

Ravesli [Ravesli](#)

- [Уроки по C++](#)
- [OpenGL](#)
- [SFML](#)
- [Qt5](#)
- [RegExр](#)
- [Ассемблер](#)
- [Купить .PDF](#)





Лямбда-захваты в C++

 [Дмитрий Бушуев](#) |

- [Уроки C++](#)

|

 Обновл. 28 Сен 2020 |

 10729

[↑](#)  9

На этом уроке мы рассмотрим, что такое лямбда-захваты в языке C++, как они работают, какие есть типы и как их использовать.

Оглавление:

1. [Зачем нужен лямбда-захват?](#)
2. [Введение в лямбда-захваты](#)
3. [Суть работы лямбда-захватов](#)
4. [Захваты переменных и const](#)
5. [Захват по значению](#)
6. [Захват по ссылке](#)
7. [Захват нескольких переменных](#)
8. [Захваты по умолчанию](#)
9. [Определение новых переменных в лямбда-захвате](#)
10. [Висячие захваченные переменные](#)
11. [Непреднамеренные копии лямбд](#)
12. [Тест](#)

Зачем нужен лямбда-захват?

На предыдущем уроке мы рассматривали следующий пример:

```
1 #include <algorithm>
2 #include <array>
3 #include <iostream>
4 #include <string_view>
5
6 int main()
7 {
8     std::array<std::string_view, 4> arr{ "apple", "banana", "walnut", "lemon" };
9
10    auto found{ std::find_if(arr.begin(), arr.end(),
11                             [](std::string_view str)
12                             {
13                                 return (str.find("nut") != std::string_view::npos);
14                             }) };
15
16    if (found == arr.end())
17    {
18        std::cout << "No nuts\n";
19    }
20    else
21    {
22        std::cout << "Found " << *found << '\n';
23    }
24
25    return 0;
26 }
```

Давайте изменим его так, чтобы пользователь сам выбирал подстроку для поиска. Это не настолько интуитивно легко, как может показаться на первый взгляд:

```
1 #include <algorithm>
2 #include <array>
3 #include <iostream>
4 #include <string_view>
5 #include <string>
6
7 int main()
8 {
9     std::array<std::string_view, 4> arr{ "apple", "banana", "walnut", "lemon" };
10
11    // Просим пользователя ввести объект для поиска
12    std::cout << "search for: ";
13
14    std::string search{};
15    std::cin >> search;
16
17    auto found{ std::find_if(arr.begin(), arr.end(), [](std::string_view str) {
18        // Ищем значение переменной search, вместо подстроки "nut"
19        return (str.find(search) != std::string_view::npos); // ошибка: Переменная search
```

```
20    }) };
21
22    if (found == arr.end())
23    {
24        std::cout << "Not found\n";
25    }
26    else
27    {
28        std::cout << "Found " << *found << '\n';
29    }
30
31    return 0;
32 }
```

Данный код не скомпилируется. В отличие от вложенных блоков кода, где любой идентификатор, определенный во внешнем блоке, доступен и во внутреннем, [лямбды в языке C++](#) могут получить доступ только к определенным видам идентификаторов: глобальные идентификаторы, объекты, известные во время компиляции и со статической продолжительностью жизни. Переменная `search` не соответствует ни одному из этих требований, поэтому лямбда не может её увидеть. Вот для этого и существует **лямбда-захват** (англ. *«capture clause»*).

Введение в лямбда-захваты

Поле `capture clause` используется для того, чтобы предоставить (косвенно) лямбде доступ к переменным из окружающей области видимости, к которым она обычно не имеет доступ. Всё, что нам нужно для этого сделать, так это перечислить в поле `capture clause` объекты, к которым мы хотим получить доступ внутри лямбды. В нашем примере мы хотим предоставить лямбде доступ к значению переменной `search`, поэтому добавляем её в захват:

```
1  #include <algorithm>
2  #include <array>
3  #include <iostream>
4  #include <string_view>
5  #include <string>
6
7  int main()
8  {
9      std::array<std::string_view, 4> arr{ "apple", "banana", "walnut", "lemon" };
10
11     std::cout << "search for: ";
12
13     std::string search{};
14     std::cin >> search;
15
16     // Захват переменной search
17     auto found{ std::find_if(arr.begin(), arr.end(), [search](std::string_view str) {
```

```
18     return (str.find(search) != std::string_view::npos);
19 }) };
20
21 if (found == arr.end())
22 {
23     std::cout << "Not found\n";
24 }
25 else
26 {
27     std::cout << "Found " << *found << '\n';
28 }
29
30 return 0;
31 }
```

Теперь пользователь сможет выполнить поиск нужного элемента в [массиве](#):

```
search for: nana
Found banana
```

Суть работы лямбда-захватов

Хотя может показаться, будто в вышеприведенном примере наша лямбда напрямую обращается к значению переменной `search` (относящейся к блоку кода функции `main()`), но это не так. Да, лямбды могут выглядеть и функционировать как вложенные блоки, но на самом деле они работают немного по-другому, и при этом существует довольно важное отличие.

Когда выполняется лямбда-определение, то для каждой захватываемой переменной внутри лямбды создается клон этой переменной (с идентичным именем). Данные переменные-клоны инициализируются с помощью переменных из внешней области видимости с тем же именем.

Таким образом, в примере, приведенном выше, при создании объекта лямбды, она получает свою собственную переменную-клон с именем `search`. Эта переменная имеет такое же значение, что и переменная `search` из функции `main()`, поэтому кажется будто мы получаем доступ непосредственно к переменной `search` функции `main()`, но это не так.

Несмотря на то, что эти переменные-клоны имеют одно и то же имя, их тип может отличаться от типа исходной переменной (об этом чуть позже).

Ключевой момент: Переменные, захваченные лямбдой, являются клонами переменных из внешней области видимости, а не фактическими «внешними» переменными.

Для продвинутых читателей: Когда компилятор обнаруживает определение лямбды, он создает для нее определение как для пользовательского объекта. Каждая захваченная переменная становится элементом данных этого объекта. Во время выполнения программы, при обнаружении определения лямбды, создается экземпляр объекта лямбды и в этот момент инициализируются члены лямбды.

