CPSC 131 Project 2: B-list

Spring 2016
Tim Finer, CSU Fullerton
tifiner@fullerton.edu

This was written by Dr. Kevin Wortman's for his CPSC 131 class.

Introduction

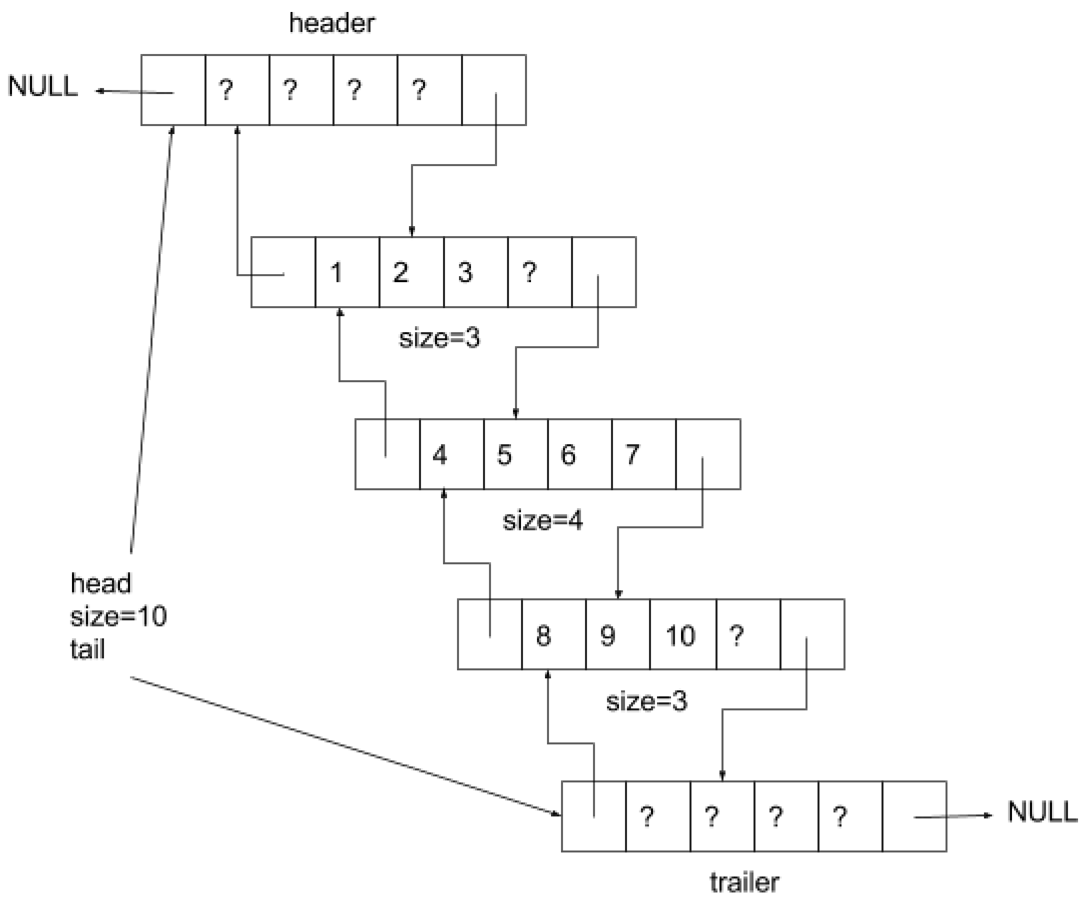
In this project you will implement a *B-list,* which is a hybrid data structure that combines the ideas of a doubly linked list and a fixed-capacity vector (a.k.a. partially-filled array).

The B-list data structure

A B-list is a deque data structure based on a doubly linked list; except that, instead of each node storing one element, each node may store anywhere between 1 and B elements, where B is some constant value greater than 1. So if $B = 4$, then each node object in the B-list stores 1, 2, 3, or 4 elements.

The B-list is inspired by another data structure called a B-tree. Very soon we will study a data structure called a *binary search tree.* Plain binary search trees use nodes, and each node stores exactly one element. However there is a variation on a binary search tree called a *B-tree* that stores multiple elements in each node. B-trees were invented in response to the observation that node objects involve a lot of overhead. Since each node stores 1 element and 2 pointers, only about ⅓ of the variables in a doubly-linked list go toward actually storing elements; the other ⅔ goes to overhead. If each node stores many elements, the proportion of space going toward overhead is reduced. For example if $B = 32$, then $\frac{32}{34} \approx 94\%$ of the variables go toward storing elements, while only $\frac{2}{34} \approx 6\%$ of the variables store overhead pointers. You may find the Wikipedia page on B-trees to be instructive, but keep in mind that this project involves B-lists, not B-trees.

The following is a sketch of a valid B-list with $B = 4$ containing the integers 1 through 10.



Observe that

1. the nodes are doubly-linked, as usual;
2. there are header and trailer nodes;
3. each node contains 1, 2, 3, or 4 elements, though not necessarily exactly 4;
4. none of the nodes are empty, except for the sentinels; and
5. there are fewer overhead pointers than there would be in a conventional 10-element doubly linked list.

Your assignment is to write a `BList` class that implements all the fundamental deque operations, and also a `BNode` class that represents a single node in a B-list. I have supplied a `BList` class that "cheats" by being a thin wrapper around the STL `deque` class. You will need to rewrite the class to actually be a B-list. You will need to write your own `BNode` class from scratch. You may decide for yourself whether to use sentinel nodes, or not; either approach can work fine. I have also supplied a `main()` function that tests whether `BList` works properly.

The code

A ZIP archive file, `project2.zip`, is available in TITANium. It contains the following files:

1. `blist.hh` is a header file that declares the `BList` class. As discussed above, this code works, but is not actually a B-list. You will need to rewrite the data members and function definitions so that `BList` really is a B-list.
2. `test_main.cc` is a source file for a program that uses the `assert(...)` function to test whether `BList` works properly.

As presented, each file should compile cleanly, and all tests will pass.

This code was developed with the `clang++` compiler on the Linux operating system. However it is portable C++, and should work properly on Windows and MacOS.

What to do

As described above, the `BList` class defined in `blist.hh` compiles and passes its tests, but is not actually a B-list. You will need to write a new `BNode` class, change the data members in `BList`, and probably rewrite all the member functions in `BList`. You should leave the declarations of all the `BList` member functions the same, so that when you're done, all the tests in `test_main.cc` still compile and pass.

Sample output

The following is the output of the test program, when all tests succeed.

```
BList tests:

BList::BList
BList::add_back
BList::is_empty
BList::size
BList::front
BList::back
BList::add_front
BList::remove_front
BList::remove_back
BList::get
BList::set
```

Deliverables

Upload the following file to TITANium:

1. `blist.hh`

Each file must be uploaded as a plain text source file. That means, for example, that you must not upload a ZIP or DOC file.

For full credit, your code must compile, link, and run cleanly against the provided `test_main.cc`. I may use the output of the test program to confirm the correctness of your code while grading. If your code passes all those tests, it is likely to be correct. However, the test program does not check every single aspect of the module. (For instance, it does not check for memory leaks.) So passing all the tests does not guarantee a 100% grade.

Deadline

The project deadline is April 8th, 2016, at 11:55 pm. Late submissions will not be accepted.

License