Умные указатели

Вернемся назад...

```
#include <iostream>
#include <stdexcept>

void func() {
    int* rawPtr = new int[100];
    std::cout << "Память выделена: " << rawPtr << std::endl;
    for (int i = 0; i < 100; ++i) {
        rawPtr[i] = i;
        if (i == 50) {
            throw std::runtime_error("Произошла непредвиденная ошибка!");
        }
    }
    std::cout << "Очищаем память..." << std::endl;
    delete[] rawPtr;
}

int main() {
    try {
        func();
    }
    catch (const std::exception& e) {
        std::cout << "Поймано исключение: " << e.what() << std::endl;
    }
    return 0;
}
```

Что не так в этой программе?

Вернемся назад...

```
void func() {
    int* rawPtr = new int[100];
    std::cout << "Память выделена: " << rawPtr << std::endl;
    for (int i = 0; i < 100; ++i) {
        rawPtr[i] = i;
        if (i == 50) {
            throw std::runtime_error("Произошла непредвиденная ошибка!");
        }
    }
    std::cout << "Очищаем память..." << std::endl;
    delete[] rawPtr;
}
```

До delete
не дошли

valgrind -s ./smartptr1

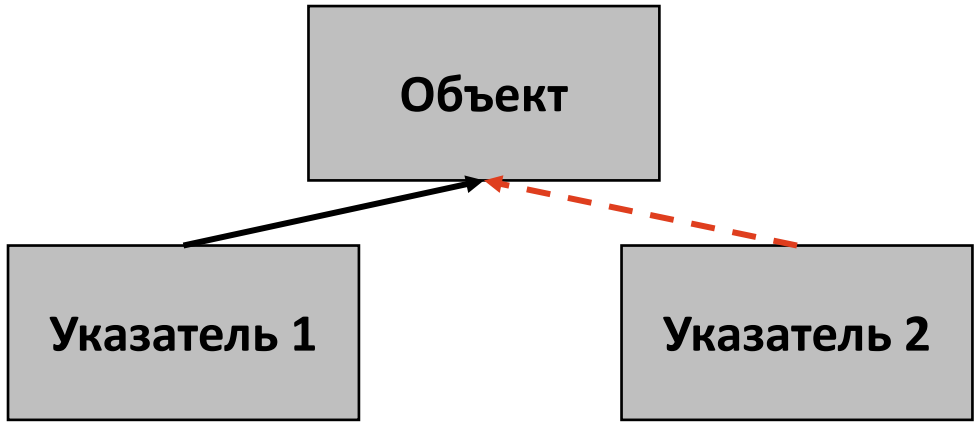
```
==3505== Memcheck, a memory error detector
==3505== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==3505== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==3505== Command: ./a.out
==3505==
Память выделена: 0x4e53080
Поймано исключение: Произошла непредвиденная ошибка!
==3505==
==3505== HEAP SUMMARY:
==3505==     in use at exit: 400 bytes in 1 blocks
==3505==   total heap usage: 5 allocs, 4 frees, 75,382 bytes allocated
==3505==
==3505== LEAK SUMMARY:
==3505==    definitely lost: 400 bytes in 1 blocks
==3505==    indirectly lost: 0 bytes in 0 blocks
==3505==      possibly lost: 0 bytes in 0 blocks
==3505==    still reachable: 0 bytes in 0 blocks
==3505==         suppressed: 0 bytes in 0 blocks
==3505== Rerun with --leak-check=full to see details of leaked memory
==3505==
```