Захваты переменных и `const`

По умолчанию переменные захватываются как **константные** значения. Это означает, что при создании лямбды, она захватывает константную копию переменной из внешней области видимости, что означает, что значения этих переменных лямбда изменить не может. В следующем примере мы захватим переменную `ammo` и попытаемся выполнить **декремент**:

```
1  #include <iostream>
2
3  int main()
4  {
5      int ammo{ 10 };
6
7      // Определяем лямбду внутри переменной с именем shoot
8      auto shoot{
9          [ammo]() {
10             // Запрещено, так как переменная ammo была захвачена в виде константной копии
11             --ammo;
12
13             std::cout << "Pew! " << ammo << " shot(s) left.\n";
14         }
15     };
16
17     // Вызов лямбды
18     shoot();
19
20     std::cout << ammo << " shot(s) left\n";
21
22     return 0;
23 }
```

В примере, приведенном выше, когда мы захватываем переменную `ammo`, внутри лямбды создается константная переменная с таким же именем и значением. Мы не можем изменить её, потому что она имеет спецификатор `const`. Подобная попытка изменения приведет к ошибке компиляции.

Захват по значению

Чтобы разрешить изменения значения переменных, которые были захвачены по значению, мы можем пометить лямбду как `mutable`. В данном контексте, **ключевое слово `mutable`** удаляет спецификатор `const` со всех переменных, захваченных по значению:

```
1  #include <iostream>
2
3  int main()
4  {
5      int ammo{ 10 };
6
7      auto shoot{
8          // Добавляем ключевое слово mutable после списка параметров
```

```
9      [ammo]() mutable {
10          // Теперь нам разрешено изменять значение переменной ammo
11          --ammo;
12
13          std::cout << "Pew! " << ammo << " shot(s) left.\n";
14      }
15  };
16
17  shoot();
18  shoot();
19
20  std::cout << ammo << " shot(s) left\n";
21
22  return 0;
23 }
```

Результат выполнения программы:

```
Pew! 9 shot(s) left.
Pew! 8 shot(s) left.
10 shot(s) left
```

Хотя теперь этот код и скомпилируется, но в нем все еще есть логическая ошибка. Какая именно? При вызове лямбда захватила копию переменной `ammo`. Затем, когда лямбда уменьшает значение переменной `ammo` с 10 до 9 и до 8, то, на самом деле, она уменьшает значение копии, а не исходной переменной.

Обратите внимание, что значение переменной `ammo` сохраняется, несмотря на вызовы лямбды.

Захват по ссылке

Подобно тому, как функции могут изменять значения аргументов, передаваемых им по ссылке, мы также можем захватывать переменные по ссылке, чтобы позволить нашей лямбде влиять на значения аргументов.

Чтобы захватить переменную по ссылке, мы должны добавить знак амперсанда (&) к имени переменной, которую хотим захватить. В отличие от переменных, которые захватываются по значению, переменные, которые захватываются по ссылке, не являются константными (если только переменная, которую они захватывают, не является изначально `const`). Если вы предпочитаете передавать аргумент функции по ссылке (например, для типов, не являющихся базовыми), то вместо захвата по значению, предпочтительнее использовать захват по ссылке.

Вот вышеприведенный код, но уже с захватом переменной `ammo` по ссылке:

```
1  #include <iostream>
2
3  int main()
4  {
5      int ammo{ 10 };
6  }
```

```

6
7 auto shoot{
8     // Ключевое слово mutable теперь нам не потребуется
9     [&ammo]() { // &ammo означает, что переменная ammo захватывается по ссылке
10        // Изменения текущей переменной ammo приведут к изменению переменной ammo из блока
11        --ammo;
12
13        std::cout << "Pew! " << ammo << " shot(s) left.\n";
14    }
15};
16
17 shoot();
18
19 std::cout << ammo << " shot(s) left\n";
20
21 return 0;
22 }

```

Результат выполнения программы:

```

Pew! 9 shot(s) left.
9 shot(s) left

```

Теперь давайте воспользуемся захватом по ссылке, чтобы подсчитать, сколько сравнений делает [алгоритм std::sort\(\)](#) при сортировке массива:

```

1 #include <algorithm>
2 #include <array>
3 #include <iostream>
4 #include <string>
5
6 struct Car
7 {
8     std::string make{};
9     std::string model{};
10 };
11
12 int main()
13 {
14     std::array<Car, 3> cars{ { { "Volkswagen", "Golf" },
15                             { "Toyota", "Corolla" },
16                             { "Honda", "Civic" } } };
17
18     int comparisons{ 0 };
19
20     std::sort(cars.begin(), cars.end(),
21        // Захват переменной comparisons по ссылке
22        [&comparisons](const auto& a, const auto& b) {
23            // Мы захватили переменную comparisons по ссылке, а это означает, что мы можем и

```

```
24     ++comparisons;
25
26     // Сортировка машин по марке
27     return (a.make < b.make);
28 });
29
30 std::cout << "Comparisons: " << comparisons << '\n';
31
32 for (const auto& car : cars)
33 {
34     std::cout << car.make << ' ' << car.model << '\n';
35 }
36
37 return 0;
38 }
```

Результат выполнения программы:

```
Comparisons: 2
Honda Civic
Toyota Corolla
Volkswagen Golf
```

Захват нескольких переменных

Мы можем захватить несколько переменных, разделив их запятыми. Мы также можем использовать как захват по значению, так и захват по ссылке:

```
1 int health{ 33 };
2 int armor{ 100 };
3 std::vector<CEntity> enemies{};
4
5 // Захватываем переменные health и armor по значению, а enemies - по ссылке
6 [health, armor, &enemies](){};
```

Захваты по умолчанию

Необходимость явно перечислять переменные для захвата иногда может быть несколько обременительной. Если вы изменяете свою лямбду, то вы можете забыть добавить или удалить захватываемые переменные. К счастью, есть возможность заручиться помощью компилятора для автоматической генерации списка переменных, которые нам нужно захватить.

Захват по умолчанию захватывает все переменные, упомянутые в лямбде. Если используется захват по умолчанию, то переменные, не упомянутые в лямбде, не будут захвачены.

Чтобы захватить все задействованные переменные по значению, используйте `=` в качестве значения для захвата. Чтобы захватить все задействованные переменные по ссылке, используйте `&` в качестве значения

для захвата.

