7.13 — Command line arguments

BY ALEX ON FEBRUARY 15TH, 2008 | LAST MODIFIED BY NASCARDRIVER ON FEBRUARY 7TH, 2020

The need for command line arguments

As you learned lesson **0.4 -- introduction to development**, when you compile and link your program, the compiler produces an executable file. When a program is run, execution starts at the top of the function called main(). Up to this point, we've declared main like this:

```
int main()
```

Notice that this version of main() takes no parameters. However, many programs need some kind of input to work with. For example, let's say you were writing a program called Thumbnail that read in an image file, and then produced a thumbnail (a smaller version of the image). How would Thumbnail know which image to read and process? The user has to have some way of telling the program which file to open. To do this, you might take this approach:

```
// Program: Thumbnail
#include <iostream>
#include <string>

int main()
{
    std::cout << "Please enter an image filename to create a thumbnail for: ";
    std::string filename{};
    std::cin >> filename;

    // open image file
    // create thumbnail
    // output thumbnail
}
```

However, there is a potential problem with this approach. Every time the program is run, the program will wait for the user to enter input. This may not be a problem if you're manually running this program once from the command line. But it is problematic in other cases, such as when you want to run this program on many files, or have this program run by another program.

Let's look into these cases further.

Consider the case where you want to create thumbnails for all the image files in a given directory. How would you do that? You could run this program as many times as there are images in the directory, typing out each filename by hand. However, if there were hundreds of images, this could take all day! A good solution here would be to write a program that loops through each filename in the directory, calling Thumbnail once for each file.

Now consider the case where you're running a website, and you want to have your website create a Thumbnail every time a user uploads an image to your website. This program isn't set up to accept input from the web, so how would the uploader enter a filename in this case? A good solution here would be to have your web server call Thumbnail automatically after upload.

In both of these cases, we really need a way for an external *program* to pass in the filename as input to our Thumbnail program when Thumbnail is launched, rather than having Thumbnail wait for the *user* to enter the filename after it has started.

Command line arguments are optional string arguments that are passed by the operating system to the program when it is launched. The program can then use them as input (or ignore them). Much like function parameters

provide a way for a function to provide inputs to another function, command line arguments provide a way for people or programs to provide inputs to a *program*.

Passing command line arguments

Executable programs can be run on the command line by invoking them by name. For example, to run the executable file "WordCount" that is located in the root directory of the C: drive on a Windows machine, you could type:

```
C:\>WordCount
```

In order to pass command line arguments to WordCount, we simply list the command line arguments after the executable name:

```
C:\>WordCount Myfile.txt
```

Now when WordCount is executed, Myfile.txt will be provided as a command line argument. A program can have multiple command line arguments, separated by spaces:

```
C:\>WordCount Myfile.txt Myotherfile.txt
```

This also works for other command line operating systems, such as Linux (although your prompt and directory structure will undoubtedly vary).

If you are running your program from an IDE, the IDE should provide a way to enter command line arguments.

In Microsoft Visual Studio, right click on your project in the solution explorer, then choose properties. Open the "Configuration Properties" tree element, and choose "Debugging". In the right pane, there is a line called "Command Arguments". You can enter your command line arguments there for testing, and they will be automatically passed to your program when you run it.

In Code::Blocks, choose "Project -> Set program's arguments".

Using command line arguments

Now that you know how to provide command line arguments to a program, the next step is to access them from within our C++ program. To do that, we use a different form of main() than we've seen before. This new form of main() takes two arguments (named argc and argv by convention) as follows:

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

You will sometimes also see it written as:

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

Even though these are treated identically, we prefer the first representation because it's intuitively easier to understand.

argc is an integer parameter containing a count of the number of arguments passed to the program (think: argc = **arg**ument **c**ount). argc will always be at least 1, because the first argument is always the name of the program itself. Each command line argument the user provides will cause argc to increase by 1.

argv is where the actual argument values are stored (think: argv = **arg**ument **v**alues, though the proper name is "argument vectors"). Although the declaration of argv looks intimidating, argv is really just an array of C-style strings. The length of this array is argc.

