COMP2113 Programming technologies [2019]
Assignment 2

General Instructions

Please read the instructions in this document carefully. In this assignment, you will solve FOUR tasks and a tester would automatically test your submitted program. Sample test cases are provided with each task in this document (**user inputs are printed in blue**). However, please note that we will also use additional test cases when marking your assignment submission.

Total: 100 points
5 points for proper code comments and indentation
5 points for program modularity using appropriate functions
90 points for program correctness

Input and output format
Please follow the instructions given by each question regarding input/output requirements for your program. If you failed to follow the instructions, the tester may not able to give a score for your program. Additionally, you should strictly follow the sample output format (including space, line breaker, etc.), otherwise, your answer might be considered as wrong.

Submission
You are provided with the following 4 C/C++ files, each a code template for a task. Download them and put them together into one directory. After completing the assignment, compress this directory as a **.zip file. Make sure to only upload the source files (***.c/*.cpp). **Do not submit any other files**. There should only be the 4 files in the zip archive. Please **use your university number to name the zip archive** and check carefully that the correct files have been submitted. We suggest you download your submitted files, extract them, and check for correctness. **You will receive 0 marks for this assignment if you submit incorrect files.** Resubmission after the deadline is not allowed.

Filename	Description
1.cpp	Task 1
2.cpp	Task 2
3.cpp	Task 3
4.cpp	Task 4

IMPORTANT: Read the comments in the given code templates carefully, and follow the instructions there.

Deadline
The deadline is announced in Moodle. **Late submission will not be accepted**.

Late submission
You will receive 0 marks if you submit after the deadline.

Wrong submission
You will receive 0 marks if you submit incorrect files.

Evaluation
Your code will be tested for technical correctness. However, the correctness of your implementation – not the tester's judgments – will be the final judge of your score. If necessary, we will review your work individually to ensure that you receive due credit for your work.

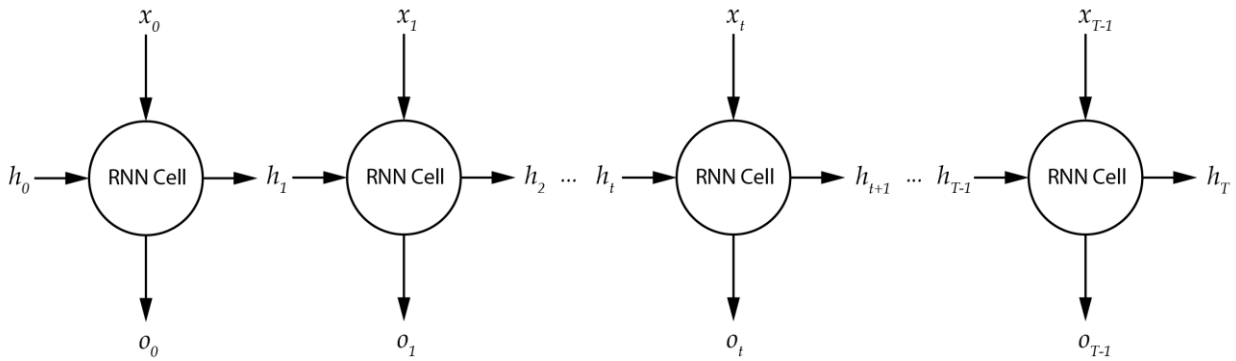
Academic Dishonesty
We will be checking your code against other submissions in the class and from the Internet for logical redundancy. If you copy someone else's code and submit it with minor changes, we will know. We trust you all to submit your own work only; please don't let us down. If you do, we will pursue the strongest consequences available to us.

Task 1 (C++) Recurrent neural network

(20 points)

"A recurrent neural network (RNN) is a class of artificial neural network where connections between nodes form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behavior. Unlike feedforward neural networks, RNNs can use their internal state (memory) to process sequences of inputs. This makes them applicable to tasks such as unsegmented, connected handwriting recognition or speech recognition." (Description from Wikipedia)

In this task, we will build a simple recurrent neural network and simulate its forward propagation. As shown the figure below, an RNN cell at time t takes an input x_t and a hidden state h_t from the previous cell, and computes a next state h_{t+1} and an output o_t. An RNN of T cells accepts an input sequence $x_0, x_1, \dots, x_{T-1}$ and an initial hidden state h_0 and outputs a sequence $o_0, o_1, \dots, o_{T-1}$.



