CSCI 1120 (Low-Level Computing), Spring 2011

Homework 2

Credit: 20 points.

1 Reading

Be sure you have read the assigned readings for classes through 9/26.

2 Programming Problems

Do the following programming problems. You will end up with at least one code file per problem. Submit your program source (and any other needed files) by sending mail to `bmassing@cs.trinity.edu`, with each file as an attachment. Please use a subject line that mentions the course number and the assignment (e.g., "csci 1120 homework 2"). You can develop your programs on any system that provides the needed functionality, but I will test them on one of the department's Linux machines, so you should probably make sure they work in that environment before turning them in.

1. (10 points) Newton's method for computing the the square root of a non-negative number x starts with an initial guess r_0 and then repeatedly refines it using the formula

   $$r_n = (r_{n-1} + (x/r_{n-1}))/2$$

   Repetition continues until the absolute value of $(r_n)^2$ - x is less than some specified threshold value. An easy if not necessarily optimal initial guess is just x.

   Write a C program that implements this algorithm and compares its results to those obtained with the library function `sqrt()`. Have the program prompt for x, the threshold value, and a maximum number of iterations; do the above-described computation; and print the result, the actual number of iterations, and the square root of x as computed using library function `sqrt()`. Also have the program print an error message if the input is invalid (non-numeric or negative).

   Here are some sample executions:

```
[bmassing@xena02]$ ./a.out
enter values for input, threshold, maximum iterations
2 .0001 10
square root of 2:
with newton's method (threshold 0.0001):  1.41422 (3 iterations)
using library function:  1.41421
difference:  2.1239e-06

[bmassing@xena02]$ ./a.out
enter values for input, threshold, maximum iterations
2 .000001 10
square root of 2:
```

```
with newton's method (threshold 1e-06):  1.41421 (4 iterations)
using library function:  1.41421
difference:  1.59472e-12

[bmassing@xena02]$ ./a.out
enter values for input, threshold, maximum iterations
-1 .00001 10
unable to compute square root of negative number

[bmassing@xena02]$ ./a.out
enter values for input, threshold, maximum iterations
xyz
invalid input
```

Hint: You may find the library function `fabs` (which computes the absolute value of a `double`) useful.

2. (10 points) Write a C program that reads characters from standard input to end-of-file and counts, for every possible character, how many times that character appears in its input. A simple way to read input a character at a time is with `getchar()`, which reads characters from "standard input" (the keyboard unless redirected). It returns an `int`, with the special value `EOF` when "end of file" is reached (control-D if input is from the keyboard under UNIX). Print counts only for the characters actually present. Here is a sample execution:

```
[bmassing@xena02]$ ./a.out
hello world
1234!@#$
count for character 10 (not printable) is 2
count for character 32 ( ) is 1
count for character 33 (!) is 1
count for character 35 (#) is 1
count for character 36 ($) is 1
count for character 49 (1) is 1
count for character 50 (2) is 1
count for character 51 (3) is 1
count for character 52 (4) is 1
count for character 64 (@) is 1
count for character 100 (d) is 1
count for character 101 (e) is 1
count for character 104 (h) is 1
count for character 108 (l) is 3
count for character 111 (o) is 2
count for character 114 (r) is 1
count for character 119 (w) is 1

21 characters in all
```

Notice that I should get the same result if I put the two lines of input in a file, say `hello.txt`, and run the program with the command

```
./a.out < hello.txt
```

(If you wonder about that non-printable character that occurs twice — it's the end-of-line character.)

Hints:

- Characters are actually small integers, and can be treated as such. If `c` is an `int` representing a character, you can print its numeric value and its value as a character with a line such as the following:
  `printf("%d %c\n", c, (char) c);`
- Possible integer values for characters range from 0 through `UCHAR_MAX`, where `UCHAR_MAX` is a constant defined in `limits.h`.
- Library function `isprint()` will tell you whether a character is "printable". (If it's not, then you may want to print only its numeric value, as my program does.)
- To get you started, here is a program that just reads input in the desired way and counts total characters: count-input.c[1].

[1] `http://www.cs.trinity.edu/~bmassing/Classes/CS1120_2011fall/Homeworks/HW02/Problems/count-input.c`