

Ravesli [Ravesli](#)


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


Урок №104. Указатели на функции

 [Юрий](#) |

- [Уроки C++](#)

|

 Обновл. 2 Дек 2020 |

 63014

[↑](#)  29

На [уроке №80](#) мы узнали, что указатель — это переменная, которая содержит адрес другой переменной. Указатели на функции аналогичны, за исключением того, что вместо обычных переменных, они указывают на функции!

Оглавление:

1. [Указатели на функции](#)
2. [Присваивание функции указателю на функцию](#)
3. [Вызов функции через указатель на функцию](#)
4. [Передача функций в качестве аргументов другим функциям](#)
5. [Параметры по умолчанию в функциях](#)
6. [Указатели на функции и псевдонимы типов](#)
7. [Использование std::function в C++11](#)
8. [Заключение](#)
9. [Тест](#)

Указатели на функции

Рассмотрим следующий фрагмент кода:

```
1 int boo()  
2 {  
3     return 7;  
4 }
```

Идентификатор `boo` — это имя функции. Но какой её тип? Функции имеют свой собственный [l-value](#) тип. В этом случае это тип функции, который возвращает целочисленное значение и не принимает никаких параметров. Подобно переменным, функции также имеют свой адрес в памяти.

Когда функция вызывается (с помощью оператора `()`), точка выполнения переходит к адресу вызываемой функции:

```
1 int boo() // код функции boo() находится в ячейке памяти 002B1050
2 {
3     return 7;
4 }
5
6 int main()
7 {
8     boo(); // переходим к адресу 002B1050
9
10    return 0;
11 }
```

Одной из распространенных ошибок новичков является:

```
1 #include <iostream>
2
3 int boo() // код функции boo() находится в ячейке памяти 002B1050
4 {
5     return 7;
6 }
7
8 int main()
9 {
10    std::cout << boo; // мы хотим вызвать boo(), но вместо этого мы просто выводим boo
11
12    return 0;
13 }
```

Вместо вызова функции `boo()` и вывода возвращаемого значения мы, совершенно случайно, отправили указатель на функцию `boo()` непосредственно в `std::cout`. Что произойдет в этом случае?

Результат на моем компьютере:

002B1050

У вас может быть и другое значение, в зависимости от того, в какой тип данных ваш компилятор решит конвертировать указатель на функцию. Если ваш компьютер не вывел адрес функции, то вы можете заставить его это сделать, конвертируя `boo` в [указатель типа void](#) и отправляя его на вывод:

```
1 #include <iostream>
2
3 int boo() // код функции boo() находится в ячейке памяти 002B1050
4
```

```
5 | {
6 |     return 7;
7 | }
8 |
9 | int main()
10 | {
11 |     std::cout << reinterpret_cast<void*>(boo); // указываем C++ конвертировать функцию
12 |
13 |     return 0;
14 | }
```

Так же, как можно объявить неконстантный указатель на обычную переменную, можно объявить и неконстантный указатель на функцию. Синтаксис создания неконстантного указателя на функцию, пожалуй, один из самых «уродливых» в языке C++:

```
1 | // fcnPtr - это указатель на функцию, которая не принимает никаких аргументов и возвращает
2 | int (*fcnPtr)();
```

В примере, приведенном выше, `fcnPtr` — это указатель на функцию, которая не имеет параметров и возвращает целочисленное значение. `fcnPtr` может указывать на любую другую функцию, соответствующую этому типу.

Скобки вокруг `*fcnPtr` необходимы для соблюдения [приоритета операций](#), в противном случае `int *fcnPtr()` будет интерпретироваться как [предварительное объявление](#) функции `fcnPtr`, которая не имеет параметров и возвращает указатель на целочисленное значение.

Для создания константного указателя на функцию используйте [const](#) после звёздочки:

```
1 | int (*const fcnPtr)();
```

Если вы поместите `const` перед `int`, это будет означать, что функция, на которую указывает указатель, возвращает `const int`.

Присваивание функции указателю на функцию

Указатель на функцию может быть инициализирован функцией (и неконстантному указателю на функцию тоже можно присвоить функцию):

```
1 | int boo()
2 | {
3 |     return 7;
4 | }
5 |
6 | int doo()
7 | {
8 |     return 8;
9 | }
```

```
10 |
11 | int main()
12 | {
13 |     int (*fcnPtr)() = boo; // fcnPtr указывает на функцию boo()
14 |     fcnPtr = doo; // fcnPtr теперь указывает на функцию doo()
15 |
16 |     return 0;
17 | }
```

Одна из распространенных ошибок, которую совершают новички:

```
1 | fcnPtr = doo();
```

Здесь мы фактически присваиваем возвращаемое значение из вызова функции `doo()` указателю `fcnPtr`, чего мы не хотим делать. Мы хотим, чтобы `fcnPtr` содержал адрес функции `doo()`, а не возвращаемое значение из `doo()`. Поэтому скобки здесь не нужны.