Умные указатели

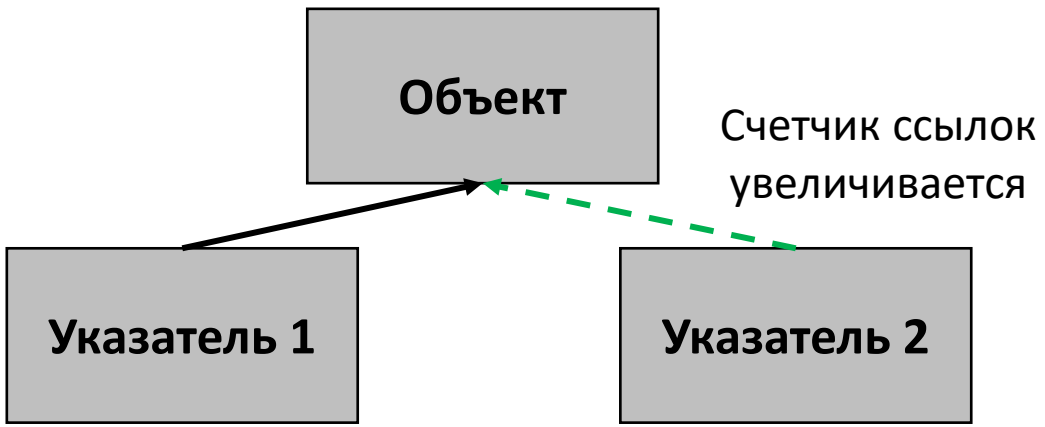
unique_ptr	умный указатель, который владеет и управляет другим объектом через указатель и удаляет этот объект, когда unique_ptr выходит за пределы области видимости.
shared_ptr	умный указатель, который сохраняет совместное владение объектом через указатель. Несколько объектов shared_ptr могут владеть одним и тем же объектом
weak_ptr	умный указатель, который содержит не владеющую ("слабую") ссылку на объект, управляемый std::shared_ptr

Умные указатели

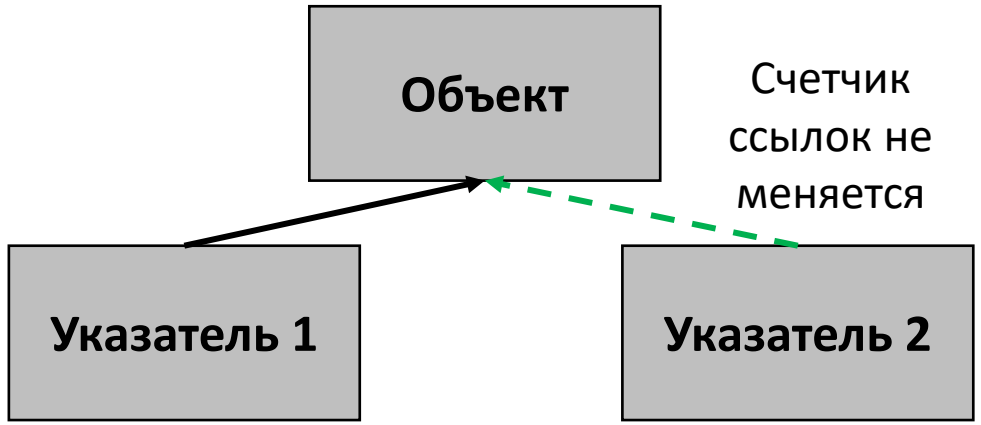
unique_ptr



shared_ptr



weak_ptr



unique_ptr

Синтаксис

```
std::unique_ptr<Type> p(new Type);
std::unique_ptr<Type> p = std::make_unique<Type>(...размер или параметры...);
```

```
#include <iostream>
#include <stdexcept>
#include <memory>


void func(std::unique_ptr<int[]> p) {
    std::cout << p[0];
}


int main()
{
    std::unique_ptr<int[]> ptr = std::make_unique<int[]>(5);
    ptr[0] = 5;
    std::unique_ptr<int[]> ptr_copy = ptr;      //тут хорошо?
    func(ptr);                                  //тут хорошо?
}
```

