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]

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.

[1] You can omit **./** prefix by adding the following line to your file **.bash_profile**, PATH=$PATH:./

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.

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

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

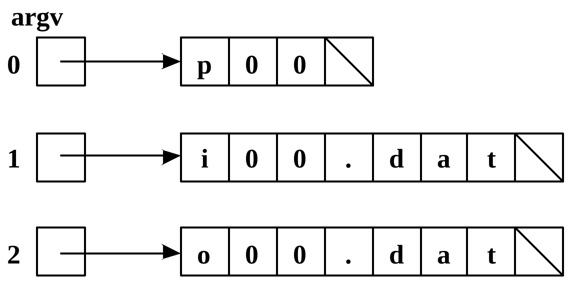

Figure 2. Command line arguments

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.

The solution to the problem posed in this lecture is shown in Figure 3.

```
//-----------------------------------------------------------------
//File L002.cpp illustrates how command line arguments are processed
//-----------------------------------------------------------------
//Author: Thomas R. Turner
//E-Mail: trturner@ucok.edu
//Date:   January, 2003
//-----------------------------------------------------------------
//Copyright January, 2003 by Thomas R. Turner.
//Do not reproduce without permission from Thomas R. Turner
//-----------------------------------------------------------------
//Standard C and C++ include files.
//-----------------------------------------------------------------
#include <iostream>
#include <fstream>
#include <string>
//-----------------------------------------------------------------
//Standard Namespace
//-----------------------------------------------------------------
using namespace std;
```
Figure 3. Code for processing command line arguments.

```
//-----------------------------------------------------------------
//FileException is thrown when a file whose name is given on the
//command line cannot be opened.
//-----------------------------------------------------------------
struct FileException {
      FileException(char* fn)
      {     cout << endl;
            cout << "File " << fn << " could not be opened.";
      }
};
//-----------------------------------------------------------------
//CommandLineException is thrown when too many arguments appear on
//the command line
//-----------------------------------------------------------------
struct CommandLineException {
      CommandLineException(int max,int actual)
      {     cout << endl;
            cout << "Too many command line arguments.";
            cout << endl;
            cout << "A maximum of " <<max<< " arguments can appear on the line.";
            cout << endl;
            cout << actual << " arguments were entered.";
      }
};
//-----------------------------------------------------------------
//Function Manager models a stub for an application manager.
//-----------------------------------------------------------------
void Manager (ifstream& i,ofstream& o)
{
      //read the input file stream i
      //process data from the input file stream
      //write to the output file stream o
}
```

Figure 3. Code for processing command line arguments, continued

```
//----------------------------------------------------------------
//Function main processes command line arguments
//----------------------------------------------------------------
int main (int argc, char* argv[])
{     try {
            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:             //Read the input file name and prompt for the output file name
                    strcpy(ifn,argv[1]);
                    cout << "Enter the output file name. ";
                    cin >> ofn;
                break;
                case 3:             //Read both file names
                    strcpy(ifn,argv[1]);
                    strcpy(ofn,argv[2]);
                break;
                default:            //Error, too many command line arguments
                    throw CommandLineException(2,argc-1);
            }
            ifstream i(ifn);  if (!i) throw FileException(ifn);
            ofstream o(ofn);  if (!o) throw FileException(ofn);
            Manager(i,o);
            i.close();
            o.close();
      } catch ( ... ) {
            cout << endl;
            cout << "Program terminated.";
            cout << endl;
            cout << "I won't be back!";
            cout << endl;
            exit(EXIT_FAILURE);
       }
       return 0;
}
```

Figure 3. Code for processing command line arguments, continued