

HW #3

CSC230 Section 001 Fall 2007

Due Date: Friday, September 14, 11:45pm EST
--

Purpose

The purpose of this assignment is to reinforce your understanding of flow of control, and writing functions, in C.

Individual Work

Homeworks are done individually; submit your own, original homework.

Getting Help

Come to my office hours, ask questions after class, go to the TA office hours, use the message board, or send email; all are good ways to get help.

Programming Requirements

As before, your programs need to compile with `gcc -Wall -std=c99` and execute properly on the Linux OS+Intel PC platform.

Programs must be formatted cleanly and consistently.

Commenting requirements are the same as before (see hw1).

Always return 0 upon successful completion of execution of your program, and something other than 0 otherwise.

Input errors that **could** happen, but that are not mentioned, will not be tested, and you do not need to worry about.

The Programs

pgm10.c :

Enter, format, comment, compile, test, and debug the following program.

- Output a question using the format specifier "Do you want to continue?\n".
- Then read a line of input from the standard input.
- If the input is a line consisting of any of "Yes\n", or "OK\n", print out "Continuing...\n" (and this completes successful execution of your program). The case of the input should not matter (e.g., all of "Yes", "YES", "yES", "ok", "OK", etc. are acceptable).

- If the input is instead a line consisting of "No\n", print out "Terminating...\n". (this also completes successful execution of your program).
- Any other line input by the user should result in the question being repeated, until the user has input an acceptable response, and a message is printed. It is guaranteed that eventually the user will give an acceptable response (i.e., you do not need to test for an EOF indicator). It is also guaranteed that there will be no white space (spaces, tabs) in the input.
- Do **not** use arrays, pointers, or string processing functions from the standard library for this program. Those will come shortly.

pgm11.c :

Enter, format, comment, compile, test, and debug the following program.

- The user will input a number n between 00 and 45, inclusive. n is guaranteed to be input as exactly 2 ASCII-encoded digits (e.g., '3' followed by '2'), followed by a newline; no error checking is required.
- Use **recursion** to compute the n th Fibonacci number, where the 0th Fibonacci number equals 0, the 1st Fibonacci number equals 1, and the n th Fibonacci number for $n > 1$ is equal to the $(n-1)$ th Fibonacci number + the $(n-2)$ th Fibonacci number. Output the result using the format specifier "The %dth Fibonacci number is %d\n". (note the spelling of "Fibonacci", please, so we don't have to adjust for variants).

Questions:

1. How long does your program take to execute when the value of n is 45? Specify on what type of machine (processor speed, amount of memory) you execute the program.
2. If it is slow, why do you think it takes so long, since the number of additions being performed is not that large? (hint: an iterative, or non-recursive, version of this program on my laptop took .001 seconds to execute).

Answer these questions as a comment embedded at the end of your program file pgm11.c. Do not submit a separate file for these answers.

pgm12.c :

Enter, format, comment, compile, test, and debug the following program.

- Read characters from the standard input until either of the following occurs: (a) the EOF indication is reached, or (b) 100 characters have been read.
- Write a switch statement that does the following for each input character:

- if it is a lower-case letter, output it to the standard output, but converted to upper case (e.g., for input 'a', output 'A', etc.).
- if it is an upper-case letter, output it to the standard output, with no change.
- if it is a digit, output the character to the standard output with no change;
- if it is the character '+', output the characters 'P', 'L', 'U', 'S' (i.e., convert the plus sign into the word PLUS).
- If it is a space or a tab or a newline, output a space character;
- For any other input character value, output the character '?'.

For instance, if the input is the following (newlines and tabs are shown explicitly as `\n` and `\t` below just so you can see what the effect is):

```
a + b + 123\nQ - R = s\n\t x + 6\n
```

the output should be

```
A PLUS B PLUS 123 Q ? R ? S X PLUS 6
```

pgm13.c:

Enter, format, comment, compile, test, and debug the following program.

A method of approximating square roots is the following: if x is a guess (approximation) for the value of \sqrt{a} , then the average of x and a/x is a better approximation. Write code that

- Reads in a value for a . The value of a will be input as exactly 4 ASCII-encoded digits (e.g., '3' followed by '4' followed by '7' followed by '6') followed by a newline; no error checking is required.
- Uses $x = a/2.0$ as the initial guess.
- Computes 10 successive approximations using the above method, all using single precision floating point variables.
- Outputs the 10 approximations using the `printf()` format specifier "Approximation %d is %16.12f\n".
- Structure your code so that the `main()` routine reads in the value of a , and outputs each successive approximation. `main()` should call a function which actually computes the next approximation.

