Programming Assignment 1

EECS 214/395 Data Structures
Fall 2013
Instructor: Peter Scheuermann

Assigned: October 9, 2013 Due: October 21, 2013 (11:59 pm)

Background

Problem Description

Consider that a freight train has ***n*** railroad cars, each to be left at different station. They're numbered 1 through n and freight train visits these stations in order n through 1. The railroad cars are labeled by their destination. To facilitate removal of the cars from the train, we must rearrange them in ascending order i.e., 1 through n from front to back. When the cars are in this order, they can be detached at each station. We rearrange cars at a shunting yard that has an ***input track***, an ***output track*** and ***k holding tracks*** between the input & output tracks

Solution Strategy

To rearrange the cars, we examine the cars on the input track from front to back. If the car being examined is the next one in the output arrangement, we move it directly to ***output track***. If not, we move it to a ***holding track*** & leave it there until it is time to place it to the ***output track***. The holding tracks operate in a **LIFO manner** as the cars enter & leave these tracks from top. When rearranging cars only the following moves are permitted:

- A car may be moved from the front (i.e. right end) of the input track to the top of one of the ***holding tracks*** or to the left end of the output track.
- A car may be moved from the top of ***holding track*** to the left end of the ***output track***.

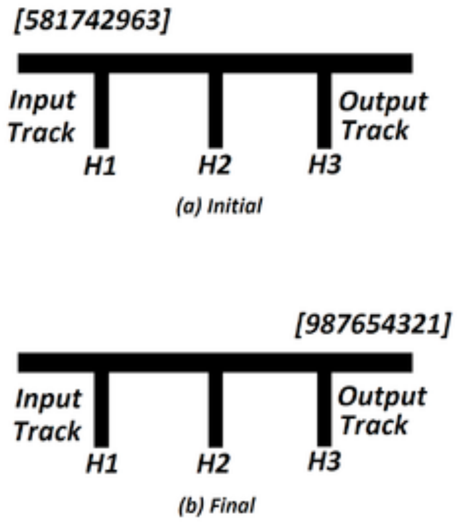

Fig. 1

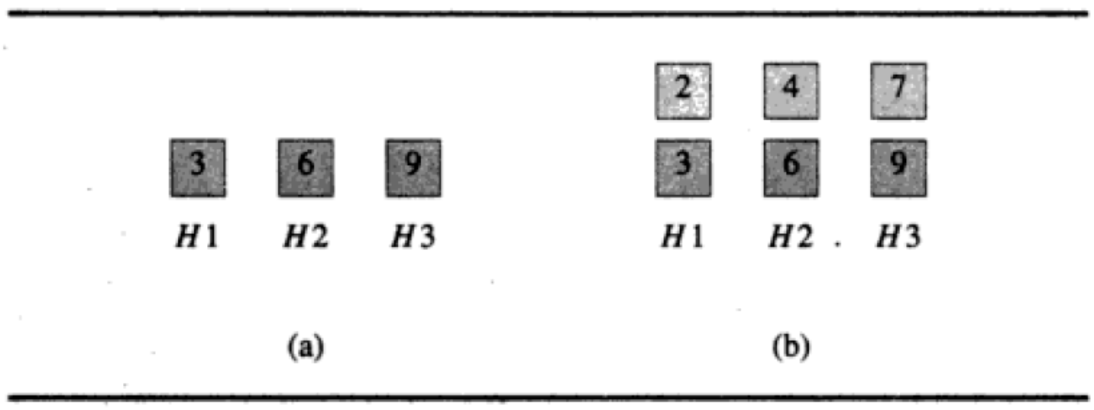

Fig. 2

Figure 1a. above shows a shunting yard with ***k*** = 3 holding tracks ***H1***, ***H2*** & ***H3***, and ***n*** = 9 cars. The ***n*** cars of the freight train begin in the input track & need to end up in the output track in order 1 through ***n*** from right to left. The cars initially are in the order 5,8,1,7,4,2,9,6,3 from back to front. Car 3 is at the top at the front and cannot be sent to output yet as it to be preceded by cars 1 & 2. So car 3 is detached & moved to holding track ***H1***. The next car 6 can't be output & it is moved to holding track ***H2***. Because we have to output car 3 before car 6 & this will not possible if we move car 6 to holding track ***H1***. Now it's obvious that we move car 9 to ***H3***. " The current state of the holding tracks is shown in Fig. 2a.