Обратите внимание, в указателя на функцию и самой функции должны совпадать тип, параметры и тип возвращаемого значения. Например:

```
1 | // Прототипы функций
2 | int boo();
3 | double doo();
4 | int moo(int a);
5 |
6 | // Присваивание значений указателям на функции
7 | int (*fcnPtr1)() = boo; // ок
8 | int (*fcnPtr2)() = doo; // не ок: тип указателя и тип возврата функции не совпадают!
9 | double (*fcnPtr4)() = doo; // ок
10 | fcnPtr1 = moo; // не ок: fcnPtr1 не имеет параметров, но moo() имеет
11 | int (*fcnPtr3)(int) = moo; // ок
```

В отличие от фундаментальных типов данных, язык C++ **неявно конвертирует** функцию в указатель на функцию, если это необходимо (поэтому вам не нужно использовать оператор адреса `&` для получения адреса функции). Однако, язык C++ не будет неявно конвертировать указатель на функцию в указатель типа `void` или наоборот.

Вызов функции через указатель на функцию

Вы также можете использовать указатель на функцию для вызова самой функции. Есть два способа сделать это. Первый — через явное разыменование:

```
1 | int boo(int a)
2 | {
3 |     return a;
4 | }
5 |
6 | int main()
7 | {
```

```
8   int (*fcnPtr)(int) = boo; // присваиваем функцию boo() указателю fcnPtr
9   (*fcnPtr)(7); // вызываем функцию boo(7), используя fcnPtr
10
11   return 0;
12 }
```

Второй — через неявное разыменование:

```
1  int boo(int a)
2  {
3      return a;
4  }
5
6  int main()
7  {
8      int (*fcnPtr)(int) = boo; // присваиваем функцию boo() указателю fcnPtr
9      fcnPtr(7); // вызываем функцию boo(7), используя fcnPtr
10
11      return 0;
12 }
```

Как вы можете видеть, способ неявного разыменования выглядит так же, как и обычный вызов функции, так как обычные имена функций являются указателями на функции!

Примечание: [Параметры по умолчанию](#) не будут работать с функциями, вызванными через указатели на функции. Параметры по умолчанию обрабатываются во время компиляции (т.е. вам нужно предоставить аргумент для параметра по умолчанию во время компиляции). Однако указатели на функции обрабатываются во время выполнения. Следовательно, параметры по умолчанию не могут обрабатываться при вызове функции через указатель на функцию. В этом случае вам нужно будет явно передать значения для параметров по умолчанию.

Передача функций в качестве аргументов другим функциям

Одна из самых полезных вещей, которую вы можете сделать с указателями на функции — это передать функцию в качестве аргумента другой функции. Функции, используемые в качестве аргументов для других функций, называются **функциями обратного вызова**.

Предположим, что вы пишете функцию для выполнения определенного задания (например, сортировки [массива](#)), но вы хотите, чтобы пользователь мог определить, каким образом выполнять эту сортировку (например, по возрастанию или по убыванию). Рассмотрим более подробно этот случай.

Все алгоритмы сортировки работают по одинаковой схеме: алгоритм выполняет итерацию по списку чисел, сравнивает пары чисел и меняет их местами, исходя из результатов этих сравнений. Следовательно, изменяя алгоритм сравнения чисел, мы можем изменить способ сортировки, не затрагивая остальные части кода.

Вот наша [сортировка методом выбора](#), рассмотренная на соответствующем уроке:

```

1  #include <algorithm> // для std::swap() (используйте <utility>, если поддерживается C++
2
3  void SelectionSort(int *array, int size)
4  {
5      // Перебираем каждый элемент массива
6      for (int startIndex = 0; startIndex < size; ++startIndex)
7      {
8          // smallestIndex - это индекс наименьшего элемента, который мы обнаружили до э
9          int smallestIndex = startIndex;
10
11         // Ищем наименьший элемент среди оставшихся в массиве (начинаем со startIndex+
12         for (int currentIndex = startIndex + 1; currentIndex < size; ++currentIndex)
13         {
14             // Если текущий элемент меньше нашего предыдущего найденного наименьшего э
15             if (array[smallestIndex] > array[currentIndex]) // СРАВНЕНИЕ ВЫПОЛНЯЕТСЯ Э
16                 // то это наш новый наименьший элемент в этой итерации
17                 smallestIndex = currentIndex;
18         }
19
20         // Меняем местами наш стартовый элемент с найденным наименьшим элементом
21         std::swap(array[startIndex], array[smallestIndex]);
22     }
23 }

```

Давайте заменим сравнение чисел на функцию сравнения. Поскольку наша функция сравнения будет сравнивать два целых числа и возвращать логическое значение для указания того, следует ли выполнять замену, то она будет выглядеть следующим образом:

```

1  bool ascending(int a, int b)
2  {
3      return a > b; // условие, при котором меняются местами элементы массива
4  }

```

А вот уже сортировка методом выбора с функцией `ascending()` для сравнения чисел:

```

1  #include <algorithm> // для std::swap() (используйте <utility>, если поддерживается C++
2
3  void SelectionSort(int *array, int size)
4  {
5      // Перебираем каждый элемент массива
6      for (int startIndex = 0; startIndex < size; ++startIndex)
7      {
8          // smallestIndex - это индекс наименьшего элемента, который мы обнаружили до э
9          int smallestIndex = startIndex;
10
11         // Ищем наименьший элемент среди оставшихся в массиве (начинаем со startIndex+
12         for (int currentIndex = startIndex + 1; currentIndex < size; ++currentIndex)
13         {

```

```

14 // Если текущий элемент меньше нашего предыдущего найденного наименьшего э
15 if (ascending(array[smallestIndex], array[currentIndex])) // СРАВНЕНИЕ ВЫП
16 // то это наш новый наименьший элемент в этой итерации
17     smallestIndex = currentIndex;
18 }
19
20 // Меняем местами наш стартовый элемент с найденным наименьшим элементом
21 std::swap(array[startIndex], array[smallestIndex]);
22 }
23 }

```

Теперь, чтобы позволить caller-у решить, каким образом будет выполняться сортировка, вместо использования нашей функции сравнения, мы разрешаем caller-у предоставить свою собственную функцию сравнения! Это делается с помощью указателя на функцию.

