

Ларри и Джен делают кату «Римские цифры» на C++



© www.cplusplus.com



© www.cplusplus.com

авторы: Ольве Маудал (Olve Maudal) и Джон Джаггер (Jon Jagger)
перевод на русский: oscat.ru

Ката — это упражнение по программированию, которое помогает разработчикам оттачивать свои навыки через практику и повторение. В 1999 году этот термин использовал Дэйв Томас, соавтор книги «Программист-прагматик», позаимствовав его из традиции японских боевых искусств (см. подробнее в [Википедии](#)).

Ларри и Джен впервые появились в презентации «Глубины C и C++»

<https://oscat.ru/?p=358>



© www.stipendinfo

Джен, сможешь мне попрактиковаться в C++?

Конечно, Ларри.



© www.stipendinfo



Как насчёт каты
«Римские цифры»?

Давай.



Для введенного целого положительного числа (например, 42) отобразите его строковое представление римскими цифрами (для числа 42 - "XLII"). Символы римских цифр:

1 ->	"I"		10 ->	"X"		100 ->	"C"		1000 ->	"M"
2 ->	"II"		20 ->	"XX"		200 ->	"CC"		2000 ->	"MM"
3 ->	"III"		30 ->	"XXX"		300 ->	"CCC"		3000 ->	"MMM"
4 ->	"IV"		40 ->	"XL"		400 ->	"CD"		4000 ->	"MMMM"
5 ->	"V"		50 ->	"L"		500 ->	"D"			
6 ->	"VI"		60 ->	"LX"		600 ->	"DC"			
7 ->	"VII"		70 ->	"LXX"		700 ->	"DCC"			
8 ->	"VIII"		80 ->	"LXXX"		800 ->	"DCCC"			
9 ->	"IX"		90 ->	"XC"		900 ->	"CM"			

Вы не можете делать так: "IM" для 999.

Википедия утверждает: «Современные римские цифры записываются путем добавления каждой цифры отдельно, начиная с крайней левой цифры и пропуском в случае нуля».

Примеры:

- o) 1990 -> "MCMXC" (1000 -> "M" + 900 -> "CM" + 90 -> "XC")
- o) 2008 -> "MMVIII" (2000 -> "MM" + 8 -> "VIII")
- o) 99 -> "XCIX" (90 -> "XC" + 9 -> "IX")
- o) 47 -> "XLVII" (40 -> "XL" + 7 -> "VII")

to_roman.tests.cpp

```
#include "to_roman.hpp"
#include <cassert>
#include <iostream>

static void test_1_is_I()
{
    assert(to_roman(1) == "I");
}

int main()
{
    std::cout << "All tests passed"
               << std::endl;
}
```



Начну с теста.
«1» должен вернуть «I».





© www.studart.ru

Вот мой заголовочный файл: to_roman.hpp

А вот файл исходного кода: to_roman.cpp

Ясно.



© www.studart.ru

```
#ifndef TO_ROMAN_INCLUDED
#define TO_ROMAN_INCLUDED

#include <string>

std::string to_roman(int number);

#endif
```

```
#include "to_roman.hpp"

std::string to_roman(int number)
{
    return "";
}
```



Как ты их собираешь и запускаешь?

Вот так.

Попробуй.

А?.. Что-то пошло не так.
Тест должен был
провалиться!

Давай опять посмотрим на
твой тест.



```
$ cc *.cpp && ./a.out  
All tests passed
```



```
#include "to_roman.hpp"
#include <cassert>
#include <iostream>

static void test_1_is_I()
{
    assert(to_roman(1) == "I");
}

int main()
{
    test_1_is_I();
    std::cout << "All tests passed"
              << std::endl;
}
```

```
$ cc *.cpp && ./a.out
Assertion failed: (to_roman(1) == "I")
```



Я могу просто изменить
возвращаемое
значение, чтобы он
прошёл успешно.



Ок.

```
#include "to_roman.hpp"

std::string to_roman(int number)
{
    return "I";
}
```

```
$ cc *.cpp && ./a.out
All tests passed
```




Давай напишем
следующий тест.
«2» должно вернуть «II».

Я думаю, сначала мы должны
решить одну проблему.

Какую?

Мы исправили проблему с
пропущенным вызовом
функции. Но ведь мы можем
забыть об этом и в
следующий раз.

Что ты предлагаешь?



© 2013 by [unreadable]



Может ли компилятор определить, что у нас объявлена функция, которая нигде не вызывается?

Наверное, у него есть ключ командной строки для этого?

Да. Компилятор может определить это при включенном флаге Wall.

Ок, давай попробуем.





Я прокомментирую
вызов функции.

И добавлю
флаг Wall.

Теперь я получил
предупреждение.



```
#include "to_roman.hpp"
#include <cassert>
#include <iostream>

static void test_1_is_I()
{
    assert(to_roman(1) == "I");
}

int main()
{
    //test_1_is_I();
    std::cout << "All tests passed"
              << std::endl;
}
```

```
$ cc -Wall *.cpp && ./a.out
warning: 'void test_1_is_I()' is
defined but not used
All tests passed
```



Еще лучше превратить
это предупреждение в
ошибку. Мы можем
сделать это при помощи
флага `Werror`.



Согласен.

```
#include "to_roman.hpp"
#include <cassert>
#include <iostream>

static void test_1_is_I()
{
    assert(to_roman(1) == "I");
}

int main()
{
    //test_1_is_I();
    std::cout << "All tests passed"
              << std::endl;
}
```

```
$ cc -Wall -Werror *.cpp && ./a.out
error: 'void test_1_is_I()' is defined
but not used
```



Я раскомментировал вызов функции,
так что тест опять выполняется
успешно.