Let's write a short program named "MyArgs" to print the value of all the command line parameters:

```
// Program: MyArgs
#include <iostream>

int main(int argc, char *argv[])
{
    std::cout << "There are " << argc << " arguments:\n";

    // Loop through each argument and print its number and value
    for (int count{ 0 }; count < argc; ++count)
        std::cout << count << " " << argv[count] << '\n';

    return 0;
}
```

Now, when we invoke this program (MyArgs) with the command line arguments “Myfile.txt” and “100”, the output will be as follows:

```
There are 3 arguments:
0 C:\MyArgs
1 Myfile.txt
2 100
```

Argument 0 is the path and name of the current program being run. Argument 1 and 2 in this case are the two command line parameters we passed in.

Dealing with numeric arguments

Command line arguments are always passed as strings, even if the value provided is numeric in nature. To use a command line argument as a number, you must convert it from a string to a number. Unfortunately, C++ makes this a little more difficult than it should be.

The C++ way to do this follows:

```
#include <iostream>
#include <string>
#include <sstream> // for std::stringstream
#include <cstdlib> // for std::exit()

int main(int argc, char *argv[])
{
    if (argc <= 1)
    {
        // On some operating systems, argv[0] can end up as an empty string instead of the pro
        // We'll conditionalize our response on whether argv[0] is empty or not.
        if (argv[0])
            std::cout << "Usage: " << argv[0] << " <number>" << '\n';
        else
            std::cout << "Usage: <program name> <number>" << '\n';

        std::exit(1);
    }

    std::stringstream convert{ argv[1] }; // set up a stringstream variable named convert, ini

    int myint{};
    if (!(convert >> myint)) // do the conversion
        myint = 0; // if conversion fails, set myint to a default value

    std::cout << "Got integer: " << myint << '\n';

```

```
28      return 0;
29  }
```

When run with input “567”, this program prints:

```
Got integer: 567
```

std::stringstream works much like std::cin. In this case, we’re initializing it with the value of argv[1], so that we can use operator>> to extract the value to an integer variable (the same as we would with std::cin).

We’ll talk more about std::stringstream in a future chapter.

The OS parses command line arguments first

When you type something at the command line (or run your program from the IDE), it is the operating system’s responsibility to translate and route that request as appropriate. This not only involves running the executable, it also involves parsing any arguments to determine how they should be handled and passed to the application.

Generally, operating systems have special rules about how special characters like double quotes and backslashes are handled.

For example:

```
MyArgs Hello world!
```

prints:

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

Typically, strings passed in double quotes are considered to be part of the same string:

```
MyArgs "Hello world!"
```

prints:

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

If you want to include a literal double quote, you have to backslash the double quote:

```
MyArgs \"Hello world!\"
```

prints:

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

Conclusion

Command line arguments provide a great way for users or other programs to pass input data into a program at startup. Consider making any input data that a program requires at startup to operate a command line parameter. If the command line isn't passed in, you can always detect that and ask the user for input. That way, your program can operate either way.

 **7.14 -- Ellipsis (and why to avoid them)**

 **Index**

 **7.12a -- Assert and static_assert**

C++ TUTORIAL | PRINT THIS POST

132 comments to 7.13 — Command line arguments

« Older Comments 1 2

kavin
February 7, 2020 at 6:30 am · Reply

Hi, in the last example line 20, we could use curly brackets to initialize like we did before right?

```
1 std::stringstream convert{argv[1]};
```

nascardriver
February 7, 2020 at 9:00 am · Reply

Yep, lesson updated!

Ged
December 11, 2019 at 9:03 am · Reply

1. Could you explain this a bit more?

You said std::stringstream works like std::cin. But what happens exactly? Variable convert gets a value of "432" (example). And how does it turn to an integer? Or was this specifically made so we could change a string with numbers to an integer?

```
1 std::stringstream convert(argv[1]);
2
3 int myint;
4 if (!(convert >> myint))
5     myint = 0;
```

Ged
December 11, 2019 at 9:07 am · Reply

And also. In one lesson you used std::exit(1), in this you used exit(1). Is exit(1) used by C and std::exit(1) used by C++?

nascardriver
December 11, 2019 at 9:14 am · Reply

Yes. C++ moved all function from C into the `std` namespace. They're still accessible without `std::` by including the header with an extension, eg.

```
1 #include <stdlib.h> // C, exit(1)
2 #include <cstdlib> // C++, std::exit(1)
```

The function might also be accessible without `std::` when you include the C++ header. That's not standard behavior, use `std::`.

nascardriver
December 11, 2019 at 9:07 am · Reply

`std::stringstream` uses the same conversion methods as `std::cin` and `std::cout`. You can input data as a string and extract it as a number, just like you can enter a string on the command line and extract it as whatever type you want.

