INDIAN STATISTICAL INSTITUTE

MTech(CS) I year 2022-2023

Subject: Computing Laboratory

Assignment 1

Due date: 09:00AM, October 17, 2022 Total: $4 \times 15 = 60$ marks

Q1. Many programming languages provide *map*() and *filter*() functions that work as follows.

map. Given two sets X, Y, $map(L, f)$ takes as input a list $L = [l_0, l_1, \ldots, l_{N-1}]$ of elements from X, and a function $f : X \to Y$, and returns $f(L) \triangleq [f(l_0), f(l_1), \ldots, f(l_{N-1})]$, a list of elements from Y.

filter. Given a set X, $filter(L, g)$ takes as input a list $L = [l_0, l_1, \ldots, l_{N-1}]$ of elements from X, and a function $g : X \to \{\text{TRUE}, \text{FALSE}\}$, and returns $L' \triangleq [l_t \mid 0 \leq t < N,\ g(l_t) = \text{TRUE}]$.

Implement the `map` and `filter` functions in C. The functions should have the following prototypes:

```
void *map(void *L, unsigned int N,
          size_t domain_elt_size, size_t range_elt_size,
          void (*f)(void *input, void *output))

int filter(void *L, unsigned int N, size_t domain_elt_size,
             int (*g)(void *input))
```

where

- `L` is the input list;
- `N` is the number of elements in `L`;
- `domain_elt_size` (and `range_elt_size`, resp.) correspond to the size of each element of the domain and range of $f : X \to Y$;
- `f` is a pointer to a C function that implements f (`input` is a pointer to an element of the domain X, and `output` is a pointer to an existing chunk of memory that is just big enough to store an element of the co-domain Y); and
- `g` is a pointer to a C function that implements g (`input` is a pointer to an element of the domain X; `g` returns a non-zero value if $g(*\texttt{input}) = \text{TRUE}$, and `0` otherwise).
- `map` should allocate memory for the array containing $f(L)$; the user of `map` is responsible for freeing this memory after use.
- `filter` should rearrange the elements of L so that all elements l for which $g(l)$ is TRUE appear in their original order before all elements l' for which $g(l')$ is FALSE (these elements should also appear in their original order). `filter` should return the number of elements of L for which $g(l)$ is TRUE.

Example. Suppose $X = \mathbb{Z}$ (the set of integers), $f(x) = x^2$ and $g(x) = \text{TRUE}$ iff x is even. Let $L = [-1, 3, -8, 2]$. Then, `map(L, f)` returns a new array of `int`s containing `[ 1, 9, 64, 4 ]`; `filter(L, g)` changes `L` to `[ -8, 2, -1, 3]` and returns `2`.

Input format:

Output format:

Q2. Polynomial arithmetic is analogous to Integer arithmetic, especially in terms of addition and multiplication. If we define the *size* of a polynomial as its degree (thus, constant polynomials have degree zero), we get a natural notion of division for polynomials $a(x)$ and $b(x)$, where $\deg(a) \geq \deg(b)$, as follows. If $a(x)$ can be written as $q(x) \cdot b(x) + r(x)$ and $0 \leq \deg(r) < \deg(b)$, then we regard $q(x)$ as the quotient and $r(x)$ as the remainder, when $a(x)$ is divided by $b(x)$. This lets us naturally define divisibility of polynomials as '$b(x)$ divides $a(x)$' if and only if $r(x) = 0$.

The *GCD* of two polynomials $a(x)$ and $b(x)$ can then be defined as the polynomial $d(x)$, of the highest possible degree, that divides both $a(x)$ and $b(x)$.

Write a C program to represent polynomials with real coefficients in a format suitable for basic arithmetic operations like addition and multiplication. Use the basic operations to implement a quotient-remainder division routine for polynomials, and then use this division routine to implement a complete GCD routine for polynomials.

Input format: The input will consist of 2 lines. Each polynomial will be specified as a list of coefficient-exponent pairs, all of which will appear on a single line. Note that the coefficients are real numbers, while the exponents are integers. Any non-zero number of blanks may be used to separate two numbers.

Output format: Your program should also print the GCD polynomial in the same format. The coefficients should be printed correct to 2 places of decimal (using `%.2f` in printf).

