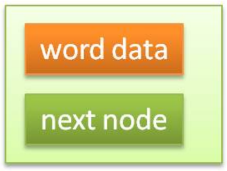
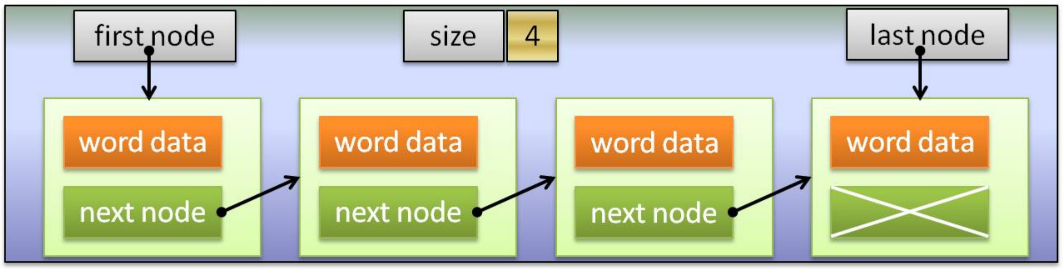
This assignment is designed to get you started with using C++ as an implementation tool, giving you practice with arrays, pointers, dynamic memory allocation and deallocation, file processing, and with writing classes.

Your task is to implements a custom data structure named `WordList` whose objects each represent a linked list of word nodes.

A word node is a simple structure named `WordNode` that stores data associated with a word, and a pointer to next word node in the list.

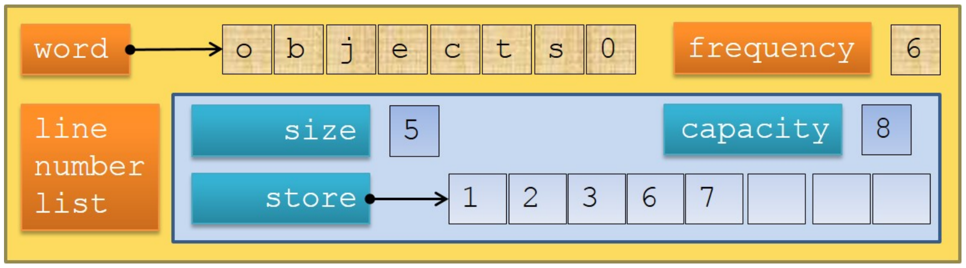


For example, a `WordList` object storing four `WordNode` objects may be depicted like this:

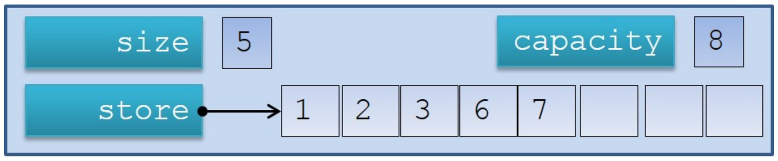


Supplied with an input text file at construction, a `WordList` object extracts all of the words in the input file, keeping track of the line numbers on which the words appear in the file. For each word extracted, the `WordList` object reflects the word and its associated line number into the word data component of one of its `WordNode` objects.

A word data object is represented by a structure named `WordData`, storing a word together with the list of line numbers associated with that word, and with the frequency of occurrence of that word in the file. For example, if the word "`objects`" occurs six times in the input file on lines 1, 2, 3, 3, 6, 7, the `WordData` object storing the word may be depicted as follows:



The line number list stored in a `WordData` is represented by an array-based list structure named `IntList`. An object of `IntList` stores a pointer to a dynamic array of integers, the list size (i.e., the number of integers currently stored in the array), and the list capacity (i.e., the storage capacity of the array). For example, an `IntList` object storing the integers 1, 2, 3, 6, 7 may be depicted as follows:



Sample Program Run

Although the `WordList` class may include many useful list operations in its public 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:

```cpp
// 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
}
```

