Homework 1 -- due Friday 17.11.2017, 23:59:59

Problem 1: Hello World (1p)

This really is a test problem to get you acquainted with the system.
Your Program should generate an output; do not read any input data. Submit a C program that does nothing else but output (using C's `printf`) the text "`Hello World!`" (without the ""), followed by a line feed ('`\n`'). Afterwards, let the program terminate correctly (remember to use "`return(0);`"). Note the capital letters '`H`' and '`W`', the whitespace between '`Hello`' and '`World`', no whitespace after '`World`', and the exclamation mark ('`!`') after '`World`'. Your output has to match "`Hello World!\n`" **exactly**. (Just to be sure: no colors or bold print on your output, just text! And the '`\n`' should not be printed as text on screen, but instead denotes a single line-feed control character).
Hint: You need to provide a complete C program, with statements such as "`int main() {`" etc. Not just one line of code.

Example Input	Required Output
	`Hello World!\n`

There is your first point. Congratulations!

Problem 2: Multiply By 2 (3p)

This really is a second test problem to get you acquainted with the system.
In this problem you should learn how to read in data from the standard input. The homework system sends test input to your program as if someone was typing on a keyboard. Use C's "`scanf`" function to parse input data.

Read an unknown quantity of numbers from the standard input (each number is an integer value followed by a line feed '`\n`'); and for each number read, your program shall print the number multiplied by two. You need to print an integer value followed by '`\n`'; no decimal point etc. **Do not print anything else on the virtual screen.**

"`scanf`" will tell you when the input data is used up (scanf returns EOF).

I recommend reading one number at a time and printing the result for this number before continuing with the next number.

Example Input	Required Output

`4\n`
`7\n`
`12\n`
	`8\n`
	`14\n`
	`24\n`

`5\n`
`3\n`
`9\n`
`34\n`
`4000\n`
`43\n`
`3\n`
	`10\n`
	`6\n`
	`18\n`
	`68\n`
	`8000\n`
	`86\n`
	`6\n`

10\n 0\n 45\n 300\n 3444\n 322\n 17\n	20\n 0\n 90\n 600\n 6888\n 644\n 34\n
2\n 4\n	4\n 8\n

Problem 3: Bubble Sort (10p)

Your program needs to read a single line of ASCII characters. The characters are constrained to '**A**' - '**Z**' only. The length of the line is unknown, but well less than 100 characters. The line is terminated by '**\n**'. Sort all individual characters using the bubble sort algorithm. After every complete pass through the data print the intermediate result terminated by '\n' (i.e. the partially sorted list). Print the final sorted list exactly once at the end of your program.
Hint: you can use `n=scanf("%s\n", s); l=strlen(s);` (where s is declared as `char s[100];` and n, l as `int n,l;`) to read the whole line at once and afterwards determine the number of characters read for this line.

Example Input	Required Output
ABC\n	ABC\n
EDCBA\n	DCBAE\n CBADE\n BACDE\n ABCDE\n
H\n	H\n
GEXKAL\n	EGKALX\n EGAKLX\n EAGKLX\n AEGKLX\n

Problem 4: Insertion Sort (10p)

Your program needs to read a single line of ASCII characters. The characters are constrained to '**A**' - '**Z**'. The length of the line is unknown, but well less than 100 characters. The line is terminated by '**\n**'. Sort all individual characters using the insertion sort algorithm. Start sorting with the second element from the beginning of the line (as the first cannot get inserted anywhere to its left). After handling an element print the intermediate result terminated by '**\n**' (i.e. the partially sorted list). Note that the output from handling the last element represents the final sorted list; so no need to create an additional output.

Example Input	Required Output
ACB\n	ACB\n ABC\n
EDCBA\n	DECBA\n CDEBA\n

	BCDEA\n ABCDE\n
H\n	H\n
GEXKAL\n	EGXKAL\n EGXKAL\n EGKXAL\n AEGKXL\n AEGKLX\n

Problem 5: Merge Sort (20p)

Your program needs to read a single line of ASCII characters. The characters are constrained to '**A**' - '**Z**' only. The length of the line is unknown, but well less than 100 characters. The line is terminated by '**\n**'. Sort all individual characters using the merge sort algorithm.
Remember that this algorithm uses a "divide&conquer" approach, by which it repeatedly breaks the sequence of characters into smaller portions before sorting (left half and right half). For your convenience, in this problem the number of input characters is a power of 2 (so you can repeatedly divide your input in halves if required).
Use **recursion** to implement merge sort; i.e. make your own "sort" function call itself recursively with smaller parts of the data. Sort the "left half" first. Whenever one instance of a recursion finishes print the **result of this recursion only**, followed by '**\n**'. Print the final sorted list exactly once at the end of your program.
Hint: Have your main program call your sort function with the full character sequence to start the recursion; this call will typically also print the final solution.

Example Input	**Required Output**
ABCD\n	AB\n CD\n ABCD\n
HGFEDCBA\n	GH\n EF\n EFGH\n CD\n AB\n ABCD\n ABCDEFGH\n

Problem 6: Breadth First Search (BFS) (15p)

Your program shall read in a graph in adjacency list format and perform a breadth first search (BFS) on the graph, starting at the node first read. Each node's name is a single letter '**A**'-'**Z**', so you can safely assume less than 27 nodes for your graph. The format of each line to be read in is specified as
<nodeName>-<string of neighboring node names>\n (see example below).
The requested output of your program is one line of node names followed by '**\n**', in which the order of names represents one possible exploration following breadth first search.
Note (1): for most inputs several different outputs are possible. Here we only ask you to find one of them!
Note (2): all graphs here are undirected, i.e. if a connection **A-...B...** exists you will also find a connection **B-...A...**.