Lesson 14: Accepting command line arguments in C using argc and argv

By Alex Allain

In C it is possible to accept command line arguments. Command-line arguments are given after the name of a program in command-line operating systems like DOS or Linux, and are passed in to the program from the operating system. To use command line arguments in your program, you must first understand the full declaration of the main function, which previously has accepted no arguments. In fact, main can actually accept two arguments: one argument is number of command line arguments, and the other argument is a full list of all of the command line arguments.

The full declaration of main looks like this:

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

The integer, argc is the **arg**ument **c**ount. It is the number of arguments passed into the program from the command line, including the name of the program.

The array of character pointers is the listing of all the arguments. argv[0] is the name of the program, or an empty string if the name is not available. After that, every element number less than argc is a command line argument. You can use each argv element just like a string, or use argv as a two dimensional array. argv[argc] is a null pointer.

How could this be used? Almost any program that wants its parameters to be set when it is executed would use this. One common use is to write a function that takes the name of a file and outputs the entire text of it onto the screen.

```
#include <stdio.h>

int main ( int argc, char *argv[] )
{
    if ( argc != 2 ) /* argc should be 2 for correct execution */
    {
        /* We print argv[0] assuming it is the program name */
        printf( "usage: %s filename", argv[0] );
    }
    else
    {
        // We assume argv[1] is a filename to open
        FILE *file = fopen( argv[1], "r" );

        /* fopen returns 0, the NULL pointer, on failure */
        if ( file == 0 )
        {
            printf( "Could not open file\n" );
        }
        else
        {
            int x;
            /* read one character at a time from file, stopping at EOF, which
               indicates the end of the file.  Note that the idiom of "assign
               to a variable, check the value" used below works because
               the assignment statement evaluates to the value assigned. */
            while  ( ( x = fgetc( file ) ) != EOF )
            {
                printf( "%c", x );
            }
            fclose( file );
        }
    }
}
```

This program is fairly short, but it incorporates the full version of main and even performs a useful function. It first checks to ensure the user added the second argument, theoretically a file name. The program then checks to see if the file is valid by

trying to open it. This is a standard operation, and if it results in the file being opened, then the return value of fopen will be a valid FILE*; otherwise, it will be 0, the NULL pointer. After that, we just execute a loop to print out one character at a time from the file. The code is self-explanatory, but is littered with comments; you should have no trouble understanding its operation this far into the tutorial. :-)

Still not getting it? Ask an expert!
Quiz yourself
Previous: Typecasting
Next: Linked Lists
Back to C Tutorial Index