PA 1b - Karel Continues to Explore

In this programming assignment, Karel continues the exploration of his world, even though he is still restricted to the same basic actions and queries as in the previous assignment.

The goal is to give you more practice on how to think like a programmer. A very important part of this is to first decompose a problem into smaller subproblems that can be more easily reasoned about. This is known as “divide-and-conquer”. To start this decomposition, it is helpful to write pseudo code first, before then translating it into a set of C functions. Being methodical about how you approach coding problems is an extremely valuable skill to learn and practice.

Summary

You will find the details of the assignment in Assignment Details. There will be starter code available for you to work from.

The goal is to write the following Karel programs:

sail.c	Create a triangular sail of items.	25 pt
level.c	Remove the same number of items from piles of items.	25 pt
sort.c	Sort stacks in descending order.	25 pt
measure.c	Measure the total height of the ceiling.	25 pt
gaps.c	(Extra credit) Count the number of gaps in a grid.	14 pt

To submit the work, please follow the Submission Instructions.

Assignment Details

Getting Started

As in the previous assignments, we are providing you with some starter code. To get this code, execute the following command in your home directory:

```
$ getStarter PA1b
```

This will create a subdirectory, called “PA1b” with all the starter code.

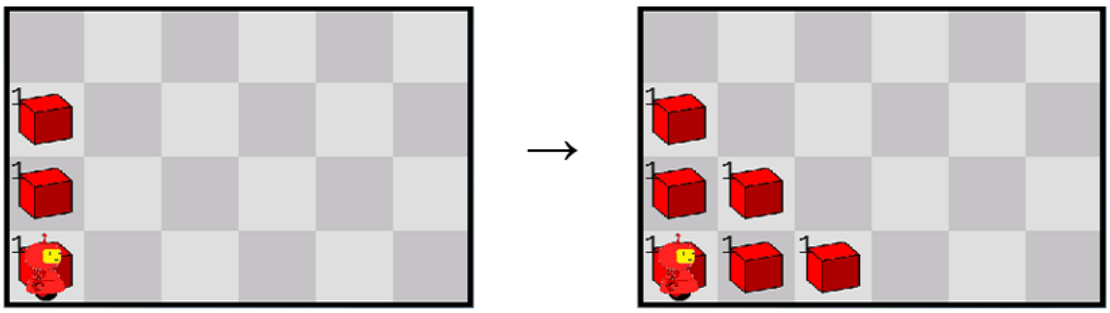
As in PA1, you are only allowed to use the actions, queries and programming constructs that were introduced in **Chapter 2. Karel the Robot**. You are NOT allowed to use any variables.

For all the problems, your solution needs to be flexible enough such that it works for different maps and situations, within the constraints of each problem. As a first step, make sure you test your code with the different test cases that are provided in the starter code. You do this by changing the start and end map in the settings file, as was also explained in the previous PA. However, these test cases are not exhaustive, and you should create your own maps as well to do further testing. **Chapter 2. Karel the Robot** explains both how to test your code with different maps and how to create your own maps.

It is also important to emphasize that you should use the top-down modular design approach to tackle these problems. Thinking about how to solve the problem at a high level first, then creating the pseudo code and finally translating it into C-code will really help you. Appropriate use of functions is extremely important: it will make it much easier to develop correct code. Use these programming assignments to practice good coding techniques and make them second nature.

Karel Unfurls a Sail

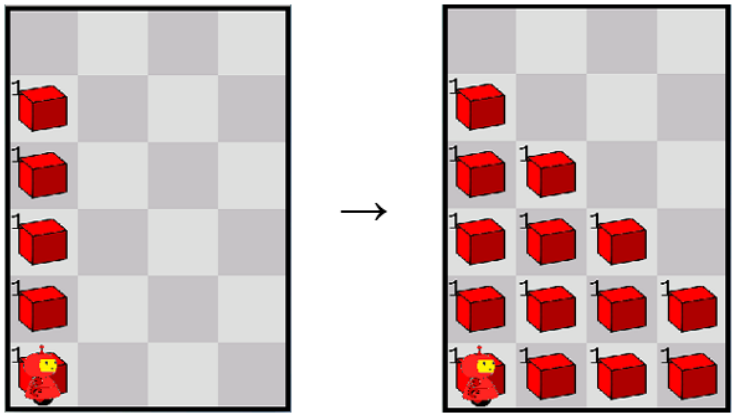
In this first problem, Karel needs to unfurl a sail. The size of the sail is given by the height of the mast, which is represented by a vertical line of items, as shown below. Unfurling the sail is done by completing the triangle.