Вот пример использования захвата по умолчанию по значению:

```
1 #include <array>
2 #include <iostream>
3
4 int main()
5 {
6     std::array areas{ 100, 25, 121, 40, 56 };
7
8     int width{};
9     int height{};
10
11     std::cout << "Enter width and height: ";
12     std::cin >> width >> height;
13
14     auto found{ std::find_if(areas.begin(), areas.end(),
15                             [=](int knownArea) { // выполняется захват по умолчанию по
16                                                     return (width * height == knownArea); // потому что они з
17                                                     }) };
18
19     if (found == areas.end())
20     {
21         std::cout << "I don't know this area :\n";
22     }
23     else
24     {
25         std::cout << "Area found :)\n";
26     }
27
28     return 0;
29 }
```

Захваты по умолчанию могут быть смешаны с обычными захватами. Вполне допускается захватить некоторые переменные по значению, а другие — по ссылке, но при этом каждая переменная может быть захвачена только один раз:

```
1 int health{ 33 };
2 int armor{ 100 };
3 std::vector<CEntity> enemies{};
4
5 // Захватываем переменные health и armor по значению, а enemies — по ссылке
6 [health, armor, &enemies](){};
7
8 // Захватываем переменную enemies по ссылке, а все остальные — по значению
9 [=, &enemies](){};
10
11 // Захватываем переменную armor по значению, а все остальные — по ссылке
```

```

12 [&, armor](){};
13
14 // Запрещено, так как мы уже определили захват по ссылке для всех переменных
15 [&, &armor](){};
16
17 // Запрещено, так как мы уже определили захват по значению для всех переменных
18 [=, armor](){};
19
20 // Запрещено, так как переменная armor используется дважды
21 [armor, &health, &armor](){};
22
23 // Запрещено, так как захват по умолчанию должен быть первым элементом в списке захвата
24 [armor, &](){};

```

Определение новых переменных в лямбда-захвате

Допустим, что нам нужно захватить переменную с небольшой модификацией или объявить новую переменную, которая видна только в области видимости лямбды. Мы можем это сделать, определив переменную в лямбда-захвате без указания её типа:

```

1  #include <array>
2  #include <iostream>
3
4  int main()
5  {
6      std::array areas{ 100, 25, 121, 40, 56 };
7
8      int width{};
9      int height{};
10
11     std::cout << "Enter width and height: ";
12     std::cin >> width >> height;
13
14     // Мы храним переменную areas, но пользователь ввел width и height.
15     // Прежде, чем выполнить операцию поиска, мы должны вычислить значение площади (area)
16     auto found{ std::find_if(areas.begin(), areas.end(),
17                             // Объявляем новую переменную, которая видима только для лямбды
18                             // Тип переменной userArea автоматически выведен как тип int
19                             [userArea{ width * height }](int knownArea) {
20                                 return (userArea == knownArea);
21                             }) };
22
23     if (found == areas.end())
24     {
25         std::cout << "I don't know this area :(\n";
26     }
27     else
28     {

```

```
29     std::cout << "Area found :)\n";
30 }
31
32 return 0;
33 }
```

Переменная `userArea` будет рассчитана только один раз: во время определения лямбды. Вычисляемая площадь хранится в объекте лямбды и одинакова для каждого вызова. Если лямбда имеет модификатор `mutable` и изменяет переменную, которая определена в захвате, то исходное значение переменной будет переопределено.

Совет: Инициализируйте переменные в захвате только в том случае, если их значения не являются слишком большими и их тип очевиден. В противном случае, лучше всего определить переменную вне лямбды, а затем захватить её.

Висячие захваченные переменные

Переменные захватываются в точке определения лямбды. Если переменная, захваченная по ссылке, прекращает свое существование до прекращения существования лямбды, то лямбда остается с висячей ссылкой:

```
1  #include <iostream>
2  #include <string>
3
4  // функция возвращает лямбду
5  auto makeWalrus(const std::string& name)
6  {
7      // Захват переменной name по ссылке и возврат лямбды
8      return [&]() {
9          std::cout << "I am a walrus, my name is " << name << '\n'; // неопределенное поведение
10     };
11 }
12
13 int main()
14 {
15     // Создаем новый объект с именем Roofus.
16     // sayName является лямбдой, возвращаемой функцией makeWalrus()
17     auto sayName{ makeWalrus("Roofus") };
18
19     // Вызов лямбды, которую возвращает функция makeWalrus()
20     sayName();
21
22     return 0;
23 }
```

Вызов функции `makeWalrus()` создает временный объект `std::string` из строкового литерала `"Roofus"`. Лямбда в функции `makeWalrus()` захватывает временную строку по ссылке. Данная строка уничтожается при выполнении возврата `makeWalrus()`, но при этом лямбда все еще ссылается на нее. Затем, когда мы

вызываем `sayName()`, происходит попытка доступа к висячей ссылке, что чревато неопределенными результатами.

Обратите внимание, это также происходит, если переменная `name` передается в функцию `makeWalrus()` по значению. Переменная `name` все равно прекратит свое существование в конце работы функции `makeWalrus()`, и лямбда останется с висячей ссылкой.

Предупреждение: Будьте особенно осторожны при захвате переменных по ссылке, особенно при указанном захвате по умолчанию по ссылке. Захваченные переменные должны существовать дольше, чем сама лямбда.

Если мы хотим, чтобы захваченная переменная `name` была валидна, когда используется лямбда, то нам нужно захватить данную переменную по значению (либо явно, либо с помощью захвата по умолчанию по значению).

Непреднамеренные копии лямбд

Поскольку лямбды являются объектами, то их можно копировать. В некоторых случаях это может вызвать проблемы. Рассмотрим следующий код:

```
1  #include <iostream>
2
3  int main()
4  {
5      int i{ 0 };
6
7      // Создаем новую лямбду с именем count
8      auto count{ [i]() mutable {
9          std::cout << ++i << '\n';
10     } };
11
12     count(); // обращаемся к count
13
14     auto otherCount{ count }; // создаем копию count
15
16     // Обращаемся к count и к копии count
17     count();
18     otherCount();
19
20     return 0;
21 }
```

Результат выполнения программы:

```
1
2
2
```

Вместо вывода 1 2 3 программа дважды выводит число 2. Создавая объект `otherCount`, как копию объекта `count`, мы копируем его текущее состояние. Значением переменной `i`, принадлежащей объекту `count`, является 1 и значением переменной `i`, принадлежащей объекту `otherCount`, так же является 1. Поскольку `otherCount` — это копия `count`, то у каждого объекта имеется своя собственная переменная `i`.

Теперь давайте рассмотрим менее очевидный пример:

```
1 #include <iostream>
2 #include <functional>
3
4 void invoke(const std::function<void(void)>& fn)
5 {
6     fn();
7 }
8
9 int main()
10 {
11     int i{ 0 };
12
13     // Выполняем инкремент и выводим на экран локальную копию переменной i
14     auto count{ [i]() mutable {
15         std::cout << ++i << '\n';
16     } };
17
18     invoke(count);
19     invoke(count);
20     invoke(count);
21
22     return 0;
23 }
```

Результат выполнения программы:

```
1
1
1
```

Данный пример демонстрирует возникновение той же проблемы, что и предыдущий пример. Когда с помощью лямбды создается объект `std::function`, то он внутри себя создает копию лямбда-объекта. Таким образом, наш вызов `fn()` фактически выполняется при использовании копии лямбды, а не самой лямбды.

