CS 2413 – Data Structures – Programming Project – 3 – Spring 2015
Due: March 11, 2015, 11:59 PM

Objectives

1. [10 pts] Implement `ArrayLinkedListRow` class.
2. [60 pts] Implement the `ArrayLinkedList` class, the points are distributed as follows.
   (a) [40 pts] Correctly implement the methods (about 2 pts for each method).
   (b) [15 pts] Template: your templated `ArrayLinkedList` class should work for both `<int>` and `<string>` in `main()` provided.
   (c) [15 pts] Implement exception handling as described in the project.
3. [20 pts] Runtime: your program correctly generate output from the `main()` provided.
4. [10 pts] Document your project thoroughly as the examples in the textbook. This includes but not limited to header comments for all classes/methods, explanatory comments for each section of code, meaningful variable and method names, and consistent indentation.

Project Description

The linked list data structure from the textbook is implemented using pointers. A linked list also can be implemented using an ***array*** as its underlying data structure. The details of the array implementation are discussed below.

Exception Handling

You will implement the following exception handling classes in ArrayLinkedList.

```
class LinkedListException: public exception{};
class LinkedListMemory: public LinkedListException{};
class LinkedListBounds: public LinkedListException{};
class LinkedListNotFound: public LinkedListException{};
```

ArrayLinkedListRow class

`ArrayLinkedListRow` class is used in `ArrayLinkedList` class to create the array that represent the nodes in the linked list.

```
template<class DT>
class ArrayLinkedListRow
{
private:
    DT* _info; // store data
    int _next; // next node
    int _nextEmpty; // next empty slot
public:
    ArrayLinkedListRow();
    // add other constructors that you need
    ~ArrayLinkedListRow();
```

```
    template<class T> friend ostream& operator<<(ostream& s,
                                                 ArrayLinkedListRow<T>& r);
     void operator=(const ArrayLinkedListRow<DT>& r);
    // add getters, setters or other methods and other methods
    // necessary for this class
};
```

ArrayLinkedList class

```
template<class DT>
class ArrayLinkedList
{
private:
    // an array used to store the nodes
    ArrayClass< ArrayLinkedListRow<DT> >* _rows;
    //*** You could use your vector class also ***//
    int _head;   // head of the list
    int _firstEmpty;   // first empty slot
    int _size;
    void copy(const ArrayLinkedList<DT>& ll);//copy from another list
    // add a new node with next as it's next node and returns
    // the index of new node
    int newNode( DT& newObject, int next);
public:
    ArrayLinkedList();    // empty and copy constructors
    ArrayLinkedList(const ArrayLinkedList<DT>& ll);
    // Constructor that create a list with newObject as the head
    ArrayLinkedList(DT& newObject);
    // Constructor with a give capacity
    ArrayLinkedList(int capacity);
    // Constructor with newObject as the head and capacity
    ArrayLinkedList(DT& newObject ,int capacity);
    ~ArrayLinkedList();   // destructor
    bool isEmpty();
    // is the list empty?
    int size();  // return the number of nodes stored
    void add(DT& newObject);    // add an object to the tail
    // insert an object at the position specified
    void insertAt(DT& newObject, int position);
    DT remove(); // remove the head
    // remove an object at the position specified
    DT removeAt(int position);
    // find the object that matches key, index of the object
    int find(DT key);
    // = operator
    void operator=(const ArrayLinkedList<DT>& ll);
    // overloading [] operator, return a reference to object at the
    // position in the linked list
    DT& operator[] (const int position);
    // ostream operator
    template<class T> friend ostream& operator<<(ostream& s,
                                                 ArrayLinkedList<T>& ll);
    // display raw data of the data members
    void displayRaw();
};
```

Data Members

You will use `ArrayClass` from the textbook (or Project 2) to create an array of `ArrayLinkedListRows<DT>` to store the nodes in the list. The default size of this array is `100`. In addition, `_head` is used to store the position of the first node in the array (`-1` if the list is empty), `_firstEmtpy` to store the position of the first empty slot in the array (-1 if none available) and _size to indicate the size of the list.

Methods

We use examples to show how the methods in `ArrayLinkedList` work. We assume the size of the array is is `5` and use `int` as the type of "`Object`". For simplicity, we show the values of the objects (for `_info`) instead of showing the pointers to objects in the table. *Keep in mind that you need use pointers in your implementation.*

Constructors

Upon the creation of the object (`myIntList`), _info, _next and _nextEmpty will be initialized as follows:

`myIntList`

_rows	_info	_next	_nextEmpty	_head = -1;
				_firstEmpty = 0;
				_size = 0;

0	NULL	-2	1	
1	NULL	-2	2	
2	NULL	-2	3	
3	NULL	-2	4	
4	NULL	-2	-1	

Linked List: empty
Empty list: 0 → 1 → 2 → 3 → 4 → NULL

In `ArrayLinkedList(Object& newObject)`, an empty slot will be obtained from `_firstEmpty`. Then a copy of `newObject` will be created and put in this slot. Assume `newObject` has the value of `5`(`int`), the data stored now updated as in the table. Changes are in red color.

`myIntList`

_rows	_info	_next	_nextEmpty	_head = 0;
				_firstEmpty = 1;
				_size = 1;

0	5	-1	-2	
1	NULL	-2	2	
2	NULL	-2	3	
3	NULL	-2	4	
4	NULL	-2	-1	

Linked List: 5 → NULL
Empty list: 1 → 2 → 3 → 4 → NULL

Other constructors are similar and you can figure it out from the comments in the code. In copy constructor, you need to use private method `copy`.