Поскольку функция сравнения caller-а будет сравнивать два целых числа и возвращать логическое значение, то указатель на эту функцию будет выглядеть следующим образом:

```
1 bool (*comparisonFcn)(int, int);
```

Мы разрешаем caller-у передавать способ сортировки массива с помощью указателя на функцию в качестве третьего параметра в нашу функцию сортировки.

Вот готовый код сортировки методом выбора с выбором способа сортировки в caller-е (т.е. в функции main()):

```

1 #include <iostream>
2 #include <algorithm> // для std::swap() (используйте <utility>, если поддерживается C++
3
4 // Обратите внимание, третьим параметром является пользовательский выбор выполнения сортировки
5 void selectionSort(int *array, int size, bool (*comparisonFcn)(int, int))
6 {
7     // Перебираем каждый элемент массива
8     for (int startIndex = 0; startIndex < size; ++startIndex)
9     {
10         // bestIndex - это индекс наименьшего/наибольшего элемента, который мы обнаружили
11         int bestIndex = startIndex;
12
13         // Ищем наименьший/наибольший элемент среди оставшихся в массиве (начинаем со startIndex + 1)
14         for (int currentIndex = startIndex + 1; currentIndex < size; ++currentIndex)
15         {
16             // Если текущий элемент меньше/больше нашего предыдущего найденного наименьшего/наибольшего
17             if (comparisonFcn(array[bestIndex], array[currentIndex])) // СРАВНЕНИЕ ВЫП
18                 // то это наш новый наименьший/наибольший элемент в этой итерации
19                 bestIndex = currentIndex;
20         }
21
22         // Меняем местами наш стартовый элемент с найденным наименьшим/наибольшим элементом
23         std::swap(array[startIndex], array[bestIndex]);

```

```
24     }
25 }
26
27 // Вот функция сравнения, которая выполняет сортировку в порядке возрастания (обратите
28 bool ascending(int a, int b)
29 {
30     return a > b; // меняем местами, если первый элемент больше второго
31 }
32
33 // Вот функция сравнения, которая выполняет сортировку в порядке убывания
34 bool descending(int a, int b)
35 {
36     return a < b; // меняем местами, если второй элемент больше первого
37 }
38
39 // Эта функция выводит значения массива
40 void printArray(int *array, int size)
41 {
42     for (int index=0; index < size; ++index)
43         std::cout << array[index] << " ";
44     std::cout << '\n';
45 }
46
47 int main()
48 {
49     int array[8] = { 4, 8, 5, 6, 2, 3, 1, 7 };
50
51     // Сортируем массив в порядке убывания, используя функцию descending()
52     selectionSort(array, 8, descending);
53     printArray(array, 8);
54
55     // Сортируем массив в порядке возрастания, используя функцию ascending()
56     selectionSort(array, 8, ascending);
57     printArray(array, 8);
58
59     return 0;
60 }
```

Результат выполнения программы:

```
8 7 6 5 4 3 2 1
1 2 3 4 5 6 7 8
```

Прикольно, правда? Мы предоставили caller-у возможность контролировать процесс сортировки чисел (caller может определить любые другие функции сравнения):

```
1 bool evensFirst(int a, int b)
2 {
3
```



```
4 // Если a - чётное число, а b - нечётное число, то a идет первым (никакого обмена)
5 if ((a % 2 == 0) && !(b % 2 == 0))
6     return false;
7
8 // Если a - нечётное число, а b - чётное число, то b идет первым (здесь уже требуется обмен)
9 if (!(a % 2 == 0) && (b % 2 == 0))
10    return true;
11
12 // В противном случае, сортируем в порядке возрастания
13 return ascending(a, b);
14 }
15
16 int main()
17 {
18     int array[8] = { 4, 8, 6, 3, 1, 2, 5, 7 };
19
20     selectionSort(array, 8, evensFirst);
21     printArray(array, 8);
22
23     return 0;
24 }
```

Результат выполнения программы:

2 4 6 8 1 3 5 7

Как вы можете видеть, использование указателя на функцию позволяет caller-у «подключить» свой собственный функционал к чему-то, что мы писали и тестировали ранее, что способствует повторному использованию кода! Раньше, если вы хотели отсортировать один массив в порядке убывания, а другой — в порядке возрастания, вам понадобилось бы написать несколько версий сортировки массива. Теперь же у вас может быть одна версия, которая будет выполнять сортировку любым способом, каким вы только захотите!

Параметры по умолчанию в функциях

Если вы позволите caller-у передавать функцию в качестве параметра, то полезным будет предоставить и некоторые стандартные функции для удобства caller-а. Например, в вышеприведенном примере с сортировкой методом выбора, было бы проще установить дефолтный (по умолчанию) способ сравнения чисел. Например:

```
1 // Сортировка по умолчанию выполняется в порядке возрастания
2 void selectionSort(int *array, int size, bool (*comparisonFcn)(int, int) = ascending);
```

В этом случае, до тех пор, пока пользователь вызывает selectionSort() обычно (а не через указатель на функцию), параметр comparisonFcn будет по умолчанию соответствовать функции ascending().

