CS1010E Practical Exam 1

AY 2023/2024 Special Term 2

Grading Guidelines:

- No mark will be given if your code cannot run, namely any syntax errors or crashing.
  - Your tutors will just grade what you have submitted and they will **not** fix your code.
  - Hint: Comment out any part that you do not want.
- Workable code (that can produce correct answers) will only give you partial marks. Only good and efficient code will give you full marks
- Marks will be deducted if your code is unnecessary long or hard-coded.
- You must use the same function names as those in the skeleton given
- You ***cannot import*** any package, library or functions. Your solution cannot use other data structures except List and string (ie., not allowed to use tuples or dictionaries or sets or objects); you cannot use *list comprehension*, *higher-order functions,* or *recursion* to solve your problems. Lastly, you cannot use the built-in `sort` function for List structures.

Important Note on Code Saving

- Throughout the exam, regularly save your code. Never develop your code in Python Console/Shell/interpreter provided for you by GLIDE.
- You should test your code in GLIDE mainly. When you submit your code to Coursemology, there are **only maximum of five attempts** for you to run your code there. If you use them up, the consequence is irreversible. Public test cases are provided in the exam template for you to test your program in GLIDE. As such, do not expect to have abundant time to test your code during submission to Coursemology.
- If your code involves execution of other support functions, remember to also submit those support functions together with the main function.
- Remember to click on **FINALIZE SUBMISSION** button to complete the coursemology submission.

Problem Dependency

- There are three Tasks to be completed. Task 1 has two sub-tasks, and it is advisable to solve subtask 1 before working on subtask 2.
- Not all questions are of the same level of difficulty. Go through all questions first to decide the order in which you solve the problems to maximize your opportunity to score.
- Cheatsheet in pdf format is provided together with the question paper. You are not allowed to bring in other cheatsheet.

Task 1: Cool Winners [30 marks]

Cool Clan always gives out presents to some lucky clan members who participate at the party organized to celebrate the birthday of some elite clan member. They call these lucky members the Cool Winners. To qualify for being a Cool winner, the selection committee looks at the last four digits of each participating member's identification number (ID, of which the last four digits definitely do not begin with digit "0"), and certifies him/her as a Cool winner if:

a. the last 4-digit ID contains the age of the birthday man/woman (who can be over 100 years old), and
b. the last 4-digit ID does not contain multiples of an unlucky number identified by the birthday man/woman.

For instance, given the last four digits of the IDs of three participating members are: 5686, 8062, and 8644. If the age of the celebrated birthday man/woman is 86, then only IDs having 5686 and 8644 are qualified; next, if the unlucky number is 4, then between these two, only 5686 is the winner, since it is the only one not a multiple of 4.

Sub-Task 1: Lucky function [15 marks]

Write a python function, **`lucky(num,age,unlucky)`** that takes a positive number of four digits (**`num`**), a positive number representing **`age`**, and an **`unlucky`** number, and returns a Boolean value. Specifically, it returns **`True`** if the number **`num`** satisfies the two selection criteria mentioned above. Otherwise, it returns **`False`**. (You can assume that **`num`** has only four digits.)

Sample runs:

```
>>> lucky(5686, 84, 4)
False
>>> lucky(5686, 86, 4)
True
```

Sub-Task 2: List of Cool winners [15 marks]

Write a python program, **`select(members, age, unlucky)`** that receives a list of participating members – represented by their respective last 4 digits of their IDs (all are numbers), the age of the celebrated birthday man/woman and the unlucky number this man/woman has identified, returns a list of numbers from the members list that satisfy the two selection criteria. The definition of lucky function is provided; you can call it though you cannot see it. Alternatively, you can provide your own; you can define additional functions to aid in your computation, but you should remember to include their definitions in your final submission.

Sample runs:

```
>>> lst2 = [2587, 5686, 4790, 1082, 8644, 9590]
>>> select(lst2,86,4)
[5686]
>>> lst3 = [3376, 4331]
>>> select(lst3,33,13)
[3376, 4331]
```

Task 2: Cargo Lifting [35 marks]

Cool Company uses a group of forklift trucks to load cargoes into ships.

