Command Line Arguments in C

This is our last topic for the semester - how to make our programs use simple command line arguments. For example, in Homework 5, you had to finish a program to compute the correlation of two lists of numbers read from a file. The name of the data file was to be "hard coded" into the C source code, e.g.,

```
FILE *fp;
fp = fopen("test.dat", "r");
```

If we wanted to use the program to compute the correlation of data in a different file, we would either have to modify the program source code *or* change the name of our data file to "test.dat". It would be nice if we could specify the name of the input file on the command line so that if we wanted to compute the correlation of the data in the file "physics_data.txt" we could just enter the command

```
./a.out physics_data.txt
```

This is an example of a command line argument - a value of a variable which we specify on the command line. The details of handling command line arguments are beyond what we can cover in this course, and we will only learn about specifying the names of input and output files on the command line.

Consider the correlation program example: we need to read the name of the input file from the command line. We have to make a few changes in main() to do this:

```
main( int argc, char *argv[] ) {
    FILE *fp;

    /* argv[1] contains the name of the input file */
    fp = fopen(argv[1], "r");

    etc.

}
```

The important thing here is that we have had to add two parameters to the definition of main(). The first, `argc`, will contain the number of command line arguments, *including the program name*, and the second argument, `argv[]`, is an array of strings where each string is one of the whitespace-separated strings from the command line.

Example: Consider running the program

```
./a.out physics_data.txt
```

The values of `argc` and `argv[]` would be:

- `argc` is 2 since there are two strings in the command line: "./a.out" and "physics_data.txt"
- `argv[0]` is "./a.out" since this is the first string in the command line

- `argv[1]` is "physics_data.txt" since this is the second string in the command line

Example: Suppose I've modified my grade statistics program to get the input file name from the command line and to write the report to an output file, the name of which also comes from the command line:

```
./a.out grades.dat grade_report.txt
```

The values of `argc` and `argv[]` are:

- `argc` is 3 since there are three strings in the command line: "./a.out", "grades.dat", and "grade_report.txt"
- `argv[0]` is "./a.out"
- `argv[1]` is "grades.dat"
- `argv[2]` is "grade_report.txt"

Here is how the main() function might look for this example:

```
main( int argc, char *argv[] ) {
     FILE *infp;      /* pointer to input file */
     FILE *outfp;     /* pointer to output file */

     [...other variable declarations go here...]

     /* Check that there are three strings in the command line:*/
     /* program name, input file name, and output file name.   */

     if ( argc != 3 ) {
          printf("Error - the command line is incorrect.");
          return 1;
     }

     /* Open the input file */

     infp = fopen(argv[1], "r");

     [...code to read data and compute statistics goes here...]

     fclose(infp);

     /* Open the output file */

     outfp = fopen(argv[2], "w");

     [...code to write report goes here...]

     fclose(outfp);

     return 0;
}
```

Note the use of `argc` in this example. Since the program requires the user to specify input and output files on the command line, it is good practice to check that they at least provided the correct number of command line arguments. Since the program expects two arguments, `argc` should be equal to 3 (two arguments plus the program name).

One last comment: you can treat the parameter definitions for main() as the necessary incantation to use command line arguments, that is, as something to memorize. However, they're not *too* hard to understand. `argc` is just an integer that contains the number of whitespace-separated strings in the command line. `argv` is an array of pointers to characters. That is, in our examples, `argv[0]` isn't *actually* the string "./a.out" but a pointer to the first character in the string "./a.out". As it turns out, that is all any string variable is - a pointer to the first character in the string - with the end of the string indicated by a null byte (byte with value 0).