

**Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования**

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ
ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
(УНИВЕРСИТЕТ ИТМО)**

Кафедра компьютерной фотоники и видеоинформатики

ОТЧЕТ ПО ПРАКТИКЕ

Выполнил:
студент группы № V3316
Герасимов В. В.

г. Санкт-Петербург
2018 г.

Оглавление

Цель проведения практики	3
Задание №1. Знакомство с системой контроля версий Git	4
Задание №2. Форматирование и стиль	7
Задание №3. TDD: Разработка через тестирование	10
Задание. Подсчет количества и характеристик зерен.....	17
Приложение.....	19
Презентация (pptx).....	19
Слайды	20

Цель проведения практики

Освоение навыков использования C++ и изучение приемов разработки программного обеспечения. Практика проходит в компьютерном классе и состоит из лекционных занятий и практических заданий.

Задание №1. Знакомство с системой контроля версий Git

1. Условие

- a. Завести аккаунт на <https://github.com/>.
- b. Создать репозиторий для лабораторных работ. 4. Склонировать репозиторий из <https://github.com/> на компьютер (git clone ...).
- c. Создать папку lab1.
- d. Написать в папке lab1 программу, вычисляющую функцию факториала.
- e. Создать .gitignore, добавив в игнорируемые файлы – промежуточные файлы компиляции (объектные файлы и другие временные файлы) и исполняемый файл. В результате должны остаться только файл проекта и исходные файлы.
- f. Сделать по крайней мере два коммита. Отправить зафиксированные изменения на <https://github.com/> (git push)
- g. Привести результаты работы “git log”.
- h. Показать при помощи “git diff <ваш_файл_ваши_коммиты>” изменение любого файла между двумя коммитами.

2. Текст программы

```
1  #include <iostream>
2  using namespace std;
3
4  long fact(int N) {
5      if (N < 0) {
6          return 0;
7      }
8
9      if (N == 0) {
10         return 1;
11     }
12     else {
13         return N * fact(N - 1);
14     }
15 }
16
17 int main() {
18     int N = 0;
19
20     cin >> N;
21
22     cout << "Factorial of " << N << " is " << fact(N) << endl;
23
24     return 0;
25 }
```

3. Выполненные команды и результаты вывода (клонирование, добавление файлов, создание коммита, diff, log)

а. Клонирование

```
Vladislav@DESKTOP-988JLKS MINGW64 ~/Documents/University/Practice/Git
$ git clone https://github.com/itmosphere/practice2018.git
Cloning into 'practice2018'...
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
```

б. Добавление файлов

```
Vladislav@DESKTOP-988JLKS MINGW64 ~/Documents/University/Practice/Git/practice2018 (master)
$ git add lab1
```

с. Создание коммита

```
Vladislav@DESKTOP-988JLKS MINGW64 ~/Documents/University/Practice/Git/practice2018 (master)
$ git commit -m "upload the main program"
[master 5a9715e] upload the main program
1 file changed, 25 insertions(+)
create mode 100644 lab1/fact.cpp
```

д. Push

```
Vladislav@DESKTOP-988JLKS MINGW64 ~/Documents/University/Practice/Git/practice2018 (master)
$ git push origin master
Counting objects: 4, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 502 bytes | 167.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0)
To https://github.com/itmosphere/practice2018.git
cac65b8..5a9715e master -> master
```

e. Diff

```
Vladislav@DESKTOP-988JLKS MINGW64 ~/Documents/University/Practice/Git/practice2018 (master)
$ git diff 5a9715e9608bbd15c66f9d9f1714b327570bc5f8 a00f511ed56ad57fdb0b64783406c8b8bc49d87f -- lab1/fact.cpp
diff --git a/lab1/fact.cpp b/lab1/fact.cpp
index eb82b00..36684b0 100644
--- a/lab1/fact.cpp
+++ b/lab1/fact.cpp
@@ -19,7 +19,7 @@ int main() {

    cin >> N;

-    cout << "Factorial of" << N << " is " << fact(N) << endl;
+    cout << "Factorial of " << N << " is " << fact(N) << endl;

    return 0;
}
\ No newline at end of file
```

Пояснение к фрагменту «@@ -19,7 +19,7 @@»: «-» относится к старому (первому) файлу, «+» - к новому (второму). Числа сообщают, о каком участке кода идет речь – «начальная строка, количество рассматриваемых строк».

