Information Processing 2 — Week 3 exercises

Complete as many exercises as you can in-class on your own laptop. You do not need to submit the answers online. Instead, ask an instructor to check your work during the class as soon as you finish answering each question. If you do not finish all the questions in-class, try to finish them outside class and have them checked *at the start of the next exercises class*.

1 [2 points] Powers of floating-point numbers

Recall from your maths classes that: $x^y = e^{y \log_e x}$

C has a library of mathematical functions that you can use after including <`math.h`> in your program. Two of those functions are

`exp(n)` returns e^n, the base-e exponential of n, and
`log(n)` returns $\log_e n$, the natural logarithm of n

both of which accept floating-point arguments and produce floating-point results. Use these two functions to implement a function

```
float power(float x, float y)
```

that raises the number `x` to the power of `y`. Test your function by printing the square roots of the numbers from `1.0` to `4.0` in steps of `0.5`.

```
1.00   1.00
1.50   1.22
2.00   1.41
2.50   1.58
3.00   1.73
3.50   1.87
4.00   2.00
```

Note that some operating systems do not include the maths functions automatically when you compile your program. If you get an 'undefined' error for `exp` or `log`, add '`-lm`' to your compile command to tell the compiler to include the maths library, like this:

```
$ gcc -o powers powers.c -lm
```

▷▷ Ask an instructor to check and record your work.

2 [2 points] Testing if a number is prime

An easy way to check if a number n is prime is to try to divide it by each integer between 2 and $n/2$ (inclusive). If the remainder on division by any of these integers is 0 then the number is not prime.

Write a function `int isPrime(int n)` that returns 1 if `n` is prime and 0 if `n` is not prime. Test your function by printing the prime numbers between 2 and 20.

Hint: The remainder (modulo) operator in C is '%'. (Formally, `(x/y)*y + x%y == x`.)

▷▷ Ask an instructor to check and record your work.

3 [1 point] Counting prime numbers

Write a program that counts (and prints) how many integers between 2 and 100,000 are prime. Check your answer, which should be 9592.

▷▷ Ask an instructor to check and record your work.

4 [2 points] Nested loops: multiplication table

Write a program that prints a two-dimensional multiplication table for the integers between 1 and 10 (inclusive).

```
 1   2   3   4   5   6   7   8   9  10
 2   4   6   8  10  12  14  16  18  20
 3   6   9  12  15  18  21  24  27  30
 4   8  12  16  20  24  28  32  36  40
 5  10  15  20  25  30  35  40  45  50
 6  12  18  24  30  36  42  48  54  60
 7  14  21  28  35  42  49  56  63  70
 8  16  24  32  40  48  56  64  72  80
 9  18  27  36  45  54  63  72  81  90
10  20  30  40  50  60  70  80  90 100
```

Hint: the easiest way to do this is with two `for` loops, one inside the other. The outer loop runs once for each row in the table and the inner loop runs once for each column in the current row.

▷▷ Ask an instructor to check and record your work.

5 [1 point] Printing long lines

Write a program that reads lines from the input and prints only those that are longer than 80 characters. Precede each printed line by a count of how many characters are on that line. Set `LINEMAX` to 100 and format each line of output accordingly.

Test your program using `03-sample.txt`, which should produce output that begins:

```
91 "!niaga krow taht odnu ot si ti drah woh dna ,eil a eveileb elpoep ekam ot si ti ysae woH"
84 ".deloof neeb evah yeht taht meht ecnivnoc ot naht elpoep eht loof ot reisae si tI"
```

▷▷ Ask an instructor to check and record your work.

6 [2 points] Reversing the order of characters in a line

Write a function `reverse(char s[], int len)` that reverses the contents of the string `s` of length `len`. Use your function to write a program that prints each line of the input with its characters reversed. Test your program on `03-sample.txt` which should produce output that begins:

```
"The welfare of humanity is always the alibi of tyrants."
-- Albert Camus
```

Hints:

- Swap the first and last characters, then the second and second-to-last, and so on. Stop when you reach the middle.
- It is very easy to leave the newline in its correct location, at the end of each line of input.

▷▷ Ask an instructor to check and record your work.

Challenges

7 [bonus point] Trimming blanks from both ends of a line

Write a function that removes spaces and tabs from the start and ends of a string. Use it to write a program that trims spaces from input lines and prints them, but which ignores (does not print) completely blank lines. For non-blank lines, print each line preceded by its length (after trimming). Hints:

- The simplest approach to trimming the line is to first remove spaces, tabs, and newline from the end (by repeatedly moving the terminating nul character left to overwrite a space, tab, or newline). Then do the same from the start (by copying the line over itself, after skipping over any initial spaces, tabs, or newline).
- You cannot know if you should ignore a line until after you have finished trimming it.
- If you write a function to trim the line, it could return the new length of the line which makes the main program easier to write.

Test your program on the file `03-blanks.txt`. The output should be:

```
3 one
3 two
5 three
4 four
4 five
3 six
5 seven
3 i    j
1 .
1 .
1 ?
```

▷▷ Ask an instructor to check and record your work.

8 [bonus point] Copying strings and using small input buffers

Write a function `int copy(char to[], char from[])` that copies the (nul-terminated) string in `from` to the array `to` (including the terminating nul character). It should return the number of characters in the string (excluding the nul at the end).

Test your function by writing a program to print the longest line read from `03-sample.txt` with `LINEMAX` set to (at least) 100. The longest line should be:

```
".tnemezama tnatsnoc ni dnuora klaw su fo tser ehT  .peelsa dnuora klaw elpoep fo tnecrep yteniN"
```

When the program works, reduce `LINEMAX` to 50. The program should now produce the wrong result:

```
".stnaryt fo ibila eht syawla si ytinamuh fo era
```

One way to improve the behaviour with a small `LINEMAX` is to modify `getchars()` so that it reads entire input lines, up to the final newline or end-of-file, but only stores at most `LINEMAX` characters in the `line` buffer (including the terminating newline and nul characters). Its result should be the actual number of input characters read, even if that is larger than the number of characters stored in `line`. With this change your output should be the first 48 characters of the longest line, which is not perfect but is a lot better than producing entirely wrong output.

```
".tnemezama tnatsnoc ni dnuora klaw su fo tser e
```

▷▷ Ask an instructor to check and record your work.