CS4023 Week08 Lab Exercise

Lab Objective: In this week's lab we will continue our focus on the `findvals` of last time. We will remove some of the output clutter by making optional the printing of a message every time a "hit" is encountered. Also today we will modify our program from last time so that it reports running time accurately, which will be needed for next week's lab. In that lab we will use the OpenMP library that will give us easy access to running our program using several threads.

Here's a quick summary of the tasks:

❶ Create a directory for this week's work

❷ Modify your "search loop" so that it now prints out the position and value of each matching element in the array *only if* the `-v` command-line option is given

❸ Secondly, modify the `findvals` program of last time so that instead of printing out an end time/date the number of CPU seconds is printed instead. You will learn a little more about how time is measured in a computer.

❹ In the weekly directory I have provided you with an executable that will be the definitive judge in any disputes concerning formatting of output, etc. Input to your program will be the same as for last week's.

❺ This week's lab will be due for handing in by the designated deadline. Once you have completed your program and your output matches mine please hand it in using the `handin` submission mechanism.

In Detail

❶ You can create this week's lab directory with

```
mkdir ~/cs4023/labs/week08
```

Now change your working directory to this since all of our compiling, etc. will be done in this directory.

❷ As a starting point for this week's lab you can use the source code for the program you wrote last time.[1] You should copy to this week's directory the three source files `findvals.c` and the two utils-related files `utils.[ch]`.

Some people like loadsa output, some like the bare minimum. We saw from last time that the program reported every element that matched the criteria and if we are only interested in knowing the number then this is just cluttering the output. You should implement a `-v` flag (for verbose) so that only if it is set will the exact hits be reported.

You will have seen from last time that the ordering of the `-r` and `-t` arguments was irrelevant. The same holds for the `-v` argument, too. That is, do not assume that it will appear as the first or last argument to the program.

❸ When logging the start time and date of a program accuracy down to the second is fine and this is what the `time()` system call gives you. However, if you want to accurately measure the clock for comparing running times you need something else. The function you need is `gettimeofday()` which has the ability to store microseconds but **this does not mean** it is accurate to this level. Due to imprecise implementations, etc. your mileage – accuracy – will vary. See a good discussion of the issues with the accuracy of a computer clock here.

The `C` struct that is used by `gettimeofday()` is different from the one used by the `time()` function of last week. This is because they do two different things: one is more concerned with time as in part of a date while the other, `gettimeofday()`, is more concerned about CPU clock ticks. The struct we need this week is called `timeval` and it really is just a pair of `int`s, one for the number of seconds since the beginning of UNIX time, 01.01.1970 and the other for "millionths" of a second. What is a bit more of a nuisance is that two different header files are required so make sure that you include the following:

```
#include <sys/time.h>
```

at the top of you program.

The Y2K38 Problem

Because UNIX systems use 01 Jan, 1970 as the epoch beginning – a reference point to you and me – and counts seconds from then in a variable of type `int`, problems could happen when the seconds count becomes larger than can be held in an `int`. On 32-bit machines this is due to happen in 2038 when the counter overflows. However, because 64-bit machines are becoming more commonplace nowadays the date when they overflow will be well after that date.
Ask Wikipedia for further information.

A good summary of the different time-related functions that are available in linux is available here. This link also discusses a more elaborate way of measuring CPU usage. If you read nothing else on this page please read the section on `gettimeofday()`.

A very useful example of `gettimeofday()`in action can be found here; this example code

[1]Alternatively, you can use the solution source code to last week's lab.

is provided in the class directory for this week's lab. The following is another example of the function being called. One the one hand the two codes call `gettimeofday()` twice but one works with microseconds as the unit while the code below converts both times to seconds, stored as `double`.

```
#include <sys/time.h>
     :
struct timeval tim;
gettimeofday(&tim, NULL);
double t1=tim.tv_sec+(tim.tv_usec/1000000.0);
     :
do_something_long_running();
     :
gettimeofday(&tim, NULL);
double t2=tim.tv_sec+(tim.tv_usec/1000000.0);
printf("%.6lf seconds elapsed\n", t2-t1);
```

The essential `gettimeofday()`

A final comment Becoming a good programmer requires learning good habits. Many of these habits will slow you down at first but over time they become second nature and are sure winners. These habits include building projects with makefiles, learning to use – and using! – a debugger all the time, and writing your code so that when you come back to a half-finished project in a few months you know what is what. One useful thing here is logging information on how your program was invoked from the command-line. Writing this information as part of the output at run-time can save a lot of head-scratching later.

logging run parameters

It is a good idea, whenever working on a coding project to have the program log information at the start of its execution. We have been doing this here by logging the start time and date but another very useful thing to log is how the program was invoked. That is, write out the command-line arguments. This is invaluable when you want to refer back to "old runs" of your program that you had saved to a file using standard output redirection.
Note: I am not asking you to log this information in this lab – it is just a good habit to get into for your own projects.

❺ This is the another lab exercies that you will be assessed on. To be assessed you will need to have the program written and working properly by the date in the handin schedule below.

The command for submitting this lab via the `handin` mechanism is:

```
~cs4023/progs/handin -m cs4023 -p w08
```

Labs that are to be handed in are open for handin-gin at 09.00 on Tuesday of the week it is *assigned* and, in order to get full marks, are due by 16.00 on Tuesday of the *following* week[2]; a lateness penalty applies to submissions made until 18.00, Friday after that. This gives you at least one week to work on each lab that is to be assessed and handin without penalty. The lab sheet will be available for reading prior to the first lab so that you can read it beforehand and come to the lab with questions.

Subject to last-minute changes, the planned schedule of lab assignments due for handin-g in are:

Lab. Week	Assessed	DueDate (16.00, Tue)
Week01	✗	
Week02	✗	
Week03	✓	Week04
Week04	✓	Week06
Week05	✗	
Week06	✓	Week07
Week07	✗	
Week08	✓	Week09
Week09	✓	Week10
Week10	✓	Week11
Week11	✗	

[2]There are a couple of exceptions where you will be given 2 weeks to complete the lab.