Grocery Buddy

Programming Assignment# 1

- **Objective:** Learn about the list ADT (Abstract Data Type). Implement list ADT as a doubly linked list. Design your solution in pseudo code and give an Object Oriented implementation in C++.
- **Points**: 50
- **Team-based**? You have the option to either work individually or work in pair. It is highly suggested that you work in pair.
- **Due**:
    - **Phase 1**: Sunday, 2015-March-29 @ 11pm
    - **Final Phase**: Monday, 2015-April-06 @ 11pm
- **What to hand in:** Refer to the section on "what to hand in" of this document.
- **Note:** Late phase 1 will not get any credit. For final phase, late assignment submitted within 24 hrs. from the deadline shall be penalized 10%. No late assignment shall be accepted after 24 hours from the deadline.

You may lose points if you don't pay special attention to the following in your program:

- *Your code should be properly indented. Separate various sections of your code with a blank line, like variable declaration, taking input, computation, printing, etc.*
- *Use appropriate data types, e.g., use integer data types where there will be no need to store floating point numbers.*
- *Avoid magic numbers, i.e., use constants instead of numeric/string literals and write a short comment that explains its purpose (it is okay to have string literals in cout for displaying messages or numeric/string literals when initializing variables).*

  E.g., `const double PI = 3.14; //value of pi`

  `area = PI * radius * radius;` *instead of* `area = 3.14 * radius * radius;`
- *Variables and constants should be named according to their purpose. E.g., if you need to store the number of cars then* `numCars` *is more descriptive then saying* `x`*.*
- *Pay attention to naming convention for variables and constants. Constants are as a practice declared in upper case; multiple words are joined by underscore. Variables are declared in lower*

case, multiple words are joined by underscore or first letter is capitalized for each word except for the first word.

- ***Document your code by writing comments where appropriate to explain what the various pieces of your code might do. Refer "additional notes" section of this document for more details.***
- *Each line should not have too much code, as a matter of practice any line longer than 80 columns should be broken down into multiple lines of code.*

Introduction

Assume you work for a company that creates computer apps to help people in their everyday world. You are currently working on a Grocery Buddy app that maintains a list of grocery items that a user would like to shop. It displays the name and quantity of desired grocery items on a list. Items can be added and removed from the list. The desired quantity of items can be updated. In addition, the app will support several other operations as described later in this document. You have been assigned the task of implementing a standard list ADT so that the app will work.

List ADT implemented as doubly linked list

A list is a linear ADT. Like any ADT, a list has two sets of members, a collection of data and a set of operations defined on the data. The data held in a list can be of varying types. A list ADT can be implemented using different data structures. In this assignment you will implement a list ADT using doubly linked list with pointers for head and tail nodes.

Getting started

Download the file Assignment1.cpp from Titanium, in this file you will find the code for your node and list classes with function prototypes that you will implement. Note that there are a lot of details to this assignment; too many to simply start programming without any forethought. For your own benefit you would want to plan your doubly linked list operations on paper by drawing out figures for the pointer manipulation. Next, you would want to work out the pseudo code for each function before you start to write your C++ code.

Operations to implement

Your grocery buddy app should support the following operations (functions):

(Note: To improve readability of your C++ code, you should define all of the below member functions outside the class using scope resolution operator.)

- **Helper functions:** Writing helper functions allows for code reuse and improves readability. Start by writing the below C++ helper functions, you may write a few more of your own as you need them.
  - `bool isEmpty() const;` checks for the list being empty, returns true if there are no items on the list or false otherwise.
  - `int length() const;` returns the number of items on the list, which is the value of `itemCount`. Note that there is no traversal of list needed to get the length. The class level data member `itemCount` keeps a running count of the items on the list. `itemCount` is updated every time a new item is added or removed from the list.
  - `void goPrevious(ItemType * & curr) const;` moves the `curr` pointer to point to the previous item. For ideas on how to write this function, you should refer to the `goNext` function that moves the pointer to point to the next item. Additionally, you should look at `printForward` function to get an idea on how to use `goNext`.

- **Destructor:** `~GroceryList();` Write a C++ destructor that deletes all items on a list.

- **Add a new grocery item to the list:** You would be writing below three C++ functions to add a new item to the list. Do not forget to increment the `itemCount` each time a new item is added.
  - `void addToFront(string itemName, double quantity);` Create a new item with `itemName` and `quantity` and add it to the front of the list. Update the `head` pointer to point to the new item.
  - `void addToBack(string itemName, double quantity);` Add a new item to the end of the list. Update the `tail` pointer to point to the new item.
  - `void add(string itemName, double quantity, int position);` Add a new item to the list at specified `position`. Items are numbered from left to right in the list. A `position` of 0 means that the item will be added to the front of the list, a `position` of 1 means that the item will become the second item on the list, and so on. A `positon` equal to,

or greater than, the length of the list means that the item is placed at the end of the list. You would want to reuse `addToFront` or `addToBack` functions depending upon the `position`.