Start position **End position**

Karel always starts in the bottom left corner facing east, with an infinite number of items in his bag. He needs to end in the same position, but his final orientation does not matter. The mast is always in the left-most column (so where Karel starts), starting from the bottom and consists of a continuous line of tiles each with a single item. The height of the mast can vary, is at least one tile high and can reach up to the height of the map.

The size of the map is variable as well, but there are never any walls inside the map. If the sail does not fit within the width of the map, it reaches up to the right wall and is cut off there, as shown in the example below.

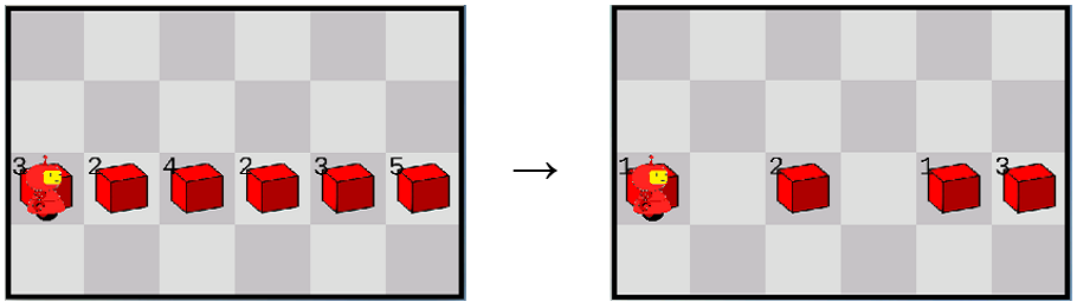


End position **Start position**

Your code should go in the file **sail.c**, which is part of the starter code.

Karel Levels the Piles

Karel is given piles of items and he needs to lower them all by the same amount. This amount should be the maximum possible. This is illustrated below. In this case, the maximum all the piles can be reduced by is 2 items.



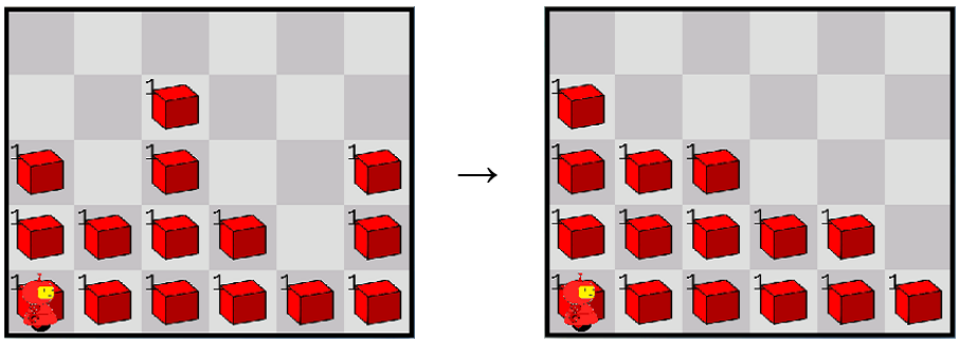
Start position **End position**

The piles are lined up in one row and reach from the west wall to the east wall. Note that if any tile in the line is empty, it is still part of the line but just has a pile of size zero. Karel starts in the same row as the line of items, at the west wall, facing east and with an empty bag. He needs to end up on the same tile, but his orientation does not matter. There are no other items anywhere except for the line of items. There are no walls anywhere inside the map.

The solution goes in file **level.c**.

Karel Sorts the Stacks

In this problem, items are organized in stacks, i.e., a sequence of neighboring tiles in a column with one item per tile. Each stack starts on the floor and can have an arbitrary height. The goal is for Karel to sort the stacks in descending order, as shown below.



Start position **End position**

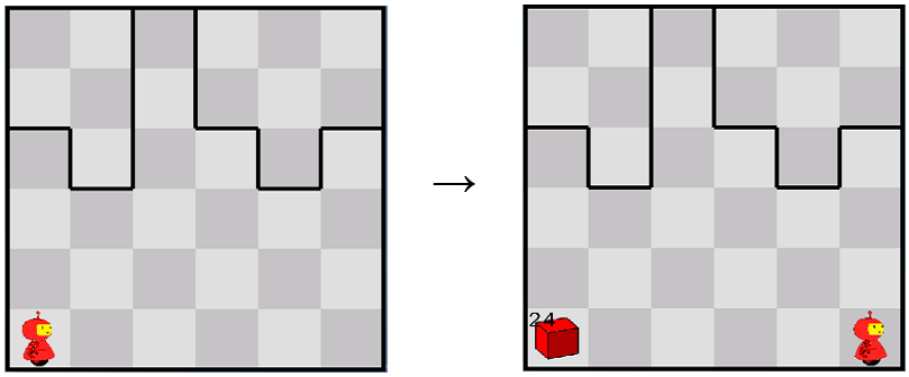
Karel starts in the south-west corner, facing east, with an empty bag. He needs to end in the same spot, but his orientation doesn't matter. There are no walls anywhere inside the map.

