

C++ Korea 7<sup>th</sup> Seminar

# 포인터 없이 게임 개발 해보기

옥찬호

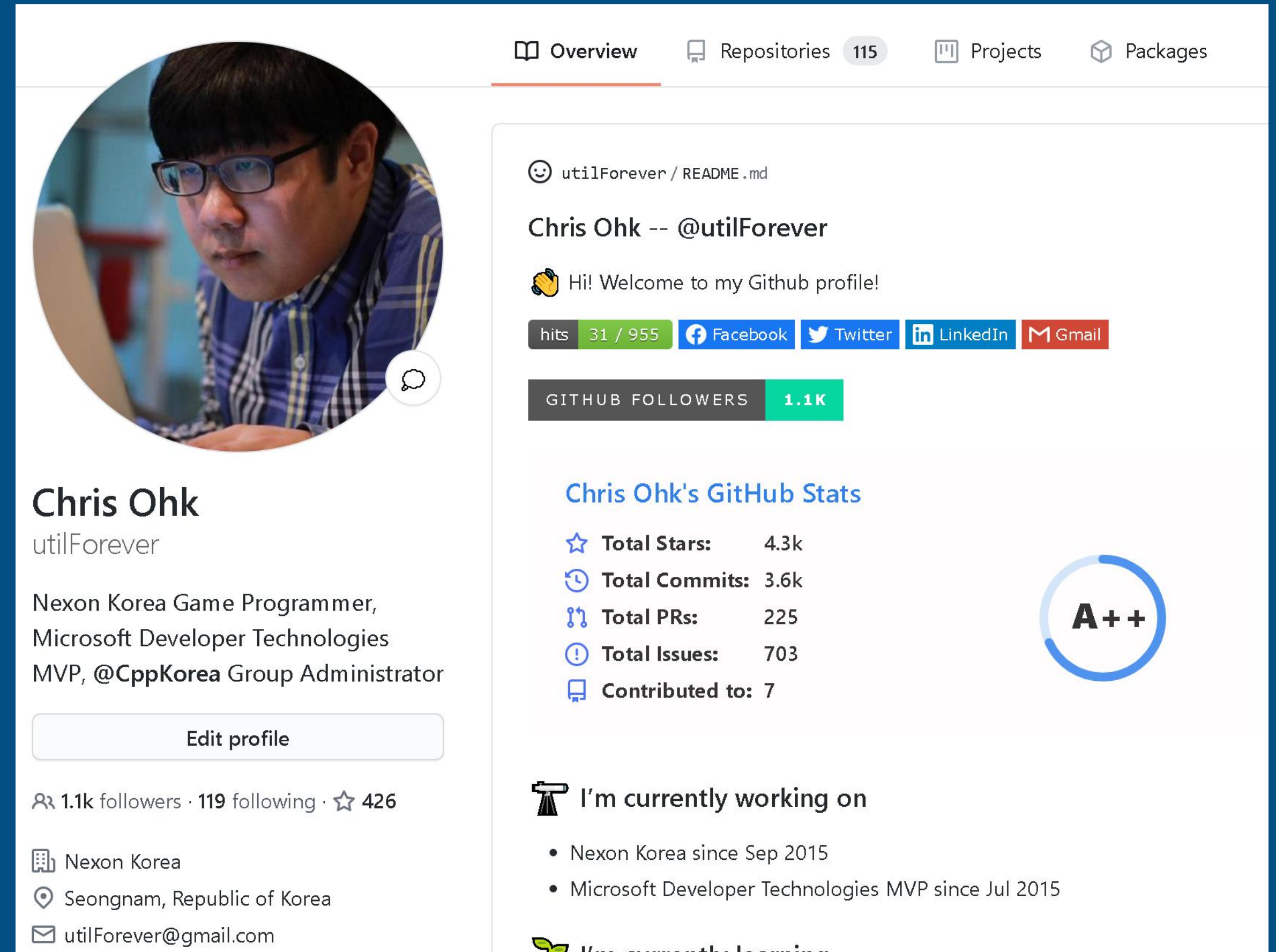
Nexon Korea, Microsoft MVP

utilForever@gmail.com

# 소개

## 옥찬호 (Chris Ohk)

- 넥슨 코리아 게임 프로그래머
- MS Developer Technologies MVP
- 페이스북 그룹 C++ Korea 대표
- IT 전문서 집필 및 번역 다수
  - 게임샐러드로 코드 한 줄 없이 게임 만들기 (2013)
  - 유니티 Shader와 Effect 제작 (2014)
  - 2D 게임 프로그래밍 (2014)
  - 러스트 핵심 노트 (2017)
  - 모던 C++ 입문 (2017)
  - C++ 최적화 (2019)



The screenshot shows the GitHub profile of Chris Ohk (@utilForever). The profile includes a circular profile picture of a man with glasses, a bio stating he is a Nexon Korea Game Programmer, Microsoft Developer Technologies MVP, and @CppKorea Group Administrator, and a location of Seongnam, Republic of Korea. The page also displays his GitHub statistics: 1.1k followers, 115 repositories, 426 stars, 3.6k commits, 225 PRs, 703 issues, and 7 contributions. A blue circular badge with 'A++' is visible. The 'About' section lists his current work on Nexon Korea since Sep 2015 and Microsoft Developer Technologies MVP since Jul 2015.

utilForever / README.md

Chris Ohk -- @utilForever

Hi! Welcome to my Github profile!

hits 31 / 955 Facebook Twitter LinkedIn Gmail

GITHUB FOLLOWERS 1.1K

Chris Ohk's GitHub Stats

- ★ Total Stars: 4.3k
- 🕒 Total Commits: 3.6k
- 🔗 Total PRs: 225
- 📌 Total Issues: 703
- 📦 Contributed to: 7

