6-2. Data & Code

창의적소프트웨어프로그래밍
2022년도 여름학기
Racin

이번 시간에는

- (근미래에)main() **정의** 내용물 적을 사람이 사용할 Data와 Code들을 한꺼번에 모아 다루기 위한 **형식**을 직접 만들어 봅니다.
  - 우리는 이미 그 재료들을 다 구경해 본 적이 있으니 그리 어렵긴 할 거예요

- 일단 CSP_6_2_yeshi_base.c를 VS에 탑재해 봅시다.
  - 탑재가 끝나면 일단 다짜고짜 실행해 보세요

'게임' 형식

- 이 프로그램은 여러 '게임'들 중 하나를 선택-플레이하도록 설계되어 있어요.

- '게임' 하나에 대한 정보는 Game **object**에 각각 담겨 있어요.
  - 게임 내에서 사용할 Data들에 대한 **위치 값**
  - 게임의 준비→진행→마무리에 해당하는 각 Code들에 대한 **위치 값**

'게임' 형식

- 이 프로그램은 여러 '게임'들 중 하나를 선택-플레이하도록 설계되어 있어요.

- '게임' 하나에 대한 정보는 Game **object**에 각각 담겨 있어요.
  - 게임 내에서 사용할 Data들에 대한 **위치 값**
  - 게임의 준비→진행→마무리에 해당하는 각 Code들에 대한 **위치 값**

- (중요)여전히 각 **object**에는 Data들(**위치 값**들)만 담겨 있어요!
  - 'Press 3 to win 게임'에 대한 Code들은 '**object** 영역 밖'에 따로 정의되어 있어요

  - 다만, New_Press_3_to_win()에 의해,
    새 Game **object**에 '이 게임은 이 세 **함수**들로 구성됨'에 대한 정보를 담아 둠으로써
    '해당 **object**가 곧 Press 3 to win 게임'을 의미하도록 구성하고 있어요

'내 게임' 직접 구성해 보기

- 잠깐 시간을 내어,
  미리 예시 코드에 마련해 둔 다섯 **함수 정의**들의 내용물을 채움으로써
  '내 게임'에 대한 내용을 추가해 봅시다.
  - ```
    void Initialize_MyGame(struct Game *)
    void        Run_MyGame(struct Game *)
    void   Finalize_MyGame(struct Game *)

    struct Game *New_MyGame()
    void Delete_MyGame()
    ```
  - 스크롤 올리면 보이는 예시 코드를 참조하면 될 듯 해요.
    - **멤버** data를 **선언**해 두긴 했지만, 자동 칸 잡기 기능을 사용해도 좋아요
      - struct Game **정의** 중괄호 안에 추가로 Data **이름**(**멤버 이름**) **선언**을 적어도 좋아요
      - 다만 컨셉을 지키기 위해 static **위치**는 사용하지 말고 진행해 봐요
    ```

'내 게임' 직접 구성해 보기

- 다섯 **함수 정의**를 잘 적어 두었다면,
  이제 Ctrl + F5를 누르면 '내 게임'을 직접 실행해 볼 수 있을 거예요!

'내 게임' 직접 구성해 보기

- 다섯 **함수 정의**를 잘 적어 두었다면,
  이제 Ctrl + F5를 누르면 '내 게임'을 직접 실행해 볼 수 있을 거예요!
  - 1번째 게임을 '내 게임'으로 설정
  - 사용자가 1을 입력했을 때, 내 **게임**의 Code 흐름에 해당하는 각 **함수**들을 호출

'내 게임' 직접 구성해 보기

- main() **정의** 내용물을 잠시 간추려 보면...

```
games[1] = New_MyGame();

game_selected = games[1];

game_selected->Initialize(game_selected);
game_selected->Run(game_selected);
game_selected->Finalize(game_selected);

Delete_MyGame(games[1]);
```

'내 게임' 직접 구성해 보기

- main() **정의** 내용물을 잠시 간추려 보면...