Ged
December 11, 2019 at 9:52 am · Reply

Wrote a program to get a better understanding. I kinda understand what it does, but having a difficult time understanding the steps, like what specifically happens at the first line and the second line. So I wrote what I think happens.

std::stringstream convert(numberString) is similar to? (std::cin >> convert = numberString); // Variable "convert" takes "45" and stops when it reaches "l";

// convert = 45 // "45 has no data type till now?"

convert >> numberInt // the value 45 is assigned to "numberInt" which converts it into data type "int"?

```
#include <iostream>
#include <string>
#include <sstream>
int main()
{
    std::string numberString{ "4519" };
    std::stringstream convert(numberString);
    int numberInt{};
    convert >> numberInt;
    std::cout << numberInt;
    return 0;
}
```

nascardriver
December 12, 2019 at 1:57 am · Reply

```
std::stringstream convert{ numberString };
convert >> numberInt;

// same as

std::stringstream convert{};
convert << numberString; // Like std::cout, but stores the text.
convert >> numberInt; // Same as std::cin. Extracts the text we inserted previ
```

Ged
December 12, 2019 at 8:51 am · Reply

Thank you very much, now I understand.

hellmet
October 27, 2019 at 3:33 am · Reply

Similarly, if I wanted this output,

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

The input would be

```
MyArgs \"Hello\ world!\"
```

arcad
July 23, 2019 at 10:12 am · Reply

Hi need help,
You gave the example of the program my Args. I want to know how do we invoke "Myfile.txt" and "100".

nascardriver
July 24, 2019 at 4:04 am · Reply

Open a console/terminal (OS-specific)
Navigate to your executable (IDE-specific)
Run your executable and pass arguments

```
MyArgs MyFile.txt 100
```

Your IDE probably has a way of passing arguments to your program so you don't have to run it manually, you'll have to google how to do it.

Muskan
July 8, 2019 at 10:35 pm · Reply

Are there any other parameter other than argc and argv which we can pass in main method???

nascardriver
July 9, 2019 at 4:38 am · Reply

No

cdecde57
June 13, 2019 at 2:12 pm · Reply

I am just wondering.

This lesson seems to be harder than the others to me and I don't completely understand it. I understand argc and argv what I don't understand is the way it can receive it. How do I have a user send input that is created into an argument to open another file? or Like edit a directory full of files then how do I directly edit them after I received the input?

```
std::cout << "Please enter the location of the file you want to open.\n";
char x[255]{" "};
std::cin.getline(x, 255);

++argc;
argv[1] = x;
std::cout << argv[1];
```

Am I a little confused on the general idea of what goes down? Sorry if I don't make sense or am getting things wrong because I just need help on this lesson.

nascardriver
June 14, 2019 at 1:02 am · Reply

If you're using Windows, you can drag a file onto your program.exe and program.exe will be launched argv[1] set the file's path.
If you're launching your program through a console/terminal, you can specify the paths by writing them after your program's name, as shown in this lesson.
You shouldn't write to `argc` or `argv`, modifying them only affect the variables. File IO is covered in lesson 18.6 and 18.7. You should be able to understand them (at least enough to use file io) already.

cdecde57
June 14, 2019 at 6:46 am · Reply

Thanks! That really makes more sense now.

I will have to review file IO sometime and maybe more in-depth, other places as well so I can make like a database or something for a program. I think that would be cool.

So, once I have the argument set (like someone dragged a file onto it) would I select argv[1] and then I can just edit it like a regular variable (depending becuase if it is like a text file then I would need to know file IO right?).

Anyways, thanks!

nascardriver
June 14, 2019 at 6:52 am · Reply

`argv[1]` holds the path of the file. You can't do anything more with it than you can do with a variable in which you manually wrote the path. You need file io.

cdecde57
June 14, 2019 at 8:45 am · Reply

Alright.
Thanks!