A++

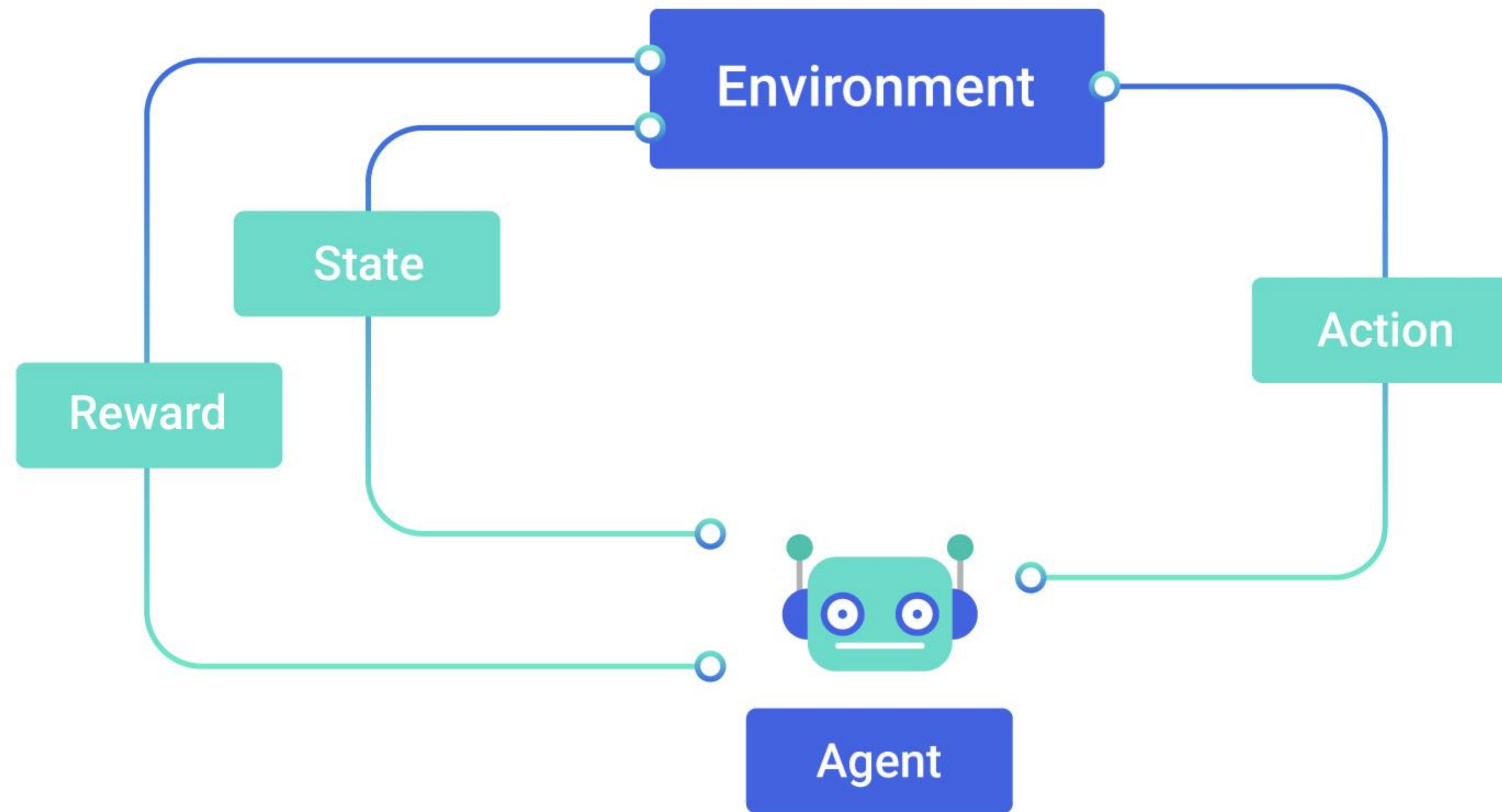
I'm currently working on

- Nexon Korea since Sep 2015
- Microsoft Developer Technologies MVP since Jul 2015