You should include in your assignment report:

- Pseudo code for your function: `void add(string itemName, double quantity, int position)`
- Include screen shot for below input and output. Start by printing an empty list. As much as possible messages should be printed from within main function and not from functions that add a new item:

```
------ Grocery List (item:quantity) ------
You do not have any item.

Adding an item via addToFront...
------ Grocery List (item:quantity) ------
1) Avocado: 5

Adding an item via addToBack...
------ Grocery List (item:quantity) ------
1) Avocado: 5
2) Lime: 2

Adding an item via add at position 1...
------ Grocery List (item:quantity) ------
1) Avocado: 5
2) Milk: 1
3) Lime: 2
```

❖ **Print the grocery list:** `void printBackward() const;`

You are asked to write a C++ function that prints the items on the list from back to front. You can inspire your solution from the `printForward`, which has been given to you.

You should include in your assignment report the screen shot for below input and output:

```
------ Grocery List (item:quantity) ------
3) Lime: 2
2) Milk: 1
1) Avocado: 5
```

- **Remove a grocery item from the list:** You would be writing below three C++ functions to remove an item from the list. Do not forget to decrement the `itemCount` each time an item is removed.
  - `void removeFirst();` remove the first item from the list. Update the `head` pointer.
  - `void removeLast();` remove the last item from the list. Update the `tail` pointer.
  - `void remove(int position);` remove an item at the given `position` on the list. Items are numbered from left to right. A `position` of 0 means that the first item on the list (item pointed at by the `head` pointer) is removed, and so on. The function does nothing if `positon` is greater or equal to the length of the list. You would want to reuse `removeFirst` or `removeLast` functions depending upon the position.

You should include in your assignment report:

- Pseudo code for your function: `void remove(int position)`
- Screen shot for below input and output. Start by printing out the below given list. As much as possible messages should be printed from within main function and not from functions that remove an item

```
------ Grocery List (item:quantity) ------
1) Avocado: 5
2) Milk: 1
3) Lime: 2
4) Apples: 6
5) Oatmeal: 3
6) Pepper: 1
7) Eggs: 12
8) Salsa: 1
9) Quinoa: 1
```

```
Removing an item via removeFirst...
------ Grocery List (item:quantity) ------
1) Milk: 1
2) Lime: 2
3) Apples: 6
4) Oatmeal: 3
5) Pepper: 1
6) Eggs: 12
7) Salsa: 1
8) Quinoa: 1

Removing an item via removeLast...
------ Grocery List (item:quantity) ------
1) Milk: 1
2) Lime: 2
3) Apples: 6
4) Oatmeal: 3
5) Pepper: 1
6) Eggs: 12
7) Salsa: 1

Removing an item via remove from position 4...
------ Grocery List (item:quantity) ------
1) Milk: 1
2) Lime: 2
3) Apples: 6
4) Oatmeal: 3
5) Eggs: 12
6) Salsa: 1
```

❖ **Lookup an item in a list:**

- `void peek(int position) const;` Write a C++ function that looks for an item at a specified `position` on the list and prints its name and quantity. Nothing gets printed if the item's `position` lies outside the list.

- `bool lookup(string itemName) const;` Write a C++ function that looks for an item by name. It returns true if the item is found or false otherwise.

You should include in your assignment report Screen shot for below input and output. Start by printing out the below given list:

```
------ Grocery List (item:quantity) ------
1) Milk: 1
2) Lime: 2
3) Apples: 6
4) Oatmeal: 3
5) Eggs: 12
6) Salsa: 1

Peek item at position 3...
Oatmeal: 3

Looking up Apples...
Apples found

Looking up Hummus...
Hummus not found
```

❖ **Divide a grocery list into two separate lists:** `void deal(GroceryList & secondList);`
Betty has a long shopping list and she is looking to get help from Nicholas. They decide to split the list such that Nicholas will shop for **every other** item on Betty's list. Your job is to write a C++ function that removes **every other** item from the original list and adds it to the `secondList`. Note that the `secondList` is empty to start with. Remember to update the `itemCount` for the two lists.

You should include in your assignment report:

- Pseudo code for your deal function.
- Screen shot for below input and output. Start by printing Betty and Nicholas's lists, call the deal and reprint their lists:

```
Printing Betty's list...
------ Grocery List (item:quantity) ------
1) Milk: 1
2) Lime: 2
3) Apples: 6
4) Oatmeal: 3
5) Eggs: 12
6) Salsa: 1
7) Avocado: 5
8) Pepper: 1
9) Quinoa: 1

Printing Nicholas's list...
------ Grocery List (item:quantity) ------
You do not have any item.

Calling deal...
Printing Betty's list...
------ Grocery List (item:quantity) ------
1) Milk: 1
2) Apples: 6
3) Eggs: 12
4) Avocado: 5
5) Quinoa: 1

Printing Nicholas's list...
------ Grocery List (item:quantity) ------
1) Lime: 2
2) Oatmeal: 3
3) Salsa: 1
4) Pepper: 1
```

What to hand in

Phase 1:

Submit your C++ source code (cpp file) for the following functions (make sure your functions work): All helper functions, destructor, all three functions that add new item, and printBackward.