Alireza
June 3, 2019 at 12:40 pm · Reply

Hello dear,
You just mentioned how to pass arguments and take them from command line. But you didn't mention how to deal with taken arguments. For example: If user inputs two numeric arguments, how do we add them ? Or something like this.(e.g. rm command in linux which renames files. Or command line music player which takes file name and plays it)

Alex
June 11, 2019 at 3:10 pm · Reply

I'm not sure what you mean by "taken arguments". If the user inputs two numeric arguments and you want to add them, convert them to numbers as shown in the lesson, and then add them. I'm not sure I understand what you're getting at with the other examples.

masterOfNothing
May 28, 2019 at 8:21 am · Reply

Sorry for this stupid question, but in case of the program with the output:

0 C:\MyArgs
1 Myfile.txt
2 100

Is MyArgs the location of a compiled .exe file from the project?

I called mine from windows command line where debug .exe file is located, e.g.,
>"C:\Users\...\...\bin\Debug\learncpp.exe" Arg1.txt 100

But the return code showed full path with .exe file extension at the end for the 0 (first) argument (which is the program file itself). You call it from the .exe file, right?

Alex
May 30, 2019 at 3:28 pm · Reply

Typically parameter 0 will be the location where the executable resides. Executables don't remember where they're compiled from.

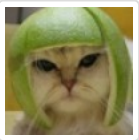
And yes, you can call executables directly from the .exe file. That's the whole point of them being executable. :)

X
April 7, 2019 at 6:23 am · Reply

How is this conversion happening?

```
1    int myint;
2    if (!(convert >> myint)) // do the conversion
3      myint = 0; // if conversion fails, set myint to a default value
```

If the conversion fails, then myint = 0. Cool. But there is no "else if" or another "if" to do the actual conversion in a non-fail case? So how is myint receiving the value from convert? Explain, please.



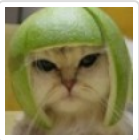
Alex
April 7, 2019 at 1:06 pm · Reply

First (convert >> myint) evaluates, which will try to do the extraction, and returns convert. Next, !convert is evaluated. This returns true if there was an error.

Thus, myint = 0 only executes if the extraction caused an error.

hassan magaji
February 6, 2019 at 10:07 am · Reply

hi Alex,
in the first paragraph you wrote "up to this point we've _decleared_ main...
shouldnt we've "defined" main...
thanks by the way



Alex
February 7, 2019 at 8:05 pm · Reply

What follows is only a declaration, not a definition, since it contains no body.

Jon
January 15, 2019 at 2:45 pm · Reply

If you're passing let's say a TXT file as a command line argument, are there any restrictions to where the file must be located? Is the default path the same directory as the program? And if you want to use a file in a different location, do you just type in the entire path for it?

Thanks!

nascardriver

January 16, 2019 at 5:44 am · Reply

Hi Jon!

Your program receives the path that you give it. It's up to you to figure out where the file is actually located.

```
#include <iostream>

int main(int argc, char *argv[])
{
  for (int i{ 0 }; i < argc; ++i)
  {
    std::cout << argv[i] << '\n';
  }

  return 0;
}
```

Compiled as "learncpp"

Sample calls
./learncpp ../link.txt

```
./learncpp
../link.txt
```

./learncpp link.txt

```
./learncpp
link.txt
```

If your terminal/console has expansion capabilities, those will happen before your program runs.

./learncpp $(ls)

```
./learncpp
CMakeCache.txt
CMakeFiles
cmake_install.cmake
learncpp
Makefile
```

Kelvin
November 19, 2018 at 4:09 am · Reply

How can I add a condition for the arguments? For example, the users can only input numerical integer.

nascardriver
November 21, 2018 at 3:56 am · Reply

The user can do what they want and you can't stop them, because the program isn't running at that time.
You have the check if the provided arguments match what you expected. For example by checking the fail bit of an @std::stringstream or the idx argument passed to @std::atoi.