f. Log

Пояснение к фрагменту «(HEAD -> master, origin/master, origin/HEAD)»: HEAD указывает на текущий коммит, стрелка справа от HEAD показывает, какая ветка (если такая есть) является текущей; origin/master ссылается на ветку master удаленного репозитория; origin/HEAD представляет ветку по умолчанию на удаленном репозитории.

```
Vladislav@DESKTOP-988JLKS MINGW64 ~/Documents/University/Practice/Git/practice2018 (master)
$ git log
commit a00f511ed56ad57fdb0b64783406c8b8bc49d87f (HEAD -> master, origin/master, origin/HEAD)
Author: ITMOSphere <s4murai.mail@gmail.com>
Date:   Fri Feb 2 02:02:01 2018 +0300

    fix typo

commit 60be6ab2bda6297711efffd123f453d544e09cea
Author: ITMOSphere <s4murai.mail@gmail.com>
Date:   Fri Feb 2 01:56:27 2018 +0300

    upload lab1 description

commit 5a9715e9608bbd15c66f9d9f1714b327570bc5f8
Author: ITMOSphere <s4murai.mail@gmail.com>
Date:   Fri Feb 2 01:49:40 2018 +0300

    upload the main program

commit cac65b888bceb59e609221891c6f26db29788c38
Author: itmosphere <36055358+itmosphere@users.noreply.github.com>
Date:   Fri Feb 2 01:36:24 2018 +0300

    Initial commit
```

Задание №2. Форматирование и стиль

1. Условие

1748. Самое сложное число

Ограничение времени: 1.0 секунды

Ограничение памяти: 64 МБ

Определим *сложность* числа как количество его делителей. Вы должны найти среди всех целых чисел от 1 до n самое сложное число. Если таких чисел несколько, требуется найти наименьшее из них.

Исходные данные

В первой строке записано количество тестов t ($1 \leq t \leq 100$). В i -й из следующих t строк записано целое число n_i ($1 \leq n_i \leq 10^{18}$).

Результат

Для каждого теста выведите ответ в отдельной строке. В i -й строке выведите через пробел самое сложное число на отрезке от 1 до n_i и его сложность.

Пример

<i>исходные данные</i>	<i>результат</i>
5	1 1
1	6 4
10	60 12
100	840 32
1000	7560 64
10000	

2. Текст программы

```
#include <cstdio>
#include <iostream>
#include <vector>

using namespace std;

vector<int> g_primes;
```

```

long long g_number = 0, g_complexity = 0, g_complexInt = 0;

bool IsPrime(int iNumber) {
    if (iNumber == 2) {
        return true;
    }
    if (iNumber % 2 == 0) {
        return false;
    }
    for (int i = 2; i <= iNumber / i; ++i) {
        if (iNumber % i == 0) {
            return false;
        }
    }
    return true;
}

void Solve(int iPosition, long long iGuess, long long iDivisor, int iLast) {
    if (iDivisor > g_complexity || (iDivisor == g_complexity && iGuess < g_complexInt)) {
        g_complexity = iDivisor;
        g_complexInt = iGuess;
    }
    if (iPosition == 16) {
        return;
    }
    for (int i = 1; i <= iLast && iGuess <= g_number / g_primes[iPosition]; ++i) {
        iGuess *= g_primes[iPosition];
        Solve(iPosition + 1, iGuess, iDivisor * (i + 1), i);
    }
}

int main() {
    // ограничение цикла до 16, т.к. primoral(16) ~ 3 * 10^19
    for (int i = 2; g_primes.size() < 16; ++i) {
        if (IsPrime(i)) {
            g_primes.push_back(i);
        }
    }
    //
    int total = 0;
    cin >> total;
    //
    while (total--) {
        cin >> g_number;
        //
        g_complexity = 1;
        g_complexInt = 1;
        //
        Solve(0, 1, 1, 16);
        //
        cout << g_complexInt << " " << g_complexity << endl;
    }
}

```


3. Резултат с Timus

ID	Date	Author	Problem	Language	Judgement result	Test #	Execution time	Memory used
7741196	02:01:00 2 Feb 2018	s4muraj	1748. The Most Complex Number	G++ 7.1	Accepted		0.078	436 KB

Задание №3. TDD: Разработка через тестирование

1. Условие

1032. Find a Multiple

Ограничение времени: 1.0 секунды

Ограничение памяти: 64 МБ

The input contains N positive integers ($N \leq 10000$). Each of that numbers is not greater than 15000. This numbers are not necessarily different (so it may happen that two or more of them will be equal). Your task is to choose a few of given numbers ($1 \leq \text{few} \leq N$) so that the sum of chosen numbers is multiple for N (i.e. $N * k = (\text{sum of chosen numbers})$ for some integer k).

Исходные данные

The first line of the input contains the single number N . Each of next N lines contains one number from the given set.

Результат

In case your program decides that the target set of numbers can not be found it should print to the output the single number 0. Otherwise it should print the number of the chosen numbers in the first line followed by the chosen numbers themselves (on a separate line each) in arbitrary order.

If there are more than one set of numbers with required properties you should print to the output only one (preferably your favorite) of them.

