CIS 313, Intermediate Data Structures
Winter 2018, University of Oregon

Programming Assignment 1

Due: January 29, 2018

0 Instructions

Submit your work through Canvas. You should submit a tar file containing all source files and a README for running your project.

More precisely, submit on Canvas a tar file named lastname.tar (where lastname is your last name) that contains:

- All source files. You can choose any language that builds and runs on ix-dev.

- A file named README that contains your name and the exact commands for building and running your project on ix-dev. If the commands you provide don't work on ix-dev, then your project can't be graded and you'll be required to resubmit with a significant penalty.

Here is an example of what to submit:

```
hampton.tar
  README
  my_class1.py
  my_class2.py
  my_class3.py
  problem.py
  another_problem.py
  ...
```

```
README
  Andrew Hampton

  Problem 1: python problem1.py input
  Problem 2: python problem2.py input
  ...
```

Note that Canvas might change the name of the file that you submit to something like lastname-N.tar. This is totally fine!

The grading for the assignment will be roughly as follows:

Task	**Points**
Problem 1	
pass given sample test case	10
pass small grading test case	5
pass large grading test case	5
print function	5
Problem 2	
pass given sample test case	10
pass small grading test case	5
pass large grading test case	5
print function	5
TOTAL	50

Passing a test case means that the `diff` utility on ix-dev produces no output. Furthermore, passing the given sample test case requires actually completing the project (not, for example, just hardcoding the given output).

1 Linear Data Types

Problem 1. Stack
Implement a stack using a linked list. Don't use any builtin utility classes (like a list or vector). You need to implement a linked list and use that to create your stack class.

Following the abstract data type described in the course textbook, your stack data structure must implement the following three methods with specified runtime:

`push(X)`: Takes a single argument X (an integer, if it matters), and puts X on the stack. *O(1)*

`pop()`: Removes the top element from the stack and returns it. *O(1)*

`is_empty()`: Returns a boolean indicating whether the stack is empty. *O(1)*

Write a driver program that takes a single command-line argument, which will be a filename. The input file will contain instructions for stack operations. The first line of the input file will be an integer $0 \leq N \leq 10^4$ giving the number of instructions. Following will be N lines, each containing an instruction. The possible instructions are:

`push X`, where $-10^5 \leq X \leq 10^5$ is an integer: For this instruction, push X onto the stack. There is no output.

`pop`: Pop the top element of the stack. Print the removed element. If the stack is empty, output *StackError*.

`print`: Print the contents of the stack separated by a single space, starting with the top (the element which would be removed with a pop). If the stack is empty, output *Empty*.

The print action should not be implementation dependent. Instead, you should write a `print_stack` function that takes a stack as an argument and accomplishes the print action using only the above three stack methods. You can, of course, also use builtin features of your language (e.g., a copy feature).

All output should be to STDOUT. Each piece of output should be separated by a newline.

Example input file:

```
7
print
push 1
push 2
print
pop
pop
pop
```

Example output:

```
Empty
2 1
2
1
StackError
```

Problem 2. Queue
Implement a queue using a linked list. Don't use any builtin utility classes (like a list or vector). You need to implement a linked list and use that to create your queue class.

Following the abstract data type described in the course textbook, your queue data structure must implement the following three methods with specified runtime:

`enqueue(X)`: Takes a single argument X (an integer, if it matters), and puts X on the queue. *O(1)*

`dequeue()`: Removes the front element from the queue and returns it. *O(1)*

`is_empty()`: Returns a boolean indicating whether the queue is empty. *O(1)*

Write a driver program that takes a single command-line argument, which will be a filename. The input file will contain instructions for queue operations. The first line of the input file will be an integer $0 \leq N \leq 10^4$ giving the number of instructions. Following will be N lines, each containing an instruction. The possible instructions are:

`enqueue X`, where $-10^5 \leq X \leq 10^5$ is an integer: For this instruction, put X on the queue. There is no output.

`dequeue`: Remove the front element of the queue. Print the removed element. If the queue is empty, output *QueueError*.

`print`: Print the contents of the queue separated by a single space, starting with the front (the element which would be removed with a dequeue). If the queue is empty, output *Empty*.

The print action should not be implementation dependent. Instead, you should write a `print_queue` function that takes a queue as an argument and accomplishes the print action using only the above three queue methods. You can, of course, also use builtin features of your language (e.g., a copy feature).

All output should be to STDOUT. Each piece of output should be separated by a newline.

Example input file:

```
7
print
enqueue 1
enqueue 2
print
dequeue
dequeue
dequeue
```

Example output:

```
Empty
1 2
1
2
QueueError
```