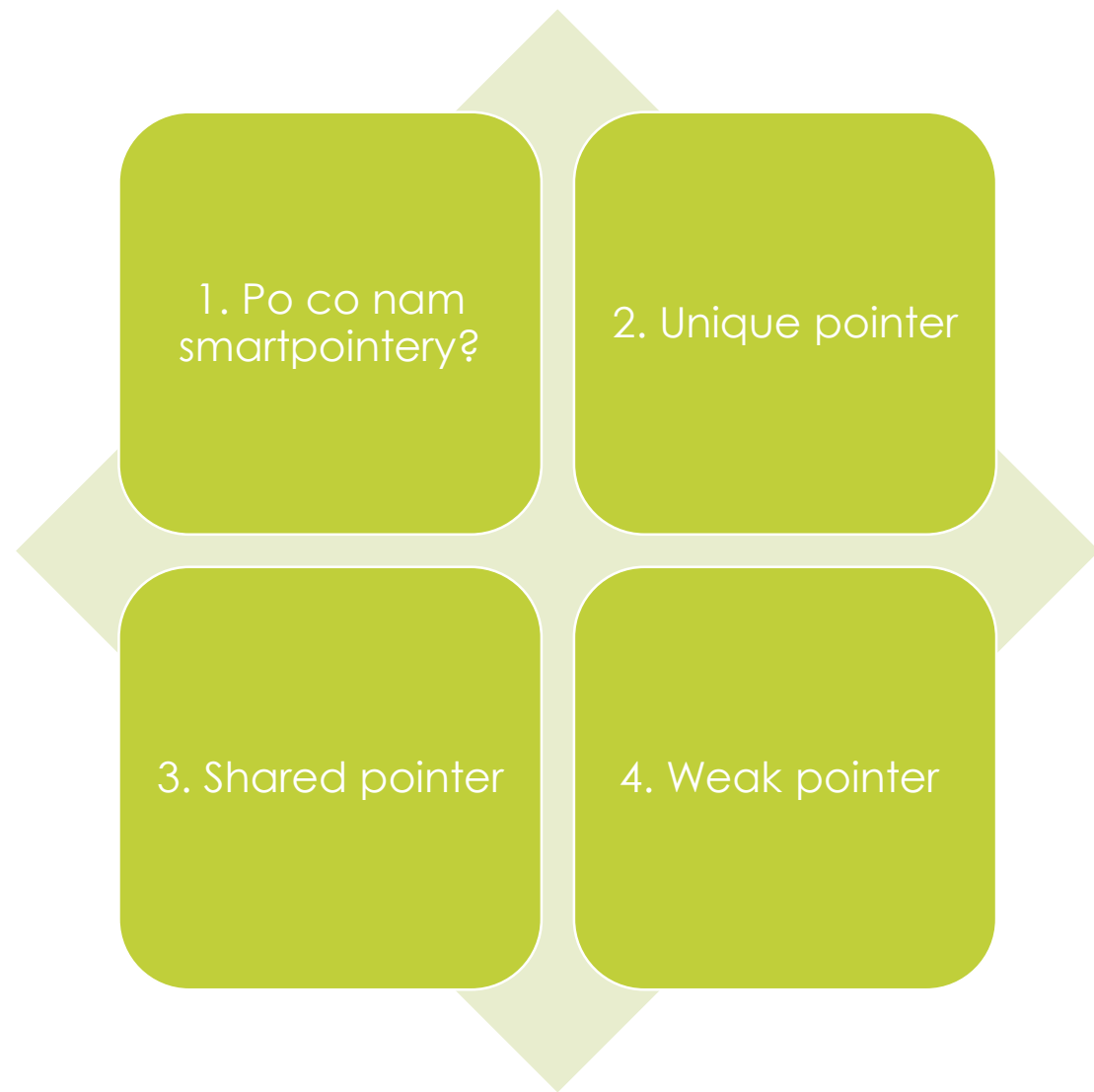




SMARTPOINTERS

Autorzy: Krupowies Wojciech, Urbaś Paulina

Program prezentacji



Co to są smart pointers?