Пример

<i>исходные данные</i>	<i>результат</i>
5	2
1	2
2	3
3	
4	
1	

2. Возможные ошибки

а. Некорректный символ на входе

```
if (!(cin >> total)) {  
    return -1;  
}
```

```
if (!(cin >> tmp)) {  
    return -1;  
}
```

б. Значение, выходящее за границы

```
if (total > 10000 || total <= 0) {  
    return -1;  
}
```

```
if (tmp > 15000 || tmp <= 0) {  
    return -1;  
}
```

3. Результат с Timus

ID	Date	Author	Problem	Language	Judgement result	Test #	Execution time	Memory used
7741223	02:30:41 2 Feb 2018	s4murai	1032. Find a Multiple	G++ 7.1	Accepted		0.031	496 KB

4. Скриншот с покрытием кода

	Hit	Total	Coverage
Lines:	20	21	95.2 %
Functions:	1	1	100.0 %
Branches:	31	46	67.4 %

	Branch data	Line data	Source code
1		:	: //include <iostream>
2		:	: #include "lab_3_pure_function.h"
3		:	:
4		:	6 : string Function(int total, vector<int> answer) {
5	[+ +] [+ +]	:	6 : if (total > 10000 total <= 0 total > answer.size()) {
6	[- +] [+ +]	:	
7	[+ -]	2 :	return "-1";
8		:	}
9	[+ -]	10 :	vector<int> used(10000, -1);
10		:	//
11	[+ +]	10008 :	for (int i = 0; i < total; ++i) {
12	[+ +] [+ +]	10006 :	if (answer[i] > 15000 answer[i] <= 0) {
13	[+ -]	2 :	return "-2";
14		:	}
15		:	}
16		:	//
17		2 :	int sum = 0;
18		2 :	used[0] = 0;
19		:	//
20	[+ -]	4 :	string result = "";
21		:	//
22	[+ -]	10002 :	for (int i = 0; i < total; ++i) {
23		10002 :	sum += answer[i];
24		:	//
25	[+ +]	10002 :	if (used[sum % total] != -1) {
26	[+ -] [+ -]	2 :	result += to_string(i - used[sum % total] + 1) + "\n";
27	[+ -]	:	//
28	[+ +]	10003 :	for (int j = used[sum % total]; j <= i; ++j) {
29	[+ -] [+ -]	10001 :	result += to_string(answer[j]) + "\n";
30	[+ -]	:	}
31		:	//
32		2 :	break;
33		:	}
34		:	else {
35		10000 :	used[sum % total] = i + 1;
36		:	}
37		:	}
38	[- +]	2 :	if (result.empty()) {
39	[# #]	0 :	return "0";
40		:	}
41		2 :	return result;
42		:	}

5. Код тестов

а. Код основной программы

```
#include "lab_3_pure_function.h"
#include <iostream>
using namespace std;

// Тесты
#define MY_DEF_USE_LIBTAP
#ifdef MY_DEF_USE_LIBTAP
// Подключение libtap для Visual Studio
// 1. Скопировать каталог libtap в папку с исходными файлами
// 2. Указать две нижеследующие строки, соблюдая порядок:
```

```

#define TAP_COMPILE // включает компиляцию libtap (указывать в одном и только одном
исходном файле)
#include "libtap/cpp_tap.h" // подключает заголовочные файлы библиотеки (указывать
в тех исходных файлах, где создаются тесты)

int main(int, char *[]) {
    using namespace std;
    vector<int> test1 = {1,2,3,4,1};
    vector<int> test2 = {42};
    vector<int> test3 = {-20};
    vector<int> test4 = {20000};
    vector<int> test5(9999, 14999);
    plan_tests(6); // указываем количество тестов
    // тесты:
    // ok(x,msg) - тест проходит если условие x выполняется, msg - название теста
    // ok1(x) - тоже что и ok(x,msg), в качестве названия автоматически подставляется
текст условия
    // тесты на обычное использование
    ok(Function(5,test1) == "2\n2\n3\n", "Test the example from Timus");
    // тесты на ошибочные значения
    ok(Function(-1, test2) == "-1", "Function of N <= 0 (= -1) - error_code = -1");
    ok(Function(20000, test2) == "-1", "Function of N > 10000 (= 20000) - error_code
= -1");
    ok(Function(1, test3) == "-2", "Function with number <= 0 (= -20) - error_code =
-2");
    ok(Function(1, test4) == "-2", "Function with number > 15000 (= 20000) -
error_code = -2");
    //
    string maxResult = "9999\n";
    for(int i = 0; i < 9999; ++i) {
        maxResult += "14999\n";
    }
    ok(Function(9999, test5) == maxResult, "Test maximum values (N = 9999, each number
is 14999)");
    //
    return exit_status(); // вывод отчета по тестам
    return 0;
}
#else

```

```
// Для проверки на timus
int main(int, char *[]) {
    return 0;
}
#endif
```