© www.clipartland.com



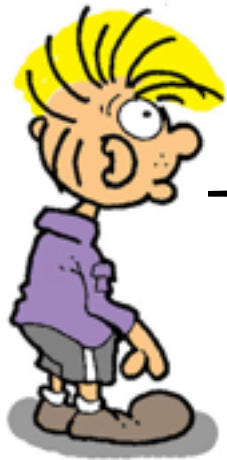
Теперь мы можем
написать
следующий тест.

```
#include "to_roman.hpp"
#include <cassert>
#include <iostream>

static void test_1_is_I()
{
    assert(to_roman(1) == "I");
}

int main()
{
    test_1_is_I();
    std::cout << "All tests passed"
              << std::endl;
}
```

```
$ cc -Wall -Werror *.cpp && ./a.out
All tests passed
```



«2» должно вернуть «II».

Тест должен провалиться, потому что to_roman всё еще возвращает «I».

Давай сделаем так, чтобы он прошел.



```
...
static void test_1_is_I()
{
    assert(to_roman(1) == "I");
}
static void test_2_is_II()
{
    assert(to_roman(2) == "II");
}
int main()
{
    test_1_is_I();
    test_2_is_II();
    std::cout << "All tests passed"
              << std::endl;
}
```

```
$ cc -Wall -Werror *.cpp && ./a.out
void test_2_is_II(): Assertion
`to_roman(2) == "II"' failed.
```



Как насчет такого?

Запускай!

А?..

А! Это потому, что если число не равно «1» или «2», то return не вызывается.

Именно.

Это легко исправить.

```
#include "to_roman.hpp"

std::string to_roman(int number)
{
    if (number == 1)
        return "I";
    if (number == 2)
        return "II";
}
```



```
$ cc -Wall -Werror *.cpp && ./a.out
error: control reaches end of non-void function
```



```
#include "to_roman.hpp"

std::string to_roman(int number)
{
    std::string roman = "";
    if (number == 1)
        roman = "I";
    if (number == 2)
        roman = "II";
    return roman;
}
```



```
$ cc -Wall -Werror *.cpp && ./a.out
All tests passed
```




Я добавлю тест для «3». Он должен провалиться.

А теперь я сделаю так, чтобы он проходил.

Хорошо.



```
...
static void test_1_is_I()
{
    assert(to_roman(1) == "I");
}

static void test_2_is_II()
{
    assert(to_roman(2) == "II");
}

static void test_3_is_III()
{
    assert(to_roman(3) == "III");
}

int main()
{
    test_1_is_I();
    test_2_is_II();
    test_3_is_III();
    std::cout << "All tests passed"
              << std::endl;
}
```

```
$ cc -Wall -Werror *.cpp && ./a.out
void test_3_is_III(): Assertion `to_roman(3) ==
"III"' failed.
..Aborted
```



```
#include "to_roman.hpp"

std::string to_roman(int number)
{
    std::string roman = "";
    if (number == 1)
        roman = "I";
    if (number == 2)
        roman = "II";
    if (number == 3)
        roman = "III";
    return roman;
}
```



```
$ cc -Wall -Werror *.cpp && ./a.out
All tests passed
```



© www.studio100.ru

Думаешь, сначала нужен рефакторинг?

Да.

Я бы начала его с тестов.



© www.studio100.ru

```
...
static void test_1_is_I()
{
    assert(to_roman(1) == "I");
}

static void test_2_is_II()
{
    assert(to_roman(2) == "II");
}

static void test_3_is_III()
{
    assert(to_roman(3) == "III");
}

int main()
{
    test_1_is_I();
    test_2_is_II();
    test_3_is_III();
    std::cout << "All tests passed"
              << std::endl;
}
```



А что с ним не так?

Название теста просто повторяет его тело. Оно не несет никакого дополнительного смысла. Как насчет того, чтобы заменить всё это одним тестом?

Покажешь, как это сделать?



```
...
static void test_1_is_I()
{
    assert(to_roman(1) == "I");
}

static void test_2_is_II()
{
    assert(to_roman(2) == "II");
}

static void test_3_is_III()
{
    assert(to_roman(3) == "III");
}

int main()
{
    test_1_is_I();
    test_2_is_II();
    test_3_is_III();
    std::cout << "All tests passed"
              << std::endl;
}
```



Например, вот так.

Всё работает,
как и раньше.



```
...
static void test_to_roman()
{
    assert(to_roman(1) == "I");
    assert(to_roman(2) == "II");
    assert(to_roman(3) == "III");
}

int main()
{
    test_to_roman();
    std::cout << "All tests passed"
              << std::endl;
}
```

```
$ cc -Wall -Werror *.cpp && ./a.out
All tests passed
```



Теперь надо провести рефакторинг кода.

Ага, сотни IF-ов едва ли будут хорошим решением...

Хочешь, покажу более изящный вариант?

Конечно.



```
#include "to_roman.hpp"

std::string to_roman(int number)
{
    std::string roman = "";
    if (number == 1)
        roman = "I";
    if (number == 2)
        roman = "II";
    if (number == 3)
        roman = "III";
    return roman;
}
```

Для начала я изменю код так, чтобы в каждом операторе IF происходило добавление к строке одного «I».



```
#include "to_roman.hpp"

std::string to_roman(int number)
{
    std::string roman = "";
    if (number >= 1)
        roman += "I";
    if (number >= 2)
        roman += "I";
    if (number >= 3)
        roman += "I";
    return roman;
}
```

```
$ cc -Wall -Werror *.cpp && ./a.out
All tests passed
```

Теперь оберну код IF-ов в фигурные скобки.



© www.Clipartland.com



© www.Clipartland.com

```
#include "to_roman.hpp"

std::string to_roman(int number)
{
    std::string roman = "";
    if (number >= 1)
    {
        roman += "I";
    }
    if (number >= 2)
    {
        roman += "I";
    }
    if (number >= 3)
    {
        roman += "I";
    }
    return roman;
}
```

```
$ cc -Wall -Werror *.cpp && ./a.out
All tests passed
```


Теперь я сделаю все три
IF-а одинаковыми.