Sample input 1:

```
1.0 4      2.0 3      2.0 2      2.0 1      1 0
1 3      2.0 2      2.0 1      1 0
```

⟵ represents $x^4 + 2x^3 + 2x^2 + 2x + 1$

⟵ represents $x^3 + 2x^2 + 2x + 1$

Sample output 1:

```
1.00 1      1.00 0
```

⟵ represents $x + 1$

Sample input 2:

```
1 4      1 2      1 1      1 0
1 3      2 2      2 1      1 0
```

⟵ represents $x^4 + x^2 + x + 1$

⟵ represents $x^3 + 2x^2 + 2x + 1$

Sample output 2:

```
1.00 0
```

⟵ represents 1

Sample input 3:

```
1.00 2      2.00 1      1 0
1.0 1      1 0
```

⟵ represents $x^2 + 2x + 1$

⟵ represents $x + 1$

Sample output 3:

```
1.00 1      1.00 0
```

⟵ represents $x + 1$

Q3. Write a program to play the "Hangman" game. The program should randomly choose a film name that consists of letters and whitespace only from `https://www.dropbox.com/s/7s1xflpfx54c9ls/movie-list.txt?dl=0` The name is presented to the player with some *non-whitespace* characters missing; blanks are inserted to mark word boundaries as usual. The goal of the player is to guess the name of the film. In each try, the player may guess a single character. If the character appears in the film name, *all* its occurrences have to be filled in by the program; otherwise, the player loses a "life". The player may also guess the full name of the film. If the guess is wrong, the player also loses a life. A player starts out with 10 lives. If the player does guess the name correctly before running out of lives, the total number of tries required becomes her score.

Your program should mark the missing letters using the underscore (_). A few of the letters should be filled in initially as hints; as above, if any of the filled in characters appears in the film name multiple times, *all* its occurrences have to be filled in initially. On average, at least one character in five (rounded up) should be filled in initially.

Your program should also display the number of remaining lives to the right of the film name. Note that uppercase and lowercase letters are *not* distinguished.

Example:

```
_____H_H      Lives: 10
z
Wrong guess!
_____H_H      Lives: 9
gupi gyne
Wrong guess!
_____H_H      Lives: 8
a
_AA__HAH      Lives: 8
b
BAA__HAH
baadshah
Correct! Your score is 5.
```

```
B____ B__     Lives: 10
E
Wrong guess!
B____ B__     Lives: 9
A
B_A__ B__     Lives: 9
O
B_A__ BO_     Lives: 9
black box
Correct! Your score is 4.
```

Q4. **Text formatting**

Write a program that takes a text file as input and prints its contents to standard output in a 'properly formatted' manner. See below for more details about formatting requirements.

Input format: Your program should take a single command line argument that specifies the name of the input file. The input file will contain one or more paragraphs of text. Paragraphs will be separated by one or more blank lines. You may assume that the text will contain only letters, digits, punctuation marks (comma, semi-colon, colon, fullstop, question mark, exclamation mark, but no dashes, hyphens, parentheses) and white space.

Output format specifications:

- Words should be separated by one or more blanks.

- There should be no whitespace between a word and a following punctuation mark, but each punctuation mark should be followed by one or more blanks.
- Paragraphs should be separated by exactly one blank line.
- Each line of each paragraph should start with and end in a non-whitespace character.
- The paragraphs should appear 'fully justified', i.e., all lines except the last line of each paragraph should be exactly 80 characters long. It will usually be necessary to add extra blank spaces in between words to achieve this effect. For a 'natural' look, the number of extra blanks inserted around a word should roughly be proportional to the length of the word.
- No word should be split across lines.
- If a single word is longer than 80 characters, it should appear by itself on one line.
- You should minimise the total number of lines required in the output, i.e., each line should contain the maximum number of non-whitespace characters possible, subject to the constraints given above.

Example: In the example below, the output is justified at 28 columns instead of 80.

Input file (`input.txt`)

```
This is the
first paragraph
 containingonelongproblematicword.

This
is a second  ,
badly written
para .
```

```
$ ./a.out input.txt
This is the first paragraph
containingonelongproblematicword.

This  is  a  second,  badly
written para.
```