Obiekty

Może wskazywać tylko pamięć przydzieloną do steru

Automatycznie są usuwane kiedy nie są potrzebne

Rodzaje: unique pointers, shared pointers, weak pointers

Auto pointer nie jest wspierany przez C++ 11

Smartpointers

Gdzie są?

- #include <memory>

Zdefiniowane przez szablony klas

- opakowanie dla surowego smartpointera (zawieranie i zarządzanie wskaźnikami)
- mogą mieć custom deleters

Unique pointer

- JEDEN wskaźnik na JEDEN obiekt
- przestaje istnieć wskaźnik = przestaje istnieć obiekt
- Nie możemy go łączyć z innymi wskaźnikami

Przykład 1

```
8
9   int counter = 1;
10
11   struct Obiekt
12   {
13
14       int ID;
15       Obiekt() : ID(counter++)
16       {
17           std::cout << "Obiekt " << ID << " stworzony" << std::endl;
18       }
19       ~Obiekt()
20       {
21           std::cout << "Obiekt " << ID << " zostal usuniety" << std::endl;
22       }
23
24       void DoSome()
25       {
26           std::cout << "Funkcja dziala" << std::endl;
27       }
28   };
29
```

Przykład 2

```
62 void Example4()  
63 {  
64     std::cout << "Example 4" << std::endl;  
65     std::unique_ptr<int> up1(new int);  
66     *up1 = 732;  
67     std::cout << up1 << std::endl; //adres  
68     std::cout << up1.get() << std::endl;  
69     std::cout << "wartosc : " << *up1 << std::endl; //warto  
70     up1.release();  
71     std::cout << "adres po release : " << up1.get() << std::endl;  
72  
73     std::unique_ptr<Obiekt> uP2(new Obiekt);  
74     if (1)  
75     {  
76         std::cout << "TUTAJ KLAMRA OTWIERAJACA" << std::endl;  
77         std::unique_ptr<Obiekt> uP2(new Obiekt);  
78     }  
79     std::cout << "wyjście z if ale nadal jesteśmy w Example 4" << std::endl;  
80 }  
81
```

Konsola debugowania programu Microsoft Visual Studio

Example 4

00000135AFDB51D0

00000135AFDB51D0

wartosc : 732

adres po release : 0000000000000000

Obiekt 1 stworzony

TUTAJ KLAMRA OTWIERAJACA

Obiekt 2 stworzony

Obiekt 2 został usunięty

wyjście z if ale nadal jesteśmy w Example 4

Obiekt 1 został usunięty

Przykład 3

```
void Example5()
{
    std::cout << "Example 5" << std::endl;
    auto obiekt = std::make_unique<Obiekt>(); //mozna tez tak tworzyc unique pointer
    obiekt->DoSome(); //funkcje wywolujemy normalnie
    std::unique_ptr<Obiekt> obiekt2 = obiekt; //tak NIE ROBIMY
}
```


Przykład 4

```
95 void Example6()  
96 {  
97     std::cout << "Example 6" << std::endl;  
98     auto pointerA = std::make_unique<Obiekt>();  
99     std::cout << "Adres wsakznika A : " << pointerA.get() << std::endl;  
100    auto pointerB = std::move(pointerA);  
101    std::cout << "Adres wsakznika A po std::move : " << pointerA.get() << std::endl;  
102    std::cout << "Adres wsakznika B po std::move : " << pointerB.get() << std::endl;  
103 }  
104
```



Konsola debugowania programu Microsoft Visual Studio

```
Example 6  
Obiekt 0 stworzony  
Adres wsakznika A : 000002A371320830  
Adres wsakznika A po std::move : 0000000000000000  
Adres wsakznika B po std::move : 000002A371320830  
Obiekt 0 zostal usuniety
```