© www.CppProject.info



© www.CppProject.info

```
#include "to_roman.hpp"

std::string to_roman(int number)
{
    std::string roman = "";
    if (number >= 1)
    {
        roman += "I";
        number--;
    }
    if (number >= 1)
    {
        roman += "I";
        number--;
    }
    if (number >= 1)
    {
        roman += "I";
        number--;
    }
    return roman;
}
```

```
$ cc -Wall -Werror *.cpp && ./a.out
All tests passed
```

А после этого заменю их на один цикл WHILE.



```
#include "to_roman.hpp"

std::string to_roman(int number)
{
    std::string roman = "";
    while (number >= 1)
    {
        roman += "I";
        number--;
    }
    return roman;
}
```

```
$ cc -Wall -Werror *.cpp && ./a.out
All tests passed
```



Круто. И правда, набор идентичных IF-ов можно представить в виде цикла.

Именно.



```
#include "to_roman.hpp"

std::string to_roman(int number)
{
    std::string roman = "";
    while (number >= 1)
    {
        roman += "I";
        number--;
    }
    return roman;
}
```



Что-то ещё исправим?

Что ты предлагаешь?

Вообще, у меня больше нет идей.

Ок. Тогда продолжим.

Я добавлю тест для «4» и «IV».

Я предлагаю сразу перейти к «10» и «X».

Ладно.





Добавлю тест для «10» и «X».

Как и ожидалось
- он провалился.



```
...
static void test_to_roman()
{
    assert(to_roman(1) == "I");
    assert(to_roman(2) == "II");
    assert(to_roman(3) == "III");
    assert(to_roman(10) == "X");
}
```

```
$ cc -Wall -Werror *.cpp && ./a.out
Assertion `to_roman(10) == "X"' failed.
Aborted
```

Чтобы он прошёл - я добавлю еще один цикл WHILE, только заменю в нём «1» и «I» на «10» и «X» соответственно.



А?.. Всё равно провалился...



```
#include "to_roman.hpp"

std::string to_roman(int number)
{
    std::string roman = "";
    while (number >= 1)
    {
        roman += "I";
        number--;
    }
    while (number >= 10)
    {
        roman += "X";
        number -= 10;
    }
    return roman;
}
```

```
$ cc -Wall -Werror *.cpp && ./a.out
Assertion `to_roman(10) == "X"' failed.
Aborted
```



Было бы удобно видеть в консоли, что в действительности вернул to_roman.

Да, ты права.

Давай так и сделаем.

Ок.

```
...
static void test_to_roman()
{
    assert(to_roman(1) == "I");
    assert(to_roman(2) == "II");
    assert(to_roman(3) == "III");
    assert(to_roman(10) == "X");
}
```

```
$ cc -Wall -Werror *.cpp && ./a.out
Assertion `to_roman(10) == "X"' failed.
Aborted
```

Добавим вызов еще не существующей функции `assert_to_roman`.

Ага.



```
...
static void test_to_roman()
{
    assert_to_roman(1, "I");
    assert_to_roman(2, "II");
    assert_to_roman(3, "III");
    assert_to_roman(10, "X");
}
```


И напишем для этой функции
вот такой код.

Теперь вывод стал гораздо понятнее.



```
static void assert_to_roman(int arabic,
                             const std::string & expected)
{
    const std::string actual = to_roman(arabic);
    if (expected != actual)
    {
        std::cerr << "expected: to_roman"
                  << '(' << arabic << ") == "
                  << "'" << expected << "'"
                  << std::endl;
        std::cerr << "  actual: to_roman"
                  << '(' << arabic << ") == "
                  << "'" << actual << "'"
                  << std::endl;
        assert(false);
    }
}
...
```

```
$ cc -Wall -Werror *.cpp && ./a.out
expected: to_roman(10) == "X"
  actual: to_roman(10) == "IIIIIIIIII"
Assertion `false' failed.
Aborted
```



Но почему же to_roman не работает для «X»?

Ох, ну конечно. У нас неверный порядок операторов WHILE.

```
#include "to_roman.hpp"

std::string to_roman(int number)
{
    std::string roman = "";
    while (number >= 1)
    {
        roman += "I";
        number--;
    }
    while (number >= 10)
    {
        roman += "X";
        number -= 10;
    }
    return roman;
}
```



```
$ cc -Wall -Werror *.cpp && ./a.out
expected: to_roman(10) == "X"
actual: to_roman(10) == "IIIIIIIIII"
Assertion `false' failed.
Aborted
```



© www.stefanec.com

Это легко исправить.

Да. И вот тест снова проходит.

```
#include "to_roman.hpp"

std::string to_roman(int number)
{
    std::string roman = "";
    while (number >= 10)
    {
        roman += "X";
        number -= 10;
    }
    while (number >= 1)
    {
        roman += "I";
        number--;
    }
    return roman;
}
```



© www.stefanec.com

```
$ cc -Wall -Werror *.cpp && ./a.out
All tests passed
```



Думаю, тесты для «20» и «30» пройдут без проблем.

Согласна.



```
#include "to_roman.hpp"
#include <cassert>
#include <iostream>
...
static void test_to_roman()
{
    assert_to_roman(1, "I");
    assert_to_roman(2, "II");
    assert_to_roman(3, "III");
    assert_to_roman(10, "X");
    assert_to_roman(20, "XX");
    assert_to_roman(30, "XXX");
}
```

```
$ cc -Wall -Werror *.cpp && ./a.out
All tests passed
```



И с тестом для «33» всё
в порядке.

