University of Zurich[UZH]

s.e.a.l. software evolution & architecture lab

Programming Languages and Paradigms Spring 2025

Assignment 1

Submission Details

You can submit your solutions to this assignment in the PLP OLAT course until Sunday, Oct 12, 2025, at 11:59pm CET. Submit a single .zip that includes all relevant files.

You must follow these submission guidelines:

1. Source code files you submit should compile/run without errors.
2. Provide a README on how to compile or run the code.
3. Also include compiled binaries if there are any.
4. For each task, also submit screenshots of the task compiling and then running in your command line, IDE, or programming environment.

With regard to your implementation, mind the following:

1. It's up to you how exactly you implement the programs internally, as long as you're using the assigned programming language. The tasks only describe the expected behavior of the programs.
2. Write idiomatic code. Your submission will be graded based on its quality in the context of the programming language that you're studying. Add classes or other abstractions as you see fit.
3. Whenever a task refers to 'lists', 'functions' or other data structures and programming concepts, use the closest equivalent in your language (*e.g.*, arrays, lists, sequences, methods, routines, etc.)
4. You are allowed to use programming aids (LLM). Please specify if you do so as I'm interested in how well they works for your language. Feel free to include chat screenshots. There is no penalty and your solution will be judged solely based on its quality.
5. Likewise, you are allowed to copy solutions from the internet, but please specify the source. Again, no penalty.

1. Task: Hello, World!

Write a command line program that prints the string "Hello, World!" including a line break (newline) at the end.

2. Task: Parameterized Fizz Buzz

The Fizz Buzz Program usually prints the numbers from 1 to 100, however substituting numbers divisble by 3 with “Fizz”, those divisible by 5 with “Buzz” and those divisible by both 3 and 5 with “FizzBuzz” (see `https://en.wikipedia.org/wiki/Fizz_buzz`).

For this task, write an interactive command line program with the purpose of generating the Fizz Buzz sequence from m to n for two arbitrary condition numbers x and y. The sequence should be printed on a single line.

The program should behave as follows:

1. When first run, it should print the following sentence: “Please enter four positive integer numbers (m n x y) separated by one or more blank spaces or type quit.”.
2. The program should then wait for a line of user input. The user is now expected to enter four positive integers followed by a newline (hitting return/enter). Here are some examples of valid input:
   - `1 100 3 5` (the original Fizz Buzz)
   - `1  100  3 5` (any number of spaces between numbers is permitted)
   - `1 100 8 6` (Fizz if div. by 8, Buzz if div. by 6, FizzBuzz if div. by 8 and 6)
   - `5 250 8 6` (same as above but from 5 to 250)
   - `11 12 20 21` (this would just print the numbers 11 and 12)
   - `1 10 1 2` (this would alternate between printing “Fizz” and “FizzBuzz” 10 times)
   - `quit` (this ends the program)
3. When the user has entered a line, the program should perform the following sanity checks.
   - Anything other than four positive integers or the word “quit” should be rejected.
   - The first number must be smaller than the second number.
   - The third and fourth numbers must differ.

   If a sanity check fails, the program should print "Invalid input, please try again". Feel free to give more accurate error messages depending on which sanity check fails – but you don’t have to.
4. If valid input has been entered, the program should print the appropriate FizzBuzz sequence on a single line, each element separated by a single blank space. There should be a newline at the end.
5. The program should then expect another line of input.
6. If the user types “quit”, the program should terminate.

3. Task: Encode, decode

Implement two functions that will encode and decode strings composed exclusively of the 26 lower case ASCII characters. Strings are encoded by adding additional characters between the original ones and applying a ROT operation[1].

Note that you don't need to necessarily follow the exact order of operations, as long as the result is correct.

Encode: The encoder is supplied with three parameters: the string to be encoded, an integer offset for the ROT operation and an additional string of characters. It should execute the following operations:

- Convert the sequence of ASCII characters to their decimal values[2].
- Between each value in the resulting sequence, insert the decimal value of a character taken from the additional string. When running out of characters, start from the beginning.
- Execute the ROT operation using the provided integer offset (which could be *any* integer).
- Convert the resulting integer sequence back to a string.

For example, given the string to be encoded $abcde$, an offset -5, and an additional string jjh, the operations would result in the following intermediate and final values:

1. Convert $abcde$ to $[97, 98, 99, 100, 101]$
2. Insert values (106, 106, 104) from the additional string (jjh) between the existing characters: $[97, 106, 98, 106, 99, 104, 100, 106, 101]$
3. Apply ROT-5: $[118, 101, 119, 101, 120, 99, 121, 101, 122]$.
4. Convert the result back to a string: $vewexcyez$

Decode: The decoder executes the reverse operation of the encoder. It is supplied with two parameters: the encoded string to be decoded and the integer offset used for the ROT operation. It should execute the following operations:

- Convert the sequence of ASCII characters to their decimal values.
- Delete/skip/omit every second element
- Unapply the ROT operation using the negated value of the provided integer offset.
- Convert the resulting sequence back to a string.

For example, given the string to be decoded $vewexcyez$ and an offset -5, the operations would result in the following intermediate and final values:

1. Convert $vewexcyez$ to $[118, 101, 119, 101, 120, 99, 121, 101, 122]$.
2. Skip every second element: $[118, 119, 120, 121, 122]$.
3. Unapply ROT: $[97, 98, 99, 100, 101]$
4. Convert the result back to a string: $abcde$

[1] https://en.wikipedia.org/wiki/ROT13
[2] https://en.wikipedia.org/wiki/ASCII#Printable_characters

4. Task: Snippets

Implement the following problems. You may implement all of them in one scope/file, embed them in a class or similar structure or have them in one scope or several files – it's up to you. Feel free to implement helper functions as needed.

Potential solutions to most of these tasks can probably be found online. You are encouraged to reach your own solution first, but you are allowed to copy from the internet at no penalty. If your solution is heavily inspired by an online solution, please provide the source url as a comment in the source code.

a). Quicksort

Implement the Quicksort algorithm. Note the following guidelines:

- Your solution need not be efficient. Try finding the most elegant and/or idiomatic solution. Try to understand why it may not be the most efficient and leave an appropriate comment in the source code.
- If your language supports something like generic types and/or a comparison interface that multiple types could implement, then make sure your implementation works with different types. If your language does not, implement Quicksort at least for integers, floating point numbers and strings (using the rules outlined at `https://en.wikipedia.org/wiki/Alphabetical_order#Ordering_in_the_Latin_script`) for sorting strings if the language has no built-in way of ordering strings.
- Call your implementation with a few examples and print the results.

b). Call an external command

Call an external command and retrieve it's output. Call one of these depending on your operating system, they should be available globally (if not, specify the full path to the executable in your source):

- Mac or Linux: `uname -a` (typically at `/usr/bin/uname`)
- Windows: `systeminfo` (typically at `C:\Windows\System32\systeminfo.exe`)

Print up to 10 lines of the output received from the command.

c). External Libraries

Write and execute/call a procedure which will download the following XML file and parse it. The program should calculate the total price of all books and then print the number of books and the total price.

`https://www.w3schools.com/xml/books.xml`

Use existing libraries (built-in or external) to perform the download and parse the XML.