б. Код тестируемой функции

```
#include "lab_3_pure_function.h"

string Function(int total, vector<int> answer) {
    if (total > 10000 || total <= 0 || total > answer.size()) {
        return "-1";
    }
    //
    vector<int> used(10000, -1);
    //
    for (int i = 0; i < total; ++i) {
        if (answer[i] > 15000 || answer[i] <= 0) {
            return "-2";
        }
    }
    //
    int sum = 0;
    used[0] = 0;
    //
    string result = "";
    //
    for (int i = 0; i < total; ++i) {
        sum += answer[i];
        //
        if (used[sum % total] != -1) {
            result += to_string(i - used[sum % total] + 1) + "\n";
            //
            for (int j = used[sum % total]; j <= i; ++j) {
                result += to_string(answer[j]) + "\n";
            }
            //
            break;
        }
    }
}
```

```

    }
    else {
        used[sum % total] = i + 1;
    }
}
if (result.empty()) {
    return "0";
}
return result;
}

```

6. Текст программы

```

#include <iostream>
#include <vector>

using namespace std;

int main() {
    int total = 0;
    //
    if (!(cin >> total)) {
        return -1;
    }
    if (total > 10000) {
        return -1;
    }
    //
    vector<int> answer(10000, 0);
    vector<int> used(10000, -1);
    //
    for (int i = 0; i < total; ++i) {
        int tmp = 0;
        //
        if (!(cin >> tmp)) {
            return -1;
        }
        if (tmp > 15000) {
            return -1;
        }
        //
        answer[i] = tmp;
    }
    //
    int sum = 0;
    used[0] = 0;
    //
    for (int i = 0; i < total; ++i) {
        sum += answer[i];
        //
        if (used[sum % total] != -1) {
            cout << i - used[sum % total] + 1 << endl;

```

```
    //
    for (int j = used[sum % total]; j <= i; ++j) {
        cout << answer[j] << endl;
    }
    //
    break;
}
else {
    used[sum % total] = i + 1;
}
}
}
```


Задание. Подсчет количества и характеристик зерен

В задании требовалось определить количество и характеристики зерен по скану со сканера.

Для распознавания изображений было решено использовать одну из наиболее распространенных библиотек для работы с изображениями на C++ – `opencv`, а для графического интерфейса программы использовался фреймворк `Qt`.

Алгоритм программы делится на три основные части: предобработку изображения для получения более качественных результатов, непосредственную обработку – поиск контуров зерен, и постобработку, заключающуюся в расчете длин, площадей и отрисовке графической составляющей.

В предобработке изображение подвергается размытию, эрозии, методу трансформации изображения в расстояния до ближайшей границы и дилатации. Размытие позволяет сделать фон изображения более равномерным; эрозия – компенсирует увеличение зерен после размытия; после метода трансформации изображения значение каждого пикселя заменяется его расстоянием до ближайшего пикселя фона, что позволяет лучше сегментировать зерна между собой; дилатация (наращивание) компенсирует размер зерен, который был потерян после трансформации.

Непосредственно поиск контуров выполняется с помощью методов модифицированного класса `opencv` – `SimpleBlobDetector`. Код класса был модифицирован таким образом, чтобы была возможность получить доступ к каждому из отдельных контуров (для расчета параметров и отрисовки контуров). Сама фильтрация происходит по возможной площади и «вытянутости» (`inertia`).

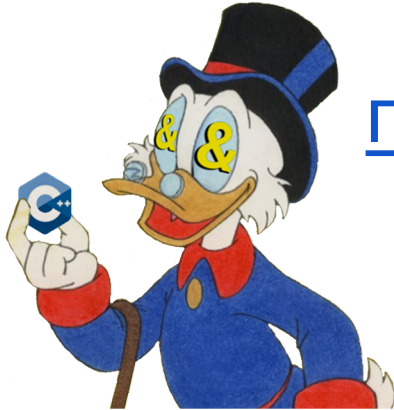
В постобработке каждый отдельный контур зерна выделяется на изображении своим случайным цветом, рядом с контуром располагается порядковый номер контура. В таблицу, рядом с порядковым номером зерна, заносятся приблизительные параметры зерна, переведенные в миллиметры – его длина, ширина и площадь. В цикле обработки осуществляется и подсчет общего количества зерен, который отображается под изображением.

Пример работы программы показан на изображении ниже.



Приложение

Презентация (pptx)



семантика перемещений

Сделал: Герасимов В.
Группа: V3316

2018 г.

Слайды



семантика перемещений

Сделал: Герасимов В.
Группа: V3316

2018 г.

Что будет?

