CS 251 Program 2 *Dequer*

Due: Thu Mar 2, 2000 5PM **Spec version: 1.0**

Introduction

The goals of this second program are to gain more experience with classes and objects, to use free store for dynamic arrays, to implement a class with non−trivial value semantics, to work with overloaded methods, and finally, to implement, test, and demonstrate a deque. To be concrete, we'll build a deque of char, and write a driver class, `Dequer`, to contain a `Deque` and perform operations on it.

We will follow the same overall approach as in Program 1, building a simple 'engine' to process commands and generate responses, according to the *Dequer* protocol described in this document. In some places the Dequer protocol is similar or identical to the Blocks protocol used in Program 1, but of course there are differences both large and small − read this specification carefully!

The Dequer Protocol

General protocol issues:

- Each protocol message in either direction is terminated with a single newline (i.e., the thing that '`endl`' generates in C++). The protocol consists of a series of *interactions*. Each interaction with the engine begins with a command to the engine followed by zero or one responses from the engine.
- The *only* thing the engine outputs is protocol responses. That implies, specifically:
  1. Your program must output ***nothing*** when it first starts up.
  2. Your program must ***not*** output any 'debugging info' while it runs. If you put in any 'debugging print statements' during development, you ***must*** be sure to remove them all before turning in your program, or it will fail testing and be marked down.
  3. Your program must ***not*** output any 'prompts' such as 'Enter command:' or anything of the sort, or it will fail testing and be marked down.
- Your program *may*, but is not required to, accept lower−case forms of the commands described below. Your program *must* generate all responses in upper−case as described below. There is no grading penalty for failing to accept lower−case commands, and no grading bonus for doing so.
- *Except where indicated otherwise*, all response arguments are separated by single space characters. There are no leading or trailing spaces in responses.
- In general, commands do not requires spaces between the command and the arguments or between the arguments and each other (for commands with more than one argument).
- There is a single newline ending each response. It is strongly advised that you use the C++ construct `endl` to generate that newline, rather than `"\n"`, because `endl` also 'flushes the output buffer' − i.e., it forces any pending output to be sent.

- Note that certain errors with certain commands − e.g., trying to remove from an empty deque − generate a FAIL response, and the engine continues operating.
- Except for the situations documented here that generate FAIL responses, the engine may deal with illegal commands and/or illegal command arguments in any way that it wants to provided that the program is not at risk of crashing. In particular, the engine *may* simply exit if a command or command error occurs, or *may* ignore the error and attempt to continue functioning, or *may* give an error message and attempt to continue functioning. It *must not* continue functioning if further execution would be unsafe.

Commands the engine reads from cin

`A`	engine produces an **A**UTHOR response containing the name of the author of the engine
`D`*`milliseconds`*	engine produces a **D**ELAY *milliseconds* (int >=0) response (Note the engine does ***NOT*** actually perform a delay or waste time here! It just outputs the DELAY response!)
`S`*`size`*	**S**et up the internal deque to have room to contain exactly *size* (int >0) chars, and generates a SIZE response
`I`*`lrch`*	**I**nsert *ch* (char) into the *lr* (char: 'L' for left or 'R' for right) side of the internal deque if there is room to do so, and generate an INSERT response. If the internal deque is full, generate a FAIL response instead. There is no space between *lr* and *ch*; any single character except newline is legal as a *ch*.
`R`*`lr`*	**R**emove a char from the *lr* (char: 'L' or 'R') side of the internal deque if it is not empty and generate a REMOVE response, or a FAIL response instead if the deque is empty.
`L`*`chars`*	**L**oad the given *chars* (0 or more char excluding newline) into the right side of the internal deque. Generates a FAIL response if some of the *chars* don't fit − in that case, as many of the *chars* as do fit are inserted, and the rest of the *chars* are discarded.
`U`	**U**nload the internal deque from the left side until it is empty, and generates an UNLOAD response.
`Q`	engine generates a QUIT response and then **Q**uits

Responses the engine writes to cout

`SIZE `*`size`*	Internal deque has been set to *size* (int >0) and emptied.
`AUTHOR "`*`name`*`"`	Author of the program is *name*. (The *name* can contain any characters except for newlines and double quotes, and is surrounded by double quotes in the AUTHOR response.)

`DELAY` *`ms`*	A delay of *ms* (int >=0) milliseconds should occur here. (Note the engine does ***NOT*** actually *do* any delaying itself!)
`INSERT` *`lrch`*	*ch* (char) has been inserted at the *lr* (char: 'L' for left or 'R' for right) side of the internal deque.
`REMOVE` *`lrch`*	*ch* (char) has been removed from the *lr* (char: 'L' for left or 'R' for right) side of the internal deque.
`FAIL` *`cmd`*	A command could not be completed (entirely). *cmd* (char: 'I' or 'R' or 'L') indicates what type of command failed.
`QUIT`	Engine is about to quit.

Deque semantics

The deque has a current size, which for purposes of this specification may be any positive number less than the largest number that fits in a 32 bit signed int. Overall memory size limits in any particular computer may restrict the maximum usable size; the Dequer program itself should not impose any additional limits. The current size is established when a 'S' command is processed, and may be changed by further 'S' commands within a single run of the engine. It is allowable for the program to abort if insufficient memory is available to satisfy a given 'S' command. At program startup it is as if an 'S1' command had been processed.

The deque contains elements of type `char`. Although the *Dequer* protocol places certain restrictions on what chars can be inserted, the entire `Deque` interface should be capable of dealing with all possible chars as data, and the underlying deque should be able to insert and remove any char.

The two ends of the deque are referred to here as 'left' and 'right'; note that these are 'logical' names and have no particular relationship to higher and lower memory addresses, or larger or smaller array indexes.

The deque class implementation must support value semantics.

The deque implementation must satisfy the following time and space limits, where *n* is current capacity of the deque as specified by the most recent 'S'ize command:

- Time to insert a char at the left or right: *O(1)*
- Time to remove a char from the left or right: *O(1)*
- Space needed for entire deque: *O(n)*
- Time to pass a deque by value: *O(n)*

Deque interface design

The specific public methods −their names, arguments, and return types − the `Deque` class provides are to be determined as part of your design, using the principles of *paranoia, stinginess,* and *naturalness* as discussed in class. Evaluation of the resulting interface will form a component of the grade on this program.

Sample interactions

In these examples, note that each command is a *single line* of input, even if it appears on multiple lines due to document formatting. Note that there may zero or one response to each command. Note, likewise, that each response is a *single line* even if it appears on multiple lines due to formatting in this document.

#	*Command*	*Response*	*Notes*
1	ILa ILb Q	INSERT a FAIL I QUIT	First insert works 2nd fails since initial size is 1
2	A Q	AUTHOR "Dave Ackley" QUIT	
3	Lpqr U U Q	FAIL L UNLOAD p UNLOAD QUIT	Not room for all but one was stored nothing there now
4	S1000 Lfoo bar RL D1234 RR U Q	SIZE 1000 REMOVE Lf DELAY 1234 REMOVE Rr UNLOAD oo ba QUIT	Room for 1000 chars (no response) an 'f' was removed an 'r' was removed rest is unloaded
5	S10 L#!(";/ RR S10 U RL Q	SIZE 10 REMOVE R/ SIZE 10 UNLOAD FAIL R QUIT	Room for 10 chars (no response) a '/' was removed Room for 10 chars nothing in it now so remove fails

You are encouraged to generate more examples of your own using the reference implementation.

Revision History

- Version 1.0, first released version,2/19/00.