MC39 Homework 1

Max Hailperin

Due: February 13, 1996

1. You may recall from the prerequisite course, MC27, that one should allow 0.75 pounds of turkey per person unless one has a turkey weighing more than 12 pounds, in which case 0.5 pounds of turkey per person is adequate.

   (a) Write a procedure, `servings`, that receives the weight of the turkey in pounds as its argument and returns the number of servings as its value. Both the weight and the return value should be of the type `double`, rather than `int`, so as to allow fractional values, such as that 12.25 pounds of turkey would serve 24.5 people.

   (b) Add a `main` procedure which prompts for and reads in the weight, uses the `servings` procedure to do the calculation, and prints out the result. Here is a sample of my compiling and running this program; the `andretti 159%` is my computer's shell prompting me for a command (and similarly with the other numbers). The first command uses `CC` to compile my program from the source file `turkey.C`, producing the executable program `turkey`. The second and third commands are examples of my running `turkey`. The numbers `6` and `12.5` were typed by me:

```
andretti 159% CC -o turkey turkey.C
andretti 160% turkey
Enter turkey weight: 6
A 6 pound turkey would serve 8 people.
andretti 161% turkey
Enter turkey weight: 12.25
A 12.25 pound turkey would serve 24.5 people.
```

   (c) Change your `main` procedure so that it repeatedly prompts for turkey weights and prints out the number of servings for each.

When the user indicates they are done by typing an end-of-file indication (control-d on Unix systems like the SGIs), then your program should exit. The way you can tell that this has happened is that an input expression such as `cin >> weight` (or anything like that) has a false value when end of file is reached, so you can do

```
if(!(cin >> weight))
  // whatever you want to do on end of file goes here
// down here the weight has been read in and can be used
```

Here is an example of my using this new version; in the last line, there is an invisible control-d that I typed after the colon, leading to the program exiting and returning me to the next shell prompt:

```
andretti 165% turkey
Enter turkey weight (control-d to end): 6
A 6 pound turkey would serve 8 people.
Enter turkey weight (control-d to end): 12
A 12 pound turkey would serve 16 people.
Enter turkey weight (control-d to end): 10
A 10 pound turkey would serve 13.3333 people.
Enter turkey weight (control-d to end): 14
A 14 pound turkey would serve 28 people.
Enter turkey weight (control-d to end): andretti 166%
```

2. When lab monitors work in the computer lab, a "logbook" program records the times they work. This was one MC39 team's project a previous year. They actually wrote a collection of several related programs, one for logging in and out, one for printing out the records, etc. The interface between these programs is certain file formats that convey data between them. In this problem, you will write an additional program that allows an MCS lab monitor to determine how many hours he or she has worked so far this month.

   There is a file that contains all the data from the month, with one line per monitoring shift. Each line contains three numbers with colons between them. The first is the user identification number, or uid, of the monitor; the second is the starting time; and the third is the ending time. The times are measured in seconds since an uninteresting reference point. So, for example, the line

```
9041:3000:6600
```

would indicate that the monitor with uid 9041 worked a shift that lasted for 3600 seconds, i.e., exactly one hour, starting at time 3000 and ending at time 6600.

What you need to do is read these lines in one by one, until end of file is reached, and for each check whether the uid is the one you are looking for. If so, the difference between the start and end times should be added to a running sum. At the end, this running sum can be divided by 3600.0 to get the number of hours the monitor worked, and this can be output.

As in the first problem, you can check for end of file by testing when the input operation, such as `cin >> uid`, has a false value. The uid and start and end times can be input into variables of type `long`, and a `long` can also be used for the running total time. To skip over the colons, you can use `cin.get(c)` as in the book's example of converting character case.

One detail is how to know what the uid of the monitor running the program is; what you can do is put the line

```
#include <unistd.h>
```

at the top of your program file and then use the `getuid()` procedure to get the uid of the user running the program. (It'd be most efficient to do this only once, rather than for each line.)

To run the program and have it read its lines of input from a file rather than from the keyboard, you can use the `<` feature of the shell, as in

```
myHours <~max/MC39/hw1.data
```

which runs the myHours program with its standard input coming from `~max/MC39/hw1.data`. This is an MCS lab logbook file that I've fudged to have the uids of the MC39 students rather than those of the actual monitors. How many hours did you work? (Be sure to also test your program with inputs simple enough for you to easily check the answer.)

Incidentally, once you've successfully completed this problem, the only minor detail standing in the way of it being a useful addition to the

logbook system is getting it to automatically access the real data file, rather than needing input fed to it with <. Although our textbook hasn't shown how to do this yet, it is a very simple matter. So, this problem is a good example of the fact that even a minor amount of custom programming can be useful to someone, particularly when used together with other existing programs.