Each RNN cell computes the following functions:

$$o_t = \text{sigmoid}(0.2x_t + 2h_t)$$

$$h_{t+1} = \tanh(0.6x_t - 2h_t)$$

where

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

$$\tanh(x) = 2 \times \text{sigmoid}(2x) - 1$$

e = 2.72

Your program should take two lines of input:

- the first line contains two numbers: an integer *T* denoting the recurrent times ($T \leq 100$) and a floating-point number h_0 denoting the initial hidden state; and
- the second line contains the input sequences which are *T* floating-point numbers $x_0, x_1, \ldots, x_{T-1}$.

Your program should then display the hidden state sequences $h_1, h_2, \ldots, h_T$ in the first line and the output sequences $o_0, o_1, \ldots, o_{T-1}$ in the second line.

You will need to complete the following functions in the provided template 1.cpp. Read the code in the template carefully to see what have been provided for you. You will find details of the function prototypes in the in-code comments too.

- ***sigmoid()*** for sigmoid activation function
- ***tanh()*** for tanh activation function
- ***ComputeH()*** for computing the next hidden value in RNN cell
- ***ComputeO()*** for computing the output value at current time step
- ***PrintSeqs()*** for printing the values in a 1D-array to screen
- ***main()*** for main function

Note:

- You may use `setprecision(n)` defined in the header `<iomanip>` to set the precision parameter of the output stream, making them the same as the sample test case.

Sample test case 1.1

```
5 0.0
1.0 2.0 3.0 4.0 5.0
0.5373192913 0.1247867951 0.9140171922 0.5170662095 0.9616286607
0.5498652773 0.8139025799 0.7005903049 0.9327484189 0.8844658501
```

Sample test case 1.2

```
1 1.0
1.0
-0.8855427140
0.9003742762
```

Sample test case 1.3

```
3 1.0
0.0 0.0 0.0
-0.9641167571 0.9586890814 -0.9578009493
0.8809297009 0.1268112559 0.8719810862
```

Task 2 (C++) Path planning

(25 points)

In this task, your program will accept a map (in form of a 2D grid) and determine the smallest number of steps it takes to go from the starting point to the destination. From any cell in the map, one can only reach to its 4-neighbours, i.e., the top / bottom / right / left neighbours.

An example 5 x 5 map is as follows:

```
-1 -1 -1 -1 -1
-1  0  0  0 -1
-1  1 -1 -2 -1
-1  0 -1  0 -1
-1 -1 -1 -1 -1
```

We denote -1 as an obstacle, 0 as feasible space, 1 as the starting point and -2 as the destination. In this example, there is a path from the starting point (1) to the destination (-2) by following the feasible space marked by the red 0, and the smallest number of steps taken from the starting point to the destination is 4.

You may assume that there is exactly one starting point and one destination in the input map. The maximal values of the height and the width of the map are no more than 1000.

The first input line to your program contains two integers giving the height (*H*) and width (*W*) of the map. After that, H rows times W columns of integers representing the map will follow.

You will need to complete the following function in the provided template 2.cpp. Read the code in the template carefully to see what have been provided for you. You will find details of the function prototypes in the in-code comments too.

- ***FindPath()*** for finding the smallest number of steps to go from the starting point to the destination on a given map.

You are NOT allowed to modify the main() body but you should add your own functions wherever appropriate for better program modularity.

Here we make use of sample test case 2.6 to give you an idea of how the smallest number of steps can be found.

