Programming Project #4
EGRE246 Spring 2018
Array Linked List

1 Overview

Suppose you are working in a object-oriented programming environment that does not allow the creation of runtime data structures, e.g. linked lists. Instead the only data structure you have available is the array which must be a fixed size defined at compile time. You have decided that you really need the advantages of a linked lists so you have decided to implement a linked list data structure in an array.

For this project you will implement in C++ the array "memory" required to implement linked lists using arrays. Though you won't do this for your project, this memory could then be used to implement a linked list ADT.

2 The Concept

Below is the code for `ALLNode.h` ("**A**rray **L**inked **L**ist **N**ode") (downloadable off the class web site) which defines a doubly-linked list node we will use to create linked lists in our code.

```
#ifndef ALLNODE_H
#define ALLNODE_H

namespace EGRE246 {
  const static int NULLADDR = -1;

  template <typename Item>
  class ALLNode {
  public:
    typedef Item value_type;

    ALLNode(const value_type& initData = value_type(),
    int prevLink = NULLADDR, int nextLink = NULLADDR){
      data = initData; prev = prevLink; next = nextLink;
    }

    void setData(const value_type& newData) { data = newData; }
    void setPrevLink(int newLink) { prev = newLink; }
    void setNextLink(int newLink) { next = newLink; }

    value_type getData() const { return data; }
    const int getPrevLink() const { return prev; }
    const int getNextLink() const { return next; }
```

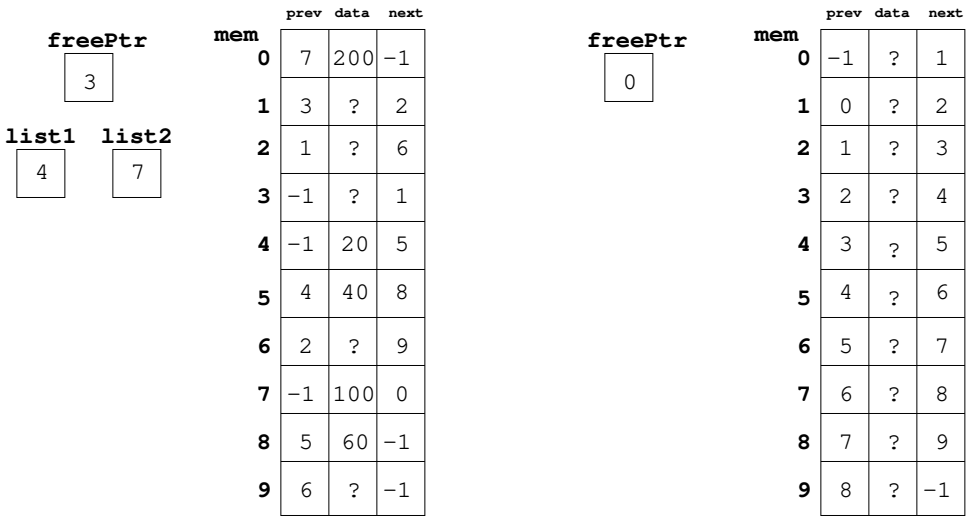
```
  private:
    value_type data;
    int prev;
    int next;
  };
}

#endif
```

Note that our previous and next "pointers" `prev` and `next` are just integers – they will be indices into our memory array. We will use -1 to indicate our NULL pointer (and name it `NULLADDR`).

We will use an array of these nodes as our memory to store our linked lists, defined as such (assuming our lists will be of type `int`):

```
ALLNode<int> *mem = new ALLNode<int>[size];
ALLMem<int>::address freePtr, list1, list2; // address will be defined as an int
```

For example, consider part **(a)** of the following diagram where `size == 10`. Three lists are represented: `list1` whose first node is at index 4 (and consists of items [20,40,60]), `list2` beginning at index 7 (consisting of 2 items, [100,200]), and a list of free or available memory called `freePtr` consisting of 5 nodes.