Ага.



```
#include "to_roman.hpp"
#include <cassert>
#include <iostream>
...
static void test_to_roman()
{
    assert_to_roman( 1, "I");
    assert_to_roman( 2, "II");
    assert_to_roman( 3, "III");
    assert_to_roman(10, "X");
    assert_to_roman(20, "XX");
    assert_to_roman(30, "XXX");
    assert_to_roman(33, "XXXIII");
}
```

```
$ cc -Wall -Werror *.cpp && ./a.out
All tests passed
```



© www.ClipPapers.com

Что дальше?

Как насчёт «100»?

Для него мы должны
получить «С».



© www.ClipPapers.com



Тест провалился - получили 10 «X».

Здорово!



```
#include "to_roman.hpp"
#include <cassert>
#include <iostream>
...
static void test_to_roman()
{
    assert_to_roman( 1,"I");
    assert_to_roman( 2,"II");
    assert_to_roman( 3,"III");
    assert_to_roman(10,"X");
    assert_to_roman(20,"XX");
    assert_to_roman(30,"XXX");
    assert_to_roman(33,"XXXIII");
    assert_to_roman(100,"C");
}
```

```
$ cc -Wall -Werror *.cpp && ./a.out
expected: to_roman(100) == "C"
  actual: to_roman(100) == "XXXXXXXXXX"
....int main(): Assertion `false' failed.
Aborted
```



Чтобы он прошел успешно - я просто добавлю еще один WHILE.



```
#include "to_roman.hpp"

std::string to_roman(int number)
{
    std::string roman = "";
    while (number >= 100)
    {
        roman += "C";
        number -= 100;
    }
    while (number >= 10)
    {
        roman += "X";
        number -= 10;
    }
    while (number >= 1)
    {
        roman += "I";
        number--;
    }
    return roman;
}
```

```
$ cc -Wall -Werror *.cpp && ./a.out
All tests passed
```




Что думаешь?

Теперь у нас три очень похожих WHILE'а. Настало время рефакторинга!



```
#include "to_roman.hpp"

std::string to_roman(int number)
{
    std::string roman = "";
    while (number >= 100)
    {
        roman += "C";
        number -= 100;
    }
    while (number >= 10)
    {
        roman += "X";
        number -= 10;
    }
    while (number >= 1)
    {
        roman += "I";
        number--;
    }
    return roman;
}
```

```
$ cc -Wall -Werror *.cpp && ./a.out
All tests passed
```



Для начала я создам простую структуру для представления одной римской цифры.



```
#include "to_roman.hpp"
struct digit
{
    int arabic;
    std::string roman;
};
std::string to_roman(int number)
{
    std::string roman = "";
    while (number >= 100)
    {
        roman += "C";
        number -= 100;
    }
    while (number >= 10)
    {
        roman += "X";
        number -= 10;
    }
    while (number >= 1)
    {
        roman += "I";
        number--;
    }
    return roman;
}
```

```
$ cc -Wall -Werror *.cpp && ./a.out
All tests passed
```



Затем я объявлю массив таких структур и добавлю в него цифру «С» со значением «100».



```
#include "to_roman.hpp"

struct digit
{
    int arabic;
    std::string roman;
};

const digit digits[] =
{
    { 100, "C" },
};

std::string to_roman(int number)
{
    std::string roman = "";
    while (number >= 100)
    {
        roman += "C";
        number -= 100;
    }
    while (number >= 10)
    {
        roman += "X";
        number -= 10;
    }
    while (number >= 1)
    {
        roman += "I";
        number--;
    }
    return roman;
}
```

```
$ cc -Wall -Werror *.cpp && ./a.out
All tests passed
```



Теперь в своем
коде я заменю
числа на элементы
этого массива.



Отлично.

```
#include "to_roman.hpp"

struct digit
{
    int arabic;
    std::string roman;
};

const digit digits[] =
{
    { 100, "C" },
};

std::string to_roman(int number)
{
    std::string roman = "";
    while (number >= digits[0].arabic)
    {
        roman += digits[0].roman;
        number -= digits[0].arabic;
    }
    while (number >= 10)
    {
        roman += "X";
        number -= 10;
    }
    while (number >= 1)
    {
        roman += "I";
        number--;
    }
    return roman;
}
```

```
$ cc -Wall -Werror *.cpp && ./a.out
All tests passed
```



Повторю то же самое для «X» и «10».



```
#include "to_roman.hpp"
...
const digit digits[] =
{
    { 100, "C" },
    { 10, "X" },
};

std::string to_roman(int number)
{
    std::string roman = "";
    while (number >= digits[0].arabic)
    {
        roman += digits[0].roman;
        number -= digits[0].arabic;
    }
    while (number >= digits[1].arabic)
    {
        roman += digits[1].roman;
        number -= digits[1].arabic;
    }
    while (number >= 1)
    {
        roman += "I";
        number--;
    }
    return roman;
}
```

```
$ cc -Wall -Werror *.cpp && ./a.out
All tests passed
```



И, наконец, для
«I» и «1».



```
#include "to_roman.hpp"
...
const digit digits[] =
{
    { 100, "C" },
    {  10, "X" },
    {   1, "I" },
};

std::string to_roman(int number)
{
    std::string roman = "";
    while (number >= digits[0].arabic)
    {
        roman += digits[0].roman;
        number -= digits[0].arabic;
    }
    while (number >= digits[1].arabic)
    {
        roman += digits[1].roman;
        number -= digits[1].arabic;
    }
    while (number >= digits[2].arabic)
    {
        roman += digits[2].roman;
        number -= digits[2].arabic;
    }
    return roman;
}
```

```
$ cc -Wall -Werror *.cpp && ./a.out
All tests passed
```