Write your solution in **sort.c**.

Hint: For this problem, you probably don't want to think about it from a human perspective, where you pick up one stack and put it in the correct position. Instead, have a close look at the start and end situation in the example. What happens to the items in a particular row?

Karel Measures the Height of his Roof

For this challenge, Karel needs to measure the total height of his roof and deposit the corresponding number of items in the bottom-left corner. What we mean by the height of the roof is essentially the height of each column measured from the floor (so from the floor to the first wall in that column). In the scenario below, the height of the roof in each column, expressed as the number of tiles, is (from left to right): 4, 3, 6, 4, 3, 4. Adding these numbers together gives a total height of 24. This is the number of items Karel needs to place in the bottom-left corner..



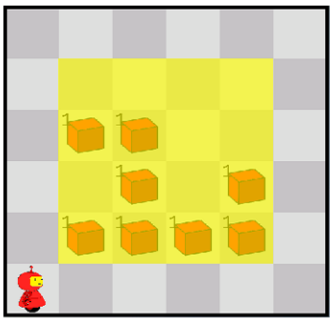
Start position **End position**

Karel always starts in the bottom-left corner, facing east, with an infinite number of items in his bag. He needs to end up in the bottom-right corner (so NOT where he started) and his orientation does not matter. There are no other items anywhere in the map or any walls other than those delineating the ceiling.

Write your solution in the file **measure.c**.

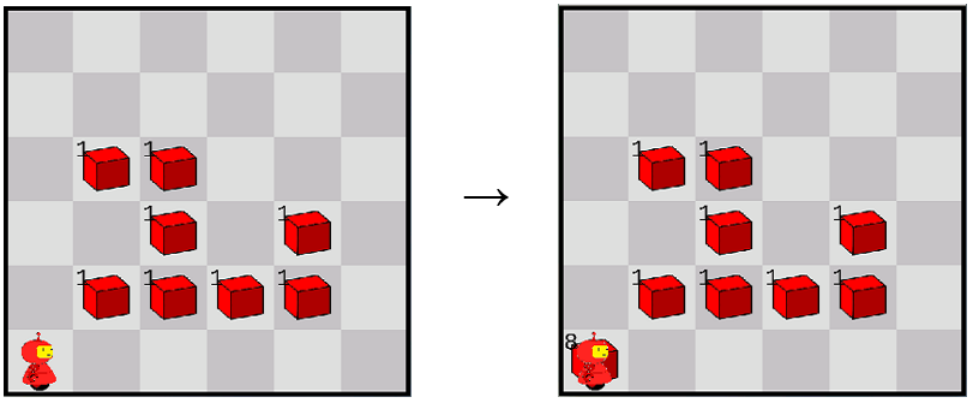
Extra Challenge: Karel Counts the Number of Gaps in a Grid

[This part is for extra credit]



For this optional challenge, Karel needs to count the number of tiles in a grid that don't have items on it. The grid is defined as the area of the map except for the top and bottom rows and the leftmost and rightmost columns. In the example on the right, the grid is shaded in yellow. For each tile in the grid, it either has one item on it or no items. The tiles that have no items on them are the "gaps" in the grid. There are also no items in the area that is not part of the grid (i.e., the top and bottom rows, and the leftmost and rightmost columns), but these tiles are not considered gaps as they do not belong to the grid. Only tiles that are part of the grid AND have no items on them are considered gaps. In the example, there are therefore 8 gaps. Gaps can be anywhere in the grid.

Karel needs to count the total number of gaps in the grid and deposit a corresponding number of items in the south-west corner of the field. This is illustrated in the example below. It is important to note that in the end position, the original grid should be intact.



Start position **End position**

Karel starts in the south-west corner, facing east and with an infinite number of items in his bag. He needs to end in the same position, but his orientation does not matter. There are no walls in the field.

Your solution should go in the file **gaps.c**.

Submission Instructions

For submitting your assignment, we will again follow the procedure outlined in the **PA Submission, Grading and Regrade Procedures** document. Remember that file names are case sensitive. Make sure you comment out, delete or disable all your pause() or say_text() statements. Your code should be able to run without being halted. For all Karel PAs (including this one), also make sure that you did not change the line `karel_setup(...)`; it is important that the code refers to the original settings .json file.

You run the submit script in the directory where your .c files are located:

```
$ submit PA1b
```

This script will submit all the .c files specified in this assignment. Remember that running the script again will overwrite your previous submission. So you could run the script as soon as you have some of the problems done, and then run it again and again as you finish more of the assignment.