CMSC 106 Project #1 Fall 2011

Date assigned: Monday, September 19, 2011
Date due: Wednesday, September 28, 11:00 p.m.

1 Introduction

The purpose of this project is to become familiar with the tools needed to complete projects for this course and to practice input, output, variable types, and mathematical expressions.

2 Project specifications

Times are sometimes represented in computers as a certain number of seconds or minutes past a given starting time. Therefore, in programming, it is often necessary to convert hours and minutes to seconds, or vice versa. Your task in this project will be to write a small C program which prints CMSC106, and also reads a time, converts it to seconds, and prints two additional facts about it. This is just practice in using the facilities of C covered so far.

Write a C program which produces several lines of output as described (see the sample execution in Section 7):

- The first output line contains only the string CMSC106.
- Your program is then to ask the user to type a time, and read the time entered. The time will consist of two integers, representing hours and minutes respectively, with a single colon appearing between them. The time entered will be in 24–hour time, so the value 14:15 would signify 15 minutes past 2:00 p.m., and 1:15 would signify the time 15 minutes past 1:00 a.m.
- On the next line your program is to print the input time, the number of seconds past midnight this time will occur, and the percentage of how far into the day this time will be. The hours and minutes in the printed time should be separated by a colon, each printed using two positions. If the number of hours is a single digit it should be printed with a preceding blank space, and if the number of minutes is a single digit it should be printed with a preceding zero. For example, the time five minutes past eight should be printed as ␣`8:05` (using ␣ to indicate a printed space).

  The percentage figure should be printed as a number between zero and 100, equal to the nearest whole number not greater than the percentage value. For example, since a day is 24 hours, at time 12:00 it will be 50% over, and 75% over at 18:00. Also see the example below. The percentage figure must be immediately followed by a percent sign (%), not separated from the number by any space.
- The next line must contain the number of seconds between the input time and the end of the day (at 24:00).
- The last line tells the current time by a different definition. You are working with the beings on another planet that has observed how long an earth day is. (This is one revolution of the earth - so this is the same as defined by earthlings and beings on this other planet.) But they divide the day into bleeps, blips and seconds. Seconds have the exact same length of time as our seconds. There are 200 seconds in a blip. There are 54 blips in a bleep. There are 8 bleeps in a day. In their format they write the time in `bleeps-blips-seconds` where the bleeps are always written as a one digit value, the blips are always written as a two digit value (possibly with preceeding 0's), and the seconds are always written as a three digit value (possibly with preceeding 0's).Note also that there are dashes between each of the parts and no spaces much like a social security number. The planet is named the "Jan Planet" so this representation of time is called "Jan Planet time".

Print a single blank line between each item mentioned above (as shown in the example below), as this will help make your output more readable.

Ordinarily a program can contain additional things as long as it improves the readability, but because we are grading this using the submit server, the program can not contain any additional output or any modification to the output.

Lastly, although in a realistic program input which is read is usually checked for correctness, you should assume that the user of your program will always enter a valid time value, so you should not attempt to verify whether the numbers of hours and minutes read represent a proper time or not.

3 Development suggestions

3.1 Development strategy

Many people like to write their whole program, or at least have a good outline of it, before starting to type any of it in. That's fine, but **never** type in more than a short part of a program without stopping to compile it, run it on sample inputs, and verify that it works correctly so far before working on the next part. Of course, at each intermediate stage the part of the program you have entered will obviously not solve the whole problem, and it must at least be a complete and valid C program in order to compile it, but you will find that testing your project a part at a time insures you can find and fix any errors more quickly and easily. If your program works right at one stage and then doesn't work right the next time you stop to test it, the problem is likely related to the part you just coded.

3.2 Finding compilation errors

It isn't sufficient to just type your program in and submit it. You have to compile it and make sure it doesn't have any syntax errors (that it follows all the rules of valid C). But what if the compiler produces syntax errors and you're not sure what they mean?