Что будет?



lvalue и
rvalue

Что будет?







lvalue и
rvalue

семантика
перемещени
й

Что будет?

	lvalue и rvalue
	семантика перемещени й
	rvalue ССЫЛКИ

Что будет?

	lvalue и rvalue	
	семантика перемещени й	реализация std::move
	rvalue ССЫЛКИ	

Что будет?

	lvalue и rvalue		
	семантика перемещени й	реализация std::move	
	rvalue ссылки	ИТОГИ	

lvalue и rvalue

Интуитивное определение

lvalue - выражение, которое может появляться на левой или на правой стороне присваивания

rvalue - выражение, которое появляется только на правой стороне

Интуитивное определение

```
int a = 42;
```

```
int b = 43;
```

```
// a и b оба l-values:
```

```
a = b; // OK
```

```
b = a; // OK
```

```
a = a * b; // OK
```

```
// a * b это rvalue:
```

```
int c = a * b; // OK, rvalue на правой стороне присваивания
```

```
a * b = 42; // ошибка, rvalue на левой стороны присваивания
```


Альтернативное определение

lvalue - выражение, которое ссылается на место в памяти и позволяет нам взять адрес этого места с помощью оператора **&**.

rvalue - выражение, которое не lvalue.

Альтернативное определение

```
// lvalues:
int i = 42;
i = 43; // ОК, i это lvalue
int* p = &i; // ОК, i это lvalue
int& foo();
foo() = 42; // ОК, foo() это lvalue
int* p1 = &foo(); // ОК, foo() это lvalue

// rvalues:
int foobar();
int j = 0;
j = foobar(); // ОК, foobar() это rvalue
int* p2 = &foobar(); // ошибка: невозможно взять адрес у rvalue
j = 42; // ОК, 42 это rvalue
```

Семантика перемещений (move semantics)

Семантика перемещений (move semantics)

Оператор присваивания копированием

```
X& X::operator=(X const & rhs)
{
    // [...]
    // Создаем копию ресурса, на который ссылается rhs.m_pResource
    // Уничтожаем ресурс, на который ссылается m_pResource
    // Прикрепляем копию к m_pResource
    // [...]
}
```

Семантика перемещений (move semantics)

```
X foo();  
X x;  
// возможно, используем x различными способами  
x = foo();
```



Строчка выше:

- копирует ресурс из временного объекта, возвращенного foo,

Семантика перемещений (move semantics)

```
X foo();  
X x;  
// возможно, используем x различными способами  
x = foo();
```



Строчка выше:

- копирует ресурс из временного объекта, возвращенного foo,
- удаляет ресурс, содержащийся в x и заменяет его клоном,



Семантика перемещений (move semantics)

```
X foo();  
X x;  
// возможно, используем x различными способами  
x = foo();
```



Строчка выше:

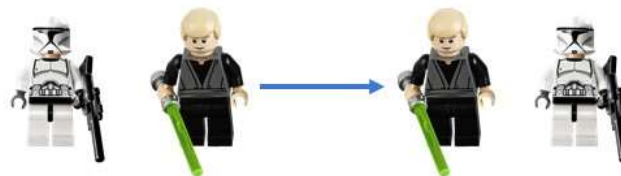
- копирует ресурс из временного объекта, возвращенного fo
- удаляет ресурс, содержащийся в x и заменяет его клоном,
- удаляет временный объект, тем самым освобождая его ресурс



Семантика перемещений (move semantics)

В специальном случае, когда на правой стороне присваивания стоит rvalue:

```
// [...]  
// поменять местами m_pResource и rhs.m_pResource  
// [...]
```



Семантика перемещений (move semantics)

В C++11 такого условного поведения можно добиться перегрузкой:

```
X& X::operator=(<загадочный тип> rhs)
{
    // [...]
    // поменять местами this->m_pResource и
rhs.m_pResource
    // [...]
}
```

Семантика перемещений (move semantics)

- Точно хотим, чтобы правостороннее выражение передавалось по ссылке
- Если есть выбор между перегрузками, **rvalue** выражения должны использовать **загадочный тип**, а **lvalue** - **обычную ссылку**



rvalue ссылки

Rvalue ссылки

Если X - любой тип, то X&& называется **rvalue ссылкой** на X.

Для различения, обычная ссылка X& теперь упоминается как **lvalue ссылка**.