I'm currently learning

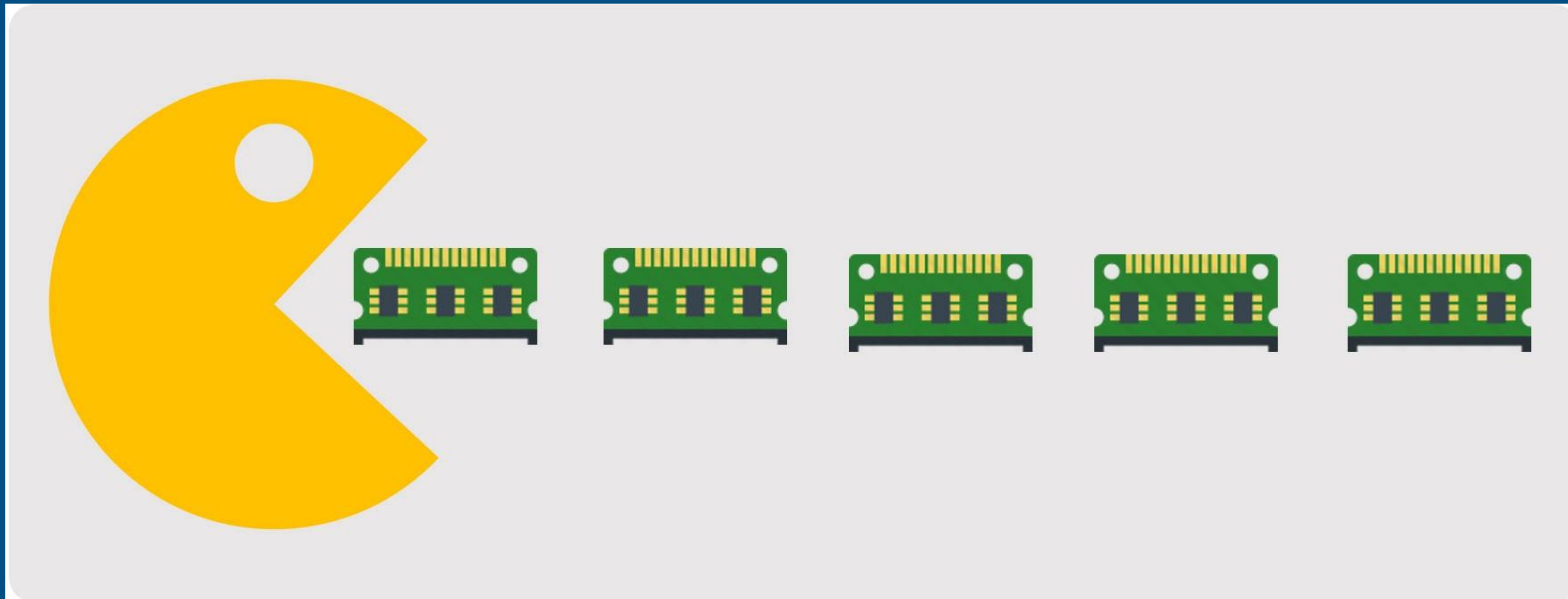
# 포인터를 사용하지 않게 된 이유

C++ Korea 7<sup>th</sup> Seminar  
포인터 없이 게임 개발 해보기



# 포인터를 사용하지 않게 된 이유

C++ Korea 7<sup>th</sup> Seminar  
포인터 없이 게임 개발 해보기





# 포인터를 사용하지 않게 된 이유

---

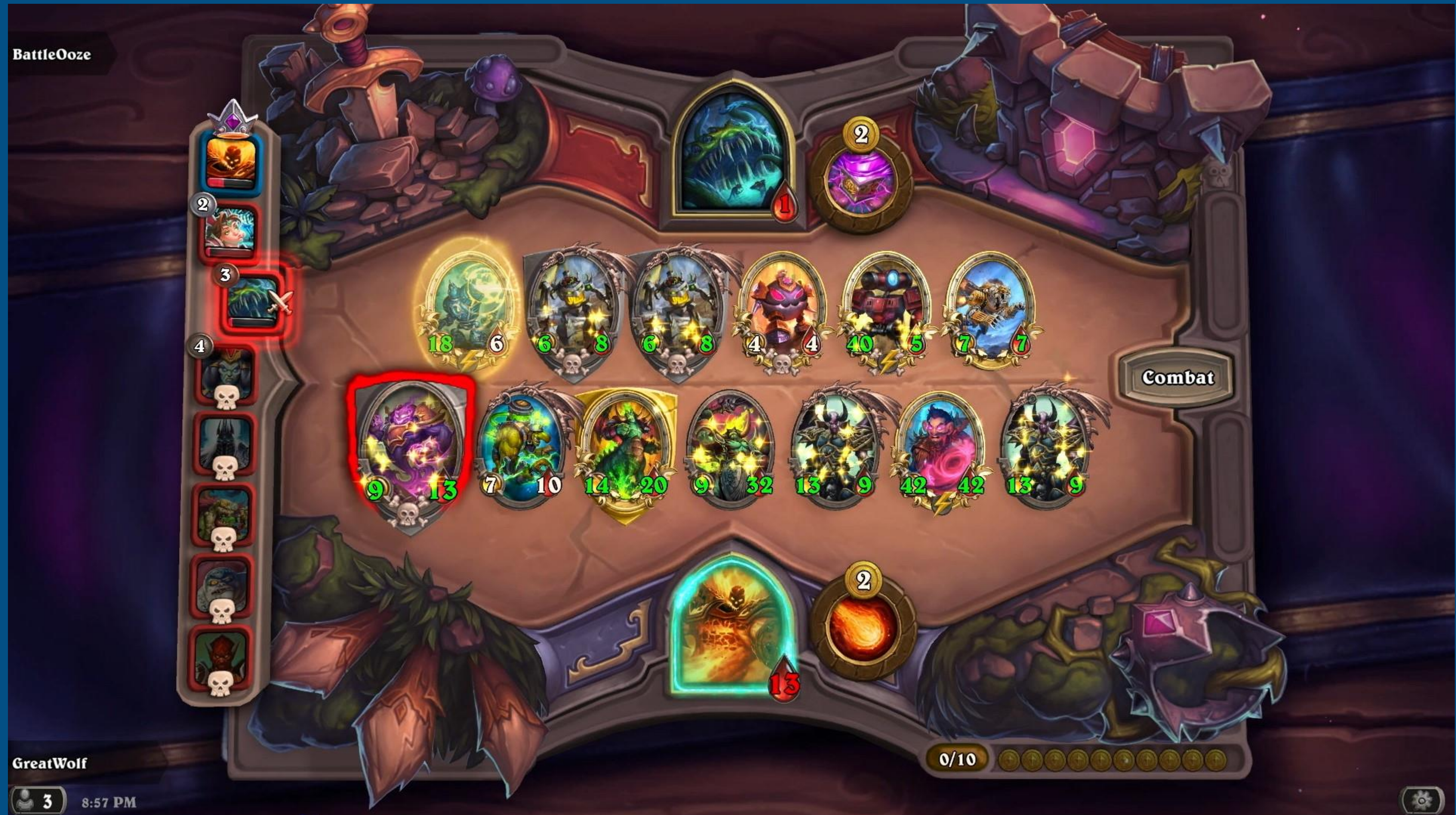
C++ Korea 7<sup>th</sup> Seminar  
포인터 없이 게임 개발 해보기





# 포인터를 사용하지 않게 된 이유

C++ Korea 7<sup>th</sup> Seminar  
포인터 없이 게임 개발 해보기





- 스마트 포인터
  - `std::unique_ptr`
  - `std::shared_ptr`
  - `std::weak_ptr`
  - ~~`std::auto_ptr`~~
- 레퍼런스 ← 오늘의 주제

- 선언과 동시에 반드시 초기화해야 한다.
  - 초기화된 후에는 다른 변수를 참조하도록 변경할 수 없다.
- 이 두 특징으로 인해 포인터의 대안으로 사용할 때 불편한 부분이 많다.



# 레퍼런스라서 불편한 부분

---

C++ Korea 7<sup>th</sup> Seminar  
포인터 없이 게임 개발 해보기

- 상호 참조/자기 자신을 참조
- 상속과 다형성
- NULL 값의 표현

- 두 클래스가 서로의 클래스를 참조하고 있는 경우

```
#include "B.h"
```

```
class A  
{  
    public:  
        B b;  
};
```



```
#include "A.h"
```

```
class B  
{  
    public:  
        A a;  
};
```



- 해결 방법 : 전방 선언 (Forward Declaration)
  - 포인터의 경우

```
class B;  
  
class A  
{  
    public:  
        B* b;  
};
```



```
#include "A.h"  
  
class B  
{  
    public:  
        A a;  
};
```

- 해결 방법 : 전방 선언 (Forward Declaration)
  - 레퍼런스의 경우

```
class B;  
  
class A  
{  
    public:  
        B& b;  
};
```



```
#include "A.h"  
  
class B  
{  
    public:  
        A a;  
};
```



- 한 클래스에서 자기 자신의 클래스 타입을 갖는 변수가 있는 경우

```
class A
{
    public:
        A& a;
};
```

- 이 변수들의 값을 당장 결정할 수 없다면...?

```
class B;  
  
class A  
{  
    public:  
        B& b = ?;  
};
```

```
class A  
{  
    public:  
        A& a = ?;  
};
```



