

Ravesli [Ravesli](#)


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


Урок №110. Аргументы командной строки

 [Юрий](#) |

- [Уроки C++](#)

|

 Обновлено: 8 Сен 2020 |

 44292

[↑](#)  [11](#)

На этом уроке мы рассмотрим, что такое аргументы командной строки в языке C++ и то, как они используются.

Оглавление:

1. [Аргументы командной строки](#)
2. [Передача аргументов командной строки](#)
3. [Использование аргументов командной строки](#)
4. [Обработка числовых аргументов](#)
5. [Анализ аргументов командной строки](#)
6. [Заключение](#)

Аргументы командной строки

Из [урока №3](#) мы уже знаем, что при компиляции и линкинге, компилятор создает исполняемый файл. Когда программа запускается, выполнение начинается с первой строки функции `main()`. До этого урока мы объявляли `main()` следующим образом:

```
1 int main()
```

Обратите внимание, в этой версии функции `main()` никаких параметров нет. Тем не менее, многие программы нуждаются в некоторых входных данных. Например, предположим, что вы пишете программу под названием `Picture`, которая принимает изображение в качестве входных данных, а затем делает из этого изображения миниатюру (уменьшенная версия изображения). Как функция `picture()` узнает, какое изображение нужно принять и обработать? Пользователь должен сообщить программе, какой файл следует открыть. Это можно сделать следующим образом:

```
1 // Программа: Picture
2 #include <iostream>
3 #include <string>
4
5 int main()
6 {
7     std::cout << "Enter name of image-file to create a thumbnail for: ";
```

```
8   std::string filename;  
9   std::cin >> filename;  
10  
11   // Открываем файл-изображение  
12   // Создаем миниатюру  
13   // Выводим миниатюру  
14 }
```

Тем не менее, здесь есть потенциальная проблема. Каждый раз при запуске программа будет ожидать пользовательский ввод. Это не проблема, если вы вручную запускаете программу из командной строки один раз для одного изображения. Но это уже проблема, если вы хотите работать с большим количеством файлов или чтобы другая программа имела возможность запустить эту программу.

Рассмотрим это детально. Например, вы хотите создать миниатюры для всех файлов-изображений, которые находятся в определенном каталоге. Как это сделать? Вы можете запускать эту программу столько раз, сколько есть изображений в каталоге, введя каждое имя файла вручную. Однако, если есть сотни изображений, такой подход будет, мягко говоря, не очень эффективным! Решением здесь будет написать программу, которая перебирала бы каждое имя файла в каталоге, вызывая каждый раз функцию `picture()` для каждого файла.

Теперь рассмотрим случай, когда у вас есть веб-сайт, и вы хотите, чтобы он создавал миниатюру каждый раз, когда пользователь загружает изображение на сайт. Эта программа не может принимать входные данные из Интернета и следует логический вопрос: «Как тогда вводить имя файла?». Выходом является вызов веб-сервером функции `picture()` автоматически каждый раз после загрузки файла.

В обоих случаях нам нужно, чтобы внешняя программа передавала имя файла в качестве входных данных в нашу программу при её запуске, вместо того, чтобы функция `picture()` сама дожидалась, пока пользователь вручную введет имя файла.

Аргументы командной строки — это необязательные строковые аргументы, передаваемые операционной системой в программу при её запуске. Программа может их использовать в качестве входных данных, либо игнорировать. Подобно тому, как параметры одной функции предоставляют данные для параметров другой функции, так и аргументы командной строки предоставляют возможность людям или программам предоставлять входные данные для программы.

Передача аргументов командной строки

Исполняемые программы могут запускаться в командной строке через вызов. Например, для запуска исполняемого файла `MyProgram`, который находится в корневом каталоге диска C в ОС Windows, вам нужно ввести:

```
C:\>MyProgram
```

Чтобы передать аргументы командной строки в `MyProgram`, вам нужно будет их просто перечислить после имени исполняемого файла:

```
C:\>MyProgram SomeContent.txt
```

Теперь, при запуске `MyProgram`, `SomeContent.txt` будет предоставлен в качестве аргумента командной строки. Программа может иметь несколько аргументов командной строки, разделенных пробелами:

```
C:\>MyProgram SomeContent.txt SomeOtherContent.txt
```

Это также работает и с ОС Linux (хотя структура каталогов будет отличаться от структуры каталогов в ОС Windows).

Если вы запускаете свою программу из среды IDE, то ваша IDE должна предоставить способ ввода аргументов командной строки.