Указатели на функции и псевдонимы типов

Посмотрим правде в глаза — синтаксис указателей на функции уродлив. Тем не менее, с помощью **typedefs** мы можем исправить эту ситуацию:

```
1 typedef bool (*validateFcn)(int, int);
```

Здесь мы определили псевдоним типа под названием `validateFcn`, который является указателем на функцию, которая принимает два значения типа `int` и возвращает значение типа `bool`.

Теперь вместо написания следующего:

```
1 bool validate(int a, int b, bool (*fcnPtr)(int, int)); // фу, какой синтаксис
```

Мы можем написать следующее:

```
1 bool validate(int a, int b, validateFcn pfcn) // вот это другое дело
```

Так гораздо лучше, не правда ли? Однако синтаксис определения самого `typedef` может быть несколько трудным для запоминания. В C++11 вместо `typedef` вы можете использовать `type alias` для создания псевдонима типа указателя на функцию:

```
1 using validateFcn = bool (*)(int, int); // type alias
```

Это уже читабельнее, чем с `typedef`, так как имя псевдонима и его определение расположены на противоположных сторонах от оператора `=`.

Использование `type alias` идентично использованию `typedef`:

```
1 bool validate(int a, int b, validateFcn pfcn) // круто, не так ли?
```

Использование `std::function` в C++11

В C++11 ввели альтернативный способ определения и хранения указателей на функции, который выполняется с использованием **`std::function`**. `std::function` является частью [заголовочного файла](#) `functional` Стандартной библиотеки C++. Для определения указателя на функцию с помощью этого способа вам нужно объявить объект `std::function` следующим образом:

```
1 #include <functional>
2
3 bool validate(int a, int b, std::function<bool(int, int)> fcn); // указываем указатель
```

Как вы можете видеть, тип возврата и параметры находятся в угловых скобках, а параметры еще и внутри круглых скобок. Если параметров нет, то внутренние скобки можно оставить пустыми. Здесь уже более понятно, какой тип возвращаемого значения и какие ожидаемые параметры функции.

Обновим наш предыдущий пример из раздела «Присваивание функции указателю на функцию» текущего урока, но уже с использованием `std::function`:

```
1 #include <iostream>
```

```
2  #include <functional>
3
4  int boo()
5  {
6      return 7;
7  }
8
9  int doo()
10 {
11     return 8;
12 }
13
14 int main()
15 {
16     std::function<int()> fcnPtr; // объявляем указатель на функцию, который возвращает
17     fcnPtr = doo; // fcnPtr теперь указывает на функцию doo()
18     std::cout << fcnPtr(); // вызываем функцию как обычно
19
20     return 0;
21 }
```

Заключение

Указатели на функции полезны, прежде всего, когда вы хотите хранить функции в массиве (или в [структуре](#)) или когда вам нужно передать одну функцию в качестве аргумента другой функции. Поскольку синтаксис объявления указателей на функции является несколько уродливым и подвержен ошибкам, то рекомендуется использовать type alias (или std::function в C++11).

Тест

Задание №1

В этот раз мы попытаемся написать версию базового калькулятора с помощью указателей на функции.

а) Напишите короткую программу, которая просит пользователя ввести два целых числа и выбрать математическую операцию: +, -, * или /. Убедитесь, что пользователь ввел корректный символ математической операции (используйте проверку).

Ответ 1.а)

```
1  #include <iostream>
2
3  int getInteger()
4  {
5      std::cout << "Enter an integer: ";
```

```
6     int a;
7     std::cin >> a;
8     return a;
9 }
10
11 char getOperation()
12 {
13     char op;
14
15     do
16     {
17         std::cout << "Enter an operation ('+', '-', '*', '/'): ";
18         std::cin >> op;
19     }
20     while (op!='+' && op!='-' && op!='*' && op!='/');
21
22     return op;
23 }
24
25 int main()
26 {
27     int a = getInteger();
28     char op = getOperation();
29     int b = getInteger();
30
31     return 0;
32 }
```

б) Напишите функции `add()`, `subtract()`, `multiply()` и `divide()`. Они должны принимать два целочисленных параметра и возвращать целочисленное значение.

Ответ 1.б)

```
1 int add(int a, int b)
2 {
3     return a + b;
4 }
5
6 int subtract(int a, int b)
7 {
8     return a - b;
9 }
10
11 int multiply(int a, int b)
12 {
13     return a * b;
14 }
15
16 int divide(int a, int b)
```

```
17 | {  
18 |     return a / b;  
19 | }
```

с) Создайте typedef с именем `arithmeticFcn` для указателя на функцию, которая принимает два целочисленных параметра и возвращает целочисленное значение.

Ответ 1.с)

```
1 | typedef int (*arithmeticFcn)(int, int);
```

д) Напишите функцию с именем `getArithmeticFcn()`, которая принимает символ выбранного математического оператора и возвращает соответствующую функцию в качестве указателя на функцию.

