Using Command Line Arguments in C/C++ and Java

Michael McAlpin

When you launch a program from the command line in Windows, Mac OS, or Linux, you can pass data called arguments or parameters for the program to use for its processing task. The arguments are written on the command line as simple character strings following the program name, separated by spaces. So, for example, if we were to launch a program called "test" from the command line with arguments for a file name, an integer, and a double, it might look like this in a Linux or Mac OS environment:

```
$ ./test  sample.txt  547  12.36
```

In Linux and Mac OS, the "$" is the command prompt and "./" is how we launch a program in the current directory. For Windows, we do not need the "./" and the command prompt looks a bit different, so the same program launch might look like:

```
C:\> test  sample.txt  547  12.36
```

Either way, we are launching the same program with the same command line arguments.

Now, the reason we use command line arguments is so we can run the program multiple times, each time using different arguments, without needing to recompile or rebuild the program for each run. It also avoids a possibly time-consuming interactive session with the program to enter the data items. This makes it easy to test programs with different input data values and different input file names quickly.

Of course, the program must be able to use the command line arguments that have been passed to it. And while it is possible to write a program that can interpret different orderings of the input parameters, it simplifies matters greatly to use a fixed ordering for them. So, if your program specification calls for parameters of particular types in a particular order, then that is the order in which they must be entered at the command line, and this ordering also guides the development of the program that must use them.

In both C/C++ and Java, the command line arguments are passed into the main function or method as parameters.

In C/C++, the arguments are passed in two parameters: (a) the int "argc", which the count of all the parameters, and (b) the pointer "argv", which points to an array of character strings that contain the command line arguments. The value of argc is always at least 1, because in C/C++, the first argument, argv[0], is always just the program name, so the user arguments start with argv[1].

In Java, the situation is a bit simpler, as all the command line arguments are passed into the main method in the String array "args". A separate counter like argc is not used, since the size of the array can be found simply using args.length. And the program name is not included in the list, so args[0] is the first user argument, if any.

The two sample programs below illustrate how input parameters for the above example can be read in and used by C/C++ and Java programs. These are not the only ways to read in the parameters, but they should give you a good start. You can find out more about using C/C++ and command line arguments at www.cprogramming.com and www.cplusplus.com. For Java, you can find out more about Java and using command line arguments at www.docs.oracle.com/javase/tutorial/essential/enfironment/cmdLineArgs.html.

C/C++ Progam

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv) {

        printf( "\ncommand line arguments including program name:\n\n");
        int i;
        for ( i = 0; i < argc; i++ ) {
                        printf( "argument %d: %s\n", i, argv[ i ] );
        }
        printf( "\n");

        char* fname = argv[ 1 ];
        FILE *file = fopen( fname, "r" );
        if ( file == 0 )
        {
                        printf( "Could not open file\n" );
        } else {
                        printf( "File opened successfully\n" );
                        fclose( file );
        }

        int intvalue = atoi( argv[ 2 ] );
        printf( "\nintvalue = %d\n", intvalue );

        double dblvalue = atof( argv[ 3 ] );
        printf( "\ndblvalue = %f\n", dblvalue );

   return 0;
}
```

Program Output:

For the launch command “test sample.key 547 12.36” and assuming that the text file “sample.txt” exists in the current directory, the output of this program is shown below. Please note how the program name is the first parameter in the list.

```
command line arguments including program name:

argument 0: ./test
argument 1: sample.txt
argument 2: 547
argument 3: 12.36

File opened successfully

intvalue = 547

dblvalue = 12.360000
```

Java Program

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;

public class JavaArgs {
	public static void main(String[] args) {

		System.out.println( "\ncommand line arguments:\n");
		for ( int i = 0; i < args.length; i++ ) {
			String s = args[ i ];
			System.out.println( "argument " + i + ": " + s );
		}
		File file = new File( args[ 0 ] );
		try {
			BufferedReader br = new BufferedReader( new FileReader( file ));
			System.out.println( "\nFile opened successfully.");
			br.close();
		} catch ( Exception e ) {
			e.printStackTrace();
		}
		int intvalue = Integer.parseInt( args[ 1 ] );
		System.out.println( "\nintvalue = " + intvalue);
		double dblvalue = Double.parseDouble( args[ 2 ] );
		System.out.println( "\ndblvalue = " + dblvalue);
	}
}
```

Program Output:

For the launch command “test sample.key 547 12.36” and assuming that the text file “sample.txt” exists in the current directory, the output of this program is shown below. Please note how the program name is not included in the parameter list.

```
	command line arguments:

	argument 0: sample.txt
	argument 1: 547
	argument 2: 12.36

	File opened successfully
	intvalue = 547

	dblvalue = 12.360000
```