Для пользователей Visual Studio: Щелкните правой кнопкой мыши по нужному проекту в меню "Обозреватель Решений" > "Свойства":

Затем выберите "Свойства конфигурации" > "Отладка". На правой панели будет строка "Аргументы команды". Вы сможете здесь ввести аргументы командной строки, и они будут автоматически переданы вашей программе при её запуске:

Пользователям Code::Blocks: Выберите "Project" > "Set program`s arguments":

Использование аргументов командной строки

Теперь, когда вы знаете, как передавать аргументы командной строки в программу, следующим шагом будет доступ к ним из программы. Для этого используется уже другая форма функции `main()`, которая принимает два аргумента (`argc` и `argv`)

следующим образом:

```
1 | int main(int argc, char *argv[])
```

Также вы можете увидеть и такой вариант:

```
1 | int main(int argc, char** argv)
```

Хоть оба эти варианта идентичны по своей сути, но рекомендуется использовать первый, так как он интуитивно понятнее.

argc (англ. *«argument count»* = «количество аргументов») — это целочисленный параметр, содержащий количество аргументов, переданных в программу. **argc** всегда будет как минимум один, так как первым аргументом всегда является имя самой программы. Каждый аргумент командной строки, который предоставляет пользователь, заставит **argc** увеличиться на единицу.

argv (англ. *«argument values»* = «значения аргументов») — это место, где хранятся фактические значения аргументов. Хотя объявление **argv** выглядит немного пугающе, но это всего лишь массив [строк C-style](#). Длинной этого массива является **argc**.

Давайте напишем короткую программу **MyArguments**, которая будет выводить значения всех аргументов командной строки:

```
1 | // Программа MyArguments
2 | #include <iostream>
3 |
4 | int main(int argc, char *argv[])
5 | {
6 |     std::cout << "There are " << argc << " arguments:\n";
7 |
8 |     // Перебираем каждый аргумент и выводим его порядковый номер и значение
9 |     for (int count=0; count < argc; ++count)
10 |         std::cout << count << " " << argv[count] << '\n';
11 |
12 |     return 0;
13 | }
```

Теперь, при вызове **MyArguments** с аргументами командной строки **SomeContent.txt** и **200**, вывод будет следующим:

There are 3 arguments:

0 C:\MyArguments

1 SomeContent.txt

2 200

Нулевой параметр — это путь и имя текущей программы. Первый и второй параметры здесь являются аргументами командной строки, которые мы передали.

Обработка числовых аргументов

Аргументы командной строки всегда передаются в качестве строк, даже если предоставленное значение является числовым. Чтобы использовать аргумент командной строки в виде числа, вам нужно будет конвертировать его из строки в число. К сожалению, в языке C++ это делается немного сложнее, чем должно быть:

```
1 | #include <iostream>
2 | #include <string>
```

```
3 #include <sstream> // для std::stringstream
4 #include <cstdlib> // для exit()
5
6 int main(int argc, char *argv[])
7 {
8     if (argc <= 1)
9     {
10         // В некоторых операционных системах argv[0] может быть просто пустой строкой, без имени про
11
12         // Обрабатываем случай, когда argv[0] может быть пустым или не пустым
13         if (argv[0])
14             std::cout << "Usage: " << argv[0] << " <number>" << '\n';
15         else
16             std::cout << "Usage: <program name> <number>" << '\n';
17
18         exit(1);
19     }
20
21     std::stringstream convert(argv[1]); // создаем переменную stringstream с именем convert, инициа
22
23     int myint;
24     if (!(convert >> myint)) // выполняем конвертацию
25         myint = 0; // если конвертация терпит неудачу, то присваиваем myint значение по умолчанию
26
27     std::cout << "Got integer: " << myint << '\n';
28
29     return 0;
30 }
```

Если мы запустим эту программу с аргументом командной строки 843, то результатом будет:

Got integer: 843

std::stringstream работает почти так же, как и std::cin. Здесь мы инициализируем переменную std::stringstream значением argv[1], так что мы можем использовать оператор >> для извлечения значения в переменную типа int.

Анализ аргументов командной строки

Когда вы пишете что-то в командной строке (или запускаете свою программу из среды IDE), то операционная система ответственна за то, чтобы ваш запрос проделал правильный путь. Это связано не только с запуском исполняемого файла, но и с анализом любых аргументов для определения того, как их следует обрабатывать и передавать в программу.

Операционные системы имеют обязательные правила обработки специальных символов (двойные кавычки, бэкслешы и т.д.).

Например:

MyArguments Hello world!

Результат:

There are 3 arguments:

0 C:\MyArguments

1 Hello

2 world!

Строки, переданные в двойных кавычках, считаются частью одной и той же строки:

```
MyArguments "Hello world!"
```

Результат:

```
There are 2 arguments:  
0 C:\MyArguments  
1 Hello world!
```

Для того, чтобы вывести каждое слово отдельно на строке, используйте бэкслешы:

```
MyArguments \"Hello world!\"
```

Результат:

```
There are 3 arguments:  
0 C:\MyArguments  
1 "Hello  
2 world!"
```

Заключение

Аргументы командной строки предоставляют отличный способ для пользователей или других программ передавать входные данные в программу при её запуске. Используйте любые входные данные, необходимые программе при запуске, в качестве аргументов командной строки. Если командная строка не передана, то вы всегда сможете это обнаружить и попросить пользователя ввести данные вручную. Таким образом, ваша программа будет работать в любом случае.