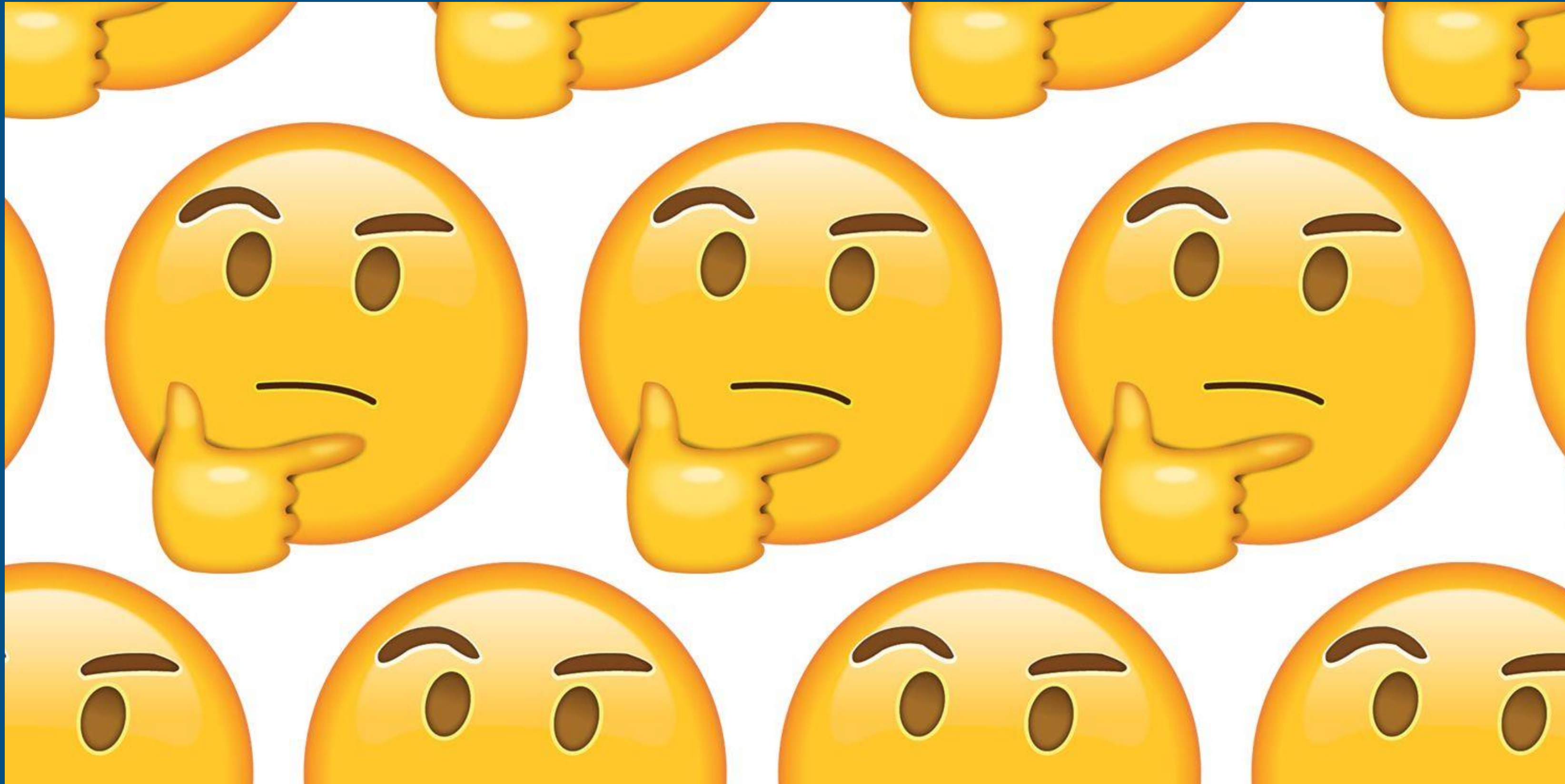
- 상속과 다형성 예제에 항상 등장하는 포인터

```
Playable* Entity::GetFromCard(Player* player, Card* card,
                               std::optional<std::map<GameTag, int>> cardTags, IZone* zone, int id) {
    Playable* result;

    switch (card->GetCardType()) {
        case CardType::HERO:
            result = new Hero(player, card, tags, id);
            break;
        case CardType::HERO_POWER:
            result = new HeroPower(player, card, tags, id);
            break;
        case CardType::MINION:
            result = new Minion(player, card, tags, id);
            break;
        case CardType::SPELL:
            result = new Spell(player, card, tags, id);
            break;
        case CardType::WEAPON:
            result = new Weapon(player, card, tags, id);
            break;
        default:
            throw std::invalid_argument("Generic::DrawCard() - Invalid card type!");
    }

    return result;
}
```

- 이걸 레퍼런스로 구현하려면...?





# NULL 값의 표현

---

C++ Korea 7<sup>th</sup> Seminar  
포인터 없이 게임 개발 해보기

- 포인터 : `nullptr`

```
int* pA = nullptr;
```

- 레퍼런스 : X (선언과 동시에 반드시 초기화해야 한다.)

```
int& rA = ?;
```

# 어떤 대안이 있을까?

---

C++ Korea 7<sup>th</sup> Seminar  
포인터 없이 게임 개발 해보기

- 람다 표현식
- `std::variant` & `std::visit`
- `std::optional`



- 함수 포인터나 함수 객체를 대체할 수 있는 방법

```
#include <array>
#include <algorithm>

struct SumFunctor {
    SumFunctor(int& number) : sum(number) {}
    void operator()(int& number) { sum += number; }

private:
    int& sum;
};

int main() {
    std::array<int, 5> numbers = { 1, 2, 3, 4, 5 };
    int sum = 0;

    std::for_each(numbers.begin(), numbers.end(), SumFunctor(sum));

    // lambda로 구현
    sum = 0;
    std::for_each(numbers.begin(), numbers.end(), [&sum](int& number) {
        sum += number;
    });
}
```

- 전장은 8명의 플레이어가 진행하지만, 턴마다 상대 플레이어가 달라진다. 따라서 Player 클래스에 상대 플레이어를 저장할 변수가 필요하다.
- 하지만 상대 플레이어가 턴마다 달라지기 때문에 레퍼런스 타입의 변수는 만들 수 없다. 게다가 포인터도 사용하지 않는 상황이다.
- 이 문제를 어떻게 대처할 수 있을까?

```
class Player
{
public:
    ...

    std::function<void(Player&)> selectHeroCallback;
    std::function<void(Player&)> prepareTavernMinionsCallback;
    std::function<void(Player&, std::size_t)> purchaseMinionCallback;
    std::function<void(int)> returnMinionCallback;
    std::function<void(Player&)> clearTavernMinionsCallback;
    std::function<void(Player&)> upgradeTavernCallback;
    std::function<void()> completeRecruitCallback;
    std::function<Player&(Player&)> getOpponentPlayerCallback;
    std::function<void(Player&)> processDefeatCallback;
};
```



```
void Game::Start()
{
    ...

    // Create callback to get opponent player
    auto getOpponentPlayerCallback = [this](Player& player) -> Player& {
        const std::size_t idx = FindPlayerNextFight(player.idx);
        return m_gameState.players[idx];
    };

    // Initialize variables and callbacks
    for (auto& player : m_gameState.players)
    {
        ...

        player.getOpponentPlayerCallback = getOpponentPlayerCallback;

        ++playerIdx;
    }
}
```

