Purpose: Repair Bitbucket problems; C input, functions, and character arrays

Details:

1. **[Make sure bitbucket is working.]** If you had problems committing and pushing your lab 1 problems, see me during lab to get the problems fixed!

2. **[Learn about `scanf`]** First, study the sample program “`input.c`” (downloadable from the website). The most important things to notice are the format specifiers (“`%d`” for `int`, “`%lf`” for `double`, “`%c`” for `char`, “`%f`” for `float`) and the use of the “`&`” operator in front of the variable names that are being input. The “`&`” operator is the “address of” operator; it points to the location where the input value is to go.

   Note that the `scanf` statement relies upon two things to “figure out” the input: white space (spaces, tabs, and newlines) and other “stop characters” depending on the format specifier. For instance, on user-supplied input `102.3`, this statement:

   ```c
   scanf("%d%lf", &i, &d);
   ```

   will set `i = 102`, `d = 0.30000`,

   ```c
   scanf("%d%c%lf", &i, &c, &d);
   ```

   will set `i = 102`, `c = '.'`, and `d = 3.00000`, and

   ```c
   scanf("%c%lf%d", &c, &d, &i);
   ```

   will set `c = '1'`, `d = 2.30000`, and then input will be paused awaiting for a value for `i` to be entered.

   There are a number of tricky cases, particularly with respect to character input and white space, as well as a large number of additional options for the format specifiers—we will not explore all of these in detail. (We will, however, introduce one more format specifier in problem ?? below.)

3. **[Write a Very Simple Function.]** Write a C program that has one function (in addition to `main`). The function (call it `mean`) takes two `int` parameters `i` and `j` and returns a `double` value equal to their arithmetic mean. **Place the `main` function first; include a function prototype for `mean` (see page 26).**
The `main` function should prompt the user to enter two integers, echo the user’s input values, and print the result of the function call; all output should be labeled in a reader-friendly manner. Sample run:

```bash
rroos$ ./prob1
Enter two numbers: 17 18
The average of 17 and 18 is 17.500000
rroos$ ./prob1
Enter two numbers: -40 -45
The average of -40 and -45 is -42.500000
```

4. **[Using the Math Library]** If you want to use math functions such as `sqrt`, `sin`, `cos`, etc., you must do two things: (1) add `#include <math.h>` to the top of your program, and (2) use the `-lm` option in your `gcc` command (this says to include the math library during compilation). Program “mathdemo.c” on the website demonstrates this.

5. **[Write More Functions.]** Write a C program that has two functions (in addition to `main`). The first function, call it `even`, takes an `unsigned int` parameter and returns true (i.e., a nonzero int value) if the parameter is even, false (i.e., an `int` value of zero) if the parameter is odd. The second function, call it `agmean` (for “arithmetic/geometric mean”) takes two `unsigned int` parameters `i` and `j` and returns a `double` value equal to their arithmetic mean (if `i + j` is an even number) or their geometric mean (if `i + j` is an odd number). The geometric mean of two numbers is the square root of their product: $\sqrt{i \times j}$.

   The format specifier for `unsigned int` is `%u`.

   **Place the main function first; include function prototypes for even and agmean.** The `main` function should prompt the user to enter two unsigned integers, echo the user’s input values, and print the result of the function call; all output should be labeled in a reader-friendly manner. Sample run:

   ```bash
rroos$ ./prob2
Enter two numbers: 1 2
agmean(1,2) is 1.414214
rroos$ ./prob2
Enter two numbers: 1 3
agmean(1,3) is 2.000000
rroos$ ./prob2
Enter two numbers: -1 -2
agmean(4294967295,4294967294) is 4294967294.500000
```

   (The weird result for the negative inputs will be explained in class this week.)

6. **[Manipulate some character strings.]** (See sample program `chars.c` on the website for an example of a program that manipulates a character string.)

   Write a C program that creates three strings (i.e., character arrays); they may contain anything, but they must all have different lengths. Then “combine them in an interesting way”.

---

Handout 5

Handed out on 7 September 2015
Your program’s purpose must be made clear to the user through both comments AND helpful labeling of the output. Examples:

- print them out “interleaved” (so if the strings are "one", "three", and "five", you would get "otfnhierveee")
- swap heads and tails of a certain length (e.g., swap the first three letters of "one" with the last three letters of "three" swap the first three letters of "three" with the last three letters of "five", and swap the first three letters of "five" with the last three letters of "one" to get "ree", "iveee", and "onee")
- Rotate the letters one place to the right, with the last letter of the first string rotating into the first position of the second string, the last letter of the second rotating to the beginning of the third, and the last letter of the third becoming the first letter of the first. (so, in our previous examples, we would get "eon", "ethre", and "efiv")
- [Preferable!] Come up with your own interesting string manipulation.

7. Push your programs, fully commented and including your name and the Honor Code Pledge in the header comments, into the repository you shared with me.

Your programs must be uploaded to your repository no later than 8 a.m., Monday, Sep. 7.

Questions about the lab? Bring them to class on Tuesday morning!