Оценить статью:

★★★★★ (173 оценок, среднее: 4,82 из 5)



[← Урок №109. assert и static_assert](#)



[Урок №111. Эллипсис →](#)

Комментариев: 11



1. *Дмитрий:*
[2 июня 2020 в 23:43](#)

огромное спасибо за статью! никто кроме вас не смог понятно изложить

[Ответить](#)



2. *hillbilly:*
[24 февраля 2020 в 16:09](#)

Как понимать фразу:

"В некоторых операционных системах, `argv[0]` может быть просто пустой строкой, без имени программы"

То есть вы хотите сказать что некоторые операционные системы инициализируют указатель `argv[0]` в `nullptr`???

Если это так то все понятно, если нет то объясните логику условия:

`if(argv[0])`

[Ответить](#)



1. *Steindvart:*

[11 мая 2020 в 19:03](#)

Пустая строка не есть `nullptr`. Указатель на что-то, если он не `nullptr` (или `NULL`) преобразуется в `true`, даже если это указатель на пустую строку `""`. То есть, которая состоит только из нуля-символа.

Поэтому тут всё корректно.

[Ответить](#)



3. *Woland:*

[26 января 2020 в 21:10](#)

Может кому пригодится такой момент, если кто-то собирается писать в универсальном стиле (мультибайт/юникод). Сразу скажу, что в параметрах `main` тип `wchar_t` не работает. Так как `argv[i]` всегда будет возвращать 0.

```
1 int main(int argc, wchar_t* argv[]) {           // Не работает как надо
2 }
```

Поэтому непонятно, что делать если вы работаете с UTF (L"Text"), а в параметрах `main` всегда `char`. (Кому пока непонятно — не беда. В конце ссылка на то, что пригодится позже.)

Пример проверки существования файла переданного как аргумент (суть в преобразовании `std::string` в `std::wstring`):

```
1 #include <sstream>                // Для работы с std::(w)string
2 #include <filesystem>             // Для работы с файловой системой
3
4 int main(int argc, char* argv[]) {
5     if (argc < 2) return 1;       // Проверка, что аргумент передан
6     #ifdef _UNICODE
7         using str = std::wstring; // Переопределение для независимости от типа кодиро
8         const std::string file{ argv[1] }; // Преобразование char[] в std::string
9         const str File(file.cbegin(), file.cend()); // Преобразование std::string в std::wstring
10    #else
11        using str = std::string;   // Переопределение для независимости от типа кодиро
12        const str File{ argv[1] };
13    #endif
14    namespace fs = std::filesystem; // Переопределение для std::filesystem
15    const fs::path fsFile{ File }; // Инициализация объекта класса path
16    if (!fs::exists(fsFile)) return 1; // Проверка файла на существование
17
18    // Далее основной код
19 }
```

Работает с версией C++ 17 и выше.

Если кто-то знает как лучше преобразовать `std::string` в `std::wstring`, напишите.

В дополнение ссылка на хорошее пояснение про мультибайт/юникод — <https://habr.com/ru/post/164193/>

Оставляете по мере изучения и практики полезные примеры в поддержку автора перевода 😊

Всем терпения и удачи в изучении! Автору спасибо за перевод! Я добил этот курс.

[Ответить](#)



4. TurboOSMaker:

[14 июня 2019 в 20:41](#)

У меня возникла такая проблема: вы привели примеры добавления аргументов для Visual Studio и Code::Blocks. А что делать пользователям Dev-C++?

[Ответить](#)



1. Илья:

[4 ноября 2019 в 13:30](#)

Запускать программу из командной строки, и передавать аргументы через неё

[Ответить](#)



2. Артём:

[24 марта 2020 в 18:28](#)

Выполнить -> Параметры...

[Ответить](#)



5. Ксения:

[21 января 2019 в 15:24](#)

Добрый день!

Не понятно, что именно происходит при обработке ситуации, когда `argv[0]` является просто пустой строкой.

`<program name>`, `<number>` — они как-то инициализируются? Что именно выведет программа в случае не пустой и пустой строки?

[Ответить](#)



1. Александр:

[1 марта 2019 в 15:01](#)

первым аргументом ВСЕГДА идет исполняемый файл программы. Поэтому такого просто не может произойти

[Ответить](#)



2. Andrey:

[3 октября 2020 в 09:20](#)

Абсолютно ничем, это можно проверить, банально поменять `if<argc>=1`, либо через отладчик посмотреть инициализацию. Отладчик и тестирование наше все 😊

[Ответить](#)

6. *Elena:*[17 октября 2018 в 22:44](#)

спасибо за отличное объяснение, понятное даже новичку. помогло разобраться с 1м домашним заданием в институте. Только вот запустить из командной строки пока не получается никак..

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

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