
#include <stdio.h>

int fill_with_three(int *p_number)
{
    *p_number = 3;
}

int main()
{
    int numbers[2];

    fill_with_three(&numbers[0]);

    numbers[1] = Five();

    printf("%d", numbers[0]);

    return 0;
}
```

그래서 이런 시나리오에서는,
내가 지정한 '어디 담을 지' 정보를 다른 **함수**에게 주며
나 대신 = **수식**을 **계산**해 줄 것을 요청하게 돼요.

포인터, 배열, 함수

- 지금 아마 완성되어 있을 코드예요:

```c
#include <stdio.h>

int fill_with_three(int *p_number)
{
    *p_number = 3;
}

int main()
{
    int numbers[2];

    fill_with_three(&numbers[0]);

    numbers[1] = Five();

    printf("%d", numbers[0]);

    return 0;
}
```

우리가 **정의**한 fill_with_three()는,
요청받은 칸에 3을 대신 담아 주는 **함수**예요.

'무얼 담을 지' 하나는 명확히 알고 있어요.
하지만 '어디 담을 지'는 스스로 정하지 않고 있어요.

포인터, 배열, 함수

- 지금 아마 완성되어 있을 코드예요:

```c
#include <stdio.h>

int fill_with_three(int *p_number)
{
    *p_number = 3;
}

int main()
{
    int numbers[2];

    fill_with_three(&numbers[0]);

    numbers[1] = Five();

    printf("%d", numbers[0]);

    return 0;
}
```

main()이, 자신이 알고 있는 '어디 담을 지' 정보를
fill_with_three() 호출식의 인수 자리에 담아 호출하면...

포인터, 배열, 함수

- 지금 아마 완성되어 있을 코드예요:

```c
#include <stdio.h>

int fill_with_three(int *p_number)
{
    *p_number = 3;
}

int main()
{
    int numbers[2];

    fill_with_three(&numbers[0]);

    numbers[1] = Five();

    printf("%d", numbers[0]);

    return 0;
}
```

main()이, 자신이 알고 있는 '어디 담을 지' 정보를
fill_with_three() 호출식의 인수 자리에 담아 호출하면...

fill_with_three() 내용물을 **실행**하는 시점에는 이제
= **수식 계산**에 필요한 모든 정보를 다 가진 셈이 돼요!

포인터, 배열, 함수

- 지금 아마 완성되어 있을 코드예요:

```c
#include <stdio.h>

int fill_with_three(int *p_number)
{
    *p_number = 3;
}

int main()
{
    int numbers[2];

    fill_with_three(&numbers[0]);

    numbers[1] = Five();

    printf("%d", numbers[0]);

    return 0;
}
```

그 의도가,
바로 여기 있는 이 = **수식**에 투영되어 있어요.

main()의 **배열** 속 한 칸에 내 **값**을 담을 수 있었어요.

포인터, 배열, 함수

- 지금 아마 완성되어 있을 코드예요:

```c
#include <stdio.h>

int fill_with_three(int *p_number)
{
    *p_number = 3;
}

int main()
{
    int numbers[2];

    fill_with_three(&numbers[0]);

    numbers[1] = Five();

    printf("%d", numbers[0]);

    return 0;
}
```

scanf()도, 기본적으로는
나 대신 내가 정한 **칸**을 좌항으로 = **수식**을 적는 친구예요.

포인터, 배열, 함수

- 지금 아마 완성되어 있을 코드예요:

```c
#include <stdio.h>

int fill_with_three(int *p_number)
{
    *p_number = 3;
}

int main()
{
    int numbers[2];

    fill_with_three(&numbers[0]);

    numbers[1] = Five();

    printf("%d", numbers[0]);

    return 0;
}
```

scanf()도, 기본적으로는
나 대신 내가 정한 **칸**을 좌항으로 = **수식**을 적는 친구예요.

printf()도 나 대신 = **수식**을 적지만,
얘는 내 **값**을 우항에, '검은 창'을 좌항에 두는 친구지요!

포인터, 배열, 함수

- 요약하면...
  - **함수**를 여럿 만들어 쓰다 보면
    '목표하는 = **수식**을 내가 / 쟤가 **계산**'하는 상황이 자주 나오게 돼요

  - 그 과정에서 종종 **포인터 값을** 주고받는 상황이 연출돼요.
    = **수식 계산**에는 '어디 담을 지'에 해당하는 **위치 값**이 필요하니까요

  - 달리 말하면, 이제 우리도 지금 시점부터는,
    Python에서처럼 C에서도 다양한 **함수**들을 만들어 써 볼 준비가 되었다 할 수 있어요!

마무리

- 오... 드디어 복습이 거의 끝나가고 있어요.
  - 방금 전에 다룬 내용만 납득할 수 있다면 그럭저럭 한 학기 동안 버틸 수 있을 거예요

- 오늘은 최종 목표는 없고,
  수업 페이지에 느낀 점만 적고 퇴근하도록 합시다.
  - 오늘도 배점 및 기한은 없어요