C/C++ programming, week of Oct. 24

Due date: Wednesday, Oct. 31. You may write your code in C or in C++ (but not both!) Code skeletons in each language are provided.

Introduction

The objective today is to learn how parse an execute line to make a standalone utility program. To illustrate, consider the standard Unix sort utility. You can invoke it from the command line by typing

```
sort -n file.dat
```

which causes sort to read its data from file.dat. (the explicit redirection <file.dat is not necessary). The command line also includes an *option*, -n, that asks sort to treat its input as numeric strings. The sort utility is quite flexible, insofar as if the file name argument is omitted, then it reads from the standard input:

```
myprog | sort -n
```

In this latter example, assuming that myprog generates a list of numbers to the standard output, the pipe *redirects* the output of myprog to the standard input of *sort*. This kind of "Tinkertoy" programming is very useful for small jobs, because you can write throwaway programs in a line (or a few lines) of code with little effort.

Command-line arguments

Command-line arguments are especially useful for running programs in scripts. *The ability to script a set of actions is especially useful when dealing with a large number of data files.* Suppose for example that you have 100 data files. Rather than answering a series of prompts for every file:

> What is the name of the file that you wish to sort?
> Is the data numeric or alphabetic?
> Do you want to sort in ascending or descending order?

you can specify the answers through the options. If your data files have a common suffix, say .dat, and they are all in the same directory (folder), then you can sort them with a POSIX shell script like this:

```
for file in *.dat
do
   base=$(basename $file  .dat)
   sort -n -r $file > $base.sort
done
```

The -r option to sort outputs the data from largest to smallest. The basename command strips the .dat suffix from the file, so the sorted output from output.dat goes into a new file called output.sort. (It is also possible to place the output into a temporary file, then overwrite the original file with the sorted data.)

Unix/Linux systems include a utility function, called getopt, that simplifies the process of parsing command-line options. (The Posix shell also has a built-in function of the same name to allow command-line parsing in shell scripts.)

Practice exercise. The files command_line.c and command_line.cc are C and C++ versions, respectively, of a simple program that reads a command line for two possible options, a and n. The latter takes one integer argument. You can compile the C++ version on Agave with the Intel compiler as

```
icpc -Wall command_line.cc -o command_line
```

and the C version with

```
icc -Wall command_line.c -o command_line
```

Try running the compiled program with

```
 ./command_line
./command_line -a
./command_line -a -n10
./command_line -an8 xyz.txt
```

and so on. See what happens if you invoke it with options other than a and n. Try adding other options (-b, say, with or without arguments). You do not need to turn in this program.

Homework exercise

Write a C or C++ program that computes the minimum, 25th quartile, median, 75th quartile, and maximum value in a set of numeric input. Your program should read from the standard input or from a file named on the command line, if present. Here are additional specifications for your program.

- Your program reads in a list of numeric values. They may be formatted in any manner: one per line or several per line, with blank lines allowed. The input

  ```
  1 2 3
  ```

  is equivalent to the input

  ```
  1
  2
  3
  ```

  and to

  ```
  1 2
  3
  ```

  and so on.

- If the input is empty, then your program prints nothing. (It must not crash.) If there is only one input value, then the output values are all the same. (The special file /dev/null acts like an empty file upon reading and discards all output upon writing.)
- By default, your program computes and outputs five values, as follows:

  min 25% value median 75% value max

  The median is the value at the 50th percentile. If *n* is odd, then the median is the value in the middle of the sorted list. If *n* is even, then the median is the average of the two middle values in the sorted list. (Similarly for the quartile values.)
- Do not decorate the output with extraneous commentary. For example, do *not* produce output like

  ```
  ================================================================
  Input file name: input.dat
  Number of input values: 1000
  Minimum value in the file: -23
  Maximum value in the file: 2314
  ```

  and so on. The rationale for a single line of exactly 5 values is that the output of your program can be used as input to awk or other postprocessing program without dealing with extra words.
- If no filename is specified, then your program reads from the standard input instead. Such a situation might arise if your program is run as part of a pipeline, for example:

  ```
  awk -f awkprog mydata.dat | quartile
  ```

  Here awk extracts some of the contents of mydata.dat and pipes it to quartile, which reads the values and outputs the quartile statistics.
- Your program must accept an option, -e, which, if present, specifies that you print only the extreme values (minimum and maximum) in the input; you do not need to sort the data in this case.
- Also include an option -h, which, if specified, prints a brief help message to the standard output summarizing the program arguments and behavior and exits.
- If the input contains non-numeric data, then your program must handle it gracefully; for example, you might print an error message and halt. (It is always helpful in an error message to provide a line number where a problem was encountered in the input file.) Your program must not crash in any case. Document what it program is supposed to do with non-numeric input.

Skeleton programs quartile.c (C) and quartile.cc (C++) are provided to get you started. You may write whatever additional functions that you need to compute the statistics. Supply the necessary additional code and create a makefile.

Your program must not generate any diagnostics when compiled on Agave with any of

```
icpc -Wall quartile.cc -o quartile
g++ -Wall quartile.cc -o quartile
icc -Wall quartile.c -o quartile
gcc -Wall quartile.c -o quartile
```

References

- The standard C++ library algorithm facility;
- The standard C library function qsort, with an example.

Submission instructions

Create a tar file with

```
tar -c -f quartile.tar quartile.cc Makefile
```

or with

```
tar -c -f quartile.tar quartile.c Makefile
```

Grading rubric

Plagiarism warning: Substantial copying of source code without attribution will result in a grade of zero on this assignment. Please see the syllabus for the rules. Oherwise, this assignment is worth 20 points, as follows.

Compiles without errors or warnings	3 points
Calls qsort (C) or the STL sort (C++)	3 points
Produces correct results in the specified format for all combinations of command-line options with numeric inputs of one or more values	6 points
Gracefully handles non-numeric input	2 points
Operates without error on empty input	2 points
Includes a working Makefile	2 points
Includes adequate and accurate documentation in the source code	2 points
Total	20 points