# 실제 예제

C++ Korea 7<sup>th</sup> Seminar  
포인터 없이 게임 개발 해보기

```
std::vector<std::reference_wrapper<Minion>> IncludeTask::GetMinions(
    EntityType entityType, [[maybe_unused]] Player& player, Minion& source)
{
    std::vector<std::reference_wrapper<Minion>> minions;

    switch (entityType)
    {
        ...

        case EntityType::ENEMY_MINIONS:
        {
            Player& opponent = player.getOpponentPlayerCallback(player);
            opponent.GetField().ForEachAlive([&](MinionData& minion) {
                minions.emplace_back(minion.value());
            });
        }
    }
}
```

# std::variant & std::visit

C++ Korea 7<sup>th</sup> Seminar  
포인터 없이 게임 개발 해보기

- std::variant : 타입 세이프한 공용체 (Union)

```
#include <variant>

int main() {
    std::variant<int, float> v = 12;

    int i = std::get<int>(v);

    try {
        // v contains int, not float: will throw
        float f = std::get<float>(v);
    }
    catch (const std::bad_variant_access&) {}
}
```



# std::variant & std::visit

C++ Korea 7<sup>th</sup> Seminar  
포인터 없이 게임 개발 해보기

- std::visit : 방문자 패턴 (Visitor Pattern)의 구현을 위해 사용

```
#include <iostream>
#include <string>
#include <variant>
#include <vector>

// the variant to visit
using var_t = std::variant<int, long, double, std::string>;

int main() {
    std::vector<var_t> vec = { 10, 151, 1.5, "hello" };

    for (auto& v : vec) {
        // void visitor, only called for side-effects (here, for I/O)
        std::visit([](auto&& arg) {std::cout << arg; }, v);
    }
}
```

- 전장에서 각 플레이어는 하수인 또는 주문 카드를 받을 수 있다.  
하수인과 주문 카드는 서로 같은 함수를 사용하지만 세부 구현은 다르다.  
기존에는 상속 및 다형성으로 이 문제를 해결했는데 포인터가 없다.
- 이 문제를 어떻게 대처할 수 있을까?

# 실제 예제

C++ Korea 7<sup>th</sup> Seminar  
포인터 없이 게임 개발 해보기

```
namespace RosettaStone::Battlegrounds
{
using CardData = std::variant<Minion, Spell>;
//!
//! \brief HandZone class.
//!
//! This class is where each player keeps the cards currently available to
//! them. The player can see their hand face-up at the bottom of the screen,
//! while the opponent's hand is shown face-down at the top of the screen.
//!
class HandZone
{
    ...

private:
    const ZoneType m_type = ZoneType::HAND;

    std::array<std::optional<CardData>, MAX_HAND_SIZE> m_cards;
    int m_count = 0;
};
```



# 실제 예제

C++ Korea 7<sup>th</sup> Seminar  
포인터 없이 게임 개발 해보기

```
const CardData HandZone::Remove(CardData& card) {
    const ZoneType cardZone = std::visit(
        [&](auto&& _card) -> ZoneType { return _card.GetZoneType(); }, card);
    if (cardZone != m_type) {
        throw std::logic_error("Couldn't remove entity from zone.");
    }

    const int cardPos = std::visit(
        [&](auto&& _card) -> int { return _card.GetZonePosition(); }, card);
    int count = m_count;

    CardData result = m_cards.at(cardPos).value();

    ...

    return result;
}
```

# 실제 예제

C++ Korea 7<sup>th</sup> Seminar  
포인터 없이 게임 개발 해보기

```
ZoneType Minion::GetZoneType() const
{
    return m_zoneType;
}

int Minion::GetZonePosition() const
{
    return m_zonePos;
}
```

```
ZoneType Spell::GetZoneType() const
{
    return m_zoneType;
}

int Spell::GetZonePosition() const
{
    return m_zonePos;
}
```

- 전장에서 사용하는 카드는 다양한 효과를 갖고 있다.  
어떤 카드는 상대 하수인에게 데미지를 입히고,  
어떤 카드는 내 하수인들에게 버프를 걸어준다.  
‘카드의 효과를 발동한다’는 동일하지만 세부 동작은 각각 다르다.  
기존에는 상속 및 다형성으로 이 문제를 해결했는데 포인터가 없다.
- 이 문제를 어떻게 대처할 수 있을까?



# 실제 예제

C++ Korea 7<sup>th</sup> Seminar  
포인터 없이 게임 개발 해보기

```
#include <variant>

namespace RosettaStone::Battlegrounds
{
    using namespace SimpleTasks;
    using TaskType =
        std::variant<AddEnchantmentTask, CountTask, DamageHeroTask, DamageTask,
                    GetGameTagTask, RandomTask, RepeatNumberEndTask,
                    RepeatNumberStartTask, SummonTask>;
} // namespace RosettaStone::Battlegrounds
```

# 실제 예제

C++ Korea 7<sup>th</sup> Seminar  
포인터 없이 게임 개발 해보기

```
void Minion::ActivateTask(PowerType type, Player& player) {
    auto tasks = GetTasks(type);
    if (tasks.empty()) {
        return;
    }

    for (auto& task : tasks) {
        if (player.taskStack.isStackingTasks &&
            !std::holds_alternative<RepeatNumberEndTask>(task)) {
            player.taskStack.tasks.emplace_back(task);
        }
        else {
            std::visit([&](auto&& _task) { _task.Run(player, *this); }, task);
        }
    }
}
```

- 값이 존재하지 않음을 우아하게 표현할 수 있는 방법

```
#include <string>
#include <functional>
#include <iostream>
#include <optional>

// optional can be used as the return type of a factory that may fail
std::optional<std::string> Create(bool b) {
    if (b) {
        return "Godzilla";
    }
    return {};
}

int main() {
    std::cout << "Create(false) returned " << Create(false).value_or("empty") << '\n';
}
```

- 전장은 최대 7장의 하수인 카드를 필드에 배치할 수 있다.  
그래서 `std::array<Minion, 7> field;`로 만들었다.
- 하지만 이 필드에는 하수인이 있을 수도 있고 없을 수도 있다.  
(카드를 필드에 배치하기 전까지는 하수인이 존재하지 않는다.)
- 포인터를 사용했을 땐 `nullptr`로 처리하면 되지만  
사용하지 않기로 한 지금, `NULL`을 표현할 방법이 없다.
- 이 문제를 어떻게 대처할 수 있을까?