Given a 5x5 map	`-1 -1 -1 -1 -1`
	`-1  0  0  0 -1`
	`-1  1 -1  0 -1`
	`-1  0  0 -2 -1`
	`-1 -1 -1 -1 -1`

1st iteration – reach out one step from the starting point "1" by filling its zero-valued (i.e., feasible space) 4-neighbours with "2"	`-1 -1 -1 -1 -1`
	`-1  2  0  0 -1`
	`-1  1 -1  0 -1`

	-1 2 0 -2 -1 -1 -1 -1 -1 -1
2nd iteration – reach out one step from the all "2"s by filling their zero-valued (i.e., feasible space) 4-neighbours with "3"	-1 -1 -1 -1 -1 -1 2 3 0 -1 -1 1 -1 0 -1 -1 2 3 -2 -1 -1 -1 -1 -1 -1
3rd iteration – reach out one step from the all "3"s by filling their zero-valued (i.e., feasible space) 4-neighbours with "4". As we find that we have also reached the destination "-2", hence the number of steps taken is 3.	-1 -1 -1 -1 -1 -1 2 3 4 -1 -1 1 -1 0 -1 -1 2 3 -2 -1 -1 -1 -1 -1 -1

Think about this: how to determine that there is no valid path in the map?

Sample test case 2.1 5 5 -1 -1 -1 -1 -1 -1 0 0 0 -1 -1 1 -1 -2 -1 -1 0 -1 0 -1 -1 -1 -1 -1 -1 4	*Sample test case 2.2* 5 5 -1 -1 -1 -1 -1 -1 0 0 0 -1 -1 1 -1 0 -1 -1 0 -1 0 -1 -1 -1 -1 -1 -1 No
Sample test case 2.3 3 5 -1 -1 -1 -1 -1 -1 -2 -1 1 -1 -1 -1 -1 -1 -1 No	*Sample test case 2.4* 3 5 -1 -1 -1 -1 -1 -1 -2 0 1 -1 -1 -1 -1 -1 -1 2
Sample test case 2.5 5 5 -2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 8	*Sample test case 2.6* 5 5 -1 -1 -1 -1 -1 -1 0 0 0 -1 -1 1 -1 0 -1 -1 0 0 -2 -1 -1 -1 -1 -1 -1 3

Task 3 (C++) Deletion and statistics of stop words

(25 points)

A common pre-processing of a piece of text is to remove stop words which carries little discriminating power for information retrieval or text mining. Stop words are the most common words or short function words, such as "a", "an", "the", "where", "and", etc. In this task, you are going to write a C++ program to remove stop words from a piece of text.

Suppose a text file named story.txt contains the following two lines of text (spaces are shown as □ below for clarity):

```
The□house□whirled□around□two□or□three□times□and□rose□slowly
through□the□air.□Dorothy□felt□as□if□she□were□going□up□in□a□balloon.
```

And another file named ***stopwords.txt*** contains some stop words:

```
a
an
A
and
The
the
in
or
```

We may then remove all stop words in ***stopwords.txt*** from the text in ***story.txt***, and save the *cleaned* contents in a file named ***cleaned_story.txt***:

```
house□whirled□around□two□□three□times□□rose□slowly
through□□air.□Dorothy□felt□as□if□she□were□going□up□□□balloon.
```

We may also record the number of stop words removed from ***story.txt*** in a new file ***stopwords_count.txt***:

```
a 1
an 0
A 0
and 1
The 1
the 1
in 1
or 1
```

Specifically, the input to your program is two strings "*text*.txt" and "*word*.txt" where *text* and *word* are any character sequence giving the file names. The file "*text*.txt" contains lines of text and "*word*.txt" contains some stop words, one on each single line (the number of stop words do not exceed 100).

Your program should then create two files "*text*_cleaned.txt" and "*word*_count.txt":

- The file "*text*_cleaned.txt" contains the same lines of text as in "*text*.txt" but with all stop words in "*word*.txt" being removed (e.g., ***cleaned_story.txt*** in the example above).

- The file "*word*_count.txt" contains the same lines of stop words as in "*word*.txt" but with each word followed by a number indicating how many times the word has been removed from "*text*.txt" (e.g., ***stopwords_count.txt*** in the example above).

Your program will also output to the screen the total number of stop words removed.

If your program fails to open any input / output file, it should display "**Failed to open [filename]**", where [filename] is the name of the problematic file, and the program will terminate.

You will need to complete the following function in the provide template 3.cpp. Read the code in the template carefully to see what have been provided for you. You will find details of the function prototypes in the in-code comments too.

