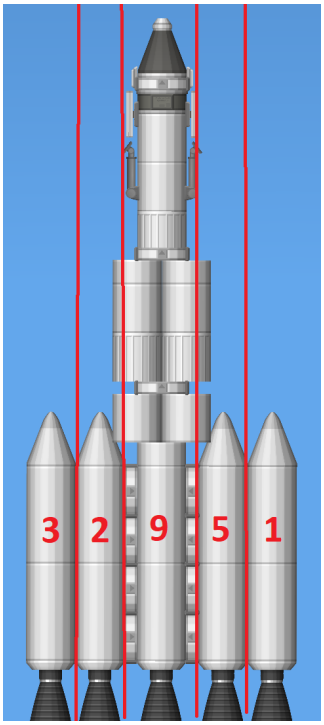
TIC1002: Introduction to Computing and Programming II
Semester II, 2019/2020

Practical Exam 1 (22nd February, 2020)
Time allowed: 2 hours

Task 1. To the stars! [8 marks]



Kerbal, the **rocket engineer**, is trying to figure out where is the **center of gravity of** his rocket. As we know, an off-balance rocket (i.e. center of gravity not in the middle of rocket) will move in an erratic fashion during takeoff.

For simplicity, we represent the weight of various parts of the rocket **as an integer array**, e.g. {3, 2, 9, 5, 1} represents the weight information of the rocket on the left.

We can then compare the weights at the opposite end of the rocket, e.g. "3" will be compared to "1", "2" compared with "5", etc.

The center of gravity **starts off at the middle column,** e.g. **column 2** in a 5-column rocket (leftmost column is column 0). A weight values pair changes the center of gravity as follows:

Center of gravity	Compare ***Left*** with ***Right***
No change	***Left*** is the same as ***Right***
Move one column to the left	***Left*** is larger than ***Right***
Move one column to the right	***Left*** is smaller than ***Right***

For example, the rocket above:

- COG (center of gravity) starts at Column 2
- 3 compares with 1 (left is heavier) ➔ COG is at column 1 (2 -1 ➔ 1)
- 2 compares with 5 (right is heavier) ➔ COG is back to column 2 (1 + 1 ➔ 2)

[Of course, this is a vast simplification of how COG is calculated in the real world 😉]

For simplicity, you can assume all rocket designs have ODD number of columns. Please read the entire question before you start, especially the "additional requirement" on next page.

Implement the following function:

`int COG( int weight[], int size);`
`weight[]` is an integer array with **`size`** number of weight data. The array always have **odd size**.
This function returns the column number of the center of gravity (COG) of the rocket.

Sample results

weight[]	size	Return result
{3, 2, 9, 5, 1}	5	**2** (COG is at column 2, i.e. balanced in this case)
{1, 2, 3, 4, 5, 6, 7}	7	**6** (COG is at column 6 as all pairs of weight are right heavy)
{2, 5, 3, 1, 2}	5	**1** (COG is at column 1, as one pair of left heavy values moved the COG to the left)

Additional Requirements

Implementation	Maximum Mark
Iterative (used **any form** of loop)	**6 marks**
Recursive (**NO** loop of **any form**)	**8 marks**

Credit:

Image is taken from the Android Game "Space Flight Simulator"

Task 2. Busiest sky in the world [12 marks]

Below is a "top down" 2D view of a **sky region**, represented as 20 rows x 15 columns of characters:

Input
----v--^-------

----PPE1-------

>-----1-NOT----
-----------h---
-----------a---
<----------r---
-----------d---
-------@---!---
------$--------
-----*---------

--------^----->
-------<-------

------v-------<

--v-------^----

Output (trails in **bold red**)
----**v**--^--^----
----**v**-----^----
----**v**-----^----
----**X**PE1--^----
----**v**-----^----
>>>>X>X>XXX>>>>
----**v**-----^h---
----**v**-----^a---
<---**v**-----^r---
----**v**-----^d---
----**v**--@--^!---
----**v**-$---^----
----**v***----^----
----**v**-----^----
----**v**---^-^--->
----**v**--<--^----
----**v**-->--^----
<<<<X<X<<<X<<<<
----**v**-----^----
--v-**v**-----^----

We want to print the trail (path) travelled by airplanes within this region. The **directions** of the airplane are:

'>'	Flying **right**
'<'	Flying **left**
'^'	Flying **up**
'v'	Flying **down**

The trails are drawn with the following rules:

R1. We only draw the trails for planes **starting at the topmost, bottommost, leftmost and leftmost** borders **AND** flying in the correct direction. e.g. trail for planes at the topmost row **flying downward** will be drawn.

R2. The direction of the airplane is used to draw the trail. e.g. the trail of plane flying right is drawn as a series of '>'.

R3. When trail crosses other trails or encounter any non-empty sky region, a '**X**' is drawn instead. **Empty** sky region is represented as a '**-**'.

Use the rules and the given sample output above to verify your understanding.

For simplicity, all test cases will respect the following assumptions:

A1. There will not be any plane in the 4 corners of the sky region.
A2. No two planes are on collision path, e.g. on a particular row, you will **not find** two planes going left AND right heading towards each other!

Implement the following function:

`void plot_trail(char region[MAXROW][MAXCOL])`
`region[]` is the 2D sky region. `MAXROW` is defined as 20 and `MAXCOL` is defined as 15. This function update the ***region*** directly with the airplane trails.

You should design **appropriate helper function(s)** to help with implementation. **Submit ONLY the function above and any new helper function(s).**

Rewarding Good Programming Style [4 marks]

4 marks out of the 12 marks are used for **programming style** check as follows:

Programming Style	Applicable to	Total
Consistent Indentation	Tasks 1 and 2	1 mark
Good variable naming	Tasks 1 and 2	1 mark
Modularization	Task 2 only	2 marks

~~~ End of PE Question ~~~