Rvalue ссылки

```
void foo(X& x); // перегрузка lvalue ссылки
```

```
void foo(X&& x); // перегрузка rvalue ссылки
```

```
X x;
```

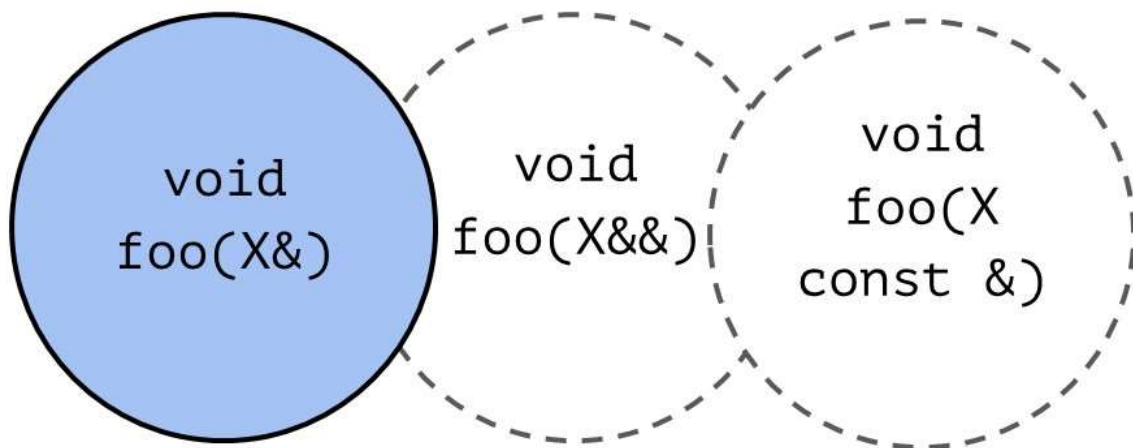
```
X foobar();
```

```
foo(x); // параметр lvalue: вызывает foo(X&)
```

```
foo(foobar()); // параметр rvalue: вызывает foo(X&&)
```

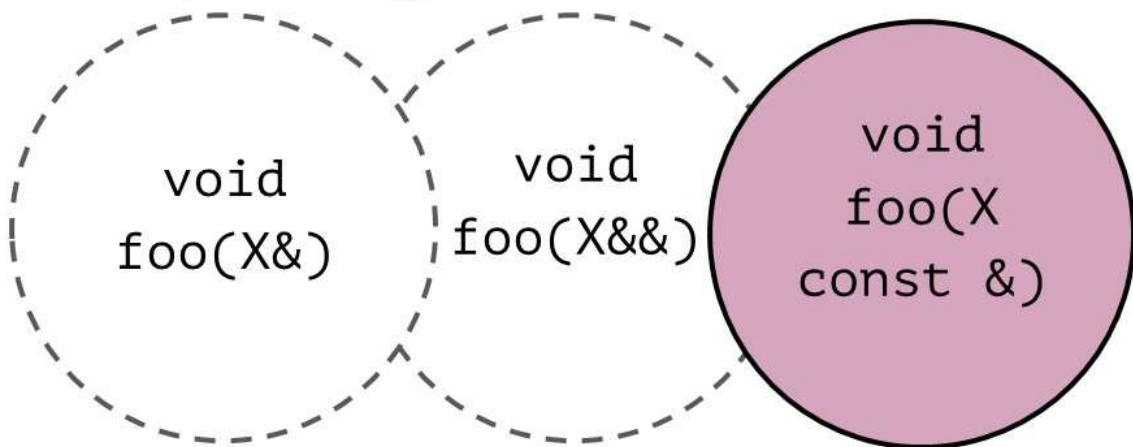

Rvalue ссылки

Поведение не меняется: foo может вызываться для *lvalue*, но не *rvalue*.



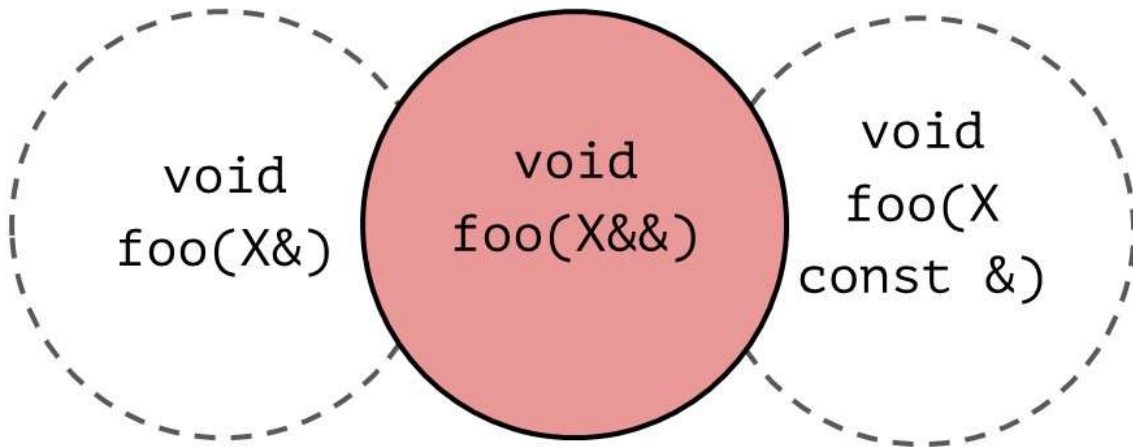
Rvalue ссылки

Поведение не меняется: foo может вызываться и для *lvalue*, и для *rvalue*, но их нельзя различить между собой.