- Usually the compiler tells you exactly which line in your program is incorrect. If you can't figure out what's wrong by looking at that line, compare a correct example of that type of statement in the text or your class notes with what you wrote.
- Sometimes the compiler can't always recognize a syntax error immediately (such as when you've forgotten the semicolon which must terminate every C statement), and doesn't detect it until the next line or statement. If the statement the compiler identifies as wrong looks correct to you, try examining the one before it for errors.
- Here are some compilation errors produced by the gcc compiler, and what they mean:

  Undefined symbol print and **ld: fatal: Symbol referencing errors:** (or `Printf`, or `scan`, etc., instead of `print`) This means you have misspelled the name of one of the standard library functions (`printf` as `print` or `Print`, `scanf` as `scan`, etc.). Every program component must be spelled exactly, and even one incorrect character will cause a program to fail to compile. Check every call to these library functions carefully.

  missing terminating " character: You probably forgot either the opening or closing double–quote mark in one of your `printf` or `scanf` argument strings.

  ld: fatal... unknown file type... File processing errors: The C compiler gcc requires that a program's filename end in ".c". That's a lowercase "c", as in a filename like "proj1.c". If your program's filename isn't in this format then rename it.

  parse error before...: This error often means you have mismatched curly braces (`{ }`).

3.3 Debugging and testing

Even when you've finished typing your program, you're not finished– you still have to test your program by running it on a variety of input values. Just because your program has no syntax errors doesn't mean it's correct– it could have semantic or logical errors. If your program doesn't work the way it should:

- Carefully trace your program through on paper for certain input values, keeping track of the current values of all of the variables.
- You may need to add debug `printf` statements to your program to print out the values of variables and calculations at various points. A debug `printf` is just a `printf` temporarily added to your program to help you figure out why it doesn't produce the right results. For example, you could print out the current values of variables and calculations at a certain point to make sure they have the values you expect. Add as many `printf`s as you need to determine what is true about the data currently stored. **Remember to remove any extra printf's before you submit the project.**
- If when you run your program you get the error "Floating exception (core dumped)", this means your program has an unrecoverable execution error. This most likely means you are either trying to calculate some expression containing a division and the denominator's current value is zero, or you are trying to perform some calculation with a `float` or `double` variable which has never been given a initial value and contains a garbage value. Add debug `printf` statements before every division printing the value of the denominator, and check carefully that all your `float` or `double` variables have initial values or that values are assigned to them or read into them before they are used in calculations.
- Another fatal execution error is "Segmentation fault (core dumped)". This may be related to calling `printf` with `&` operators preceding variables to be printed, or calling `scanf` without `&` operators preceding variables to be read into.
- If your program just hangs and doesn't do anything after you type values to be read into variables and you've pressed the enter key, you may have characters (such as the `\n` escape sequence) in your `scanf` format string besides format specifiers.
- If you've investigated the above and still can't figure out why your program doesn't work, come to office hours, and we can show how to track the problem down.
- Once you think your program is correct, test it against a wide selection of input values, not just those in the sample output section below.

3.4 Other hints

- Start to work on projects when they're assigned. If you end up having a problem which you can't solve on your own you'll have time to come to office hours for help.
- In addition to the automatic backups made of your extra course disk space, keep several backup copies of your program saved under different filenames or in different subdirectories. Before making any major changes to your program, copy it to a new backup file with a different name. This will save you a lot of time if you accidentally delete your file, or if it turns out that your changes were incorrect and you want to revert to the previous version.
- The **very last thing** you should do before submitting your program is to make sure it compiles, and test it again to make sure it produces correct results. Don't test your program, then add comments, then submit it. It only takes one incorrectly–typed character to cause a program to fail to compile. Test your program **after** adding your comments, not before. Better yet, add the comments while you're developing the program, not just at the end.

4 Project requirements

1. All of your projects for this course must have a comment near their top which contains your name, your Glue ID, your section number, your TA's name, and your university ID number (not your Social Security number).