Теперь при обращении к массиву я могу заменить индекс 0 на переменную i.



```
...
std::string to_roman(int number)
{
    std::string roman = "";
    std::size_t i = 0;
    while (number >= digits[i].arabic)
    {
        roman += digits[i].roman;
        number -= digits[i].arabic;
    }
    while (number >= digits[1].arabic)
    {
        roman += digits[1].roman;
        number -= digits[1].arabic;
    }
    while (number >= digits[2].arabic)
    {
        roman += digits[2].roman;
        number -= digits[2].arabic;
    }
    return roman;
}
```

```
$ cc -Wall -Werror *.cpp && ./a.out
All tests passed
```



И индекс 1 тоже.



```
...
std::string to_roman(int number)
{
    std::string roman = "";
    std::size_t i = 0;
    while (number >= digits[i].arabic)
    {
        roman += digits[i].roman;
        number -= digits[i].arabic;
    }
    ++i;
    while (number >= digits[i].arabic)
    {
        roman += digits[i].roman;
        number -= digits[i].arabic;
    }
    while (number >= digits[2].arabic)
    {
        roman += digits[2].roman;
        number -= digits[2].arabic;
    }
    return roman;
}
```

```
$ cc -Wall -Werror *.cpp && ./a.out
All tests passed
```




И индекс 2.



```
...
std::string to_roman(int number)
{
    std::string roman = "";
    std::size_t i = 0;
    while (number >= digits[i].arabic)
    {
        roman += digits[i].roman;
        number -= digits[i].arabic;
    }
    ++i;
    while (number >= digits[i].arabic)
    {
        roman += digits[i].roman;
        number -= digits[i].arabic;
    }
    ++i;
    while (number >= digits[i].arabic)
    {
        roman += digits[i].roman;
        number -= digits[i].arabic;
    }
    return roman;
}
```

```
$ cc -Wall -Werror *.cpp && ./a.out
All tests passed
```



© www.ClipPaperLife

А теперь
превращаю все 3
WHILE'а в один.



© www.ClipPaperLife

```
...
std::string to_roman(int number)
{
    std::string roman = "";
    for (std::size_t i = 0; i != 3; ++i)
        while (number >= digits[i].arabic)
        {
            roman += digits[i].roman;
            number -= digits[i].arabic;
        }
    return roman;
}
```

```
$ cc -Wall -Werror *.cpp && ./a.out
All tests passed
```



Вместо числа «3»
лучше использовать
число элементов
массива.

© www.clipart.com



Прекрасно.

© www.clipart.com

```
#include "to_roman.hpp"
...
const digit digits[] =
{
    { 100, "C" },
    {  10, "X" },
    {   1, "I" },
};

const std::size_t n =
    sizeof digits / sizeof digits[0];

std::string to_roman(int number)
{
    std::string roman = "";
    for (std::size_t i = 0; i != n; ++i)
        while (number >= digits[i].arabic)
        {
            roman += digits[i].roman;
            number -= digits[i].arabic;
        }
    return roman;
}
```

```
$ cc -Wall -Werror *.cpp && ./a.out
All tests passed
```



ГОТОВО.

© www.ClipPaperLife



Что-нибудь
еще?

© www.ClipPaperLife

```
#include "to_roman.hpp"

struct digit
{
    int arabic;
    std::string roman;
};

const digit digits[] =
{
    { 100, "C" },
    { 10, "X" },
    { 1, "I" },
};

const std::size_t n =
    sizeof digits / sizeof digits[0];

std::string to_roman(int number)
{
    std::string roman = "";
    for (std::size_t i = 0; i != n; ++i)
        while (number >= digits[i].arabic)
        {
            roman += digits[i].roman;
            number -= digits[i].arabic;
        }
    return roman;
}
```



Твои
предложения?



digits и n имеют
внутреннее
связывание,
потому что это
константы, но тип
digit имеет
внешнее
связывание.

```
#include "to_roman.hpp"

struct digit
{
    int arabic;
    std::string roman;
};

const digit digits[] =
{
    { 100, "C" },
    { 10, "X" },
    { 1, "I" },
};

const std::size_t n =
    sizeof digits / sizeof digits[0];

std::string to_roman(int number)
{
    std::string roman = "";
    for (std::size_t i = 0; i != n; ++i)
        while (number >= digits[i].arabic)
        {
            roman += digits[i].roman;
            number -= digits[i].arabic;
        }
    return roman;
}
```



Как я могу
задать для типа
внутреннее
связывание?



С помощью
анонимного
пространства
имен.

```
#include "to_roman.hpp"

struct digit
{
    int arabic;
    std::string roman;
};

const digit digits[] =
{
    { 100, "C" },
    { 10, "X" },
    { 1, "I" },
};

const std::size_t n =
    sizeof digits / sizeof digits[0];

std::string to_roman(int number)
{
    std::string roman = "";
    for (std::size_t i = 0; i != n; ++i)
        while (number >= digits[i].arabic)
        {
            roman += digits[i].roman;
            number -= digits[i].arabic;
        }
    return roman;
}
```



Можешь
показать на
примере?

© www.ClipProject.info



Вот так.

© www.ClipProject.info

```
#include "to_roman.hpp"

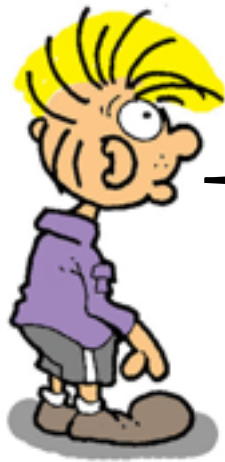
namespace
{
    struct digit
    {
        int arabic;
        std::string roman;
    };

    const digit digits[] =
    {
        { 100, "C" },
        { 10, "X" },
        { 1, "I" },
    };

    const std::size_t n =
        sizeof digits / sizeof digits[0];
}

...
```

```
$ cc -Wall -Werror *.cpp && ./a.out
All tests passed
```



Что еще?