Using an input text file that contains the following sample text

```
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 these objects by value,
make sure that the class of these objects 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 again!
```

the program should produce an output formatted as follows:[1]

```
Word Collection Source File: input.txt
============================
<A>
              a  (3) 8 9 10
          again  (1) 12
            and  (4) 10 12
     assignment  (1) 9
<B>
         better  (1) 4
            Big  (1) 11
            But  (1) 6
             by  (5) 1 3 4 6
<C>
              C  (1) 12
         called  (1) 11
          class  (1) 7
          const  (1) 4
    constructor  (1) 8
           copy  (2) 8 9
<D>
        defines  (1) 7
     destructor  (1) 10
             Do  (2) 1 3
        dynamic  (1) 2
<E>
     emphasized  (1) 12
           even  (1) 4
           ever  (1) 3
<F>
      functions  (2) 1 3
<G>
<H>
         handle  (1) 2
           heap  (1) 2
<I>
            i.e  (1) 2
             if  (2) 2 6
             in  (1) 12
        instead  (1) 4
             is  (1) 12
```

```
<J>
<K>
<L>
     literature  (1) 12
<M>
           make  (1) 7
         memory  (2) 2
           must  (1) 6
<N>
            not  (2) 1 3
<O>
        objects  (5) 1 2 3 6 7
             of  (2) 7 11
       operator  (1) 9
             or  (2) 4 11
           over  (3) 12
<P>
           pass  (5) 1 3 4 6
<Q>
<R>
      reference  (2) 4
           Rule  (1) 11
<S>
           sure  (1) 7
<T>
           that  (1) 7
         That's  (1) 11
            the  (5) 2 7 11 12
          these  (2) 6 7
          Three  (2) 11
             to  (2) 1 3
<U>
<V>
          value  (3) 1 3 6
<W>
<X>
<Y>
            you  (1) 6
<Z>
```

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

On line 8 in the driver code above, by the time the construction of the `WordList` object `wl` is complete, `wl` stores a sorted list of word nodes, each uniquely representing a word in the supplied text file `input.txt`.

[1]The actual output of the program consists of only one unboxed column, of course.

On line 9, the function call `wl.print(cout)` prints the entire list in 26 groups according to the first letter in 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.

Requirements

You will implement the four data structures depicted above in terms of four C++ classes: `IntList`, `WordData`, `WordNode` and `WordList`.

Implementation of these classes will provide plenty of opportunity for you to practice managing dynamic memory as well as using pointer and reference variables in C++. To avoid obscuring the objectives of this assignment, the operations that these classes are required to provide are kept to a minimum. Feel free to implement other operations in addition to those listed for each class. However, your implementation should carry out the tasks involved *without* the use of the container classes and algorithms from the C++ standard template library (STL).

`IntList`

This class represents array-based lists, each 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.

The `IntList` class 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. Moreover, when full during an add operation, an `IntList` 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.

`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
- ➢ An `IntList` 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; its constructor initializes its data members according to a supplied pair of word and a number. The public interface of the class also includes the following operations:

- ➢ Append a given number to the list line numbers, incrementing the frequency.
- ➢ Get frequency count.
- ➢ Get a read-only pointer to the stored word.
- ➢ Get a read-only reference to the `IntList` 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`).

`WordNode`

This class represents the nodes in a singly linked list represented by the `WordList` class, and is defined as a private `struct` nested within the `WordList` class.

A `WordNode` object stores two public data members:

- ➢ A `WordData` object
- ➢ A pointer to a `WordNode` object

and provides the following public member functions:

- ➢ A constructor that initializes the data members above using a pair of values supplied through its parameters.

`WordList`

This class represents singly linked lists of `WordData` 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 3.

To facilitae its internal operations, the class implements the following private operations:

➢ Load `this` list using the words in an input text file.
➢ Get a pointer to the node whose word data object stores a given word.
➢ Reflect a given a word and its corresponding line number into `this` list as follows: if the given word does not exist in the list then create and insert a new `WordNode` object to the list at its sorted position; otherwise, just append the given number to the list of line numbers of the given word.

Deliverables

Create a a new folder that contains the files listed below, and then zip up your folder and *please* submit it exactly *as instructed* in the course outline.

1. Header files: `IntList.h`, `WordData.h`, `WordList.h`
2. Implementation files: `IntList.cpp`, `WordData.cpp`, `WordList.cpp`, `wordListDriver.cpp`
3. Input and output files
4. A `README.txt` text file.

Marking scheme

60%	Program correctness
20%	Proper use of pointers, dynamic memory management, and C++ concepts
10%	Format, clarity, completeness of output
10%	Concise documentation of nontrivial steps in code, choice of variable names, indentation and readability of program