```
games[1] = New_MyGame();

game_selected = games[1];

game_selected->Initialize(game_selected);
game_selected->Run(game_selected);
game_selected->Finalize(game_selected);

Delete_MyGame(games[1]);
```

여기 있는 각 **문장**들의 의미를
함께 살짝 떠올려 보도록 합시다.

'내 게임' 직접 구성해 보기

- main() **정의** 내용물을 잠시 간추려 보면…

```
games[1] = New_MyGame();

game_selected = games[1];

game_selected->Initialize(game_selected);
game_selected->Run(game_selected);
game_selected->Finalize(game_selected);

Delete_MyGame(games[1]);
```

'내 게임' **object**가 형성되어 담김

'내 게임' **object**가 선택됨

'내 게임' **object**가 소멸함

'내 게임' 직접 구성해 보기

- main() **정의** 내용물을 잠시 간추려 보면...

```
games[1] = New_MyGame();

game_selected = games[1];

game_selected->Initialize(game_selected);
game_selected->Run(game_selected);
game_selected->Finalize(game_selected);

Delete_MyGame(games[1]);
```

이 세 줄은 그 의미가 눈에 잘 들어오나요?
왜 동일한 **이름**을 **수식** 왼쪽, 오른쪽에 각각 적고 있나요?

'내 게임' 직접 구성해 보기

- main() **정의** 내용물을 잠시 간추려 보면...

```
games[1] = New_MyGame();

game_selected = games[1];

game_selected->Initialize(game_selected);
game_selected->Run(game_selected);
game_selected->Finalize(game_selected);

Delete_MyGame(games[1]);
```

'내 게임' **object**에는 **실행**할 Code에 대한 정보가 담겨 있어요.
따라서 이런 식으로 **수식**을 적어야 할 거예요.
(games[0]의 것을 호출하면 안 됨!)

'내 게임' 직접 구성해 보기

- main() **정의** 내용물을 잠시 간추려 보면...

```
games[1] = New_MyGame()

game_selected = games[1

game_selected->Initialize(game_selected);
game_selected->Run(game_selected);
game_selected->Finalize(game_selected);

Delete_MyGame(games[1]);
```

반면, '내 게임용 Code를 **실행**하기 위해 필요한 Data' 또한 '내 게임' **object**에 그 정보가 담겨 있어요. 따라서 우리는 선택한 **함수**를 호출하기 위해 다시 한 번 동일한 **이름**을 적어 주어야 해요. (games[0]을 담아 호출하면 안 됨!)

'내 게임' 직접 구성해 보기

- main() **정의** 내용물을 잠시 간추려 보면...

```
games[1] = New_MyGam

game_selected = game

game_selected->Initialize(game_selected);
game_selected->Run(game_selected);
game_selected->Finalize(game_selected);

Delete_MyGame(games[1]);
```

'내 게임' Code를 '내 게임' Data와 연계하여 **실행**해야 함은 자명해요.
따라서 이 **수식** 또한,
두 곳에 동일한 **이름**을 적어야만 그 의도를 달성할 수 있게 될 거예요!

'내 게임' 직접 구성해 보기

- main() **정의** 내용물을 잠시 간추려 보면...

```
games[1] = New_MyGame();

game_selected = game

game_selected->Init
game_selected->Run(game_se
game_selected->Finalize(g          elected);

Delete_MyGame(games[1]);
```

비슷한 느낌으로,
만약 games[1]에 '내 게임' **object**를 형성시켜 (그 시작 **위치**를)담는다면,
해당 **object**를 소멸시킬 때도 반드시 '내 게임용 **함수**'를 써야 할 거예요.
(뭐 지금은 큰 차이가 없긴 하지만 의미 측면에선 그래요)

'내 게임' 직접 구성해 보기

- main() **정의** 내용물을 잠시 간추려 보면…

```
games[1] = New_
game_selected
game_selected
game_selected->Run(game_selected);
game_selected->Finalize(game_selected);

Delete_MyGame(games[1]);
```

중요한 것은, 이 코드들이 지금 main()에 적혀 있다는 것이에요.
미래에 main()을 짜게 될 사람은 우리가 만든 '게임 시스템'을 사용하기 위해
방금 본 두 가지 규칙을 반드시 잘 준수하도록 Code / Data 흐름을 구성해야 해요.

문제점

- main() **정의** 내용물을 잠시 간추려 보면...