# 실제 예제

C++ Korea 7<sup>th</sup> Seminar  
포인터 없이 게임 개발 해보기

```
#include <array>
#include <optional>

namespace RosettaStone::Battlegrounds
{
    using MinionData = std::optional<Minion>;

    class FieldZone
    {
        ...

    private:
        const ZoneType m_type = ZoneType::PLAY;

        std::array<MinionData, MAX_FIELD_SIZE> m_minions;
        int m_count = 0;
    };
}
```

# 실제 예제

C++ Korea 7<sup>th</sup> Seminar  
포인터 없이 게임 개발 해보기

```
Minion& FieldZone::operator[](int zonePos)
{
    return m_minions.at(zonePos).value();
}

const Minion& FieldZone::operator[](int zonePos) const
{
    return m_minions.at(zonePos).value();
}
```

# 실제 예제

C++ Korea 7<sup>th</sup> Seminar  
포인터 없이 게임 개발 해보기

```
void FieldZone::Add(Minion& minion, int zonePos) {
    if (zonePos > m_count) {
        throw std::invalid_argument("Zone position isn't in a valid range.");
    }

    const int pos = zonePos < 0 ? m_count : zonePos;
    if (IsFull()) {
        return;
    }
    if (pos < 0 || pos == m_count) {
        m_minions[m_count] = minion;
    }
    else {
        for (int i = m_count - 1; i >= pos; --i) {
            m_minions[i + 1] = m_minions[i];
        }
        m_minions[pos] = minion;
    }

    ++m_count;
    m_minions[pos].value().SetZoneType(m_type);
    Reposition(zonePos);
}
```

- `std::variant` & `std::visit`
- `std::optional`



# std::variant & std::visit

C++ Korea 7<sup>th</sup> Seminar  
포인터 없이 게임 개발 해보기

- std::visit를 사용할 때 함수를 호출할 경우 std::variant 타입에 정의되어 있는 모든 클래스에 해당 함수를 선언 및 정의해야 합니다.

```
ZoneType Minion::GetZoneType() const
{
    return m_zoneType;
}

int Minion::GetZonePosition() const
{
    return m_zonePos;
}
```

```
ZoneType Spell::GetZoneType() const
{
    return m_zoneType;
}

int Spell::GetZonePosition() const
{
    return m_zonePos;
}
```

- 잘못 사용할 경우 의도치 않은 결과가 나올 수 있습니다.

```
#include <optional>
#include <iostream>
#include <array>

struct A {
    ~A() { a = 10; }
    int a = 30;
};

int main() {
    std::array<std::optional<A>, 3> arr;
    arr[0] = A();
    arr[1] = A();

    arr[0]->a = 100;
    arr[1]->a = 200;

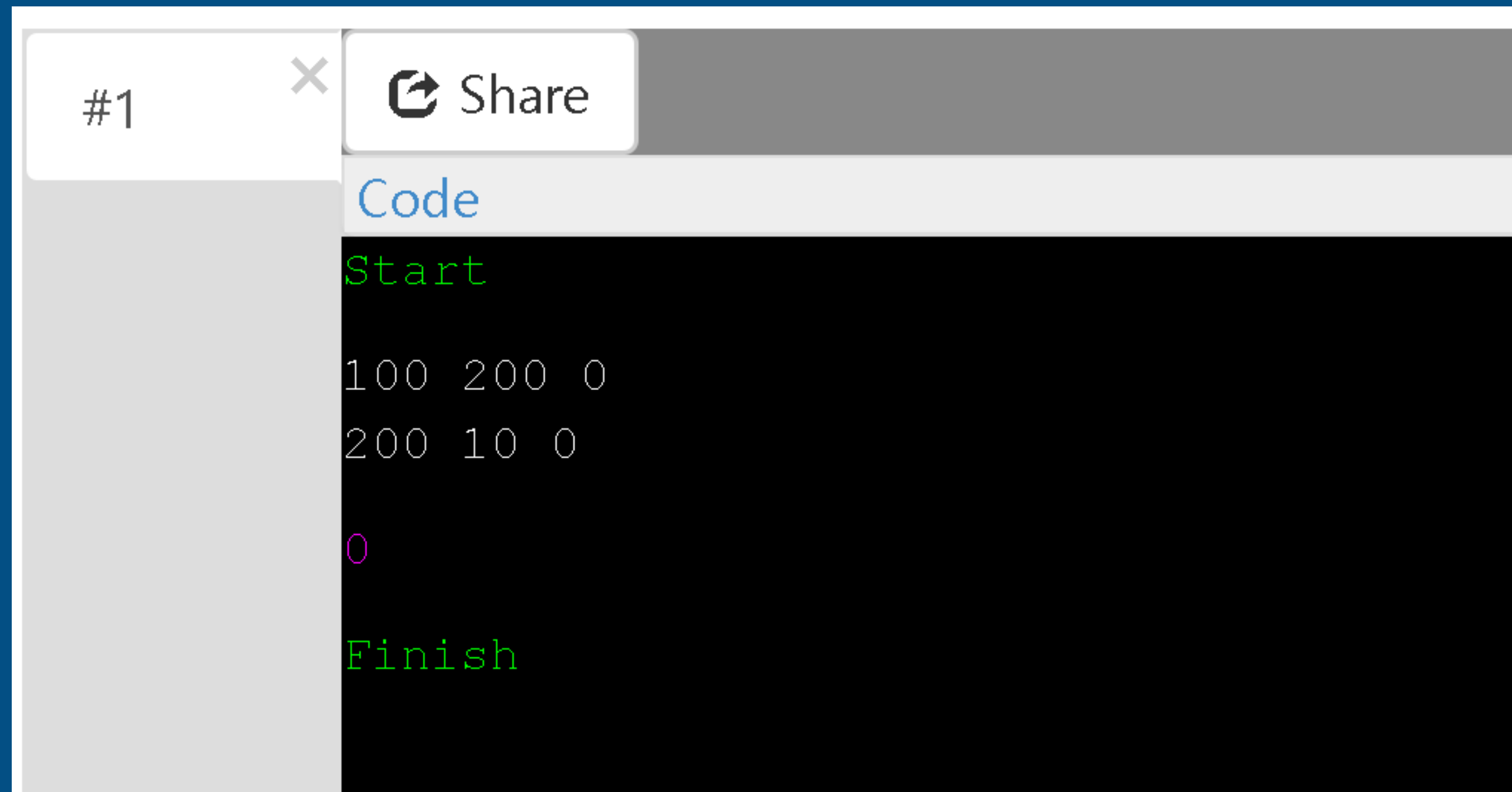
    std::cout << arr[0]->a << ' ' << arr[1]->a << ' ' << arr[2]->a << '\n';

    arr[0] = arr[1];
    arr[1] = arr[2];
    arr[2] = std::nullopt;

    std::cout << arr[0]->a << ' ' << arr[1]->a << ' ' << arr[2]->a << '\n';
}
```

# std::optional

- 잘못 사용할 경우 의도치 않은 결과가 나올 수 있습니다.