Предлагаю
переименовать
переменную
number в arabic.

```
#include "to_roman.hpp"

namespace
{
    ...
}

std::string to_roman(int arabic)
{
    std::string roman = "";
    for (std::size_t i = 0; i != n; ++i)
        while (arabic >= digits[i].arabic)
        {
            roman += digits[i].roman;
            arabic -= digits[i].arabic;
        }
    return roman;
}
```

```
$ cc -Wall -Werror *.cpp && ./a.out
All tests passed
```




Запустим наши
тесты.

Думаю, «200» и
«300» должны
пройти.



«333»
тоже прошел.

Отлично.

```
#include "to_roman.hpp"
#include <cassert>
#include <iostream>
...
static void test_to_roman()
{
    assert_to_roman( 1, "I");
    assert_to_roman( 2, "II");
    assert_to_roman( 3, "III");
    assert_to_roman(10, "X");
    assert_to_roman(20, "XX");
    assert_to_roman(30, "XXX");
    assert_to_roman(33, "XXXIII");
    assert_to_roman(100, "C");
    assert_to_roman(200, "CC");
    assert_to_roman(300, "CCC");
    assert_to_roman(333, "CCCXXXIII");
}
```

```
$ cc -Wall -Werror *.cpp && ./a.out
All tests passed
```



Вызовы наших тестов фактически дублируют друг друга. Давай проведем рефакторинг.



Не уверена, что оно того стоит.

Но давай попробуем и посмотрим, станет ли лучше.

```
#include "to_roman.hpp"
#include <cassert>
#include <iostream>
...
static void test_to_roman()
{
    assert_to_roman( 1, "I");
    assert_to_roman( 2, "II");
    assert_to_roman( 3, "III");
    assert_to_roman( 10, "X");
    assert_to_roman( 20, "XX");
    assert_to_roman( 30, "XXX");
    assert_to_roman( 33, "XXXIII");
    assert_to_roman(100, "C");
    assert_to_roman(200, "CC");
    assert_to_roman(300, "CCC");
    assert_to_roman(333, "CCCXXXIII");
}
```



Как и раньше - создам структуру. В ней добавлю поля типа int и string.



```
#include "to_roman.hpp"
#include <cassert>
#include <iostream>
...

struct test_data
{
    int arabic;
    std::string roman;
};

static void test_to_roman()
{
    assert_to_roman( 1, "I");
    assert_to_roman( 2, "II");
    assert_to_roman( 3, "III");
    assert_to_roman( 10, "X");
    assert_to_roman( 20, "XX");
    assert_to_roman( 30, "XXX");
    assert_to_roman( 33, "XXXIII");
    assert_to_roman(100, "C");
    assert_to_roman(200, "CC");
    assert_to_roman(300, "CCC");
    assert_to_roman(333, "CCCXXXIII");
}
```



Теперь объявлю
массив таких структур.
Начну с двух.



```
#include "to_roman.hpp"
#include <cassert>
#include <iostream>
...
struct test_data
{
    int arabic;
    std::string roman;
};

const test_data tests[] =
{
    { 1, "I" },
    { 2, "II" },
};

static void test_to_roman()
{
    assert_to_roman( 1, "I");
    assert_to_roman( 2, "II");
    assert_to_roman( 3, "III");
    assert_to_roman( 10, "X");
    assert_to_roman( 20, "XX");
    assert_to_roman( 30, "XXX");
    assert_to_roman( 33, "XXXIII");
    assert_to_roman(100, "C");
    assert_to_roman(200, "CC");
    assert_to_roman(300, "CCC");
    assert_to_roman(333, "CCCXXXIII");
}
```



Перепишу код с
ИСПОЛЬЗОВАНИЕМ ЭТОГО
МАССИВА.



```
#include "to_roman.hpp"
#include <cassert>
#include <iostream>
...
struct test_data
{
    int arabic;
    std::string roman;
};

const test_data tests[] =
{
    { 1, "I" },
    { 2, "II" },
};

int main()
{
    for (size_t i = 0; i != 2; ++i)
        assert_to_roman(
            tests[i].arabic,
            tests[i].roman);

    test_to_roman();
    std::cout << "All tests passed"
              << std::endl;
}
```

```
$ cc -Wall -Werror *.cpp && ./a.out
All tests passed
```



И заменю литерал «2»
на число элементов
массива.



```
#include "to_roman.hpp"
#include <cassert>
#include <iostream>
...
struct test_data
{
    int arabic;
    std::string roman;
};

const test_data tests[] =
{
    { 1, "I" },
    { 2, "II" },
};

int main()
{
    const size_t n =
        sizeof tests / sizeof tests[0];

    for (size_t i = 0; i != n; ++i)
        assert_to_roman(
            tests[i].arabic,
            tests[i].roman);

    test_to_roman();
    std::cout << "All tests passed"
              << std::endl;
}
```

```
$ cc -Wall -Werror *.cpp && ./a.out
All tests passed
```



Добавлю в массив
остальные элементы.



```
#include "to_roman.hpp"
#include <cassert>
#include <iostream>
...
struct test_data
{
    int arabic;
    std::string roman;
};

const test_data tests[] =
{
    { 1, "I" },
    { 2, "II" },
    { 3, "III" },
    { 10, "X" },
    { 20, "XX" },
    { 30, "XXX" },
    { 33, "XXXIII" },
    { 100, "C" },
    { 200, "CC" },
    { 300, "CCC" },
    { 333, "CCCXXXIII" },
};
...
```

```
$ cc -Wall -Werror *.cpp && ./a.out
All tests passed
```



Теперь я могу
удалить
test_to_roman() и ее
ВЫЗОВ.



Ага.

```
...
const test_data tests[] =
{
    { 1, "I" },
    ...
    { 333, "CCCXXXIII" },
};

int main()
{
    const size_t n =
        sizeof tests / sizeof tests[0];

    for (size_t i = 0; i != n; ++i)
        assert_to_roman(
            tests[i].arabic,
            tests[i].roman);

    std::cout << "All tests passed"
              << std::endl;
}
```

```
$ cc -Wall -Werror *.cpp && ./a.out
All tests passed
```




У тебя есть еще предложения?



