

## CSE 332 Studio Session on C++ Data Representation, Storage, and Movement

These studio exercises are intended to gain familiarity with basic C++ data input, output, storage, and movement concepts and techniques, some of which may be somewhat familiar from your previous programming experience, and to add details specific to C++ that may not be obvious at first glance.

In this studio you will again work in groups of 2 or 3 people. Students who are more familiar with the material are encouraged to help those who are less familiar with it and asking questions of your professor and teaching assistants during the studio sessions is highly encouraged as well.

Please record your answers you work through the following exercises. After you have finished please post your answers to the required exercises, and to any of the enrichment exercises you completed, to the course message board, as a reply to my posting titled “Data and IO Studio”. The enrichment exercises are optional but are a good way to dig into the material a little deeper, especially if you breeze through the required ones.

### PART I: REQUIRED EXERCISES

1. Form a team of 2 or 3 people of your choice (we’ll shuffle the team assignments starting next time, with this week’s list as the starting point) and write down the names of the team members as the answer to this exercise.

2. Log into one of the studio lab’s Windows machines, open up Visual Studio 2008, and create a new Win32 Console Application project (for example, named **DataStudio** or something similar that identifies which studio this is for). Change the signature of the main function to be **int main (int argc, char \* argv[ ])** and add a line that prints out position **0** of **argv** (so named because it holds the program’s arguments vector) to **cout** (the standard output stream). Add any necessary precompiler and compiler directives, build your project, fix any remaining warnings or errors.

Once the program builds correctly open up a console window as you did in the previous studio, change to the directory where the executable program for the project was created, and run it. As the answer to this exercise please say (1) what output the program produced, and (2) what that tells you about what is always contained in position **0** of a program’s arguments vector.

3. Use **argc** (the program’s argument count) and **argv** to iterate through all of the program’s command line arguments from position **0** to position **argc-1** (which is the last valid position in **argv**), as in the lecture slides available at [http://www.cse.wustl.edu/~cdgill/courses/cse332/C++\\_programs.ppt](http://www.cse.wustl.edu/~cdgill/courses/cse332/C++_programs.ppt), but instead of printing each one push it back into a vector of C-style strings (of type **vector<char \*>**) and then iterate through the vector (**hint**: a vector also starts with position **0** and the number of positions in a vector is given by its **size** method) and print out each position in the vector. Run the program with all the team members’ names (first, middle, and last) as arguments after the program name on the command line, and as the answer to this exercise (1) show the output from the program, and (2) say whether or not all of the command line arguments given to the program appeared in that output.

4. Change your program so that instead of pushing the command line arguments into a vector, it concatenates them into a C++-style string with spaces in between them. Wrap the completed string in an input string stream (of type `istringstream`) and then use its extraction operator (the `>>` operator) to pull off one argument at a time from the string stream into a C++-style string and then print each one on its own line. Again run the program with all the team members' names (first, middle, and last) as arguments after the program name on the command line, and as the answer to this exercise (1) show the output from the program, and (2) say whether or not all of the command line arguments given to the program appeared in that output.

5. Repeat the previous exercise, but instead of printing out each argument to `cout` print each one to a file using an output file stream (of type `ofstream`) and its insertion operator (the `<<` operator). Make sure to check whether or not the file opened successfully (if not print out an error message and return a non-zero integer value to indicate failure) and make sure that after all arguments are written to it that you also close the file. Again run the program with all the team members' names (first, middle, and last) as arguments after the program name on the command line, and open up the file in notepad or another editor. As the answer to this exercise (1) show the output from the program, and (2) say whether or not all of the command line arguments given to the program appeared in that output.

6. Close the editor you used to look at the output file produced in the previous exercise. Modify your program so that instead of reading all the command line arguments, it first checks that exactly one argument in addition to the program name was given (and if not prints an error message and returns a non-zero value) and uses that argument as the name of an input file, opens that file, reads in data from the file into a C++-style string using its extraction operator (the `>>` operator), and each time prints out the string to `cout` on its own line. The program should again check that the input file opened correctly and if not should print out an error message and return a non-zero value to indicate failure. The program also should close the file after it has read all its contents (which is indicated by the `>>` operator returning false, so one cool trick is to write a while loop that uses the `>>` operator in its iteration control test expression).

Run the program with the name of the file produced in the previous exercise as its only argument. As the answer to this exercise (1) show the output from the program, and (2) say whether or not all of the data in the file from the previous exercise appeared in that output.

## PART II: ENRICHMENT EXERCISES (optional, feel free to skip some and do ones that interest you)

7. Try running the program from the previous exercise with inputs that generate all of the error cases it handles, including running it: with more than one argument, with no arguments, or with the name of a file that does not exist. After the program runs, you can find out what it returned by immediately issuing the command `echo %errorlevel%` on the command line. For each case, show the error message, and also what value the program returned.

8. Repeat exercise 4 or 6 (or both) but instead of treating all the data as strings, give integer values to the program and read those values into integer variables. The string and file input streams will do the conversions for you, which can really ease some programming tasks (including some in lab 1).