Rvalue ссылки

Согласно C++11, foo может вызываться для **rvalue**, а вызовы для **lvalue** повлекут ошибку компиляции.



Обязывание семантики перемещений

Обязывание семантики перемещений

C++11 позволяет использовать семантику перемещений не только на **rvalue**, но, на усмотрение пользователя, и на **lvalue**.

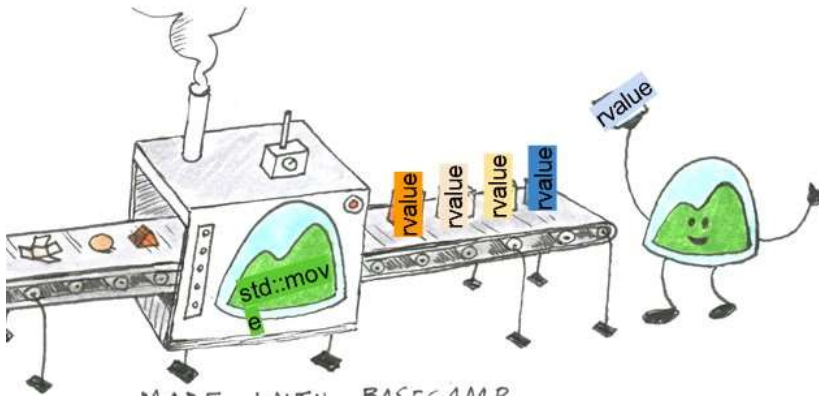
Обязывание семантики перемещений

```
template<class T>
void swap(T& a, T& b)
{
    T tmp(a);
    a = b;
    b = tmp;
}
```

```
X a, b;
swap(a, b);
```

Обязывание семантики перемещений

В библиотеке `std` C++11 есть функция `std::move`, которая нам поможет. Это функция, которая преобразует переданный параметр в `rvalue`, не делая ничего больше.



Обязывание семантики перемещений

```
template<class T>
void swap(T& a, T& b)
{
    T tmp(std::move(a));
    a = std::move(b);
    b = std::move(tmp);
}
```

```
X a, b;
swap(a, b);
```

Преимущества

Преимущества

- Многие стандартные алгоритмы и операции воспользуются семантикой перемещений, набирая потенциально значительный прирост в производительности
- Теперь мы можем использовать типы, являющиеся *moveable*, но не *copyable* (например, `unique_ptr`) во многих местах, где ранее они не разрешались



rvalue ссылка - rvalue?

rvalue ссылка - rvalue?

```
void foo(X&& x)
{
    X anotherX = x;
    // ...
}
```

rvalue ссылка - rvalue?

```
void foo(X&& x)
{
    X anotherX = x;
    // ...
}

X(X&& rhs);
```

rvalue ссылка - rvalue?

Всё, что объявлено как **rvalue ссылка**, может быть как **lvalue**, так и **rvalue**.

Критерий различимости: **если у этого есть имя**, то это **lvalue**. В ином случае, это **rvalue**.



rvalue ссылка - rvalue?

```
void foo(X&& x)
{
    X anotherX = x; // вызывает X(X const & rhs)
}
```

rvalue ссылка - rvalue?

```
X&& goo();
X x = goo(); // вызывает X(X&& rhs), т.к. значение справа
              // не имеет имени
```

rvalue ссылка - rvalue?

`std::move` пропускает через себя параметр по ссылке, не совершая над ним никаких операций, а результатом является **rvalue** ссылка. Выражение

```
std::move(x)
```

является **rvalue ссылкой** и не имеет имени. То есть, это **rvalue**.

rvalue ссылка - rvalue?

```
Base(Base const & rhs); // не семантика перемещений
```

```
Base(Base&& rhs); // семантика перемещений
```


rvalue ссылка - rvalue?

```
Base(Base const & rhs); // не семантика перемещений  
Base(Base&& rhs); // семантика перемещений
```

```
Derived(Derived const & rhs)  
: Base(rhs)  
{  
    // Выражения, связанные с Derived  
}
```

rvalue ссылка - rvalue?

```
Base(Base const & rhs); // не семантика перемещений  
Base(Base&& rhs); // семантика перемещений  
Derived(Derived const & rhs)  
: Base(rhs)  
{  
    // Выражения, связанные с Derived  
}  
Derived(Derived&& rhs)  
: Base(rhs) // ошибка: rhs это lvalue  
{  
    // Выражения, связанные с Derived  
}
```

rvalue ссылка - rvalue?

```
Base(Base const & rhs); // не семантика перемещений
Base(Base&& rhs); // семантика перемещений
Derived(Derived const & rhs)
: Base(rhs)
{
    // Выражения, связанные с Derived
}
Derived(Derived&& rhs)
: Base(std::move(rhs)) // ОК, вызывает Base(Base&& rhs)
{
    // Выражения, связанные с Derived
}
```

