Основы программирования на C++

Занятие 13. Умные указатели

Дерево языка

Вернемся назад...

```cpp
#include <iostream>
#include <memory>
using namespace std;
const int N = 1000;
int main() {

    int* x = new int[N];
    for (size_t i = 0; i < N; i++)
    {
        x[i] = rand() % 100 - 50;
    }

    for (size_t i = 0; i < N; i++)
    {
        cout << x[i] << "\n";
    }
    return 0;
}
```

Что не так в этой программе?

Полный код см. в репозитории

Вернемся назад...

```cpp
#include <iostream>
#include <memory>
using namespace std;
const int N = 1000;
int main() {

    int* x = new int[N];
    for (size_t i = 0; i < N; i++)
    {
        x[i] = rand() % 100 - 50;
    }

    for (size_t i = 0; i < N; i++)
    {
        cout << x[i] << "\n";
    }
    return 0;
}
```

valgrind -s ./smartptr1

```
==111==
==111== HEAP SUMMARY:
==111==     in use at exit: 4,000 bytes in 1 blocks
==111==   total heap usage: 3 allocs, 2 frees, 77,728 bytes allocated
==111==
==111== LEAK SUMMARY:
==111==    definitely lost: 4,000 bytes in 1 blocks
==111==    indirectly lost: 0 bytes in 0 blocks
==111==      possibly lost: 0 bytes in 0 blocks
==111==    still reachable: 0 bytes in 0 blocks
==111==         suppressed: 0 bytes in 0 blocks
==111== Rerun with --leak-check=full to see details of leaked memory
==111==
==111== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Вернемся назад...

```cpp
#include <iostream>
#include <memory>
using namespace std;
const int N = 10;
int main() {

    int* x = new int[N];
    for (size_t i = 0; i < N; i++)
    {
        x[i] = rand() % 100 - 50;
    }

    for (size_t i = 0; i < N; i++)
    {
        cout << x[i] << "\n";
    }
    delete []x;
    return 0;
}
```

```
==118==
==118== HEAP SUMMARY:
==118==     in use at exit: 0 bytes in 0 blocks
==118==   total heap usage: 3 allocs, 3 frees, 77,728 bytes allocated
==118==
==118== All heap blocks were freed -- no leaks are possible
==118==
==118== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Очистка памяти

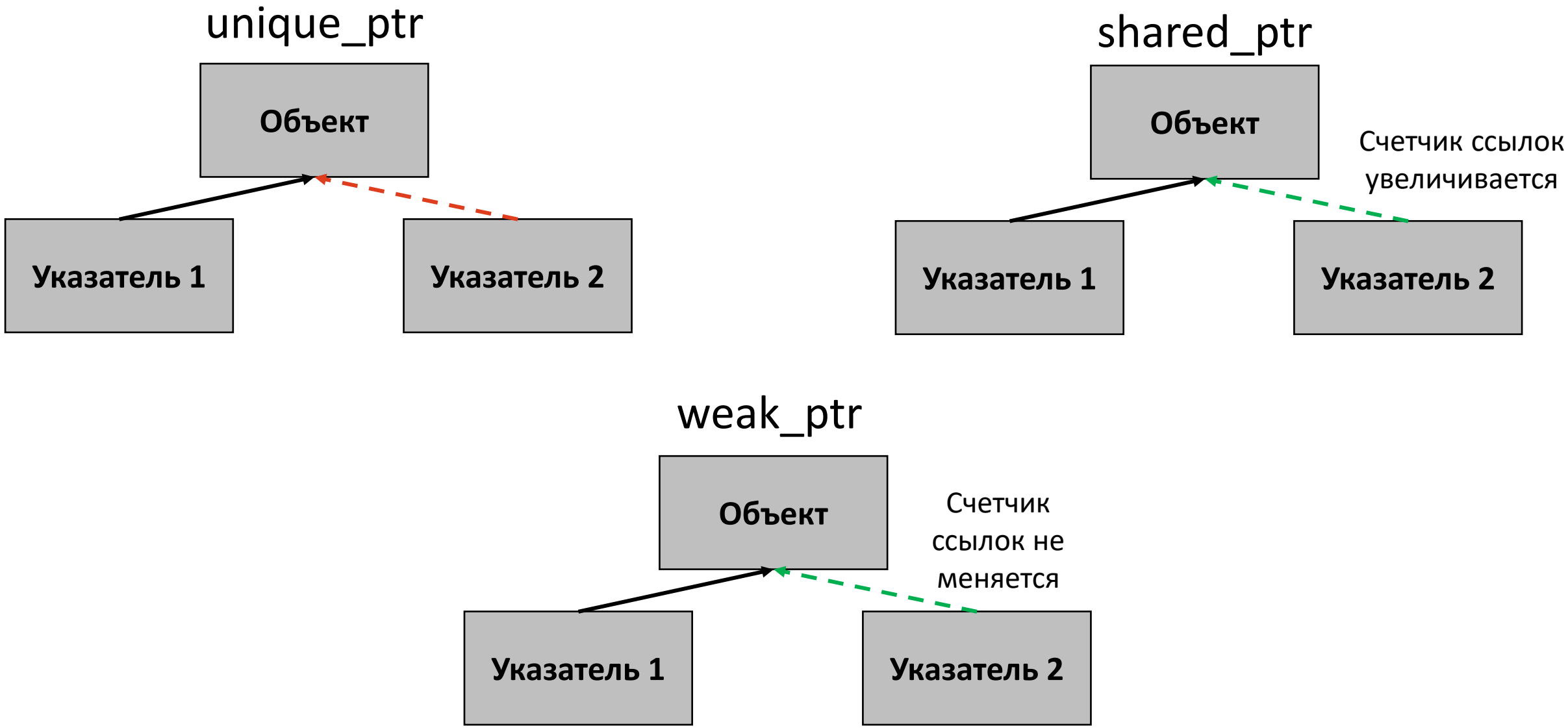
```
#include <iostream>
#include <memory>
using namespace std;
const int N = 10;
int main() {
    srand(time(NULL));
    int* x = new int[N];
    for (size_t i = 0; i < N; i++)
    {
        x[i] = rand() % 100 - 50;
    }

    if ((rand() % 2) == 0) exit(-1);

    for (size_t i = 0; i < N; i++)
    {
        cout << x[i] << "\n";
    }
    delete []x;
    return 0;
}
```

