1 Objectives

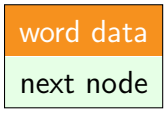
This assignment is designed to get you started with using C++ as an implementation tool, giving you practice with arrays, pointers, dynamic memory allocation and deallocation, and with writing classes. The goal hear is for you to learn about pointers and references. No matter how hard you try, you just can't completely escape pointers and references because their presence in C++ is pervasive. The more you know about them, the better you know about both how to deal with them and when and how to avoid them.

If you are new to the C++ language, you might find this assignment challenging at first while learning about pointers, references, dynamic memory, etc. However, you will be happy to know that these notions will soon become second nature, leaving you pleased with your work on this assignment.

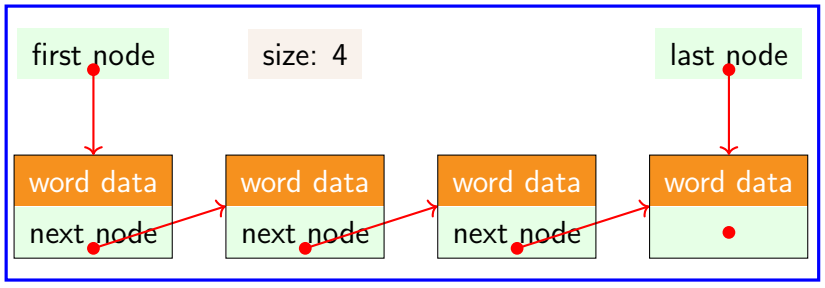
2 Assignment Background

This assignment involves implementing a custom linked list data structure called `WordList` that will store the *words* in a given text file, keeping track of their frequency of appearance in the file as well as maintaining a list of the numbers of the lines on which they each appear. A *word* in this assignment is defined as a sequence of characters that begin and end with letters.

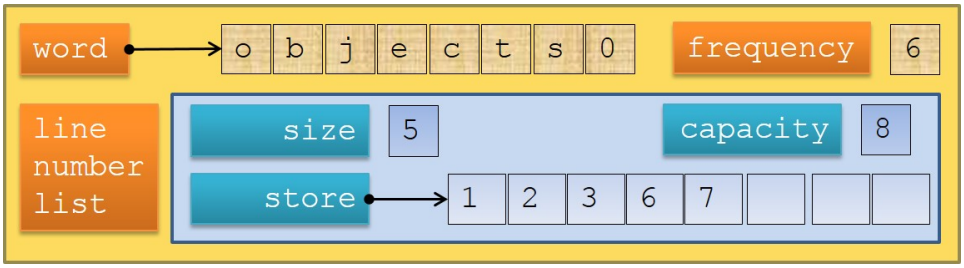
A node in `WordList` is a simple structure named `WordNode` that stores data associated with a *word* as well as a pointer to the next node in the list.



For example, a `WordList` object storing four `WordNode` objects may be depicted like this:

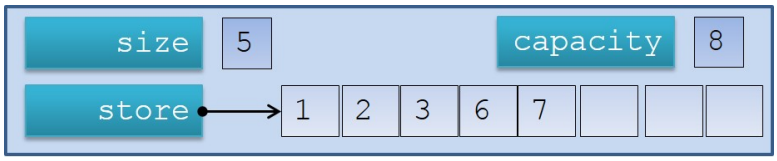


A word and the data associated with it is represented by a data type called `WordData`, an instance of which is depicted as follows:



The WordData instance object above stores the word "objects", recording that the word has appeared six times in the input file.

The line numbers of the lines on which a word appears are maintained in an array_based structure called NumList. For example, the structure



depicts an instance of NumList storing five integers $1, 2, 3, 6,$ and 7 in an array of capacity 8.

Sample Driver Program

Although the WordList class may include many useful operations in its interface, this assignment requires implementation of only a few, including a print operation to print the entire list. The implementation of these operations alone involves enough basic C++ concepts and issues to meet the objectives of this assignment.

Use the following driver code to test your WordList class:

```
// wordListDriver.cpp
#include<iostream>
using namespace std;
#include "WordList.h"

int main()
{
   WordList wl("input.txt"); // build a word list from an input file,
   wl.print(cout);           // write the entire word list to standard output,
   return 0;                 // report success
}
```

Running the driver program above against an input text file named `input.txt` with the following content

Sample input file `input.txt`

```
Do not pass objects to functions by value;
if the objects handle dynamic memory (i.e., heap memory)
do not ever pass objects to functions by value;
instead pass by reference, or even better, pass by const reference.

But if you must pass an object by value,
make sure that the class of that object defines
a copy constructor,
a copy assignment operator,
and a destructor.
That's called "the rule of three", or "the big three",
and is emphasized in the C++ literature over and over and over again!

To optimize performance, C++11 introduced
move constructor,  and
move assignment operator.
As a result, we must now follow "the rule of five" in C++.
There are 19 lines in this file.
Lines 5, and 13 are empty lines.
```

should produce an output similar to the text displayed inside the two boxes shown on the next page; please note that the actual output consists of a single column of text with no boxes or vertical line numbers.