Если нам нужно передать изменяемую лямбду, и при этом мы хотим избежать непреднамеренного копирования, то есть два варианта решения данной проблемы. Один из них — использовать вместо этого лямбду, не содержащую захватов. В примере, приведенном выше, мы могли бы удалить захват и отслеживать наше состояние, используя статическую локальную переменную. Но статические локальные переменные могут быть трудны для отслеживания и делают наш код менее читабельным. Лучший вариант — это с самого начала не допустить возможности копирования нашей лямбды. Но, поскольку мы

не можем повлиять на реализацию `std::function` (или любой другой функции или объекта из Стандартной библиотеки C++), как мы можем это сделать?

К счастью, C++ предоставляет тип `std::ref` (как часть [заголовочного файла](#) `functional`), который позволяет нам передавать обычный тип, как если бы это была ссылка. Обёртывая нашу лямбду в `std::ref` всякий раз, когда кто-либо пытается сделать копию нашей лямбды, он будет делать копию ссылки, а не фактического объекта.

Вот наш обновленный код с использованием `std::ref`:

```
1  #include <iostream>
2  #include <functional>
3
4  void invoke(const std::function<void(void)> &fn)
5  {
6      fn();
7  }
8
9  int main()
10 {
11     int i{ 0 };
12
13     // Выполняем инкремент и выводим на экран локальную копию переменной i
14     auto count{ [i]() mutable {
15         std::cout << ++i << '\n';
16     } };
17
18     // std::ref(count) гарантирует, что count рассматривается, как ссылка.
19     // Таким образом, всё, что пытается скопировать count, фактически является ссылкой
20     // гарантируя тем самым существование только одного объекта count
21     invoke(std::ref(count));
22     invoke(std::ref(count));
23     invoke(std::ref(count));
24
25     return 0;
26 }
```

Результат выполнения программы:

```
1
2
3
```

Обратите внимание, выходные данные не изменяются, даже если `invoke()` принимает `fn()` по значению. `std::function` не создает копию лямбды, если мы используем `std::ref`.

Правило: Стандартные библиотечные функции могут копировать функциональные объекты (напомним, что лямбды принадлежат к категории функциональных объектов). Если вы хотите

использовать лямбду вместе с изменяемыми захваченными переменными, то передавайте их по ссылке с помощью `std::ref`.

Тест

Задание №1

Какие из следующих переменных могут использоваться лямбдой в функции `main()` без их явного захвата?

```
1  int i{};
2  static int j{};
3
4  int getValue()
5  {
6      return 0;
7  }
8
9  int main()
10 {
11     int a{};
12     constexpr int b{};
13     static int c{};
14     static constexpr int d{};
15     const int e{};
16     const int f{ getValue() };
17     static const int g{};
18     static const int h{ getValue() };
19
20     [](){
21         // Попробуйте использовать переменные без их явного захвата
22         a;
23         b;
24         c;
25         d;
26         e;
27         f;
28         g;
29         h;
30         i;
31         j;
32     }();
33
34     return 0;
35 }
```

Ответ №1

Переменная

Использование без явного захвата

- a Нет. Переменная a имеет автоматическую продолжительность.
- b Да. Переменная b используется в константном выражении.
- c Да. Переменная c имеет статическую продолжительность.
- d Да.
- e Да. Переменная e используется в константном выражении.
- f Нет. Значение переменной f зависит от getValue(), что может потребовать запуска программы.
- g Да.
- h Да. Переменная h имеет статическую продолжительность.
- i Да. Переменная i является глобальной переменной.
- j Да. Переменная j доступна в целом файле.

Задание №2

Что выведет на экран следующая программа? Не запускайте код, а выполните его в уме:

```
1 #include <iostream>
2 #include <string>
3
4 int main()
5 {
6     std::string favoriteFruit{ "grapes" };
7
8     auto printFavoriteFruit{
9         [=]() {
10             std::cout << "I like " << favoriteFruit << '\n';
11         }
12     };
13
14     favoriteFruit = "bananas with chocolate";
15
16     printFavoriteFruit();
17
18     return 0;
19 }
```

Ответ №2

Результат выполнения программы:

I like grapes

Лямбда printFavoriteFruit захватывает favoriteFruit по значению. Изменение переменной favoriteFruit в функции main() никак не влияет на изменение переменной favoriteFruit в лямбде.

Задание №3

Мы собираемся написать небольшую игру с квадратами чисел.

Суть игры:

- ➔ Попросите пользователя ввести 2 числа: первое — стартовое число, которое нужно возвести в квадрат, второе — количество чисел, которые нужно возвести в квадрат.
- ➔ Сгенерируйте случайное целое число от 2 до 4 и возведите в квадрат указанное пользователем количество чисел, начиная со стартового.
- ➔ Умножьте каждое возведенное в квадрат число на сгенерированное ранее число (от 2 до 4).
- ➔ Пользователь должен вычислить, какие числа были сгенерированы — он указывает свои предположения.
- ➔ Программа проверяет, угадал ли пользователь число, и, если угадал — удаляет угаданное число из списка.
- ➔ Если пользователь не угадал число, то игра заканчивается, и программа выводит число, которое было ближе всего к окончательному предположению пользователя, но только если последнее предположение не отличалось больше чем на 4 единицы от числа из списка.

Вот первый запуск игры:

```
Start where? 4
How many? 8
I generated 8 square numbers. Do you know what each number is after multiplying
it by 2?
> 32
Nice! 7 number(s) left.
> 72
Nice! 6 number(s) left.
> 50
Nice! 5 number(s) left.
> 126
126 is wrong! Try 128 next time.
```

Разбираемся:

- ➔ Пользователь решил начать с числа 4 и хочет 8 чисел.
- ➔ Квадрат каждого числа будет умножен на 2. Число 2 было выбрано программой случайным образом.
- ➔ Программа сгенерировала 8 квадратов чисел, начиная с числа 4: 16 25 36 49 64 81 100 121.
- ➔ Но при этом каждое число было умножено на 2, поэтому мы получаем следующие числа: 32 50 72 98 128 162 200 242.
- ➔ Теперь пользователь начинает угадывать. Порядок, в котором вводятся догадки, не имеет значения.
- ➔ Число 32 значится в списке.