unique_ptr

Синтаксис

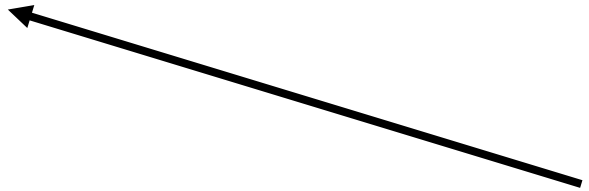
```
std::unique_ptr<Type> p(new Type);
std::unique_ptr<Type> p = std::make_unique<Type>(...размер или параметры...);
```

```
#include <iostream>
#include <stdexcept>
#include <memory>


void func(std::unique_ptr<int[]> p) {
    std::cout << p[0];
}


int main()
{
    std::unique_ptr<int[]> ptr = std::make_unique<int[]>(5);
    ptr[0] = 5;
    std::unique_ptr<int[]> ptr_copy = ptr;      //CE
    func(ptr);                                  //CE
}
```

Вернемся назад...

```
void func() {
    std::unique_ptr<int[]> rawPtr(new int[100]);
    std::cout << "Память выделена: " << std::endl;
    for (int i = 0; i < 100; ++i) {
        rawPtr[i] = i;
        if (i == 50) {
            throw std::runtime_error("Произошла непредвиденная ошибка!");
        }
    }
    std::cout << "Очищаем память..." << std::endl;
}
```

valgrind -s ./smartptr1



delete
вообще нет

```
==4635== Memcheck, a memory error detector
==4635== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==4635== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==4635== Command: ./a.out
==4635==
Память выделена:
Поймано исключение: Произошла непредвиденная ошибка!
==4635==
==4635== HEAP SUMMARY:
==4635==     in use at exit: 0 bytes in 0 blocks
==4635==   total heap usage: 5 allocs, 5 frees, 75,382 bytes allocated
==4635==
==4635== All heap blocks were freed -- no leaks are possible
==4635==
==4635== For lists of detected and suppressed errors, rerun with: -s
==4635== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Умные указатели

```
#include <iostream>
#include <memory>

using namespace std;

const int N = 10;

int main() {
    srand(time(NULL));
    unique_ptr<int[]> x(new int[N]);
    unique_ptr<int[]> y;
    for (size_t i = 0; i < N; i++)
    {
        x[i] = rand() % 100 - 50;
    }

    y = x;
    return 0;
}
```

ОШИБКА!!!

Умные указатели

```
#include <iostream>
#include <memory>

using namespace std;

const int N = 10;

int main() {
    srand(time(NULL));
    shared_ptr<int[]> x(new int[N]);
    shared_ptr<int[]> y;
    for (size_t i = 0; i < N; i++)
    {
        x[i] = rand() % 100 - 50;
    }

    y = x;
    return 0;
}
```

ОШИБКИ НЕТ

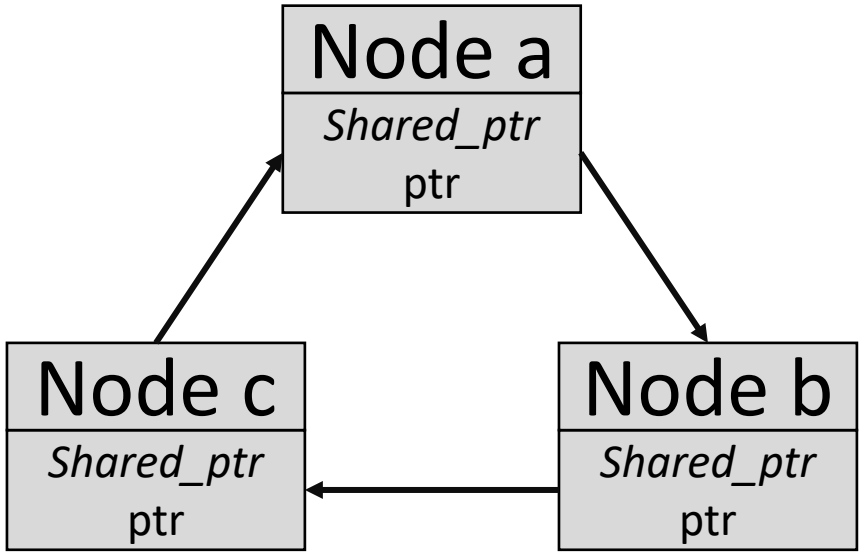
Умные указатели