Car 2 is considered next. It can be moved into any of the holding tracks while satisfying the requirement that car labels in any holding track be in increasing order, but moving it to ***H1*** is preferred. If car 2 is moved to ***H3***, then we have no place to move cars 7 and 8. If we move it to ***H2***, then the next car, car 4, will have to be moved to ***H3*** and we will have not place for cars 5, 7, and 8. The least restrictions on future car placement arise when the new car *u* is moved to the holding track that has at its top a car with smallest label *v* such that *u*>*v*. We will use this assignment rule to select the holding track, as shown in Fig. 2b.

The objective of this assignment is to write a function *Railroad* that determines a sequence of moves that results in rearranging cars with initial ordering *inputOrder (int numberofCars)* using the most *numberHoldingtracks* holding tracks. If such a sequence does not exist, *Railroad* returns false; otherwise it returns true.

Tasks to be performed

(1) The LIFO manner of a holding track matches the property of Stack. The implementation should be based on the **LinkedStack** class shown below.

To implement Stack using Linked List class, you need to first define the structure of a node in list. An example format is as below:

```
struct node
 {
     int d;
```

```
    node *next;

};
```

Then you can define the following specification:

```
class LinkedStack
{
    node *first;
    public:
        linkedstack(){...}//constructor
        ~linkedstack(){...}//destructor
        linkedstack(linkedstack & other){...}//copy constructor
    void push(int num);
    int pop();
    bool empty();
    int top();
};
```

Provide the implementation of the member functions in the LinkedStack class. (do NOT use iterators from STL)

Make sure that your Stack works well, because the following problems depend on that.

(2) The main function to be implemented is *boolean Railroad (int [ ] inputOrder, int numberof Cars, int numberofTracks).*
Railroad should begin by allocating an array of holding tracks. The main loop of Railroad considers the next car in the input sequence and decides if it can be moved to the output sequence. If this car can be moved to the output sequence, it may be possible to move additional cars from the holding tracks. Otherwise, determine to which holding track to move the input element. The function Railroad makes calls to two auxiliary functions. The first, called *outputFromHoldTrack( )* outputs instructions to move a car from a holding track to the output track. The second auxiliary function *putinHold(int c)* which moves car *c* from the input track to one of the holding tracks if possible, else it returns false.

(3) We will give you two input files "input.txt", "input2.txt" to test your program. Each file contains 3 lines: ***n***, ***k***, permutation of cars. Your program should print the sequence of moves executed for a given input. For example, as car ***i*** is about to be moved to holding track ***j,*** your program should print: " holding track ***j*** hold car ***i***".

When car ***i*** is about to be moved to output track directly, your program should print: "Move car ***i*** from input to output".

As car ***i*** is about to be moved from holding track ***j*** to output track*,* your program should print: "Move car ***i*** from holding track ***j*** to output".

(4) If a given permutation is not feasible your program should only print a statement to that affect. In other words if the permutation 1-2-3-4 is not feasible your program should print only:

1-2-3-4 is not feasible

and not print any moves prior to determining that the desired permutation is not possible. This will require you to "save" the moves and print them only after determining feasibility.

(5) Given any permutation of cars with input numbers from 1 to ***n***. Give an example of an input sequence for which the rearrangement process requires ***n-1*** holding tracks.

(6) Bonus (10 points) Modify your algorithm so that it works for any given input sequence of ***n*** = 9 cars. *Hint:* the rearrangement process needs to be changed to find and print the minimal number of stacks required as your program terminated given any permutation of cars

Requirements

Environment:
Make sure your program is able to work on T-lab machine. (Note the version of compiler: g++ (GCC) 4.4.7 20120313 (Red Hat 4.4.7-3))

Program:

Your code should be well commented and explain what you are doing. Make sure your runs correctly, efficiently. The output should follow the format mentioned in Problem (2).

Writing:

For each problem, give a brief and clear explanation (**less than 1 page**) about your design and thought.

Submission:

The format of your writing is required to be in .pdf format

Put your implementation of problem (1) and problem (2) in separate files. You may include the implementation of stack as a header or a .cpp file in your program for problem (2).

Everything you do for this project should be included in a folder (e.g. named as project). Compress it in tar.gz format named by your “FirstName_LastName”. Submit it to blackboard.

Late policy:
Programs will be accepted for at most two days later with a penalty of 10% off for each day. After submission, if want to modify and submit it, you may upload it again via blackboard.

Good Luck!