If you choose to work in pair, **both students must individually** upload the cpp file via Titanium. Write *yours and your partner's name, section number, and "Assignment 1"* at the top of your ***cpp*** file as comments.

Note: For phase 1 you do not need to turn in any report, screenshots of output or pseudo code. Only turn in the above mentioned cpp file.

Final Phase:

A) Submit your C++ source code (cpp file) with **all** functions. If you choose to work in pair, **both students must individually** upload the cpp file via Titanium. Write *yours and your partner's name, section number, and "Assignment 1"* at the top of your ***cpp*** file as comments.

B) Submit a written report electronically as a PDF or MS Word doc file through Titanium. Again, if you worked in pair then each student uploads the report individually. Write *yours and your partner's name, section number, and "Assignment 1" on the first page of your report*. **Your report should include:**

1. Give pseudo code for each of those functions as previously asked in this document. (Refer "additional notes" section of this document for tips on how to write good pseudo code).
2. Screen shots of your input and output for each of those functions as previously asked in this document. **Note that typed or hand written input/output will not be considered. You must provide screen shots of your actual program run.**
3. Do not include any C++ code in the report. It is to be uploaded separately as a .cpp file as asked in A).

Additional Notes

- **Pseudo code**

  Word of caution: It may be tempting to jump into C++ coding and later convert it to some sort of pseudo code. This is contrary to the purpose of writing pseudo code. If you find yourself in such a situation then you are missing out on the benefits that result from detailed planning.

  The idea behind pseudo code is that it is easier for people to understand than conventional programming language code, and that it is an efficient and environment-independent description of the key principles of your solution.

  Another advantage is that pseudo code is a useful tool for planning your program design. It allows you to sketch out the structure of your program and perform stepwise refinement before the actual coding takes place.

  Below is a sample C++ style pseudo code for the game of fizz buzz. It contains some programming language elements augmented with high level English description. The goal is to have enough useful details while leaving out elements not essential for human understanding of your program, like, variable declaration and system specific code.

```
fizzbuzz()
Input: none
Return: none
   for i <- 1 to 100 do
       print_number <- true
       if i is divisible by 3
           print "Fizz"
           print_number <- false
       if i is divisible by 5
           print "Buzz"
           print_number <- false
       if print_number == true
           print i
       print a newline
```

  Pseudo code style that you follow for your program does not have to be exactly as above. You may choose the style described in your textbook. In addition, there are numerous resources available on the internet that describe various popular styles, feel free to look up and choose one.

How detailed? Check for balance. If the pseudo code is hard for a person to read (too close to a particular programming language's syntax) or difficult to translate into working code, i.e., too vague, (or worse yet, both!), then something is wrong with the level of detail you have chosen to use.

- **Code Writing:** A bad approach is to write entire code all inside one function and later perform a massive refactoring, i.e., split your code into multiple cohesive functions. It leads to increased effort in integration, testing, and debugging. Good approach is to plan ahead and start by writing small cohesive functions (up to a maximum of 30 lines or what might easily fit on one screen) with reasonable refactoring of code as you go along.

- **Choice of loops:** Choose appropriate loop type (for, while or do while) in a way that it makes your program logic simpler and code more readable.

- **Commenting your code**

1. **File comments**: Start your cpp file with a description of your program, names of authors, date authored, and version. Add other comments as asked in the "what to hand in" section of this document.

2. **Commenting function header:** Right above the header of each of your function you should have a comment. The comment will have three pieces to it:

   Describe what the function accomplishes.

   Inputs: Indicates what the required parameters are for the function. "nothing" for no parameters.

   Return: Clearly states what the function computes and returns. "nothing" for void functions.

   Here is an example:

```
/*******************************************************
 * greetings will print a welcome message, specifically *
 * addressing the name supplied                         *
 * Inputs:                                              *
 *      name, a string, is the name of the person being *
 *      addressed                                       *
 * Return:                                              *
 *      nothing                                         *
 *******************************************************/

void greetings(string name)
{
    string message = "Hello, " + name + ". How are you?";
    cout << message << endl;
}
```

3. **Section comments:** Add comments for each section of your code within your function like variable declaration, taking input, computation, display output, etc.

4. **Explanatory comments** should be added for tricky or complicated or important code blocks.

5. **Line comments:** Lines that are non-obvious should get a comment at the end of the line. These end-of-line comments should be separated from the code by 2 spaces. Break long comments into multiple lines.

- **Testing:** You should test each function as you code it. Use of big bang approach to testing, i.e., postponing to test after you have written all functions would make it very difficult to isolate errors in your code.