→ Число 72 значится в списке.

→ Числа 126 нет в списке, поэтому пользователь проиграл. В списке есть число 128, которое отличается не более чем на 4 единицы от предположения пользователя, поэтому его мы и выводим в качестве подсказки.

Вот второй запуск игры:

```
Start where? 1
How many? 3
I generated 3 square numbers. Do you know what each number is after multiplying
it by 4?
> 4
Nice! 2 numbers left.
> 16
Nice! 1 numbers left.
> 36
Nice! You found all numbers, good job!
```

Разбираемся:

- Пользователь решил начать с числа 1 и хочет 3 числа.
- Квадрат каждого числа будет умножен на 4.
- Программа сгенерировала следующие числа: 1 4 9.
- Умножаем их на 4: 4 16 36.
- Пользователь выиграл, угадав все числа.

Вот третий запуск игры:

```
Start where? 2
How many? 2
I generated 2 square numbers. Do you know what each number is after multiplying
it by 4?
> 21
21 is wrong!
```

Разбираемся:

- Пользователь решил начать с числа 2 и хочет 2 числа.
- Квадрат каждого числа умножается на 4.
- Программа сгенерировала следующие числа: 16 36.
- Пользователь выдвигает предположение — 21, и проигрывает. 21 не достаточно близко к любому из оставшихся чисел, поэтому число-подсказка не выводится.

Подсказки:

- Используйте `std::find()` для поиска номера в списке.
- Используйте `std::vector::erase()` для удаления элемента, например:

```
1 auto found{ std::find(/* ... */) };
2
3 // Убедитесь, что элемент был найден
4
5 myVector.erase(found);
```

- Используйте `std::min_element()` и лямбду, чтобы найти число, наиболее близкое к предположению пользователя. `std::min_element()` работает аналогично `std::max_element()` из теста предыдущего урока.
- Используйте `std::abs()` из заголовочного файла `cmath`, чтобы вычислить положительную разницу между двумя числами:

```
1 int distance{ std::abs(5 - 3) }; // 2
```

Ответ №3

```
1 #include <algorithm> // для std::generate(), std::find() и std::min_element()
2 #include <cmath> // для std::abs()
3 #include <ctime>
4 #include <iostream>
5 #include <random>
6 #include <vector>
7
8 using list_type = std::vector<int>;
9
10 namespace config
11 {
12     constexpr int multiplierMin{ 2 };
13     constexpr int multiplierMax{ 4 };
14     constexpr int maximumWrongAnswer{ 4 };
15 }
16
17 int getRandomInt(int min, int max)
18 {
19     static std::mt19937 mt{ static_cast<std::mt19937::result_type>(std::time(nullptr)) };
20
21     return std::uniform_int_distribution{ min, max }(mt);
22 }
23
24 // Генерируем количество чисел, указанное в count, начиная со start*start, и умножаем
25 list_type generateNumbers(int start, int count, int multiplier)
26 {
27     list_type numbers(static_cast<list_type::size_type>(count));
28 }
```

```
29     int i{ start };
30
31     for (auto& number : numbers)
32     {
33         number = ((i * i) * multiplier);
34         ++i;
35     }
36
37     return numbers;
38 }
39
40 // Просим пользователя ввести стартовое число и общее количество чисел, а затем вызываем
41 list_type generateUserNumbers(int multiplier)
42 {
43     int start{};
44     int count{};
45
46     std::cout << "Start where? ";
47     std::cin >> start;
48
49     std::cout << "How many? ";
50     std::cin >> count;
51
52     // Здесь пропущена проверка пользовательского ввода. Все функции подразумевают корректный ввод.
53
54     return generateNumbers(start, count, multiplier);
55 }
56
57 int getUserGuess()
58 {
59     int guess{};
60
61     std::cout << "> ";
62     std::cin >> guess;
63
64     return guess;
65 }
66
67 // Ищем значение guess в numbers и удаляем его.
68 // Возвращаем true, если значение было найдено, в противном случае - возвращаем false
69 bool findAndRemove(list_type& numbers, int guess)
70 {
71     if (auto found{ std::find(numbers.begin(), numbers.end(), guess) };
72         found != numbers.end())
73     {
74         return false;
75     }
76     else
77
```

```
78     {
79         numbers.erase(found);
80         return true;
81     }
82 }
83
84 // Находим значение в numbers, которое ближе всего к guess
85 int findClosestNumber(const list_type& numbers, int guess)
86 {
87     return *std::min_element(numbers.begin(), numbers.end(), [=](int a, int b) {
88         return (std::abs(a - guess) < std::abs(b - guess));
89     });
90 }
91
92 void printTask(list_type::size_type count, int multiplier)
93 {
94     std::cout << "I generated " << count
95               << " square numbers. Do you know what each number is after multiplying it
96               << multiplier << "?\n";
97 }
98
99 // Вызывается, когда пользователь правильно угадывает число
100 void printSuccess(list_type::size_type numbersLeft)
101 {
102     std::cout << "Nice! ";
103
104     if (numbersLeft == 0)
105     {
106         std::cout << "You found all numbers, good job!\n";
107     }
108     else
109     {
110         std::cout << numbersLeft << " number(s) left.\n";
111     }
112 }
113
114 // Вызывается, когда пользователь указывает число, которого нет в numbers
115 void printFailure(const list_type& numbers, int guess)
116 {
117     int closest{ findClosestNumber(numbers, guess) };
118
119     std::cout << guess << " is wrong!";
120
121     if (std::abs(closest - guess) <= config::maximumWrongAnswer)
122     {
123         std::cout << " Try " << closest << " next time.\n";
124     }
125     else
126
```

```
127 {
128     std::cout << '\n';
129 }
130 }
131
132 // Возвращаем false, если игра окончена, в противном случае - возвращаем true
133 bool playRound(list_type& numbers)
134 {
135     int guess{ getUserGuess() };
136
137     if (findAndRemove(numbers, guess))
138     {
139         printSuccess(numbers.size());
140
141         return !numbers.empty();
142     }
143     else
144     {
145         printFailure(numbers, guess);
146         return false;
147     }
148 }
149
150 int main()
151 {
152     int multiplier{ getRandomInt(config::multiplierMin, config::multiplierMax) };
153     list_type numbers{ generateUserNumbers(multiplier) };
154
155     printTask(numbers.size(), multiplier);
156
157     while (playRound(numbers));
158
159     return 0;
160 }
```

Оценить статью:

★★★★★ (43 оценок, среднее: 4,70 из 5)



← [Лямбда-выражения \(анонимные функции\) в C++](#)