- ***ReadStopWordFromFile()*** for reading stop words from a file and storing them in an array of `struct stop_words` (each struct stores a stop word and the corresponding removal count.)
- ***RemoveWordFromLine()*** for deleting all occurrences of a word from a line of text
- ***RemoveAllStopwordsFromLine()*** for deleting all occurrences of all stop words from a given line of text; this function should call ***RemoveWordFromLine()*** to delete all occurrences of a single stop word from the given line
- ***WriteStopWordCountToFile()*** for writing the stop words and corresponding removal counts to an output file
- ***main()*** – the main body to read in the stop words and process the text line by line by removing all stop words from each line, record the removal count and write to the output files. The main function should call ***ReadStopWordFromFile()***, ***RemoveAllStopwordsFromLine()***, and ***WriteStopWordCountToFile()***

You should add your own functions wherever appropriate for better program modularity.

Sample test case 3.1

```
stop_words2.txt story2.txt
Number of stop words removed = 7
```

Input file – stop_words2.txt

```
is
are
be
```

Input file – story2.txt

```
Monkey is a common name that may refer to groups or species of mammals,
in part, the simians of infraorder Simiiformes. The term is applied
descriptively to groups of primates, such as families of new world
monkeys and old world monkeys. Many monkey species are tree-dwelling
(arboreal), although there are species that live primarily on the ground,
such as baboons. Most species are also active during the day (diurnal).
Monkeys are generally considered to be intelligent, especially the old
world monkeys of Catarrhini.
```

Output file – stop_words2_count.txt

```
is 2
```

```
are 4
be 1
```

Output file – story2_cleaned.txt

```
Monkey  a common name that may refer to groups or species of mammals, in
part, the simians of infraorder Simiiformes. The term  applied
descriptively to groups of primates, such as families of new world
monkeys and old world monkeys. Many monkey species  tree-dwelling
(arboreal), although there  species that live primarily on the ground,
such as baboons. Most species  also active during the day (diurnal).
Monkeys  generally considered to  intelligent, especially the old world
monkeys of Catarrhini.
```

Sample test case 3.2 (if the input files cannot be opened)

```
words.txt text.txt
```

Failed to open words.txt

Task 4 (C++) Array split

(20 points)

In this question, you are trying to split an array into three equal sum sub-arrays.

Write a C++ program to split an array. Consider an array A of n integers. Determine if array A can be split into three consecutive parts such that sum of each part is equal. If yes, print the index pair(i, j) such that (1) sum(arr[0..i]) = sum(arr[i+1..j]) = sum(arr[j+1..n-1]) and (2) 'i+j' is the smallest among all feasible index pairs; otherwise print -1.

Sample test case 4.1:
Input:

```
5
1 3 4 0 4
```

Output:
(1,2)
Reason:
First, Sum of subarray arr[0..1] is equal to
sum of subarray arr[2..2] and also to
sum of subarray arr[3..4]. The sum is 4. -> we have pair (1,2)
Second, Sum of subarray arr[0..1] is equal to
sum of subarray arr[2..3] and also to
sum of subarray arr[4..4]. The sum is 4. -> we have pair (1,3)
Since '1+2' < '1+3', we finally output (1,2)

Sample test case 4.2:

Input:
11
0 2 1 -6 6 -7 9 1 2 0 1
Output:
(2,7)
Reason:
0 + 2 + 1 = -6 + 6 - 7 + 9 + 1 = 2 + 0 + 1

Sample test case 4.3:
Input:
3
2 3 4
Output:
-1
Reason:
No three subarrays exist which have equal sum.

Program input. The first line contains the length of the input array. The second line consists all array values.

Program output. If we can split the array into three equal-sum sub-arrays, then print the index pair; Otherwise, print -1.

You will need to complete the following function in the provided template 4.cpp. Read the code in the template carefully to see what have been provided for you. You will find details of the function prototypes in the in-code comments too.

- ***findSplit()*** for finding a spliting solution.

Note:

- **You are NOT allowed to modify the main() body but you should add your own functions wherever appropriate for better program modularity.**

Hint: You may first calculate the entire sum of the array, and check whether the sum can be split into three equal sum set or not. If yes, you may use two variables to record sum/3 and 2*sum/3, and then try to find such split from the beginning of the array recursively.