```
games[1] = New_
game_selected
game_selected
game_selected->Run(game_selected);
game_selected->Finalize(game_selected);

Delete_MyGame(games[1]);
```

중요한 것은, 이 코드들이 지금 main()에 적혀 있다는 것이에요.
미래에 main()을 짜게 될 사람은 우리가 만든 '게임 시스템'을 사용하기 위해
방금 본 두 가지 규칙을 반드시 잘 준수하도록 Code / Data 흐름을 구성해야 해요.

6-1에서 보았듯, 이 부분이 문제의 원인이 될 가능성이 높아졌지요.

C++를 썼다면...

- 만약 우리가 C++를 썼다면...

```
games[1] = New_MyGame();

game_selected = games[1];

game_selected->Initialize(game_selected);
game_selected->Run(game_selected);
game_selected->Finalize(game_selected);

Delete_MyGame(games[1]);
```

C++를 썼다면...

- 만약 우리가 C++를 썼다면 이런 식으로 코드를 적을 수 있게 돼요!

```
games[1] = New_MyGame();

game_selected = games[1];

game_selected->Initialize(game_selected);
game_selected->Run(game_selected);
game_selected->Finalize(game_selected);

Delete_MyGame(games[1]);
```

```
games[1] = new MyGame();

game_selected = games[1];

game_selected->Initialize();
game_selected->Run();
game_selected->Finalize();

delete games[1];
```

C++를 썼다면...

- 만약 우리가 C++를 썼다면 이런 식으로 코드를 적을 수 있게 돼요!

```
games[1
game_selected = games[1];

game_selected->Initialize(game_selected);
game_selected->Run(game_selected);
game_selected->Finalize(game_selected);

Delete_MyGame(games[1]);
```

항상 game_selected의 Data들을 쓸 예정이므로
굳이 연거푸 적지 않아도 됨!
(알아서 0번째 인수 자리에 담아 줌!)

```
games[1] = new MyGame();

game_selected = games[1];

game_selected->Initialize();
game_selected->Run();
game_selected->Finalize();

delete games[1];
```

항상 '형성했던 그대로 소멸'시킬 예정이므로
굳이 연거푸 적지 않아도 됨!
(알아서 적절한 소멸용 **함수**를 호출해 줌!)

C++를 썼다면...

- 오... C++를 썼다면
  `main()` **정의** 내용물 적을 사람이 '어처구니 없는 실수'를 할 가능성을
  대폭 차단할 수 있는 것 같아요.
  - 오류의 대부분은 오타로 인해 발생하므로(의도 전달 수단이 글자니까 당연),
    '아예 안 적어도 되도록 만드는 것'만으로도 오류의 가능성을 크게 줄일 수 있어요!

  - 특히 방금 본 것처럼,
    'C 문법으로는 막히지 않는, 그러나 의미 관점에서 반드시 일치해야 할'
    ...상황에서는 오타를 C 컴파일러가 잡아주는 것이 거의 불가능하므로
    내 동료가 디버깅 장인이 아닌 이상 C++를 쓰는 것이 더 안전할 것 같아요
    - 한 **object**에 서로 연계되는 Data와 Code에 대한 정보를 몰아서 담아 두는 케이스가 대표적!

마무리

- 좋아요. 이번 시간에 우리는
  C에서 C++로 넘어가기 위한 기본적인 needs를 잠시 느껴 보았어요.
  - 잠시 쉬었다가 새로운 프로그래밍 언어로 살짝 뛰어들어 보도록 합시다