[Глава №7. Итоговый тест](#) →

Комментариев: 9



1. *Furxie Fluke:*

[30 августа 2020 в 00:42](#)

Хех, как я понимаю, ответ на третье задание сделан для того чтобы те кто будет пытаться вникнуть в это, поняли как много простых деталей\мелочей\важностей они ещё не понимают? :3

Но конечно здорово разобраться, много интересного понять оттуда можно

```
1  int getNumber()
2  {
3      int number;
4      for (;;)
5      {
6          cin >> number;
7          if (cin.fail())
8          {
9              cin.clear();
10             cin.ignore(32767, '\n');
11         }
12         else
13         {
14             cin.ignore(32767, '\n');
15             return number;
16         }
17     }
18 }
19
20
21
22 int main()
23 {
24     SetConsoleCP(1251);
25     SetConsoleOutputCP(1251);
26     for (;;)
27     {
28         cout << "Введи два числа!\nПервое - начальное число которое нужно возвести в степень\nВторое - количество чисел в последовательности\n";
29         int initialNumber = getNumber();
30         cout << "И второе число, сколько чисел в общем должно быть: ";
31         int numberCount = getNumber();
32
33         srand(static_cast<unsigned int>(time(0)));
34         char r = rand();
35         int randomThing = static_cast<int>(rand() / 10940 + 2);
36
37     }
```

```

38     std::vector<int> numbers;
39     numbers.reserve(8);
40
41
42     for (int tempNumber = initialNumber, iter = 0; iter < numberCount; tempNum
43         numbers.push_back(tempNumber);
44
45     std::for_each(numbers.begin(), numbers.end(), [](auto& a) {a *= a; });
46
47     //for (int& a : numbers)
48     //    cout << "\n " << a;
49
50     std::for_each(numbers.begin(), numbers.end(), [randomThing](auto& a) {a *=
51
52     //for (int& a : numbers)
53     //    cout << '\n' << a;
54     int playerGuess;
55     const int& refPG = playerGuess;
56     auto findIfLamb{ [&refPG](const auto& a) {if (a + 4 >= refPG && a - 4 <=
57
58     cout << '\n' << "Я сгенерировал " << numberCount << " квадратных циферок!
59     for (;;)
60     {
61         cout << "\nПиши догадку: ";
62         playerGuess = getNumber();
63
64         auto result(std::find(numbers.begin(), numbers.end(), playerGuess));
65         auto resultFail(std::find_if(numbers.begin(), numbers.end(), findIfLa
66         if (result != numbers.end())
67         {
68             cout << "\nМолодец, ты угадал! ^^";
69             Sleep(1000);
70             numbers.erase(result);
71         }
72         else
73         {
74             if (resultFail != numbers.end())
75             {
76                 cout << "\nНе угадал! Но был близок! В следующий раз попробуй
77                 Sleep(1000);
78                 break;
79             }
80             else
81             {
82                 cout << "\nНе угадал вовсе.... ты проиграл... \n\n\n";
83                 Sleep(1000);
84                 break;
85             }
86

```



```

87         }
88         if (numbers.size() == 0)
89         {
90             cout << "\n\nумниииичкаа!!! Ты смог! Да! :3\nТы всё угадал, начинай заново!";
91             Sleep(5000);
92             break;
93         }
94     }
95 }

```

[Ответить](#)



2. Валера:

[4 августа 2020 в 17:32](#)

```

1  #include <iostream>
2  #include <ctime>
3  #include <cmath>
4  #include <vector>
5  #include <algorithm>
6
7  int getRandomInt(int min, int max)
8  {
9      static const double fraction = 1.0 / (static_cast<double>(RAND_MAX) + 1.0);
10     return static_cast<int>(rand() * fraction * (max - min + 1) + min);
11 }
12
13 int main()
14 {
15     std::cout << "number:";
16     int num;
17     std::cin >> num;
18
19     std::cout << "count of numbers:";
20     int count;
21     std::cin >> count;
22
23     std::vector<int> numbers(count);
24
25     srand(time(0));
26     rand();
27
28     int mul = getRandomInt(2, 4);
29
30     std::for_each(numbers.begin(), numbers.end(), [mul,&num](int &i)
31     {
32         i = pow(num++, 2) * mul;

```

```
33     }
34 );
35
36 system("cls");
37
38 std::cout << "I generated " << count << " square numbers multiplied by " << m
39
40 bool isEnd(false);
41 int userNum;
42
43 while (!isEnd)
44 {
45     std::cout << "your number:";
46     std::cin >> userNum;
47
48     auto found{ std::find_if(numbers.begin(), numbers.end(), [userNum](const in
49         {
50             if (userNum != i)
51             {
52                 if (abs(userNum - i) <= 4) std::cout << "closest number: " << i <
53                 return false;
54             }
55             return true;
56         }
57     ) };
58     if (found != numbers.end())
59     {
60         --count;
61         numbers.erase(found);
62         std::cout << "Nice! " << count << " number(s) left.\n";
63     }
64     else
65     {
66         isEnd = true;
67         std::cout << "You lose!";
68     }
69     if (count == 0)
70     {
71         isEnd = true;
72         std::cout << "You win!";
73     }
74 }
75 }
```

[Ответить](#)

3. *Vampi:*

[29 июля 2020 в 01:16](#)

Оставлю так же свой вариант:

```
1  #include <iostream>
2  #include <random>
3  #include <cmath>
4  #include <algorithm>
5
6  int get_value()
7  {
8      while (true)
9      {
10         int a;
11         std::cin >> a;
12
13         if (std::cin.fail())
14         {
15             std::cin.clear();
16             std::cin.ignore(32767, '\n');
17             std::cout << "Oops, that input is invalid. Please try again.\n";
18         }
19         else
20         {
21             std::cin.ignore(32767, '\n');
22
23             return a;
24         }
25     }
26 }
27
28 int main()
29 {
30     constexpr int min = 2;
31     constexpr int max = 4;
32     constexpr int max_dist = 5;
33
34     std::cout << "Start where?: ";
35     const int start = get_value();
36
37     std::cout << "How many?: ";
38     const int n = get_value();
39
40     std::vector<int> v_numbers(n);
41
42     std::random_device rd;
43     std::mt19937 rng(rd());
44     std::uniform_int_distribution<int> uni(min, max);
45
46     const auto random_integer = uni(rng);
47 }
```

```

48     std::generate(v_numbers.begin(), v_numbers.end(), [start = start - 1, random_
49                                     {start++; return start * start;});
50
51     std::cout << "I generated " << n << " square numbers. Do you know what each num
52     std::cout << random_integer << "?" << std::endl;
53
54     while (true)
55     {
56         int num;
57         std::cin >> num;
58
59         auto found = std::find(std::begin(v_numbers), std::end(v_numbers), num);
60
61         if (found != std::end(v_numbers) && v_numbers.size() != 1)
62         {
63             v_numbers.erase(found);
64             std::cout << "Nice!" << v_numbers.size() << " numbers left." << std::endl;
65         }
66         else if (found == std::end(v_numbers) && v_numbers.size() > 0)
67         {
68             std::cout << "Nice! You found all numbers, good job!" << std::endl;
69             break;
70         }
71         else
72         {
73             std::cout << num << " is wrong!";
74             auto found_dist = std::min_element(std::begin(v_numbers), std::end(v_
75                                     {return std::abs(x - num) < std::abs(*found_dist - num);});
76             if (std::abs(*found_dist - num) < max_dist)
77             {
78                 std::cout << " Try " << *found_dist << " next time." << std::endl;
79             }
80             else
81             {
82                 std::cout << " Try another number next time." << std::endl;
83             }
84             break;
85         }
86     }
87
88     return EXIT_SUCCESS;
89 }