The screenshot shows a code editor window with a tab labeled '#1'. The code is as follows:

```
Code
Start
100 200 0
200 10 0

0

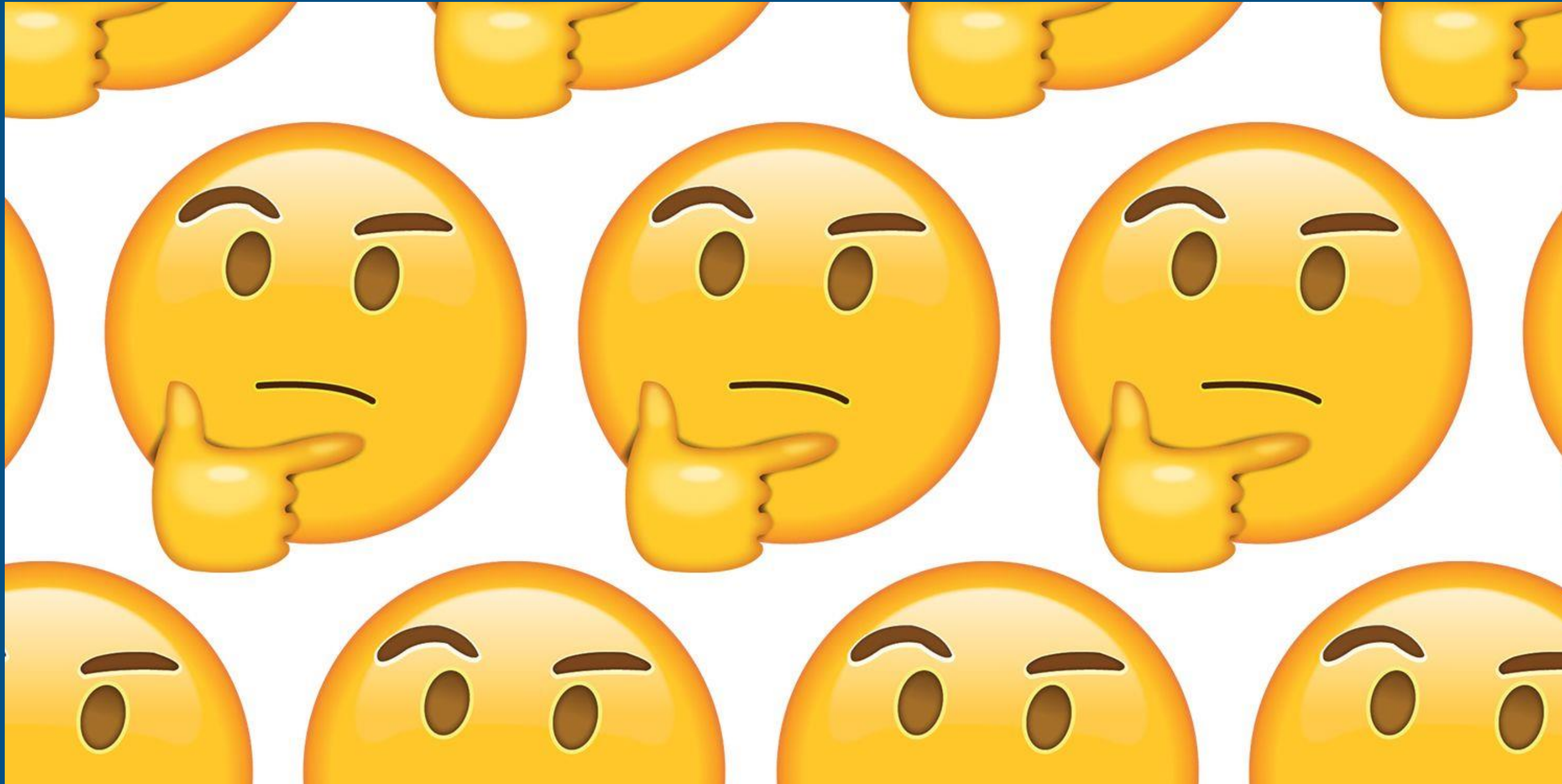
Finish
```

# std::optional

---

C++ Korea 7<sup>th</sup> Seminar  
포인터 없이 게임 개발 해보기

- 잘못 사용할 경우 의도치 않은 결과가 나올 수 있습니다.





- 잘못 사용할 경우 의도치 않은 결과가 나올 수 있습니다.

```
#include <optional>
#include <iostream>
#include <array>

struct A {
    ~A() { a = 10; }
    int a = 30;
};

int main() {
    std::array<std::optional<A>, 3> arr;
    arr[0] = A();
    arr[1] = A();

    arr[0]->a = 100;
    arr[1]->a = 200;

    std::cout << arr[0]->a << ' ' << arr[1]->a << ' ' << arr[2]->a << '\n';

    arr[0] = arr[1];
    arr[1] = arr[2];
    arr[2] = std::nullopt;

    std::cout << arr[0]->a << ' ' << arr[1]->a << ' ' << arr[2]->a << '\n';
}
```

```
// observers [optional.object.observe]
_NODISCARD constexpr const _Ty* operator->() const {
    return _STD addressof(this->_Value);
}

_NODISCARD constexpr _Ty* operator->() {
    return _STD addressof(this->_Value);
}
```

- 잘못 사용할 경우 의도치 않은 결과가 나올 수 있습니다.

```
#include <optional>
#include <iostream>
#include <array>

struct A {
    ~A() { a = 10; }
    int a = 30;
};

int main() {
    std::array<std::optional<A>, 3> arr;
    arr[0] = A();
    arr[1] = A();

    arr[0]->a = 100;
    arr[1]->a = 200;

    std::cout << arr[0]->a << ' ' << arr[1]->a << ' ' << arr[2]->a << '\n';

    arr[0] = arr[1];
    arr[1] = arr[2];
    arr[2] = std::nullopt;

    std::cout << arr[0]->a << ' ' << arr[1]->a << ' ' << arr[2]->a << '\n';
}
```

```
// assignment [optional.object.assign]
optional& operator=(nullopt_t) noexcept {
    reset();
    return *this;
}
```

```
void reset() noexcept {
    if (_Has_value) {
        _Destroy_in_place(_Value);
        _Has_value = false;
    }
}
```