Cargoes come with different weights, and they are piled up in stacks. A forklift truck can lift up several cargoes at a time from top of the stack, bounded that a maximum weight limit. Given:

a. A non-empty list (of **`cargoes`**) representing a stack of cargoes (represented by positive integers), with the leftmost/rightmost list element representing the weight (with "100KG" as the unit of measurement) of the cargo at the top/bottom of the stack, respectively;
b. A positive number (**`cap`**) representing the maximum weight the forklift truck can lift up in one go.

Write a program **`forklift(cargoes,cap)`** to return the shortest list of indexes (in increasing order) into the cargoes list where each index in the list indicates the point in which the truck can lift up the existing piles of cargoes so that it takes minimum number of rounds of lifting to clear the cargo stack.

For instance, given the cargo list **`[1, 2, 3, 2, 3, 1, 2, 3, 1, 1, 2]`** (depicted in the picture shown on the right) and the maximum weight a forklift truck can pick up is `6` (x100KG) units. Then, the truck will take minimally four rounds to clear the cargo stack; the points (indices) at which cargoes to be lifted are shown below. Here, the cargo list is broken up into 4 sub-lists (underlined), each sublist has a total weight of not more than 6 units, and represents the current pile of cargoes that the truck can lift up at one go. This is the minimum number of sublists needed to clear the cargo list.

	[1, 2, 3,			**2, 3, 1,**			**2, 3, 1,**			**1, 2]**	
Index:	0	1	**2**	3	4	**5**	6	7	**8**	9	**10**

The (indexed) positions in the list for the truck to lift up the cargoes are shown here: `[2,5,8,10]`. (Note that there can be more than one possible solution with the same length of the returned index list, and you just need to provide one solution.)

Lastly, if it becomes impossible for the forklift to clear all the cargoes in the stack, the forklift will return an empty list `[ ]`, indicating no solution can be found.

Sample runs:

```
>>> cargoes = [1, 2, 3, 2, 3, 1, 2, 3, 1, 1, 2]
>>> cap = 6
>>> forklift(cargoes, cap)
[2, 5, 8, 10]
>>> cap = 7
>>> forklift(cargoes, cap)
[2, 5, 9, 10]
>>> cap = 2
>>> forklift(cargoes, cap)
[ ]
```

Task 3: Line Editor [35 marks]

During the "line editor era", computers accept keyboard inputs line by line. That is, before the "ENTER" key is pressed, whatever have been typed are captured by the computer as a character string; including the characters that are meant for editing that character string.
For instance, in the following string received by a computer:

Not a pretty sight, and>8 someone you better look >005into this.

Here,

a. We scan through characters in the string from left to right to detect the existence of any **forward delete (">") operator**.
b. The substring "**>8**" is encountered; it indicates that the next 8 characters (ie., the text " **someone**" – including a space as the first next character) are to be deleted.
c. The substring "**>005**" is encountered next; it indicates that the next 5 characters (yes, "005" means number 5; ie., the text "**into** ") are to be deleted.
d. Consequently, the resulting text after editing is:

Not a pretty sight, and you better look this.

In general, here are the rules for forward delete:

a. A forward delete starts with "**>**" may be followed by an arbitrary number of digits (from "**0**" to "**9**"). The digits combined indicate the number of following characters to be deleted, regardless of whether these characters represent delete operators. Thus:
   (i) "**abc>04uvxyz**" will result in a new string "**abcz**" with four characters appearing after the operator and the number (ie., "**uvxy**") being deleted.
   (ii) "**abcdef>00xyz**" will produce a new string "**abcdefxyz**" with no character getting deleted because the associated number is 0.
b. If the number of characters to be deleted is more than the available characters, all characters will be deleted. Thus:
   (i) "**abcdef>40xyz**" will produce a new string "**abcdef**" with all characters after "**>**" being deleted.
c. If "**>**" is not followed by any digit, one character will be deleted. Thus:
   (i) "**abcd100>xyz**" will produce a new string "**abcd100yz**" with the "**x**" after "**>**" being deleted.

Write a python program `edit(string)` to perform these editings.

Sample runs:

```
>>> edit('Not a pretty sight, and>8 someone you better look >005into
this.')
'Not a pretty sight, and you better look this.'

>>> edit("abcdef>00xyz")
'abcdefxyz'

>>> edit("abc>04uvxyz")
'abcz'
```

```
>>> edit("abcdef>40xyz")
'abcdef'

>>> edit("abcd100>xyz")
'abcd100yz'
```

This is the end of this paper.