```

[Ответить](#)



4. *pleb:*

[27 июля 2020 в 10:55](#)

Приму ваши замечания

```
1 #include <iostream>
2 #include <random>
3 #define MAX_RANDOM_DENOMINATOR 10
4 #define MIN_RANDOM_DENOMINATOR 2
5 #define RANGE_NUMBER 4
6 void EnterValue(int& value) { //verify the spelling
7     using namespace std;
8     while (1) {
9         cin >> value;
10        if (cin.fail() || value <= 0) {
11            cout << "you\'ve printed some bullshit, try again\n";
12            cin.clear();
13            cin.ignore(32767, '\n');
14            continue;
15        }
16        else break;
17    }
18 }
19
20 int main()
21 {
22
23     int startNumber, numberQuantity, increaseDenominator;
24
25     std::mt19937 randomGenerator(static_cast<std::mt19937::result_type>(time(NULL)));
26     increaseDenominator = randomGenerator() % (MAX_RANDOM_DENOMINATOR - MIN_RANDOM_DENOMINATOR + 1);
27
28     std::cout << "Enter number to start with: ";
29     EnterValue(startNumber);
30     std::cout << "Now enter the quantity of numbers: ";
31     EnterValue(numberQuantity);
32
33     std::vector<long long> numbers(numberQuantity);
34     std::for_each(numbers.begin(), numbers.end(), [&startNumber, increaseDenominator](long long& i) {
35         i = startNumber * startNumber * increaseDenominator;
36         startNumber++;
37     });
38
39     std::cout << "I generated " << numberQuantity << " square numbers. Do you know if any of them is a prime number?" << endl;
40
41     int checkIfExists;
42     while (numberQuantity-- > 0) {
43         EnterValue(checkIfExists);
44
45         auto found = std::find(numbers.begin(), numbers.end(), checkIfExists);
46         if (found == numbers.end()) {
47             std::cout << "No prime numbers found in the generated numbers." << endl;
48         } else {
49             std::cout << "A prime number found in the generated numbers." << endl;
50         }
51     }
52 }
```

```

48         auto closest = std::min_element(numbers.begin(), numbers.end(), [checkIfExists](int i, int j) {
49             return abs(checkIfExists - i) < abs(checkIfExists - j);
50         });
51
52         std::cout << checkIfExists << " is wrong!";
53         if (abs(*closest - checkIfExists) <= RANGE_NUMBER) std::cout << "Try again!";
54         std::cout << "\n";
55         break;
56     }
57     else {
58         numbers.erase(found);
59         if (numbers.size()) std::cout << "Nice! " << numberQuantity << " numbers left!\n";
60         else break;
61     }
62 }
63 if (numbers.empty()) std::cout << "Nice! You found all numbers, good job!";
}

```

Ответить



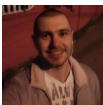
5. *Viktor:*

[5 июля 2020 в 15:04](#)

Здравствуйте) Возможно я просто чего то не понимаю, но во время прочтения главы у меня возник вопрос по поводу `std::min_element`.

Я вообще не понимаю на что влияет лямбда, которую мы передали третьим параметром. Если сама функция возвращает указатель на наименьший элемент из вектора или массива, то какую функцию в таком случае выполняет лямбда, нам достаточно указать всего начало и конец? Если честно, я не понимаю как и в каких случаях использовать третий параметр(В интернете ответа на этот вопрос не нашол. Помогите если кто то знает. Заранее спасибо)

Ответить



1. *Дмитрий Бушуев:*

[7 июля 2020 в 00:11](#)

>>Я вообще не понимаю на что влияет лямбда, которую мы передали третьим параметром. Это в каком моменте?

Ответить



2. *Максим:*

[2 октября 2020 в 22:54](#)

То, что для работы `std::min_element` достаточно указать всего начало и конец массива, ты прав, но отчасти.

В твоём случае функция пройдет по массиву и выведет указатель наименьшего элемента. Это, так сказать, "First version" использования функции.

Есть, еще "Second version" для этой функции, когда можно дописать третий параметр (указатель на функцию или лямбду) как свой вариант сравнения элементов массива.

В данном тесте этот третий параметр сравнивает и находит наиболее "близкий" элемент из массива к введенному значению от пользователя (guess):

```
1 //-----
2 return *std::min_element(numbers.begin(), numbers.end(), [=](int a, int b) {
3     return (std::abs(a - guess) < std::abs(b - guess));
4 });
5 //-----
```

В интернете есть описание работы функции `std::min_element`:

First version:

```
1 //-----
2 template<class ForwardIt>
3 ForwardIt min_element(ForwardIt first, ForwardIt last)
4 {
5     if (first == last) return last;
6
7     ForwardIt smallest = first;
8     ++first;
9     for (; first != last; ++first) {
10         if (*first < *smallest) {
11             smallest = first;
12         }
13     }
14     return smallest;
15 }
16 //-----
```

Second version:

```
1 //-----
2 template<class ForwardIt, class Compare>
3 ForwardIt min_element(ForwardIt first, ForwardIt last, Compare comp)
4 {
5     if (first == last) return last;
6
7     ForwardIt smallest = first;
8     ++first;
9     for (; first != last; ++first) {
10         if (comp(*first, *smallest)) {
11             smallest = first;
12         }
13     }
14 }
```

```

13     }
14     return smallest;
15 }
16 //-----