- 잘못 사용할 경우 의도치 않은 결과가 나올 수 있습니다.

```
#include <optional>
#include <iostream>
#include <array>

struct A {
    ~A() { a = 10; }
    int a = 30;
};

int main() {
    std::array<std::optional<A>, 2> arr;
    arr[0] = A();
    arr[1] = std::nullopt;

    arr[0]->a = 100;
    arr[1]->a = 200;

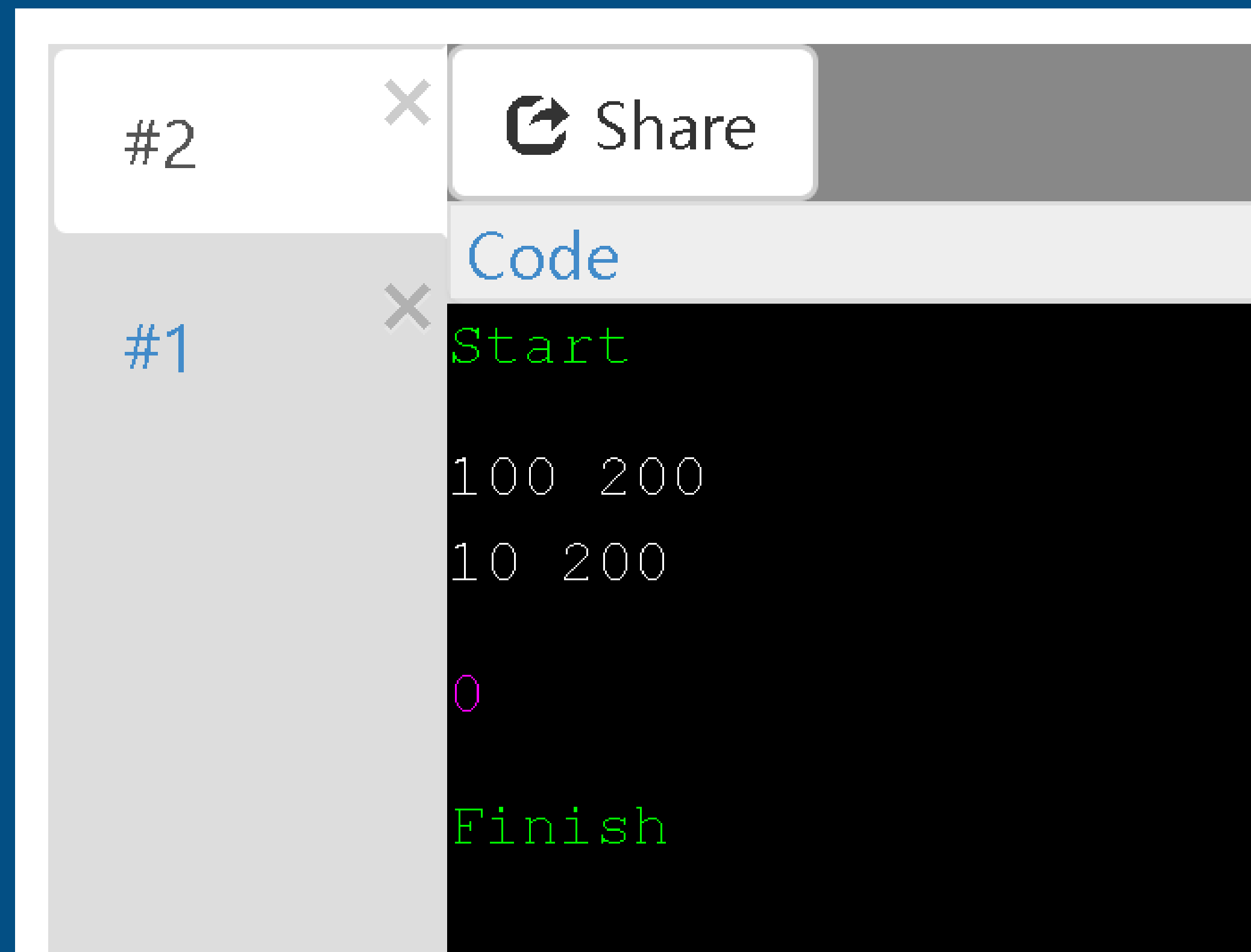
    std::cout << arr[0]->a << ' ' << arr[1]->a << '\n';

    arr[0] = arr[1];
    arr[1] = std::nullopt;

    std::cout << arr[0]->a << ' ' << arr[1]->a << '\n';
}
```

# std::optional

- 잘못 사용할 경우 의도치 않은 결과가 나올 수 있습니다.





- 잘못 사용할 경우 의도치 않은 결과가 나올 수 있습니다.

```
#include <optional>
#include <iostream>
#include <array>

struct A {
    ~A() { a = 10; }
    int a = 30;
};

int main() {
    std::array<std::optional<A>, 2> arr;
    arr[0] = A();
    arr[1] = std::nullopt;

    arr[0]->a = 100;
    arr[1]->a = 200;

    std::cout << arr[0]->a << ' ' << arr[1]->a << '\n';

    arr[0] = arr[1];
    arr[1] = std::nullopt;

    std::cout << arr[0]->a << ' ' << arr[1]->a << '\n';
}
```

```
// assignment [optional.object.assign]
optional& operator=(nullopt_t) noexcept {
    reset();
    return *this;
}
```

```
void reset() noexcept {
    _Has_value = false;
}
```

- 포인터를 사용하지 않고 게임 개발을 할 수 있습니다. (~~하지 마세요.~~)
  - 상호 참조/자기 자신을 참조 문제는 콜백 함수 with 람다식을 사용하면 됩니다.
  - 상속 및 다형성 문제는 std::variant & std::visit를 사용하면 됩니다.
  - NULL 값의 표현 문제는 std::optional을 사용하면 됩니다.
- 하지만 포인터를 사용하지 않기 때문에 한계도 존재합니다.
  - std::variant와 std::visit을 사용하는 모든 클래스에 똑같은 함수를 선언 및 정의해야 합니다.
  - std::optional은 잘못 사용하면 의도치 않은 결과가 나올 수 있습니다.
- 용도에 맞게 (스마트) 포인터 또는 위 방법을 사용합시다.

- <https://github.com/utilForever/RosettaStone>
  - Includes/Rosetta/Battlegrounds
  - Sources/Rosetta/Battlegrounds
  - Tests/UnitTests/Battlegrounds

# 감사합니다

<http://github.com/utilForever>

질문 환영합니다!