Forward Declaration w C++

Omówienie

17 marca 2025

1 Problem cyklicznych zależności

Założenie: Mamy dwie klasy, które odwołują się do siebie nawzajem.

1.1 Kod powodujący problem

Plik A.h

```
#include "B.h"
class A {
    B objB; // Kompilator jeszcze nie zna klasy B
};
```

Plik B.h

```
#include "A.h"
class B {
    A objA; // Kompilator jeszcze nie zna klasy A
};
```

1.2 Problem

Problem: Kompilator wpada w nieskończoną pętlę – która klasa jest pierwsza?

2 Rozwiązanie: Forward Declaration

Forward Declaration pozwala kompilatorowi wiedzieć, że klasa istnieje, bez załączania całego nagłówka.

2.1 Przykład Forward Declaration

Zamiast `#include "B.h"` w `A.h`, piszemy:

```
class B;  // Forward Declaration
class A {
    B* objB;  // Można używać wskaźników i referencji
};
```

Podobnie w `B.h`:

```
class A;  // Forward Declaration
class B {
    A* objA;
};
```

3 Forward Declaration w praktyce - Przykład z Employee

Plik Employee.h

```cpp
#ifndef EMPLOYEE_H
#define EMPLOYEE_H

#include <string>
class Department;  // Forward declaration

class Employee {
private:
    std::string _name;
    Department* _department;  // Można używać wskaźnika
public:
    explicit Employee(const std::string& name, Department* department = nullptr);
};

#endif
```

4 Dlaczego Forward Declaration jest konieczne?

- Unika cyklicznych zależności – `Employee` i `Department` mogą się wzajemnie znać.
- Przyspiesza kompilację – nie ładuje zbędnych nagłówków.
- Pozwala używać wskaźników i referencji do niepełnych typów.
- Nie można tworzyć obiektów niekompletnej klasy – `Department` musi być w pełni zdefiniowany w `Employee.cpp`.

5 Podsumowanie

- Forward Declaration mówi kompilatorowi: „ta klasa istnieje, ale nie mówimy, jak wygląda”.
- Działa tylko dla wskaźników (`Class* ptr`) i referencji (`Class& ref`).
- Nie działa dla obiektów (`Class obj;` wymaga pełnej definicji klasy).
- Pomaga unikać niepotrzebnych `#include` i problemów z cyklicznymi zależnościami.