Ответ 1.д)

```
1 | arithmeticFcn getArithmeticFcn(char op)  
2 | {  
3 |     switch (op)  
4 |     {  
5 |         default: // функцией по умолчанию будет add()  
6 |         case '+': return add;  
7 |         case '-': return subtract;  
8 |         case '*': return multiply;  
9 |         case '/': return divide;  
10 |     }  
11 | }
```

е) Добавьте в функцию `main()` вызов функции `getArithmeticFcn()`.

Ответ 1.е)

```
1 | #include <iostream>  
2 |  
3 | int main()  
4 | {  
5 |     int a = getInteger();  
6 |     char op = getOperation();  
7 |     int b = getInteger();  
8 |  
9 |     arithmeticFcn fcn = getArithmeticFcn(op);  
10 |     std::cout << a << ' ' << op << ' ' << b << " = " << fcn(a, b) << '\n';  
11 |  
12 |     return 0;  
13 | }
```

ф) Соедините все части вместе.

Полная программа

```
1  #include <iostream>
2
3  int getInteger()
4  {
5      std::cout << "Enter an integer: ";
6      int a;
7      std::cin >> a;
8      return a;
9  }
10
11 char getOperation()
12 {
13     char op;
14
15     do
16     {
17         std::cout << "Enter an operation ('+', '-', '*', '/'): ";
18         std::cin >> op;
19     }
20     while (op!='+' && op!='-' && op!='*' && op!='/');
21
22     return op;
23 }
24
25 int add(int a, int b)
26 {
27     return a + b;
28 }
29
30 int subtract(int a, int b)
31 {
32     return a - b;
33 }
34
35 int multiply(int a, int b)
36 {
37     return a * b;
38 }
39
40 int divide(int a, int b)
41 {
42     return a / b;
43 }
44
45 typedef int (*arithmeticFcn)(int, int);
46
47 arithmeticFcn getArithmeticFcn(char op)
48 {
49     switch (op)
```

```
50 {
51     default: // функцией по умолчанию будет add()
52     case '+': return add;
53     case '-': return subtract;
54     case '*': return multiply;
55     case '/': return divide;
56 }
57 }
58
59 int main()
60 {
61     int a = getInteger();
62     char op = getOperation();
63     int b = getInteger();
64
65     arithmeticFcn fcn = getArithmeticFcn(op);
66     std::cout << a << ' ' << op << ' ' << b << " = " << fcn(a, b) << '\n';
67
68     return 0;
69 }
```

Задание №2

Теперь давайте изменим программу, которую мы написали в 1-м задании, чтобы переместить логику из `getArithmeticFcn` в массив.

а) Создайте структуру с именем `arithmeticStruct`, которая имеет два члена: математический оператор типа `char` и указатель на функцию `arithmeticFcn`.

Ответ 2.а)

```
1 struct arithmeticStruct
2 {
3     char op;
4     arithmeticFcn fcn;
5 };
```

б) Создайте статический глобальный массив `arithmeticArray`, используя структуру `arithmeticStruct`, который будет инициализирован каждой из 4-х математических операций.

Ответ 2.б)

```
1 // Версия до C++11:
2 static arithmeticStruct arithmeticArray[] = {
3     { '+', add },
4     { '-', subtract },
5     { '*', multiply },
6     { '/', divide }
7 };
```

```
8
9 // Версия C++11 с использованием uniform-инициализации
10 static arithmeticStruct arithmeticArray[] {
11     { '+', add },
12     { '-', subtract },
13     { '*', multiply },
14     { '/', divide }
15 };
```

с) Измените `getArithmeticFcn` для выполнения цикла по массиву и возврата соответствующего указателя на функцию.

Подсказка: Используйте [цикл foreach](#).

Ответ 2.с)

```
1 arithmeticFcn getArithmeticFcn(char op)
2 {
3     for (auto &arith : arithmeticArray)
4     {
5         if (arith.op == op)
6             return arith.fcn;
7     }
8
9     return add; // функцией по умолчанию будет add()
10 }
```

д) Соедините все части вместе.

Полная программа

```
1 #include <iostream>
2
3 int getInteger()
4 {
5     std::cout << "Enter an integer: ";
6     int a;
7     std::cin >> a;
8     return a;
9 }
10
11 char getOperation()
12 {
13     char op;
14
15     do
16     {
17         std::cout << "Enter an operation ('+', '-', '*', '/'): ";
18         std::cin >> op;
19     } while (op != '+' && op != '-' && op != '*' && op != '/');
```



```
20
21     return op;
22 }
23
24 int add(int a, int b)
25 {
26     return a + b;
27 }
28
29 int subtract(int a, int b)
30 {
31     return a - b;
32 }
33
34 int multiply(int a, int b)
35 {
36     return a * b;
37 }
38
39 int divide(int a, int b)
40 {
41     return a / b;
42 }
43
44 typedef int(*arithmeticFcn)(int, int);
45
46 struct arithmeticStruct
47 {
48     char op;
49     arithmeticFcn fcn;
50 };
51
52 static arithmeticStruct arithmeticArray[] {
53     { '+', add },
54     { '-', subtract },
55     { '*', multiply },
56     { '/', divide }
57 };
58
59
60 arithmeticFcn getArithmeticFcn(char op)
61 {
62     for (auto &arith : arithmeticArray)
63     {
64         if (arith.op == op)
65             return arith.fcn;
66     }
67
68
```

```

69     return add; // функцией по умолчанию будет add()
70 }
71
72 int main()
73 {
74     int a = getInteger();
75     char op = getOperation();
76     int b = getInteger();
77
78     arithmeticFcn fcn = getArithmeticFcn(op);
79     std::cout << a << ' ' << op << ' ' << b << " = " << fcn(a, b) << '\n';
80
81     return 0;
82 }

```

Оценить статью:

★★★★★ (174 оценок, среднее: 4,83 из 5)



[← Урок №103. Параметры по умолчанию](#)



[Урок №105. Стек и Куча](#) →

Комментариев: 29



1. Alex:

[20 марта 2020 в 21:35](#)

Не понял. А зачем делать массив static? Для того чтобы область видимости ограничивалась только этим файлом?