Может, сделаем assert_to_roman() методом структуры test_data?

```
...
struct test_data
{
    int arabic;
    std::string roman;
};

const test_data tests[] =
{
    { 1, "I" },
    ...
    { 333, "CCCXXXIII" },
};

int main()
{
    const size_t n =
        sizeof tests / sizeof tests[0];

    for (size_t i = 0; i != n; ++i)
        assert_to_roman(
            tests[i].arabic,
            tests[i].roman);

    std::cout << "All tests passed"
              << std::endl;
}
```



Согласен. Сначала
ИЗМЕНЮ ВЫЗОВ.



```
...
struct test_data
{
    int arabic;
    std::string roman;
};

const test_data tests[] =
{
    { 1, "I" },
    ...
    { 333, "CCCXXXIII" },
};

int main()
{
    const size_t n =
        sizeof tests / sizeof tests[0];

    for (size_t i = 0; i != n; ++i)
        tests[i].assert_equal();

    std::cout << "All tests passed"
              << std::endl;
}
```



Теперь добавлю
объявление метода.

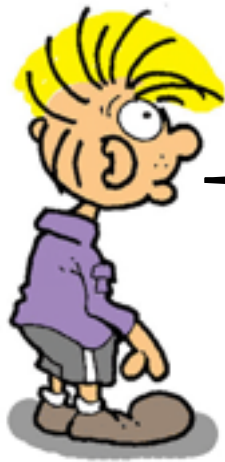


```
...
struct test_data
{
    int arabic;
    std::string roman;
    void assert_equal() const;
};

int main()
{
    const size_t n =
        sizeof tests / sizeof tests[0];

    for (size_t i = 0; i != n; ++i)
        tests[i].assert_equal();

    std::cout << "All tests passed"
              << std::endl;
}
```

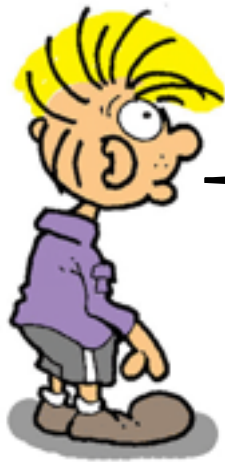


И создам его реализацию.



```
...
struct test_data
{
    int arabic;
    std::string roman;
    void assert_equal() const;
};

void test_data::assert_equal() const
{
    const std::string actual = to_roman(arabic);
    if (expected != actual)
    {
        std::cerr << "expected: to_roman"
                  << '(' << arabic << ") == "
                  << "'" << expected << "'"
                  << std::endl;
        std::cerr << "  actual: to_roman"
                  << '(' << arabic << ") == "
                  << "'" << actual << "'"
                  << std::endl;
        assert(false);
    }
}
```



Нужно будет переименовать «roman» в «expected».

© www.clipartland.com



Ок.

© www.clipartland.com

```
...
struct test_data
{
    int arabic;
    std::string expected;
    void assert_equal() const;
};

void test_data::assert_equal() const
{
    const std::string actual = to_roman(arabic);
    if (expected != actual)
    {
        std::cerr << "expected: to_roman"
                  << '(' << arabic << ") == "
                  << "'" << expected << "'"
                  << std::endl;
        std::cerr << "  actual: to_roman"
                  << '(' << arabic << ") == "
                  << "'" << actual << "'"
                  << std::endl;
        assert(false);
    }
}
```

```
$ cc -Wall -Werror *.cpp && ./a.out
All tests passed
```



Что-нибудь еще?

Имя «test_data»
звучит так себе.

И эта структура не
должна иметь
внешнего связывания.

Согласен.

Но мы можем вернуться к
этому позже.





Давай сравним наши
тесты до и после
рефакторинга.



Точно! Я и забыл, ради чего
всё это было!



Вот так наши
тесты выглядели
раньше.

В их прямоте
была
своеобразная
элегантность.



```
#include "to_roman.hpp"
#include <cassert>
#include <iostream>
...
static void test_to_roman()
{
    assert_to_roman( 1, "I");
    assert_to_roman( 2, "II");
    assert_to_roman( 3, "III");
    assert_to_roman( 10, "X");
    assert_to_roman( 20, "XX");
    assert_to_roman( 30, "XXX");
    assert_to_roman( 33, "XXXIII");
    assert_to_roman(100, "C");
    assert_to_roman(200, "CC");
    assert_to_roman(300, "CCC");
    assert_to_roman(333, "CCCXXXIII");
}

int main()
{
    test_to_roman();
    std::cout << "All tests passed"
              << std::endl;
}
```




А вот такими они
стали после
рефакторинга.



Код выглядит
совсем иначе. И его
стало больше! Не
знаю, стоило ли оно
того.

```
...
struct test_data
{
    int arabic;
    std::string expected;
    void assert_equal() const;
};

const test_data tests[] =
{
    { 1, "I" },
    { 2, "II" },
    { 3, "III" },
    { 10, "X" },
    { 20, "XX" },
    { 30, "XXX" },
    { 33, "XXXIII" },
    { 100, "C" },
    { 200, "CC" },
    { 300, "CCC" },
    { 333, "CCCXXXIII" },
};

int main()
{
    const size_t n =
        sizeof tests / sizeof tests[0];

    for (size_t i = 0; i != n; ++i)
        tests[i].assert_equal();

    std::cout << "All tests passed"
              << std::endl;
}
```

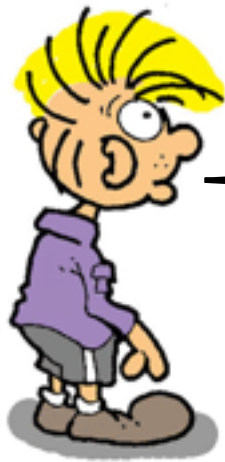


Что теперь?