The input text file consists of tokens separated by whitespace, where a token is a string of characters other than whitespace. In this assignment, a *word* is a token, stripped of any leading and trailing *non-alphabetical* characters; for example, the token (i.e., reduces to the word i.e, and the token C++ to the word C.

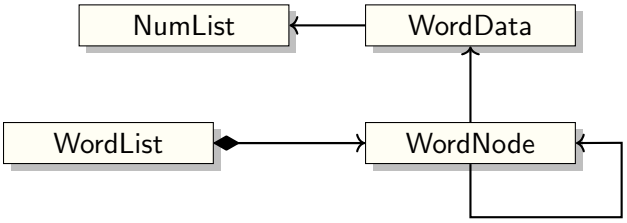
The `WordList` object `wl` defined on line 8 of the driver program stores a sorted list of all the words in the input file along with their associated frequencies and line numbers.

On line 9 of the driver program, the function call `wl.print(cout)` prints the entire list in 26 groups according to the first letter of the words. Each word in the list is printed in a field of width 15, right justified, followed by the frequency count of the word in parentheses, and followed by the list of line numbers of the lines on which the word appears.

```
WordList Source File: input.txt
============================
<A>
                a  (4) 8 9 10 17
            again  (1) 12
               an  (1) 6
              and  (6) 10 12 15 19
              are  (2) 18 19
               as  (1) 17
       assignment  (2) 9 16
<B>
           better  (1) 4
              big  (1) 11
              but  (1) 6
               by  (5) 1 3 4 6
<C>
                c  (3) 12 14 17
           called  (1) 11
            class  (1) 7
            const  (1) 4
      constructor  (2) 8 15
             copy  (2) 8 9
<D>
          defines  (1) 7
       destructor  (1) 10
               do  (2) 1 3
          dynamic  (1) 2
<E>
       emphasized  (1) 12
            empty  (1) 19
             even  (1) 4
             ever  (1) 3
<F>
             file  (1) 18
             five  (1) 17
           follow  (1) 17
        functions  (2) 1 3
<G>
<H>
           handle  (1) 2
             heap  (1) 2
<I>
              i.e  (1) 2
               if  (2) 2 6
               in  (3) 12 17 18
          instead  (1) 4
       introduced  (1) 14
               is  (1) 12
<J>
<K>
<L>
            lines  (3) 18 19
       literature  (1) 12
<M>
             make  (1) 7
           memory  (2) 2
             move  (2) 15 16
             must  (2) 6 17
<N>
              not  (2) 1 3
              now  (1) 17
<O>
           object  (2) 6 7
          objects  (3) 1 2 3
               of  (3) 7 11 17
         operator  (2) 9 16
         optimize  (1) 14
               or  (2) 4 11
             over  (3) 12
<P>
             pass  (5) 1 3 4 6
      performance  (1) 14
<Q>
<R>
        reference  (2) 4
           result  (1) 17
             rule  (2) 11 17
<S>
             sure  (1) 7
<T>
             that  (2) 7
           that's  (1) 11
              the  (6) 2 7 11 12 17
            there  (1) 18
             this  (1) 18
            three  (2) 11
               to  (3) 1 3 14
<U>
<V>
            value  (3) 1 3 6
<W>
               we  (1) 17
<X>
<Y>
              you  (1) 6
<Z>
============================
```

3 Your Task

Implement the following class diagram.



Since, for most of you, this maybe your first encounter with C++ programming, this assignment will provide you with the following detailed description of the four classes involved:

3.1 NumList

This class models an array-based list storing and managing three data members:

1. A *pointer* to a dynamically allocated array of integer elements,
2. The *size* of the list; that is, the number of elements currently stored in the array,
3. The current *capacity* of the list; that is, the number of elements for which memory has been allocated.

Class `NumList` manages dynamic memory through a default constructor, a copy constructor, a copy assignment operator, and a destructor; its default constructor creates a list with capacity 1 and size 0. If full during an add operation, a `NumList` object doubles the capacity of its internal array storage; the class implements this resizing operation in a private helper method. The public interface of the class also includes the following basic array list operations:

- ➢ Determine whether the list is empty.
- ➢ Determine whether a given element exists in the list.
- ➢ Append an element to the end of the list.
- ➢ Get/set an element at a specified position.
- ➢ Get size/capacity of the list.
- ➢ Get a read-only pointer to the underlying array.