[Ответить](#)



1. Артурка:

[29 октября 2020 в 01:08](#)

Ну в данном случае static несет в себе именно этот посыл. Просто для того что бы другие юниты трансляции не имели доступа к этой переменной. С таким же успехом можно его объявить спецификатором const вместо static.

[Ответить](#)



2. *Бауртаа:*

[10 марта 2020 в 21:22](#)

Написать функцию, которая считает количество слов в переданной строке. Слова отделены друг от друга пробелами и знаками препинания (.,!?-). Учтите, что эти знаки могут идти друг за другом. К примеру, «Привет ... Мир!» – здесь 2 слова.

[Ответить](#)



3. *Даниил:*

[9 марта 2020 в 11:36](#)

Объясните пожалуйста про псевдоним в тесте.(1с) тут просто не недопонимания у меня

[Ответить](#)



4. *Peter:*

[18 января 2020 в 17:44](#)

Где я провтыкал с указателем? Подскажите..

```
1  #include <iostream>
2
3  using namespace std;
4
5  void d(int &a, bool (*how)()) {
6      if (how()==true)
7          a++;
8      else
9          a--;
10 }
11
12 bool minus() { return false; }
13 bool plus() { return true; }
14
15 int main()
16 {
17     int a(5);
18     d(a, minus);
19     return 0;
20 }
```

[Ответить](#)



1. *Andrew Gulenko:*

[6 сентября 2020 в 16:03](#)

Все нормально с указателями. Переименуй функции `minus()` и `plus()`. Потому что такие названия для функций, являются неоднозначными в некоторых компиляторах.

```
1  #include <iostream>
2  using namespace std;
3
4  void d(int &a, bool(*how)() ) {
5      if (how()==true)
6          a++;
7      else
8          a--;
9  }
10
11 bool minus_func() { return false; }
12 bool plus_func() { return true; }
13
14 int main()
15 {
16     int a(5);
17     d(a, minus_func);
18     return 0;
19 }
```

[Ответить](#)



5. *Cv287:*

[21 декабря 2019 в 17:31](#)

Спасибо огромное за ваш непомерно большой труд! Ваша коллекция статей — это один из лучших русскоязычных ресурсов для изучения C++. Все очень удобно и доступно, UX у меня сложился очень хороший, еще раз, большое спасибо!

[Ответить](#)

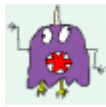
R

1. *Юрий:*

[22 декабря 2019 в 21:57](#)

Очень приятно, что Равесли вам помог, за UX отдельное спасибо))

[Ответить](#)



6. *Kinonik:*

[3 ноября 2019 в 08:54](#)

Как такое величественное в книге за час изучить? XD
Огромное спасибо автору за все уроки, они просто прекрасны!

[Ответить](#)



1. *Юрий:*

[3 ноября 2019 в 10:57](#)

Пожалуйста))

[Ответить](#)



7. *Уа:*

[30 июля 2019 в 15:35](#)

Чем дальше в лес, тем злее волки 😞 В голове каша от всех этих указателей, ссылок, указателей на ссылки, указателей на указатели

[Ответить](#)



1. *Алексей:*

[30 августа 2019 в 11:22](#)

Вчера тоже голова была кругом, лучше еще раз перечитать и покодировать ссылки, указатели на переменные и функции.

Достаточно все просто, брак опыта и только.

Попытаюсь объяснить, что тут происходит.

Смотри — мы инициализировали функции на принятие двух переменных и оператора.

```
1  int getInteger()
2  {
3      std::cout << "Enter an integer: ";
4      int a;
5      std::cin >> a;
6      system("cls");
7      return a;
8  }
9
10 char getOperation()
11 {
12     char op;
13
14     do
15     {
16         std::cout << "Enter an operation ('+', '-', '*', '/'): ";
17         std::cin >> op;
18         system("cls");
19     } while (op != '+' && op != '-' && op != '*' && op != '/');
20
21     return op;
22 }
```

Вот, далее мы инициализируем функции на математические вычисления. Называем соответственно.

```
1 int add(int a, int b)
2 {
3     return a + b;
4 }
5
6 int subtract(int a, int b)
7 {
8     return a - b;
9 }
10
11 int multiply(int a, int b)
12 {
13     return a * b;
14 }
15
16 int divide(int a, int b)
17 {
18     return a / b;
19 }
```

Все просто. Тут мы приходим к инициализации функции, ссылки на функции через typedef, она в свою очередь работает с двумя целочисловыми переменными, также как и те 4 функции, на мат. операции.

```
1 typedef int(*arithmeticFcn)(int, int);
```

Далее инициализируем структуру для удобства, в которой есть наш оператор и ссылка на функцию(функции мат. вычислений).