Part **(b)** above illustrates what the initial configuration of the array might be where all the nodes are available (i.e. free or unallocated).

3 Project Overview

This project entails you writing a template class to implement the array of nodes model presented above. You must use the following specification for your project; the only thing you should do with this file is to implement the routines; do not change any of the specifications or add ones of your own! This file is downloadable off of the class web pages.

```
#ifndef ALLMEM_H
#define ALLMEM_H
#include "ALLNode.h"

#define SIZE 100 // default size of the node array; arbitrary

namespace EGRE246 {
  template <typename Item>
  class ALLMem {
  public:
    typedef Item value_type;
    typedef int size_type;
    typedef size_type address;

    ALLMem(size_type size_=SIZE) { /*...*/ }

    size_type getMemSize() const { /*...*/ } // returns total size of memory
    address getFreePtr() const { /*...*/ }   // returns pointer to free memory list
    size_type memFree() const    { /*...*/ } // returns amount of free memory

    address alloc() { /*...*/ }
      // allocates and returns address of new node removing it from free
      // memory; prints error message and exits if no memory available

    void dealloc(address addr) { /*...*/ }
      // deallocates node pointed to by addr (i.e. returns it to free memory list)
      // prints error message and exits if:
      //   1) no memory to deallocate, or
      //   2) address node is not allocated (it is free), or
      //   3) address is illegal

    value_type getData(const address& addr) { /*...*/ }
      // returns data in node pointed to by addr prints error message and
      // exits if illegal address

    address getPrev(const address& addr) { /*...*/ }
    address getNext(const address& addr) { /*...*/ }
      // returns address of prev/next node, prints error message and exits
      // if illegal address

    void setData(const address& addr, const value_type val) ...
      // sets node data at addr to val; prints error message and exits
      // if illegal address

    void setPrev(const address& addr, const address naddr) { /*...*/ }
```

```
    void setNext(const address& addr, const address naddr) { /*...*/ }
      // sets prev/next at addr to val; prints error message and exits
      // if illegal address

  private:
    size_type size;     // size of memory
    size_type freeMem;  // size of available memory
    ALLNode<Item> *mem; // memory array of nodes
    address freePtr;    // pointer to available memory list

  };

}

#endif
```

Node that you will implement all of your routines in this class file. The `alloc`/`dealloc` routines above essentially perform the same tasks as `new`/`delete` in C++. You should turn in a `.h` file for this project instead of the usual `.cpp` file.

4 Sample Use

File `allnodeHANDOUT.cpp`:

```
#include <iostream>
using namespace std;
#include "ALLMem.h"
using namespace EGRE246;

int main(void){
  ALLMem<int> mem(10);
  ALLMem<int>::address list1, lastnode;

  // create list #1 adding nodes to the end
  for (int i=1; i <= 8; i++) {
    ALLMem<int>::address newnode = mem.alloc();
    mem.setData(newnode,i*10);
    mem.setNext(newnode,NULLADDR);
    if (i==1) {
      list1 = newnode;
      mem.setPrev(newnode,NULLADDR);
    } else {
      mem.setPrev(newnode,lastnode);
      mem.setNext(lastnode,newnode);
    }
    lastnode = newnode;
```

```
  }

  cout << "dealloc node = " << mem.getNext(mem.getNext(list1));
  cout << " (value: " << mem.getData(mem.getNext(mem.getNext(list1))) << ")"<< endl;
  mem.dealloc(mem.getNext(mem.getNext(list1)));

  ALLMem<int>::address ptr = list1;
  while (ptr!=NULLADDR) {
    cout << mem.getData(ptr) << " ";
    ptr = mem.getNext(ptr);
  }
  cout << endl;
}
```

5 Deliverables

You are to turn in your project through the project submission link on the class web page. Name your source code file `proj4`*XXXX*`.h` where *XXXX* is the last 4 digits of your student V number.

Due date: Tuesday April 10