2. All your C programs in this course should be written in ANSI C, which means they must compile and run correctly when compiled with the compiler `gcc`, with the options `-Wall`, `-Werror`, `-ansi` and `-pedantic-errors`. This will be much easier if you ran the setup program discussed in the lab section when you did the setup.

3. Please review the project grading policy in the course syllabus. Note that you will **lose credit** during grading if your program produces any warning messages when it is compiled. Note again that not only does your program have to compile correctly, it must also produce correct output as well, so you should check the results it prints carefully.

4. You **may not** use any C language features except those introduced **up through Chapter 3** of your textbook, plus those presented in lecture and discussion section while the material in those chapters was being described. Note that using any features of C other than these will result in **losing credit** when your project is graded. For this project you may not use arrays or conditionals or loops.

5. The Campus Senate has adopted a policy asking students to include the following statement on each assignment in every course: "I pledge on my honor that I have not given or received any unauthorized assistance on this assignment." Consequently you're asked to include this pledge in a comment near the top of your program. See below for important information regarding violations of academic integrity.

6. Your program should be written using good programming style and formatting, which for this project is considered to consist of:
   - having an initial block comment describing the purpose of and task solved by the program,
   - use of adequate descriptive comments throughout, to indicate what your program is doing and how,
   - neat and readable indentation and formatting, including consistent alignment of braces and indentation of subsidiary statements, and avoiding writing any statements or comments which are wider than the standard terminal window width of 80 columns,
   - writing clear and readable code,
   - use of meaningful and descriptive variable names,
   - use of symbolic constants for any special values which don't change and which are used in several places in your code, and
   - avoiding disallowed C language features as discussed above.

5 Submitting your project

1. Turn in your program using the "submit" program provided by running "submit". **This submits only the .c file containing your source code, not the executable version of your program**!

2. Your project must be electronically submitted by the date and time above to avoid losing credit. No projects more than three days late will be accepted for credit without prior permission and a valid, documented excuse, as described in the course syllabus.

   Note there is **no grace period** for project submissions (the submission deadline is enforced exactly at the time above, the 24–hour late deadline is enforced exactly 24 hours later, etc.) and exceptions cannot be made; see the course syllabus for details.

3. Only the version(s) of the project which you electronically submit can be graded; it is your responsibility to test the program you are submitting and verify that it works properly before submitting.

4. **Do not delay or wait until the last minute to submit your program.** If you do, and you have any trouble, it will be too late to come to office hours, and you'll lose credit on the project. But if you start early and have questions or run into any difficulty, you can come to the TAs' office hours for assistance.

   It is recommended that you complete and submit your project **at least one day prior** to the due date and then reread this project assignment to insure that you haven't missed anything important which could cause you to lose credit on your project. If you have, this will give you ample time to correct it.

6 Academic integrity statement

Please **carefully read** the academic honesty section of the course syllabus. **Any evidence** of impermissible cooperation on projects, use of disallowed materials or resources, or unauthorized use of computer accounts, **will be submitted** to the Student Honor Council, which could result in an XF for the course, or suspension or expulsion from the University. Be sure you understand what you are and what you are not permitted to do in regards to academic integrity when it comes to project assignments. These policies apply to all students, and the Student Honor Council does not consider lack of knowledge of the policies to be a defense for violating them. Full information is found in the course syllabus– please review it at this time.

7 Sample output

Here is a sample output assuming the executable version of their program is in a file named "proj1.x" and the prompt on the linux.grace.umd.edu system is shown as as ">. The seconds and percentage printed would vary depending upon the time which was entered. Underlined text is typed as input when the program is run, while everything else is its output.

```
> proj1.x

CMSC106

Enter a time, in 24-hour format (hours:minutes): 16:13

16:13 is 58380 seconds past midnight, and the day is 67% over.

There are 28020 seconds from this time until the end of the day.

Jan Planet time is: 5-21-180
>
```