После этого инициализация массива через эту структуру. Определенному оператору, определенная функция.

Тут самое интересное — передача данных другой функции.

```
1 arithmeticFcn getArithmeticFcn(char op)
2 {
3     for (auto &arith : arithmeticArray)
4     {
5         if (arith.op == op)
6             return arith.fcn;
7     }
8
9     return add; // функцией по умолчанию будет add
10 }
```

Тут foreach, в котором идет объявление ссылки на переменную, для быстрого действия, которая принимает каждое значение массива и проверяется ввод — найден, держи функцию.

В `main()` идет выполнения нашего ввода и передача данных с действий описанных выше и передача в `fcn`.

`fcn(a, b)` просто принимает ввод и идет вычисление.

[Ответить](#)



1. *Lore:*

[27 марта 2020 в 20:22](#)

Почему мы во втором задании, когда возвращаем при нахождении `op` `arith.fcn`?

допустим `arith` сразу нашёл `op` (`op`) под строкой 0 (т.е. `+`) и вот он сравнил и оказалось что ввели и в правду `+` и он возвращает `return arith.fcn`;
и на что он указывает и что он означает ?

[Ответить](#)



1. *Ростислав:*

[18 августа 2020 в 13:47](#)

Он указывает как раз таки на функцию `add()`



8. *deb:*

[18 мая 2019 в 11:34](#)

Спасибо за содержательный урок! Особенно понравились `typedef`, `<functional>` и `alias`

[Ответить](#)

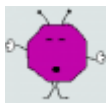


1. *Юрий:*

[6 июня 2019 в 15:07](#)

Пожалуйста. Используйте на здоровье 😊

[Ответить](#)



9. *Александр:*

[1 марта 2019 в 12:15](#)

@@@

Все алгоритмы сортировки работают по одинаковой схеме: алгоритм сортировки выполняет итерацию по списку чисел, сравнивает пары чисел и меняет их местами, исходя из результатов этих сравнений.

@@@

Я бы советовал эту строку убрать или скорректировать. Это ведь неправда 😊

По такой схеме работают сортировки, основанные на сравнении и обмене. Есть сортировки, основанные на сравнении, но в которых нет обменов. Есть сортировки, НЕ основанные на сравнении.

Например, сортировка слиянием — сравнения есть, обменов нет

некоторые реализации "быстрой сортировки" — нет сравнения между элементами (хотя есть сравнения в принципе), обменов может не быть

Сортировка подсчетом (а так же корзинная и радикас) — нет сравнений вообще

Сортировка вставками — нет сравнений между элементами (но есть сравнения вообще), нет обменов

[Ответить](#)



10. *kmish:*

[14 февраля 2019 в 15:09](#)

По 2b тоже не дано пояснений достаточных в предыдущих уроках, чтобы выполнить. Нигде (в предыдущих уроках) не был показан синтаксис, как заполнять массив типа Структура и, что он получается двумерным, по крайней мере из ответа пришел к такому выводу.

[Ответить](#)



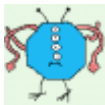
1. *Юрий:*

[14 февраля 2019 в 17:08](#)

Массивы, область видимости (глобальная, локальная), структуры, инициализация, даже математические операторы — мы рассматривали? Рассматривали! В соответствующих уроках.

Понимаешь, программирование — это не скопировал/вставил. Это не 2 минуты и готово. Это не "покажите мне пошагово, построочно, посимвольно, попиксельно, со всеми существующими и несуществующими нюансами, подробностями, деталями, вариациями, способами использования и т.д.". Программирование — это думать, очень много думать, анализировать, пробовать, учиться. Если вы к этому не готовы, то что я могу сделать?

[Ответить](#)



1. *kmish:*

[14 февраля 2019 в 17:20](#)

А я не про копировать-вставить говорю, а про синтаксис. Причем тут думать, если синтаксис не давался? Как я могу пониманием прийти вот к этому?:

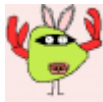
```
1 static arithmeticStruct arithmeticArray[] = {
2     { '+', add }, //нигде не говорилось как заполнить Структуру
3     { '-', subtract },
4     { '*', multiply },
```



```
5 | { '/', divide }
6 | };
```

Это должно было быть прокомментировано в уроке о массивах или структурах. Это не камень в огород, а мои замечания. После просмотра ответов я это понял синтаксис, но его нельзя получить "пониманием". Синтаксис — это правила, а не выводы. А правила должны быть описаны.

[Ответить](#)



1. *Михаил:*

[8 августа 2019 в 08:30](#)

Вполне возможно, что этот синтаксис и не разбирался в данном пособии. Хотя не факт, я не проверял просто. Но кто мешает Вам написать 8 строк и присвоить эти значения по отдельности каждому полю? Потом посмотреть в ответ и уяснить для себя, что возможен и такой синтаксис?

Нет в мире ничего идеального и данное пособие, скорее всего, тоже страдает огрехами. Хотя более полного БЕСПЛАТНОГО труда я не видел! И огромное спасибо Юрию за его бескорыстное вложение сил и средств!

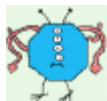
Но, что самое смешное, Вы пытаетесь предъявить претензии переводчику, а не автору!