Оптимизация компилятора

Оптимизация компилятора

```
X foo()  
{  
    X x;  
    // возможно, используем x различными способами  
    return x;  
}
```

Оптимизация компилятора

```
X foo()  
{  
    X x;  
    // возможно, используем x различными способами  
    return std::move(x); // Только хуже!  
}
```

Оптимизация компилятора

Любой современный компилятор применит оптимизацию возвращаемого значения. Другими словами, вместо создания *X* локально и последующего его копирования, компилятор создаст *X* прямо на месте возвращаемого значения *foo*.



Реализация `std::move`

Реализация std::move

```
template<class T>
typename remove_reference<T>::type&&
std::move(T&& a) noexcept
{
    typedef typename remove_reference<T>::type&& RvalRef;
    return static_cast<RvalRef>(a);
}
```

Реализация std::move

```
template<class T>
typename remove_reference<T>::type&&
std::move(T&& a) noexcept
{
    typedef typename remove_reference<T>::type&& RvalRef;
    return static_cast<RvalRef>(a);
}
```

```
X x;
std::move(x);
```

Реализация std::move

По правилу вывода аргументов, шаблонный параметр T станет X&. Значит, компилятор создаст следующие экземпляры:

```
typename remove_reference<X&>::type&&
std::move(X& && a) noexcept
{
    typedef typename remove_reference<X&>::type&& RvalRef;
    return static_cast<RvalRef>(a);
}
```

После выполнения remove_reference и применения правил сворачивания ссылок:

```
X&& std::move(X& a) noexcept
{
    return static_cast<X&&>(a);
}
```

Реализация std::move

Вместо

```
std::move(x);
```

можно написать

```
static_cast<X&&>(x);
```

Но std::move строго рекомендуется, т.к. он лучше передает смысл.

rvalue ссылки и исключения

rvalue ссылки и исключения

При перегрузке конструктора копирования и оператора присваивания копированием с целью внедрения семантики перемещений, сильно рекомендуется сделать следующее:

1. Стараться написать перегрузки таким образом, чтобы они не выкидывали исключений. Зачастую это простая задача, поскольку семантика перемещений обычно лишь меняет местами указатели (или дескрипторы) между двумя объектами
2. Если это удалось, то следует отметить этот факт с помощью ключевого слова `noexcept`

rvalue ссылки и исключения

Если не выполнить оба этих пункта, то возникнет как минимум одна часто возникающая ситуация, где семантика перемещений не может быть применена, хотя ожидается обратное: когда `std::vector` меняет размер, мы точно хотим использовать семантику перемещений при перемещении существующих элементов на новый блок памяти. Но это произойдет только в том случае, если и п.1, и п.2 выполнены.

Вопросы

Вопросы

- В чем разница этих двух перегрузок?

```
void foo(X& x);  
void foo(X&& x);
```

- разницы нет; они одинаковые
- перегрузка X& для lvalue ссылки, перегрузка X&& для rvalue ссылки
- перегрузка X& для rvalue ссылки, перегрузка X&& для lvalue ссылки

Вопросы

- В чем разница этих двух перегрузок?

```
void foo(X& x);  
void foo(X&& x);
```

- ~~● разницы нет; они одинаковые~~
- перегрузка X& для lvalue ссылки, перегрузка X&& для rvalue ссылки
- ~~● перегрузка X& для rvalue ссылки, перегрузка X&& для lvalue ссылки~~

Вопросы

- Какое главное приложение у перегрузок из прошлого слайда?
- Позволяют использовать в классах *copyable*, но не *moveable* типы
- Используются для обхода оптимизации компилятора
- Перегрузка у классов *конструктора копирования* и *оператора присваивания копированием* для внедрения семантики перемещений

Вопросы

- Какое главное приложение у перегрузок из прошлого слайда?
- ~~● Позволяют использовать в классах *copyable*, но не *moveable* типы~~
- ~~● Используются для обхода оптимизации компилятора~~
- Перегрузка у классов *конструктора копирования* и *оператора присваивания копированием* для внедрения семантики перемещений

Вопросы

- Что делает `std::move`?

- превращает параметр в rvalue
- превращает параметр в lvalue
- тоже самое, что и `std::swap`

Вопросы

- Что делает `std::move`?

- превращает параметр в rvalue
- ~~● превращает параметр в lvalue~~
- ~~● тоже самое, что и `std::swap`~~

ИСТОЧНИКИ

- **Becker, T. (2013).** *C++ Rvalue References Explained.*
http://thbecker.net/articles/rvalue_references/section_01.html