```
#include <iostream>
#include <memory>

using namespace std;

const int N = 10;

int main() {
    srand(time(NULL));
    shared_ptr<int[]> x(new int[N]);
    shared_ptr<int[]> y;
    for (size_t i = 0; i < N; i++)
    {
        x.get()[i] = rand() % 100 - 50;
    }

    y = x;
    return 0;
}
```

x.get()[i] ~ x[i]

Умные указатели

```
#include <memory>
#include <iostream>

using namespace std;
typedef struct node {
    uint16_t value;
    std::shared_ptr<node> ptr;
} Node;

int main() {
    shared_ptr<Node> a = shared_ptr<Node>(new Node);
    shared_ptr<Node> b = shared_ptr<Node>(new Node);
    shared_ptr<Node> c = shared_ptr<Node>(new Node);
    a->ptr = b;
    b->ptr = c;
    c->ptr = a;
    return 0;
}
```



Умные указатели

```
#include <memory>
#include <iostream>

using namespace std;
typedef struct node {
    uint16_t value;
    std::shared_ptr<node> ptr;
} Node;

int main() {
    shared_ptr<Node> a = shared_ptr<Node>(new Node);
    shared_ptr<Node> b = shared_ptr<Node>(new Node);
    shared_ptr<Node> c = shared_ptr<Node>(new Node);
    a->ptr = b;
    b->ptr = c;
    c->ptr = a;
    return 0;
}
```

```
==449== LEAK SUMMARY:
==449==    definitely lost: 24 bytes in 1 blocks
==449==    indirectly lost: 120 bytes in 5 blocks
==449==      possibly lost: 0 bytes in 0 blocks
==449==    still reachable: 0 bytes in 0 blocks
==449==         suppressed: 0 bytes in 0 blocks
```

Умные указатели

```
#include <memory>
#include <iostream>

using namespace std;
typedef struct node {
    uint16_t value;
    std::weak_ptr<node> ptr;
} Node;

int main() {
    shared_ptr<Node> a = shared_ptr<Node>(new Node);
    shared_ptr<Node> b = shared_ptr<Node>(new Node);
    shared_ptr<Node> c = shared_ptr<Node>(new Node);
    a->ptr = b;
    b->ptr = c;
    c->ptr = a;
    return 0;
}
```

```
==455== HEAP SUMMARY:
==455==     in use at exit: 0 bytes in 0 blocks
==455==   total heap usage: 7 allocs, 7 frees, 72,848 bytes allocated
```

Умные указатели

Если weak_ptr может ссылаться на несуществующий объект, то как правильно реализовать доступ к значениям по данному указателю?

```
std::weak_ptr<int> wp;
wp.lock();
```

Метод Lock()

Возвращает shared_ptr в случае наличия связи

Возвращает пустой указатель, в случае отсутствия связи

Умные указатели

```
#include <memory>
#include <iostream>

int main()
{
    std::weak_ptr<int> wp;
    {
        std::shared_ptr<int> sp(new int[10]);
        for (size_t i = 0; i < 10; i++)
        {
            sp.get()[i] = i;
        }
        wp = sp;
        std::cout << "{\n" << "wp.lock() == " << (wp.lock()) << std::endl <<
            "wp.lock()[5] == " << (wp.lock().get()[5]) << "\n}" << std::endl;
    }
    std::cout << "wp.lock() == " << (wp.lock()) << std::endl <<
        "wp.lock()[5] == " << (wp.lock().get()[5]) << std::endl;
    return 0;
}
```

```
{
wp.lock() == 0x5d3d33f452b0
wp.lock()[5] == 5
}
wp.lock() == 0
Segmentation fault (core dumped)
```