Давай добавим тест для «5».

Хорошо.





Я просто добавлю еще один элемент в массив.



Как и ожидалось - тест провалился.

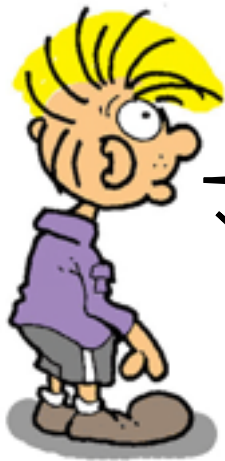
```
...
const test_data tests[] =
{
    { 1, "I" },
    { 2, "II" },
    { 3, "III" },
    { 5, "V" },
    { 10, "X" },
    { 20, "XX" },
    { 30, "XXX" },
    { 33, "XXXIII" },
    { 100, "C" },
    { 200, "CC" },
    { 300, "CCC" },
    { 333, "CCCXXXIII" },
};

int main()
{
    const size_t n =
        sizeof tests / sizeof tests[0];

    for (size_t i = 0; i != n; ++i)
        tests[i].assert_equal();

    std::cout << "All tests passed"
               << std::endl;
}
```

```
$ cc -Wall -Werror *.cpp && ./a.out
expected: to_roman(5) == "V"
actual: to_roman(5) == "IIIII"
Assertion failed: (false), ...
```



Теперь сделаем так, чтобы он прошел.

Нужно просто указать, что «5» - это «V».



Отлично.

```
...
namespace
{
    struct digit
    {
        int arabic;
        std::string roman;
    };

    const digit digits[] =
    {
        { 100, "C" },
        {  10, "X" },
        {   5, "V" },
        {   1, "I" },
    };

    const std::size_t n =
        sizeof digits / sizeof digits[0];
}

std::string to_roman(int arabic)
{
    std::string roman = "";
    for (std::size_t i = 0; i != n; ++i)
        while (arabic >= digits[i].arabic)
        {
            roman += digits[i].roman;
            arabic -= digits[i].arabic;
        }
    return roman;
}
```

```
$ cc -Wall -Werror *.cpp && ./a.out
All tests passed
```



Тесты для «6» и «338»
должны пройти
успешно.

© www.ClipProject.com

Согласна.



© www.ClipProject.com

```
...
const test_data tests[] =
{
    { 1, "I" },
    { 2, "II" },
    { 3, "III" },
    { 5, "V" },
    { 6, "VI" },
    { 10, "X" },
    { 20, "XX" },
    { 30, "XXX" },
    { 33, "XXXIII" },
    { 100, "C" },
    { 200, "CC" },
    { 300, "CCC" },
    { 333, "CCCXXXIII" },
    { 338, "CCCXXXVIII" },
};

int main()
{
    const size_t n =
        sizeof tests / sizeof tests[0];

    for (size_t i = 0; i != n; ++i)
        tests[i].assert_equal();

    std::cout << "All tests passed"
              << std::endl;
}
```

```
$ cc -Wall -Werror *.cpp && ./a.out
All tests passed
```



© www.stardred.ru

Давай теперь
добавим «4» и
«IV».

Ок.

Мы уже поддержали «1»/«I»
и «5»/«V». Возможно,
мы могли бы как-то
объединить их, чтобы
поддержать «4»/«IV».

Хм...

Ну, думаю, это
будет довольно
сложно.

Ага...



© www.stardred.ru



© www.studio100.ru

А ты как считаешь?

Я думаю, что когда что-то становится сложным - это признак того, что мы на неверном пути.

Мы могли бы просто добавить «4» в массив наших цифр! Хотя это похоже на костыль...

Думаю, это отличная идея. И я не считаю это костылем! Это простое и рабочее решение.

Ок!



© www.studio100.ru



Запущу тест для «4».

Он провалился,
как и должен
был.



```
...
const test_data tests[] =
{
    { 1, "I" },
    { 2, "II" },
    { 3, "III" },
    { 4, "IV" },
    { 5, "V" },
    { 6, "VI" },
    { 10, "X" },
    { 20, "XX" },
    { 30, "XXX" },
    { 33, "XXXIII" },
    { 100, "C" },
    { 200, "CC" },
    { 300, "CCC" },
    { 333, "CCCXXXIII" },
    { 338, "CCCXXXVIII" },
};
```

```
$ cc -Wall -Werror *.cpp && ./a.out
expected: to_roman(4) == "IV"
actual: to_roman(4) == "IIII"
Assertion failed: (false), ...
```




Теперь сделаю так, чтобы он прошел.

Просто укажу, что «4» - это «IV».



Прекрасно.

```
...
namespace
{
    struct digit
    {
        int arabic;
        std::string roman;
    };

    const digit digits[] =
    {
        { 100, "C" },
        { 10, "X" },
        { 5, "V" },
        { 4, "IV" },
        { 1, "I" },
    };

    const std::size_t n =
        sizeof digits / sizeof digits[0];
}

std::string to_roman(int arabic)
{
    std::string roman = "";
    for (std::size_t i = 0; i != n; ++i)
        while (arabic >= digits[i].arabic)
        {
            roman += digits[i].roman;
            arabic -= digits[i].arabic;
        }
    return roman;
}
```

```
$ cc -Wall -Werror *.cpp && ./a.out
All tests passed
```



Мы можем и дальше развивать эту идею -
добавить «9»/«IX», «40»/«XL» и т.д.

Согласна.

Так что мы сделали эту кату.
Дальше просто
механическая работа.

Да. Хотя, думаю, здесь есть еще
место для нескольких мелких
улучшений. И раньше мы
договорились, что уберем
внешнее связывание
для структуры test_data.

Спасибо тебе за помощь.

Обращайся.

