

Ravesli [Ravesli](#)


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


## Урок №92. Указатели типа void

 [Юрий](#) |

- [Уроки C++](#)

|

 Обновл. 17 Сен 2020 |

 74089

[1](#)  [7](#)

**Указатель типа void** (или «*общий указатель*») — это специальный тип [указателя](#), который может указывать на объекты любого типа данных! Объявляется он как обычный указатель, только вместо типа данных используется **ключевое слово [void](#)**:

```
1 void *ptr; // ptr - это указатель типа void
```

Указатель типа void может указывать на объекты любого типа данных:

```
1 int nResult;  
2 float fResult;  
3  
4 struct Something  
5 {  
6     int n;  
7     float f;  
8 };  
9  
10 Something sResult;  
11  
12 void *ptr;  
13 ptr = &nResult; // допустимо  
14 ptr = &fResult; // допустимо  
15 ptr = &sResult; // допустимо
```

Однако, поскольку указатель типа `void` сам не знает, на объект какого типа он будет указывать, разыменовать его напрямую не получится! Вам сначала нужно будет явно преобразовать указатель типа `void` с помощью [оператора `static\_cast`](#) в другой тип данных, а затем уже его разыменовать:

```
1 #include <iostream>
2
3 int main()
4 {
5     int value = 7;
6     void *voidPtr = &value;
7
8     //std::cout << *voidPtr << std::endl; // запрещено: нельзя разыменовать указатель
9
10    int *intPtr = static_cast<int*>(voidPtr); // однако, если мы конвертируем наш указ
11    std::cout << *intPtr << std::endl; // то мы сможем его разыменовать, будто бы это
12
13    return 0;
14 }
```

Результат выполнения программы:

7

Возникает вопрос: «Если указатель типа `void` сам не знает, на что он указывает, то как мы тогда можем знать, в какой тип данных его следует явно конвертировать с помощью оператора `static_cast`?». Никак, это уже на ваше усмотрение, вам самим придется выбрать нужный тип. Например:

```
1 #include <iostream>
2
3 enum Type
4 {
5     INT,
6     DOUBLE,
7     CSTRING
8 };
9
10 void printValue(void *ptr, Type type)
11 {
12     switch (type)
13     {
14         case INT:
15             std::cout << *static_cast<int*>(ptr) << '\n'; // конвертируем в указатель
16             break;
17         case DOUBLE:
18             std::cout << *static_cast<double*>(ptr) << '\n'; // конвертируем в указате
19             break;
20         case CSTRING:
21             std::cout << static_cast<char*>(ptr) << '\n'; // конвертируем в указатель
22             // std::cout знает, что char* следует обрабатывать как строку C-style.
```

```
23      // Если бы мы разыменовали результат (целое выражение), то тогда бы вывелс
24      break;
25  }
26 }
27
28 int main()
29 {
30     int nValue = 7;
31     double dValue = 9.3;
32     char szValue[] = "Jackie";
33
34     printValue(&nValue, INT);
35     printValue(&dValue, DOUBLE);
36     printValue(szValue, CSTRING);
37
38     return 0;
39 }
```

Результат выполнения программы:

```
7
9.3
Jackie
```

Указателям типа void можно присвоить нулевое значение:

```
1 void *ptr = 0; // ptr - это указатель типа void, который сейчас является нулевым
```

Хотя некоторые компиляторы позволяют удалять указатели типа void, которые указывают на динамически выделенную память, делать это не рекомендуется, так как результаты могут быть неожиданными.

Также не получится выполнить адресную арифметику с указателями типа void, так как для этого требуется, чтобы указатель знал размер объекта, на который он указывает (для выполнения корректного инкремента/декремента). Также нет такого понятия, как ссылка на void.

## Заключение

В общем, использовать указатели типа void рекомендуется только в самых крайних случаях, когда без этого не обойтись, так как с их использованием проверку типов данных ни вам, ни компилятору выполнить не удастся. А это, в свою очередь, позволит вам случайно сделать то, что не имеет смысла, и компилятор на это жаловаться не будет. Например:

```
1 int nResult = 7;
2 printResult(&nResult, CSTRING);
```

Здесь компилятор промолчит. Но что будет в результате? Непонятно!

Хотя код, приведенный выше, кажется аккуратным способом заставить одну функцию обрабатывать несколько типов данных, в языке C++ есть гораздо лучший способ сделать то же самое (через перегрузку функций), в котором сохраняется проверка типов для предотвращения неправильного использования. Также для обработки нескольких типов данных можно использовать шаблоны, которые обеспечивают хорошую проверку типов (но об этом уже на следующих уроках).

Если вам все же придется использовать указатель типа void, то убедитесь, что нет лучшего (более безопасного) способа сделать то же самое, но с использованием других механизмов языка C++!

## Тест

В чём разница между нулевым указателем и указателем типа void?

### Ответ

Указатель типа void — это указатель, который может указывать на объект любого типа данных, но он сам не знает, какой это будет тип. Для разыменования указатель типа void должен быть явно преобразован с помощью оператора `static_cast` в другой тип данных. Нулевой указатель — это указатель, который не указывает на адрес. Указатель типа void может быть нулевым указателем.

Оценить статью:

★★★★★ (221 оценок, среднее: 4,94 из 5)



← [Урок №91. Цикл foreach](#)

[Урок №93. Указатели на указатели](#) →



## Комментариев: 7



1. Анастасия:  
[30 июня 2019 в 18:14](#)

а я не поняла, почему со строковым типом данным работать на так, как с остальными? Почему

```
1 | printValue(szValue, CSTRING);
```

а не

```
1 | printValue(&szValue, CSTRING);
```

?

[Ответить](#)1. *Anton:*[27 июля 2019 в 17:48](#)

Потому, что строка — это массив значений типа `char`, а массивы, как мы знаем, по умолчанию передаются по ссылке, поэтому указывать `&` нет необходимости.

[Ответить](#)1. *Анастасия:*[28 июля 2019 в 17:16](#)

Спасибо за Ваш ответ! Понятно.

[Ответить](#)2. *Ivan:*[6 декабря 2018 в 15:57](#)

```
1 | printResult(&nResult, CSTRING);
```

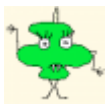
в последнем примере имеется ввиду вызов

```
1 | void printValue(void *ptr, Type type)
```

из примера выше?

[Ответить](#)1. *somebox:*[30 мая 2019 в 23:20](#)

Я вот тоже с ходу не понял, что это за кусок кода.

[Ответить](#)1. *Алексей:*[21 августа 2019 в 12:04](#)

```
1 | char szValue[] = "Jackie";
```

Это ведь массив. Ранее мы изучали указатели на массивы и тд и тп.

Надо просто повторить.

[Ответить](#)1. *Алексей:*[21 августа 2019 в 12:09](#)

Переменная array содержит адрес первого элемента массива, как если бы это был указатель! Например:

```
1  #include <iostream>
2
3  int main()
4  {
5      int array[4] = { 5, 8, 6, 4 };
6
7      // Выводим значение массива (переменной array)
8      std::cout << "The array has address: " << array << '\n';
9
10     // Выводим адрес элемента массива
11     std::cout << "Element 0 has address: " << &array[0] << '\n';
12
13     return 0;
14 }
```

Результат на моём компьютере:

The array has address: 004BF968

Element 0 has address: 004BF968

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

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