Shared pointer

- Zapewniają współdzieloną własność obiektu stosu
- `Shared_ptr <T>`
 - wskazuje na cel typu T na stercie
 - nie jest unikalny, może być wiele `shared_ptr` wskazujących na ten sam obiekt na stercie
 - ustanawia relację współwłasności
 - może być przydzielony i kopiowany
 - może być przenoszony
 - domyślnie nie obsługuje zarządzania tablicami
 - gdy liczba użycia wynosi zero, obiekt zarządzany na stercie jest niszczone

```
void Example1 ()
{
    std::shared_ptr<int> p1{ new int {100} };
    std::cout << *p1 << std::endl;
    std::cout << p1.use_count() << std::endl;


    *p1 = 200;
    std::cout << *p1 << std::endl;
    std::shared_ptr<int> p2{ p1 }; //can we do it?
    std::cout << p1.use_count() << std::endl;

    p1.reset();
    std::cout << p1.use_count() << std::endl;
    std::cout << p2.use_count() << std::endl;
} //automatically deleted
```

SHARED POINTERS PRZYKŁAD

SHARED POINTERS — VECTOR | MOVE

```
void Example2()
{
    std::cout << "Example 2" << std::endl;
    std::vector<std::shared_ptr<int>> vec;
    std::shared_ptr<int> ptr{ new int {100} };
    vec.push_back(ptr); //can we do it?
    std::cout << ptr.use_count() << std::endl;
    std::cout << std::endl;
} //automatically deleted
```



```
void Example3()
{
    std::cout << "Example 3" << std::endl;
    std::shared_ptr<int> p1 = std::make_shared<int>(100);
    std::shared_ptr<int> p2{ p1 };
    std::shared_ptr<int> p3;
    p3 = p1;
    std::cout << p1.use_count() << std::endl;
    std::cout << std::endl;
} //automatically deleted
```

SHARED POINTERS— MAKE SHARE

Weak pointer

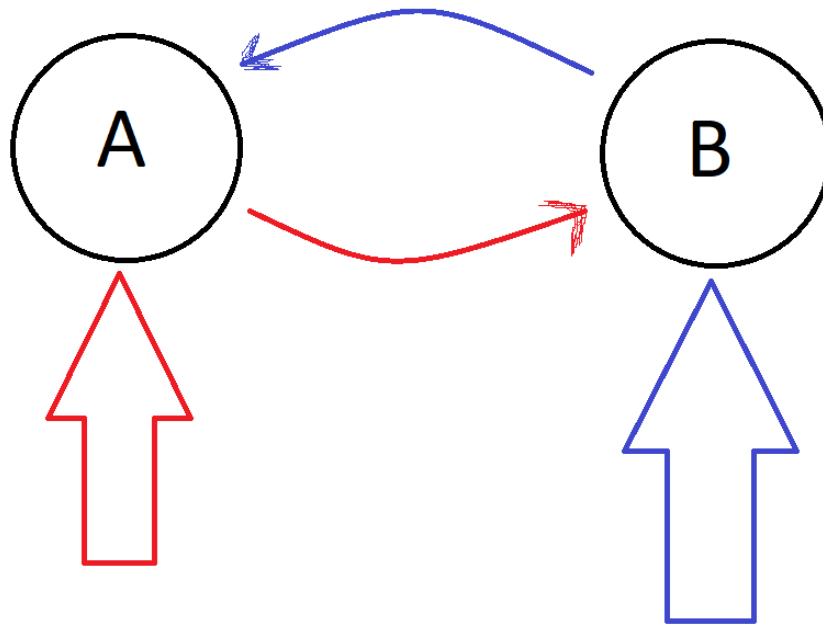
- Nie inkrementuje licznika referencji
- Stosujemy aby uniknąć odniesienia cyklicznego (Circular Reference)

CIRCULAR REFERENCE

- Odniesienie cykliczne

REFERENCE
COUNTER :

0
↓
1
↓
2
↓
1





DZIĘKUJEMY ZA UWAGĘ

<https://github.com/paulinaurbas/Smartpointers>