Winter 2016 CS 32

Width: 1280

Project 4 Warmup

There are several parts to Project 4, one of which involves implementing a disk-based data structure. We will give you a class to use to help you do that, but we anticipate that debugging your implementation that uses that class may be complicated by other aspects of Project 4. To save you time, we'll give you this warmup exercise that lets you face some problems in a simpler context. The understanding you gain by doing this exercise will help you in the full Project 4, and the code you write for this exercise can be used with a few small modifications in the full project 4.

The exercise is to implement a DiskList class representing a singly linked list of C strings. The novelty is that the list nodes must be stored in a disk file instead of in RAM. Here is the public interface of this class:

```
class DiskList
{
  public:
    DiskList(const std::string& filename);
    bool push_front(const char* data);
    bool remove(const char* data);
    void printAll();
};
```

The constructor creates an empty list in a disk file of the specified name.

For the `push_front` function, if data is a C string of fewer than 256 characters (not counting the zero byte), then a new node with a copy of the C string (including a zero byte) is added to the front of the list, and the function returns true. If the C string has at least 256 non-zero-byte characters, the function returns false without changing anything.

Every node whose C string is equal to data is removed from the list by the `remove` function. Nodes not removed must remain in the same order in the linked list as they were in before. The function returns true if at least one node was removed, otherwise false.

The `printAll` function writes, one per line, all the C strings in the list in the order they appear in the list.

Except for the constructor, you should not be surprised by the following:

```
int main()
{
        DiskList x("mylist.dat");
        x.push_front("Fred");
        x.push_front("Lucy");
        x.push_front("Ethel");
        x.push_front("Ethel");
        x.push_front("Lucy");
        x.push_front("Fred");
        x.push_front("Ethel");
        x.push_front("Ricky");
        x.push_front("Lucy");
        x.remove("Lucy");
        x.push_front("Fred");
        x.push_front("Ricky");
        x.printAll();  // writes, one per line
          // Ricky  Fred  Ricky  Ethel  Fred  Ethel  Ethel  Fred
}
```

Your task for this exercise is to implement this class using the BinaryFile class, one that manages creating, writing to, and reading from binary files stored on disk. The code for the class is in BinaryFile.h. Your DiskList implementation will have a data member of type BinaryFile and possibly an additional one or two simple members. It must **not** have any members that are arrays, strings, vectors, lists, or other containers. All data in the linked list, as well as the links from one node to the next, must be in the disk file written by the BinaryFile member.

For your initial implementation, don't worry about trying to reuse portions of the disk file that aren't currently part of the linked list (because the nodes that used those portions were removed from the list).

Only after your implementation works completely correctly, enhance it so that the portions of the disk file that contained nodes that were removed

can be reused. In other words, `push_front` should not grow the disk file if it doesn't have to; if it can, the new node it adds should overwrite a portion of the disk file that held a node that was later removed. This means that you will have to keep track of where the removed list nodes are in the disk file, and push_front will have to efficiently either locate one or determine that none are available (which is the only circumstance in which it should do something that would increase the size of the disk file).

When you have completed this exercise, you will have done an important chunk of the work you need to do for Project 4.