2. *Юрий:*

[10 августа 2019 в 16:28](#)

Согласен с вашими словами))



11. *kmish:*

[14 февраля 2019 в 14:21](#)

В данном уроке ничего не сказано о пункте D задания 1, а конкретнее про "возвращение функцией указателя на функцию". Синтаксис `int* getArithmeticFcn(char буква);` не работает для указателя и только посмотрев ответ стало понятно, что нужно `typedef` использовать для типа возврата. А если я не хочу использовать `typedef`? Как тогда задать тип возврата указатель на функцию? В уроке синтаксис тоже не указан.

В общем не совсем полноценно все объяснено. Прошу пояснить перечисленные вопросы.

[Ответить](#)



12. *Владимир:*

[8 декабря 2018 в 16:09](#)

```
1 | #include <iostream>
2 | #include <stdint.h>
```

```
3  #include <array>
4
5  using std::cout;
6  using std::cin;
7  using arithmeticFcn = int32_t (*)(int32_t, int32_t);
8
9  //Ввод и проверка ввода целочисленных значений;
10 int32_t inputIntControl()
11 {
12     int32_t value;
13
14     cout << "Enter an integer value: ";
15     while (true)
16     {
17         cin >> value;
18         if (cin.fail())
19         {
20             cin.clear();
21             cin.ignore(25000, '\n');
22
23             cout << "Input failure. Too high or unknown value. Try again\n";
24
25             continue;
26         }
27         else
28         {
29             cin.ignore(25000, '\n');
30             return value;
31         }
32     }
33 }
34
35
36 //Ввод и проверка ввода оператора;
37 char inputOperControl()
38 {
39     while (true)
40     {
41         cout << "Enter an operator (+, -, / or *)\n";
42
43         char oper;
44         cin >> oper;
45
46         if (cin.fail())
47         {
48             cin.clear();
49             cin.ignore(25000, '\n');
50
51
```

```
52         cout << "Input failure. Unknown operator\n";
53
54         continue;
55     }
56     else
57     {
58         cin.ignore(25000, '\n');
59
60         if ((oper == '+') || (oper == '-') || (oper == '*') || (oper == '/'))
61             else cout << "Input failure. Unknown operator\n";
62     }
63 }
64 }
65
66 int32_t add(int32_t a, int32_t b)
67 {
68     return (a + b);
69 }
70
71 int32_t subtract(int32_t a, int32_t b)
72 {
73     return (a - b);
74 }
75
76 int32_t multiply(int32_t a, int32_t b)
77 {
78     return (a * b);
79 }
80
81 int32_t divide(int32_t a, int32_t b)
82 {
83     if (b) return(a / b);
84     else cout << "Error. Can't divide on 0\n";
85
86     exit(-1);
87 }
88
89 struct arithmeticStruct
90 {
91     char oper;
92     arithmeticFcn function;
93 };
94
95 static std::array<arithmeticStruct, 4> arithmeticArray = //Для разнообразия
96 {
97     '+', add,
98     '-', subtract,
99     '*', multiply,
100
```

```

101     '/', divide
102 };
103
104 arithmeticFcn getArithmeticFcn(char opr)
105 {
106     for (const auto &i : arithmeticArray)
107     {
108         if (i.oper == opr) return i.function;
109     }
110
111     return nullptr;
112 }
113
114 int main()
115 {
116
117     int32_t a{ inputIntControl() }, b{ inputIntControl() };
118
119     char oper{ inputOperControl() };
120
121     arithmeticFcn func{ getArithmeticFcn(oper) };
122     if (!func)
123     {
124         cout << "Critical error. Unknown arithmetic function.\n";
125         exit(-2);
126     }
127
128     cout << a << oper << b << '=' << func(a, b) << '\n';
129
130     return 0;
131 }

```

[Ответить](#)13. *Anastasia:*[16 августа 2018 в 11:54](#)

Как в данном задании записать массив через std::array? Т.к. ругается, что "слишком много инициализаторов" (с 3-й строчки).

```

1 static std::array<arithmeticStruct, 4> arithmeticArray {
2     { '+', add },
3 { '-', subtract },
4 { '*', multiply },
5 { '/', divide }};

```

[Ответить](#)



1. *Алексей:*

[30 августа 2019 в 12:11](#)

```
1 | static std::array<arithmeticStruct, 4> arithmeticArray{ '+', add, '-', subtra
```

[Ответить](#)



14. *Petr:*

[11 июля 2018 в 12:16](#)

А зачем в строке (62) "for (auto &arith : arithmeticArray)" использовать "&" ведь и без него будет работать "for (auto arith : arithmeticArray)".

[Ответить](#)

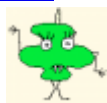


1. *Petr:*

[11 июля 2018 в 12:27](#)

Забыл добавить, что при таком объёме данных ссылку нет смысла использовать. там нет таких затрат память, что бы ссылка хоть как то помогала.

[Ответить](#)



1. *Алексей:*

[30 августа 2019 в 12:20](#)

На самом деле — соглашусь.

Суть в том, что входит все в привычку, образуется условный рефлекс и потом, когда будет этих данных много.

Скажем пользователь и пароль для системы, а это уже тысячи, забыл поставить, ибо в таких маленьких не ставил и пиши пропало.

[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.

Отправить комментарий

[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 \(Интегрированной Среды Разработки\)](#)

- [Ravesli](#)
- - [О проекте/Контакты](#) -
- - [Пользовательское Соглашение](#) -
- - [Все статьи](#) -
- Copyright © 2015 - 2020