Questions:

1. For an input value of $a = 3476$, how many decimal digits of your final approximation are correct, and how many iterations were required before the approximations stopped changing?
2. Answer the same questions, but modify your program to use double precision arithmetic for your computations. (You do not need to submit the code for that version of your program).

Put your answers as a comment at the end of file `pgm13.c`; do not submit a separate file with your answers.

pgm14a.c and pgm14b.c :

Enter, format, comment, compile, test, and debug the following program.

The famous mathematician Gauss invented an algorithm for computing the date of Easter Sunday, which is as follows:

1. Let y be the year (such as 1800, or 2007)
2. Divide y by 19 and call the remainder a . Ignore the quotient.
3. Divide y by 100 to get a quotient b and a remainder c .
4. Divide b by 4 to get a quotient d and a remainder e .
5. Divide $8 * b + 13$ by 25 to get a quotient g . Ignore the remainder.
6. Divide $19 * a + b - d - g + 15$ by 30 to get a remainder h . Ignore the quotient.
7. Divide c by 4 to get a quotient j and a remainder k .
8. Divide $a + 11 * h$ by 319 to get a quotient m . Ignore the remainder.
9. Divide $2 * e + 2 * j - k - h + m + 32$ by 7 to get a remainder r . Ignore the quotient.
10. Divide $h - m + r + 90$ by 25 to get a quotient n . Ignore the remainder.
11. Divide $h - m + r + n + 19$ by 32 to get a remainder p . Ignore the quotient.

Then Easter falls on day p of month n . For example, if y is 2001:

$a = 6$, $b = 20$, $c = 1$, $d = 5$, $e = 0$, $g = 6$, $h = 18$, $j = 0$, $k = 1$, $m = 0$, $r = 6$, $n = 4$, and $p = 15$.

- Write a program which implements this algorithm.
- The user will input the value of y as exactly 4 ASCII-encoded digits followed by a newline; no error checking is required, and the equivalent integer value is guaranteed to be between 1800 and 2007, inclusive.
- Print your result using the format specifier "Easter Sunday in the year %d fell on day %d of month %d\n". You do not need to convert the month number into a month name.

You are required to write your program **two** ways, called `pgm14a.c` and `pgm14b.c`:

In **version a**, each of the above steps is a function, i.e., function `int step1(void)` has no input parameters and (after reading the user input) returns the integer value of the year; function `int step2(int y)` has one input parameter, and returns the integer value of a ; etc. For steps that return two results, like step 3, use two functions; function `step3b()`, for instance, returns the value of b , and function `step3c()` returns the value of c . Do **not** use any global parameters in version a of your program.

In **version b** of your program, use **only** global parameters, i.e., still use a single function for each output of each step, but no function will have any input parameters. All inputs to every function must be global parameters. I do **not**

recommend this to you as a programming style; I simply want you to experience the difference for yourself.

Testing

Along with the homework, I have put on the class website executables for these homework problems that you can run (on the common platform). This will give you a reference standard to compare with your program's behavior. I have also provided examples of program inputs that you can use for testing purposes (not meant to be exhaustive, and we will test with different inputs than the ones provided to you).

Instructions for Submission

Submit exactly 6 files using <http://submit.ncsu.edu>. The names of these files are `pgm10.c`, `pgm11.c`, `pgm12.c`, `pgm13.c`, `pgm14a.c`, and `pgm14c.c`, all lower case. Do not zip the files first, or tar them, or pack them in any way, and do not put them in a subdirectory; just 6, separate, uncompressed files with the names shown.

Make sure your programs compile on Linux OS+Intel PCs cleanly, using the options given!

Grading

- 4 points for each of the 6 programs (total of 24 points) for:
 - following instructions on submission, with filenames given
 - formatting the program properly
 - acceptable amount of comments, including an “identification and purpose comment” at the head of the file
 - program compiles cleanly with no warnings or error messages, on the common platform
 - program returns the value 0 on successful completion
- 12 points each for successfully implementing each of programs `pgm10.c`—`pgm14b.c` according to the instructions
 - partial credit is pro-rated; if the program is half right, 6 points, etc.
- 4 points “freebie”, so the total adds to 100 ☺