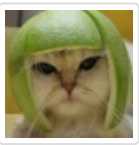
Adam
October 28, 2018 at 2:57 pm · Reply

Hi,
I have seen code examples where they define the main function argument type but don't provide any parameters (from the OpenCV website):

```
 #include "opencv2/imgproc.hpp"
#include "opencv2/highgui.hpp"
using namespace cv;
int main(int, char**) //what is going on here??
{
    VideoCapture cap(0);
    if(!cap.isOpened()) return -1;
    Mat frame, edges;
    namedWindow("edges",1);
    for(;;)
    {
        cap >> frame;
        cvtColor(frame, edges, COLOR_BGR2GRAY);
        GaussianBlur(edges, edges, Size(7,7), 1.5, 1.5);
        Canny(edges, edges, 0, 30, 3);
        imshow("edges", edges);
        if(waitKey(30) >= 0) break;
    }
    return 0;
}
```

What is the purpose of this?

Thanks,
Adam



Alex
October 28, 2018 at 5:58 pm · Reply

There's no purpose to this. If you're not going to use the parameters, it's cleaner to use the `int main()` form.

pete
July 15, 2018 at 4:48 am · Reply

Hi alex,
can you discuss (or point to..) how to extract data from other files as an input?
thanks!

nascardriver
July 15, 2018 at 5:28 am · Reply

Hi pete!
Have a look at less 18.6, you should be able to understand it from what has been covered so far.

References
* Lesson 18.6 - Basic file I/O

pete
July 15, 2018 at 10:56 am · Reply

i will, thanks nascardriver!

Ishak
July 7, 2018 at 5:24 am · Reply

can you explain this piece of code, please? if argc is less than 1 this piece of code starts running, why? what are we trying to do ? "On some operating systems, argv[0] can end up as an empty string instead of the program's name. We'll conditionalize our response on whether argv[0] is empty or not" ? I don't understand what the means, and I don't understand whats the purpose of this code and how it works. I tried reading this multiple times. Thanks for any help

```
if (argc <= 1)
  {
      // On some operating systems, argv[0] can end up as an empty string instead of the
      // We'll conditionalize our response on whether argv[0] is empty or not.
      if (argv[0])
          std::cout << "Usage: " << argv[0] << " <number>" << '\n';
      else
          std::cout << "Usage: <program name> <number>" << '\n';

      exit(1);
```

nascardriver
July 8, 2018 at 7:52 am · Reply

```
if (argc <= 1) // If we have received too few arguments
{
  if (argv[0]) // And we know what out program is called
  {
    // Print usage with program name
    std::cout << "Usage: " << argv[0] << " <number>" << '\n';
  }
  else // If we don't know what our program is called
  {
    // Print usage with a placeholder as the program name
    std::cout << "Usage: <program name> <number>" << '\n';
  }

  std::exit(1);
}
```

Ishak
July 8, 2018 at 11:10 am · Reply

Thank you nascardriver for the help. I have better understanding now. I have some follow up questions if you don't mind.

- Should we use that piece of code to check for too few arguments in program "MyArgs" (the one before dealing with numeric arguments)?

- Why can't we just do this argv[1] >> myint?

- What exactly is a string-stream variable ?

Thanks again for the help, and the amazing material.

nascardriver
July 9, 2018 at 2:01 am · Reply

1. You should always check if there are enough elements (arguments) before accessing an array (argv).
I wouldn't use the check provided here. I'd use preprocessor to determine the OS and shift the index that way. You don't need to worry about this too much, do what works on your OS and fix it once you want to port your code.

2. argv[1] is a const char*. A const char* doesn't have an operator>>.

3. It's like @std::cin and @std::cout but it doesn't print anything. You can use it to format strings and numbers and get the resulting string from it.

Petr
May 29, 2019 at 11:19 am · Reply

This is confusing, because the condition

```
1 if (argc <= 1)
```

is evaluated as true even when argc == 0! In that case,

```
1 std::cout << "Usage: " << argv[0] << " <number>" << '\n';
```

probably causes segmentation faul.

nascardriver
May 29, 2019 at 11:40 pm · Reply

Hi Petr!

Well spotted. It won't cause a segmentation fault however. The value of argv[argc] must be 0, so argv[0] always exists.

Anon
September 3, 2017 at 1:06 am · Reply

Hey Alex,

Why don't you just use the atoi function from the C standard library.

Thanks so much for all your help.

God bless!