Добавим конструкцию, которая случайным образом может выйти из программы

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



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]);
    for (size_t i = 0; i < N; i++)
    {
        x.get()[i] = rand() % 100 - 50;
    }

    for (size_t i = 0; i < N; i++)
    {
        cout << x.get()[i] << "\n";
    }
    return 0;
}
```

```
==247== HEAP SUMMARY:
==247==     in use at exit: 0 bytes in 0 blocks
==247==   total heap usage: 3 allocs, 3 frees, 73,768 bytes allocated
==247==
==247== All heap blocks were freed -- no leaks are possible
==247==
==247== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

new есть!

delete нет!

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.get()[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.get()[i] = rand() % 100 - 50;
    }

    y = x;
    return 0;
}
```

ОШИБКИ НЕТ

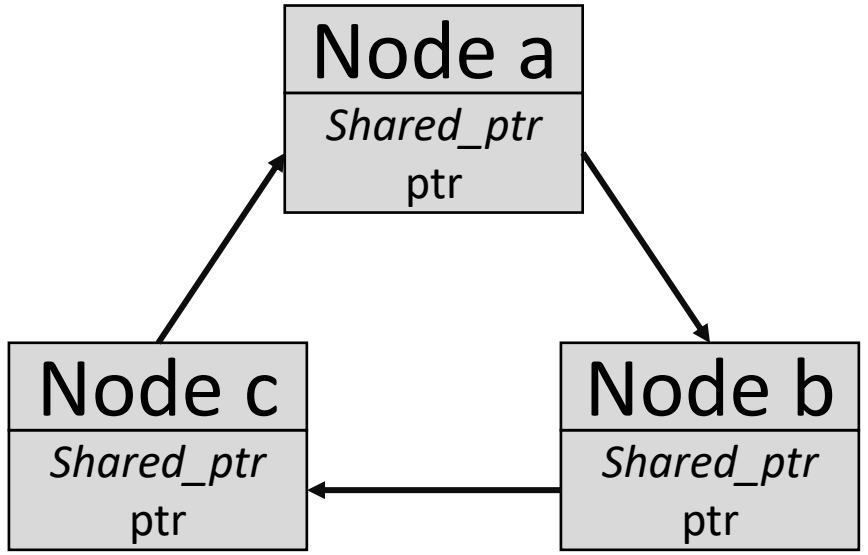
Умные указатели

```cpp
#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;
}
```

Для shared_ptr
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

```cpp
#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 << "wp.lock() == " << (wp.lock()) << std::endl <<
            "wp.lock()[5] == " << (wp.lock().get()[5]) << std::endl;
    }
    std::cout << "wp.lock() == " << (wp.lock()) << std::endl <<
        "wp.lock()[5] == " << (wp.lock().get()[5]) << std::endl;

    return (0);
}
```

```
wp.lock() == 0x562e951deeb0
wp.lock()[5] == 5
wp.lock() == 0
Segmentation fault
```

Спасибо за внимание!