3.2 WordData

The `WordData` class represents objects that store and manage the following data members:

- ➢ A *pointer* to a dynamically allocated array of characters that stores a specified word,
- ➢ The *frequency* count of the word
- ➢ A `NumList` object storing a list of line numbers associated with the word

The class manages dynamic memory through a constructor, a copy constructor, a copy assignment operator, and a destructor. The constructor of the class should accept a word and a line number; it should add the line number to its line number list only if it is not already there; however, it always increments the frequency count. For the sake of simplicity in this assignment, the constructor should convert the upper case letters in the incoming word to lowercase before storing it.

The public interface of the class also includes the following operations:

- Determine whether a line number is in the line number list.
- Append a given number to the list of line numbers, only if it is not already there.
- Get the frequency count.
- Get a read-only pointer to the stored word.
- Get a read-only reference to the `NumList` object.
- Determine whether the stored word compares equal to, or comes before or after a given word. Use case insensitive alphabetical ordering of strings of characters when comparing two words.
- Print the word together with its frequency count and list of line numbers to a specified `ostream` object (like `cout`).

3.3 WordNode

This class represents the nodes in a singly linked list represented by the `WordList` class. Since `WordNode` is specifically designed to represent the nodes in a `WordList`, `WordNode` should be defined as a private `struct` data type nested within the `WordList` class.

A `WordNode` object stores two public data members:

- A `WordData` object
- A pointer called `next` pointing to another `WordNode` object

and provides the following public member functions:

- A constructor that accepts a word, setting the `next` data member to `nullptr`.
- A constructor that accepts a word and a pointer to another `WordNode` object, initializing the corresponding data members.

3.4 WordList

This class represents singly linked lists of `WordNode` objects. A `WordList` object stores and manages three data members:

- A *pointer* to the first node of the list
- A *pointer* to the last node of the list
- The size of the list

The class manages dynamic memory through a default constructor, a copy constructor, a copy assignment operator, and a destructor; its default constructor creates an empty list; it also

provides the following operations in its public interface.

- ➢ Get the size of the list.
- ➢ Print the list formatted as shown on page 4.

To facilitate its internal operations, the class should implement the following `private` operations:

- ➢ Load `this` list using the words in a specified text file.
- ➢ Get a pointer to the node whose word data object stores a given word.
- ➢ Accept both a given word and a line number and then reflect them both into `this` list as follows: if the given word is already represented by a `WordNode` in the list, then simply append the given number to the list of line numbers of the `WordData` in that `WordNode`. Otherwise, create and insert a new `WordNode` object to the list at its sorted position.

4 Requirements

You may use the C++ `string` class during input file processing:

```
string filename = "input.txt";
ifstream fin(filename);
if (!fin)
{
   cout << "could not open input file: " << filename << endl;
   exit(1);
}
int linenum = 0;
string line;
getline(fin, line);  // attempt to read a line
while (fin)
{
   ++linenum;
   istringstream sin(line); // convert the line just read into an input stream
   string word;
   while (sin >> word)  // extract the words
   {
      // Take this opportnity to trim word.
      // ...
      // Your use of C++ string class ends here.
      // Back to using C-style strings
      char * charArrayWord = new char[word.length() + 1];
      std::strcpy(charArrayWord, word.c_str());
      // ...
      // process charArrayWord and linenum
      // ...
      delete[] charArrayWord; // clean up
   }
   getline(fin, line);
}
fin.close();
```

Apart from input processing above, your implementation must carry out the tasks involved *without*

the use of the container classes and algorithms from the C++ standard template library (STL).

5 Deliverables

Following the instructions given in the course outline, your assignment folder for this assignment must include the following files:

1. Header files: `NumList.h`, `WordData.h`, `WordList.h`
2. Implementation files: `NumList.cpp`, `WordData.cpp`, `WordList.cpp`, `wordListDriver.cpp`. Note that there is NO file named `WordNode.h` or `WordNode.cpp` because class `WordNode` is embedded as an inner `struct` in class `WordList`.
3. Input and output files
4. A `README.txt` text file.

6 Marking scheme

60%	Program correctness: 25% `WordList` 5% `WordNode` 15% `WordData` 15% `NumList`
20%	Proper use of pointers, dynamic memory management, and C++ concepts. No C-style memory functions such as **malloc**, **alloc**, **realloc**, **free**, etc. No C-style coding.
10%	Format, clarity, completeness of output
10%	Concise documentation of nontrivial steps in code, choice of variable names, indentation and readability of program