

Problem: We want to parameterize our programming projects so that files containing input data and files providing results are named on the command line. If such files are not provided on the command line, we want our program to prompt for such files.

Programming project **p00**, for example, accepts a single input file and produces a single output file. Under ordinary circumstances the command line would look like:

**\$ p00 i00.dat o00.dat**

1. File **p00** is the first string that appears on the command line and contains the executable form of project 1.
2. File **i00.dat** is the second string that appears on the command line and contains input data. File **i00.dat** is the first command line parameter.
3. File **o00.dat** is the third string that appears on the command line and contains results produced by project **p00**. File **o00.dat** is the second command line parameter.

Another acceptable way to execute project **p00** is:

**\$ p00 i00.dat**

Enter the output file name: **o00.dat**

1. File **p00** is the first string that appears on the command line contains the executable form of project 1.
2. File **i00.dat** is the second string that appears on command line and contains input data
3. The prompt

**Enter the output file name:**

is produced by the program **p00** when fewer than three strings appear on the command line.

4. The response, **o00.dat**, is entered by the user.

The third and last way program **p00** can be invoked is:

**\$ ./p00**

Enter the input file name: **i00.dat**

Enter the output file name: **o00.dat**

1. File **p00** is the first and only string that appears on the command line contains the executable form of project 1.
2. The prompt

**Enter the input file name:**

is produced by program **p00** when only one string appears on the command line.

3. The user enters the response, **i00.dat**.
4. The prompt

**Enter the output file name:**

is produced by the program **p00** when fewer than three strings appear on the command line.

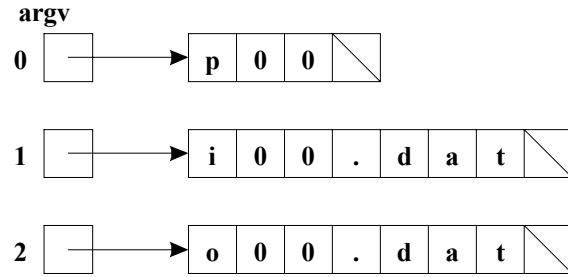
5. The response, **o00.dat**, is entered by the user.

Command line arguments are stored as an array of strings. For example, given the command  
**\$ p00 i00.dat o00.dat**

and the C++ program in Figure 1.

```
int main(int argc, char* argv[]) { return 0; }
```

**Figure 1.** C++ program declarations for command line arguments



**Figure 2.** Command line arguments

Integer parameter *argc* stores the number of arguments, the argument count. Array *argv* contains pointers to the separate strings on the command line.

Processing command line arguments proceeds by determining the number of arguments. If fewer than the requisite number of arguments are supplied, then the missing arguments must be obtained from the user. Once all the arguments are obtained, corresponding files may be opened.

Consider the first part of the problem. Process command line arguments. The program in Figure 3 processes command line arguments.

```

#include <iostream>
#include <fstream>
#include <string>
using namespace std;
void CommandLineException(int max,int actual)
{
    cout << endl;
    cout << "Too many command line arguments." << endl;
    cout << "A maximum of " << max << " arguments are permitted." << endl;
    cout << actual << " arguments were entered." << endl;
    cout << "Program terminated." << endl;
    exit(EXIT_FAILURE);
}
void FileException(char* fn)
{
    cout << endl;
    cout << "File " << fn << " could not be opened." << endl;
    cout << "Program terminated." << endl;
    exit(EXIT_FAILURE);
}
int main(int argc, char* argv[])
{
    char ifn[255];           //Input File Name
    char ofn[255];           //Output File Name

    switch (argc) {
        case 1:             //Prompt for both file names
            cout << "Enter the input file name. ";
            cin >> ifn;
            cout << "Enter the output file name. ";
            cin >> ofn;
            break;
        case 2:             //Prompt for the output file name
            strcpy(ifn,argv[1]);
            cout << "Enter the output file name. ";
            cin >> ofn;
            break;
        case 3:             //Both file names are arguments
            strcpy(ifn,argv[1]);
            strcpy(ofn,argv[2]);
            break;
        default:
            CommandLineException(2,argc-1);
            break;
    }
    ifstream i(ifn); if (!i) FileException(ifn);
    ofstream o(ofn); if (!o) FileException(ofn);
    //Read the input file, process input data, and write to the output file here
    o.close();
    i.close();
    return 0;
}

```

Figure 3. C++ program that processes command line arguments

Notes:

1. Include file *iostream* defines standard input and output classes for C++.
2. Include file *fstream* defines file structures and operations files (streams) in C++.
3. Include file *string* defines functions for C++ strings and standard C strings.
4. `cout <<` is the C++ equivalent of `printf(...)`
5. `cin >>` is the C++ equivalent of `scanf(...)`
6. Declarations need only appear before they are used. Declarations do not need to precede executable statements. Thus, the declaration

```
ifstream i ...;
```

does not appear directly after a {.

7. The declaration

```
ifstream i ...;
```

defines variable *i*. Variable *i* is an input file stream. An input file stream is equivalent to type FILE.

8. Variable *i* is initialized to the string referenced by variable *ifn* in the declaration `ifstream i(ifn);` The equivalent declaration in C is `FILE* i=fopen(ifn,"r");`
9. Input file stream *i* is closed by the statement `i.close();` The equivalent C syntax is `fclose(i);`
10. Function *CommandLineException* is called when more than two file names are entered on the command line. Function *CommandLineException* prints the appropriate error message and terminates program execution.
11. Function *FileException* is called when the input file cannot be opened. The most likely reason for this failure is that the input file does not exist in your local directory. Copy or create the input file whose name you entered on the command line in your local directory. Like function *CommandLineException*, function *FileException* terminates program execution.