```

Если нет третьего параметра, происходит сравнение элементов:

```
1 if (*first < *smallest)
```

Если есть третий параметр (указатель или лямбда), функция использует его для сравнения элементов массива, "передавая" ему сравниваемые элементы в качестве аргументов:

```
1 if (comp(*first, *smallest))
```

[Ответить](#)



6. *Михаил:*

[20 июня 2020 в 03:32](#)

Мой вариант. Работает, проверял.

```

1 #include <cstdlib>
2 #include <time.h>
3 #include <vector>
4 #include <iostream>
5 #include <algorithm>
6 using namespace std;
7
8 int main() {
9     srand (time(NULL));
10    int first, count, multiplier {rand() % 2 + 2};
11    cout << "Start where? " << endl;
12    cin >> first;
13    cout << "How many? " << endl;
14    cin >> count;
15    vector<int> numbers;
16    numbers.push_back(first*first*multiplier);
17    int next {first};
18    for (int i = 0; i < count - 1; i++) {
19        next++;
20        numbers.push_back(next*next*multiplier);
21    }
22    cout << "I generated " << count << " square numbers. Do you know what "
23         << "each number is after multiplying it by " << multiplier << "?\n";
24    while (!numbers.empty()) {
25        cout << ">";
26        int guess;
27        cin >> guess;
28        auto found {find(numbers.begin(), numbers.end(), guess)};

```



```

29     if (found != numbers.end()) {
30         numbers.erase(found);
31         cout << "Nice! ";
32     }
33     else {
34         auto found_approx {min_element(numbers.begin(), numbers.end(),
35             [guess](const int &number, const int &min){
36                 return abs(number - guess) < abs(guess - min);
37             }
38         )};
39         if (abs(guess - *found_approx) <= 4) {
40             cout << guess << " is wrong! Try " << *found_approx << " next time. ";
41             numbers.erase(found_approx);
42         }
43         else {
44             cout << guess << " is wrong! ";
45         }
46     }
47     cout << numbers.size() << " number(s) left.\n";
48 }
49 cout << "Finished!\n";
50 return 0;
51 }

```

Ответить



7. Анастасия:

[8 мая 2020 в 17:31](#)

Оставлю здесь своё решение для третьего задания, т.к. оно получилось гораздо короче, чем в ответе, а `std::min_element` мой компилятор почему-то не знает, и лямбду я с ним не использовала.

```

1  #include <iostream>
2  #include <string>
3  #include <ctime>      // для randomize
4  #include <cmath>      // abs()
5  #include <vector>
6
7  using std::cin; using std::cout;
8
9  // получает натуральное число от пользователя
10 unsigned get_number(const std::string& text_for_user)
11 {
12     cout << text_for_user;
13     int answer;
14     cin >> answer;
15     while (cin.fail() || (answer < 0))
16     {

```

```

17     cin.clear();
18     cin.ignore(1000, '\n');
19     cout << "Нужно ввести натуральное число, повторите ввод: ";
20     cin >> answer;
21 }
22 return static_cast<unsigned>(answer);
23 }
24
25 // возвращает число массива, наиболее близкое к заданному и его разницу с числом
26 unsigned find_the_closest_number(const std::vector<unsigned>& numbers_to_guess, co
27 {
28     unsigned the_closest_number{ numbers_to_guess[0] };
29     distance = std::abs(static_cast<int>(the_closest_number) - static_cast<int>(u
30     for (const auto each_number : numbers_to_guess)
31         if (std::abs(static_cast<int>(each_number) - static_cast<int>(users_try))
32         {
33             the_closest_number = each_number;
34             distance = std::abs(static_cast<int>(each_number) - static_cast<int>(
35         }
36     return the_closest_number;
37 }
38
39 int main()
40 {
41     setlocale(LC_CTYPE, "rus"); // подключаем кириллицу для выво
42     srand(static_cast<unsigned>(time(0))); // аналог randomize
43
44     const unsigned start_number{ get_number("Введите стартовое число, которое нуж
45     const unsigned n_number{ get_number("Введите количество чисел, которые нужно
46     const unsigned random_number_from_2_to_4{ static_cast<unsigned>(rand()) % 3 +
47
48     // создаём числа для угадывания
49     std::vector<unsigned> numbers_to_guess{};
50     for (unsigned i = 0; i < n_number; ++i)
51         numbers_to_guess.push_back((start_number + i) * (start_number + i) * rand
52     cout << "Сгенерировано " << n_number << " чисел/число/числа. Попробуйте угада
53
54     while (true) // цикл угадывания очередного ч
55     {
56         unsigned user_tryes_to_guess{ get_number("Ваше предположение: ") };
57         // лямбда, которая возвращает true, если такое число есть в списке
58         auto found{ [&numbers_to_guess](const auto number_to_find)
59             {return std::find(numbers_to_guess.begin(), numbers_to_guess.e
60         if (found(user_tryes_to_guess) < numbers_to_guess.end())
61         {
62             numbers_to_guess.erase(found(user_tryes_to_guess));
63             if (numbers_to_guess.empty())
64             {
65

```

```
66         cout << "Очень здорово! Вы угадали все числа! \n";
67         break;
68     }
69     cout << "Верно! Вы угадали. Осталось отгадать " << numbers_to_guess.s
70 }
71 else
72 {
73     cout << "К сожалению, Вы не угадали число. ";
74     unsigned distance{ 1 };           // разница между названным числом и
75     unsigned the_closest_number{ find_the_closest_number(numbers_to_guess
76     if (distance < 5)
77         cout << "Было бы лучше, если бы Вы назвали " << the_closest_number
78         break;
79     }
80 }
81 cout << "Игра окончена.\n";
82 return 0;
}
```

[Ответить](#)

Добавить комментарий

Ваш E-mail не будет опубликован. Обязательные поля помечены *

Имя *

Email *

Комментарий






☐ Сохранить моё Имя и E-mail. Видеть комментарии, отправленные на модерацию

☐ Получать уведомления о новых комментариях по электронной почте. Вы можете [подписаться](#) без комментирования.

[TELEGRAM](#)  [КАНАЛ](#)

[ПАБЛИК](#) 

ТОП СТАТЬИ

-  [Словарь программиста. Сленг, который должен знать каждый кодер](#)
-  [Урок №1. Введение в программирование](#)
-  [70+ бесплатных ресурсов для изучения программирования](#)
-  [Урок №1: Введение в создание игры «SameGame» на C++/MFC](#)
-  [Урок №4. Установка IDE \(Интегрированной Среды Разработки\)](#)