Destructor

Release the memory allocated for `ArrayLinkedListRows<DT>`.

copy [private method]

Copy data from another `ArrayLinkedList` object.

int newNode(Object& newObject, int next) [private method]

Used by `add()` and `insertAt()`. `newNode()` obtains an empty spot from `_firstEmpty`. "`next`" specifies (index of) the node following the new node. Throw `LinkedListMemory()` if the list is full.

bool isEmpty()

Returns `true` if the list is empty, `false` otherwise.

int size()

Returns `_size`, the number of objects stored in the list.

void add(Object& newObject)

Make a copy of `newObject` and add it to the tail of the list. Examples are shown below.

Add objects with the value of 7, 5 then 6 (to the last table).

myIntList				
_rows	_info	_next	_nextEmpty	_head = 0;
0	5	1	-2	_firstEmpty = 4;
1	7	2	-2	_size = 4;
2	5	3	-2	
3	6	-1	-2	
4	NULL	-2	-1	
Linked List: 5(0) → 7 → 5(2) → 6 → NULL // (i) indicate the position in _info Empty list: 4 → NULL				

`LinkedListMemory()` will be thrown by `newNode()` called by this method if the list is full.

void insertAt(Object& newObject, int position)

Make a copy of `newObject` and insert it at "`position`" in the list. Assume newObject has the value 5, insertAt(newObject, 2) will give the following result.

myIntList				
_rows	_info	_next	_nextEmpty	_head = 0;

0	5	1	-2	_firstEmpty =-1;
1	7	4	-2	_size = 5;
2	5	3	-2	
3	6	-1	-2	
4	5	2	-2	
Linked List: 5(0) → 7 → 5(4) → 5(2) → 6 → NULL // (i) indicate the position in _info Empty list: NULL				

`LinkedListMemory()` will be thrown by `newNode()` called by this method if the list is full. Throw `LinkedListBounds()` if `position` is greater than `_size` or less than 0.

Object remove()

Remove the head and return the value of the object. Since the object will be destructed, you will need to store it in a local variable before delete the object. Then you can return this local variable at the end of this method. *The position of the object removed will become the new `_firstEmpty`.* The following table shows the result of `remove()` call from last step.

myIntList				
_rows	_info	_next	_nextEmpty	_head = 1;
0	NULL	-2	-1	_firstEmpty = 0;
1	7	4	-2	_size = 4;
2	5	3	-2	
3	6	-1	-2	
4	5	2	-2	
Linked List: 7 → 5(4) → 5(2) → 6 → NULL // (i) indicate the position in _info Empty list: 0 → NULL				

Now we add object with value `4 (to the end)`.

myIntList				
_rows	_info	_next	_nextEmpty	_head = 1;
0	4	-1	-2	_firstEmpty =-1;
1	7	4	-2	_size = 5;
2	5	3	-2	
3	6	0	-2	
4	5	2	-2	
Linked List: 7 → 5(4) → 5(2) → 6 → 4→ NULL // (i) indicate the position in _info Empty list: NULL				

Throw `LinkedListBounds()` if `the list is empty`.

Object removeAt(int position)

Remove the object at `position` (in the list) and return the value of the object. *The position of the object removed will become the new `_firstEmpty`.* The result of `removeAt(3)` is shown below.

myIntList				
_rows	_info	_next	_nextEmpty	_head = 1;
0	4	-1	-2	_firstEmpty = 3;

1	7	4	-2	`_size = 4;`
2	5	0	-2	
3	NULL	-2	-1	
4	5	2	-2	
Linked List: 7 → 5(4) → 5(2) → 4→ NULL // (i) indicate the position in _info Empty list: 3 → NULL				

Throw `LinkedListBounds()` if `the list is empty` or `position` is greater than `_size - 1`.

int find(Object key)

Returns the index (in `_rows`) of the first object in the list that matches "`key`. Although you can simply search through the array but here we require you to search through the "list". For example, `find(5)` from the result in `removeAt()` section returns `4` which is the first match *in the list*. Throw `LinkedListBounds()` if `the key value is not found.`

void operator=(const ArrayLinkedList<Object>& ll)

Assignment operator, use `copy()` to simplify the task.

Object& operator[] (const int position)

Subscript operator which returns the reference to the object at `position` in the "list". `myIntList[3]` returns the object stored at `_row[0]` in the example, which has the value of `4`. Throw `LinkedListMemory()` if `position` is greater than `_size`.

ostream& operator<<(ostream& s, ArrayLinkedList<Object>& ll)

Returns an ostream& that will generate the output in the following format (using myIntList as an example):

```
7(1) --> 5(4) --> 5(2) --> 4(0)
```

The integer in `( )` indicates the position of the object in the array.

void displayRaw()

This method is used to check if you have implemented this data structure correctly. It displays the contents of the data members in the following format (for the result in `removeAt` method) :

```
Head: 1
First Emtpy: 3
Size: 4
info: 4 7 5 X 5
next:  -1 4 0 -2 2
nextEmpty: -2 -2 -2 -1 -2
```

* For the empty slots, print out the character 'X'.

main()

Due to the space limitation, the main() method for grading this project will be upload to http://learn.ou.edu. You should not make any modification to main() except commenting out the part that hasn't been implemented in your code.

Constraints

1. In this project, the header files you will have are <iostream> and <string>. <string> is used by the main() to test your program. No other libraries will be used/allowed. You will also use `ArrayClass` from the textbook in this project.
2. You are NOT allowed to add any data members to the classes. However, you can create additional private methods to simplify your work in implementing the methods.
3. None of the projects will be a group project. Consulting with other members of this class on programming projects is strictly not allowed and plagiarism charges will be imposed on students who do not follow this.