Alex
September 5, 2017 at 9:30 am · Reply

You certainly can use atoi if you desire.

Angmar

August 21, 2017 at 1:51 am · Reply

Hello again, Alex.

Would you recommend using std::stoi instead of stringstream in that particular case? stoi looks a bit simpler to me as a beginner.

[code]#include <iostream>
#include <string>
using namespace std;

int
main(int argc, char** argv)
{
 for (int i = 1; i < argc; i++) {
  cout << stoi(*(i + argv)) << endl;
 }
 return 0;
}

Alex
August 21, 2017 at 10:03 pm · Reply

Yes, if it meets your needs.

Tal
July 27, 2017 at 9:06 am · Reply

Is there way to check if the input is a float?
I used

```
if (!(convert >> myint)) // do the conversion
        myint = 0; // if conversion fails, set myint to a default value
```

But the conversion doesn't fail if the input is 10.9 for example.

Alex
July 27, 2017 at 9:49 pm · Reply

As you noticed, operator>> will extract whatever it can. So if you try to extract 10.9 to an int, it will extract the 10, and leave the .9 in the input stream.

std::stringstream doesn't provide any way to determine if the value is a float. Of course, you could always just extract to a float value if that's a valid choice. Alternatively, you could read your input into a std::string and then look at the string to see if it has a decimal.

Georges Theodosiou
March 22, 2017 at 6:36 am · Reply

My dear c++ Teacher,
Please accept my many thanks for you answered my question about main() arguments and for your suggestion to read here and post my question.
First of all I use compilers online and I do not understand what is "command line".
I run program

```
// Program: MyArgs
```

```
#include <iostream>

int main(int argc, char *argv[])
{
    std::cout << "There are " << argc << " arguments:\n";

    // Loop through each argument and print its number and value
    for (int count=0; count < argc; ++count)
        std::cout << count << " " << argv[count] << '\n';

    return 0;
}
```

by compiler
http://codepad.org/kfWQHi1v
and output was
There are 1 arguments:
0 /t
I run by
https://www.tutorialspoint.com/compile_cpp_online.php
and output was
There are 1 arguments:
0 main
With regards and friendship.

Alex
March 24, 2017 at 3:38 pm · Reply

Once you compile your program using https://www.tutorialspoint.com/compile_cpp_online.php, you can type "main myFile.txt 100" in the green box labeled "Default Term". This will run the program you compiled with the command line arguments you typed in.

Georges Theodosiou
March 25, 2017 at 5:57 am · Reply

My dear c++ Teacher,
Please accept my many thanks for you answered my question and for your helpful answer. I typed that you suggest me and output is
There are 3 arguments:
0 main
1 my file.txt
2 100
However I understand nothing but that have to learn every past lesson before learn present.
With regards and friendship.

ngabo
March 16, 2017 at 1:35 pm · Reply

Hi, Alex my name is Ngabo, i am from in Africa
I like so much your great tutorials.

so, i have a question about this top <<command line arguments>>
is it possible to give a pop up message at start up program like " # MyArgs please enter your argument? : myfile.txt"

is my question clear ?

Alex
March 17, 2017 at 2:21 pm · Reply

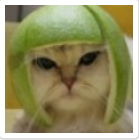
Hello Ngabo from Africa, thanks for visiting the site!

Once the program has started executing, it's too late to have the user pass in command line arguments. But you are free to have your program output anything or ask the user to input anything once it is running.

Perhaps I missed the point of your question?

Rohit
March 12, 2017 at 5:54 am · Reply

Hi Alex! Got some doubts.
1) stringstream creates a variable convert, then how can we convert the initilized value to int using conver>>myint. I mean to say how this statement works? please explain.
2) what is the diff b/w exit(0) and exit(1)? and is there anything exit(2) or exit(3) or.....?



Alex
March 12, 2017 at 11:51 am · Reply

1) stringstream works like std::cin does -- except instead of getting input from the user, you can give it input. Then you use operator<< to extract the output to a variable of your choice.
2) exit returns an error code back to the OS. 0 means "everything went okay". A non-zero values means "something went wrong". In most cases, this error code is ignored, but it can be useful if you have another program calling your program and that other program needs to know whether your program succeeded or not.

« Older Comments 1 2