Program 1: Hello Shell

CSCI 4547 / 6647 Systems Programming
Fall 2013

1 Goals

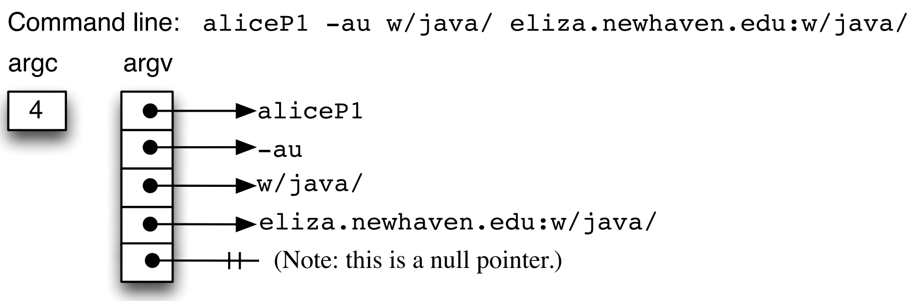
1. To access a Unix system. If you are running a Windows system, you must install a Unix system (Fedora Linux) and desktop (KDE) on your machine. Mac users already have a version of Unix. Be sure the Developer Tools are installed.
2. To open a shell in a terminal window on your system. I use tcsh. You will probably use bash.
3. To learn the form of and parts of a command line.
4. To learn about what a shell does with a command line, including quotes and wild cards.
5. To compile and run a C or C++ program without using an IDE. (Use your IDE to write it.)
6. To use standard file output in either C or C++. (Don't cut-and-paste or turn in screen shots).

2 The Program

- Write a program to print out command line arguments. Your main function should accept arguments from the command shell, as follows:

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

  The array named `argv` contains the white-space delimited text fields from the command line, with patterns expanded and quotes "managed"; argc is the number of elements in this array, excluding the last array element, which contains NULL.



Command line: aliceP1 -au w/java/ eliza.newhaven.edu:w/java/
argc
argv
4
aliceP1
-au
w/java/
eliza.newhaven.edu:w/java/
(Note: this is a null pointer.)

- In your program, open an output stream in append mode, using your own name and P1 as part of the file name:
  In C: `FILE* out = fopen( "JonesP1.txt", "a")`
  In C++: `ofstream myOut( "JonesP1.txt", ios::out | ios::app)`

  By using append mode, you will be able to store the results from all the runs in one file.

- To begin, print a dashed line, as a divider. Then, starting with argv[0], list, the parts of the command, one per line, with a label for each. For argv[0], print "command". If the arg starts with a "-", print "switch". Otherwise, print "argument". The output for one test should look like this:

  ```
  -----------------------------------
  command  aliceP1
  switch   au
  argument w/java/
  argument eliza.newhaven.edu:w/java/
  ```

- In C, use printf to print your results to the screen and ALSO use fprintf to write to your file. In C++, use `<<` to send output to both cout and your own stream.

3 Installation

1. If you need help installing Fedora Linux or KDE, ask Mark Morton.
2. If you want to run both Windows and Unix, the best way is to make Unix the basic system, install VirtualBox, and run Windows inside of the Virtual box. Second best is to make a dual-boot system.
3. If you can't get a Unix system installed by Tuesday, use one of the Macs in the lab for your homework. DO NOT procrastinate.

4 Testing

1. Navigate to the directory that contains your .c or .cpp file. Then use one of these compile commands for Program 1:

   ```
   gcc -std=c99 -Wall -o yourName yourSource.c
   g++ -std=c++11 -Wall -o yourName yourSource.cpp
   clang++ -std=c++11 -Wall -o yourName yourSource.cpp
   ```

   Example: `gcc -o aliceP1 main.c`

   This will produce an executable file (command) named "aliceP1" in the current working directory.

2. In that same directory, create two or three dummy files (they can be empty) with names ending in .bak
3. Execute the command `ls -l > JonesP1.txt` (Use your own name.)
   This will put a directory listing into your output so that I can grade your work.
4. Run your new command six times, using the tests below, but substituting the name of your executable file for my name.

   (a) `aliceP1`

   (b) `aliceP1 -o myCommand myMain.c myClass.c`

   (c) `aliceP1 w/java bash-3.2 ls -l`

   (d) `aliceP1 -au --verbose w/java/ kira:w/java/`

   (e) `aliceP1 -i "CSCI 6657" *.html > mytemp.txt`

   (f) `aliceP1 *.bak *.log`

5. When testing is finished, five of the tests will also be in JonesP1.txt and one will be in mytemp.txt. Write some comments at the end of your source code file that explain what you see and what you do not see for tests (e) and (f).
6. Turn in your code (with your explanations) and the two output files. Due Friday, Sept.13.