CSS 501 - Application 4 - Concordance Generator
Design and Specification
Fall, 2017

Thurman Gillespy
11/22/17

## Problem Statement

Develop an application that creates a concordance (concordance generator) from a text document (corpus). A concordance is a list of words (the key words) in a document (with some exceptions) with (in this program) 5 words before and after the key word (the context). A key word that appears more than once will have a separate record and output of each context. Common words like "a", "an", and "the" are excluded (stop words), and are maintained in a separate list.

Program output conforms to the Keys Words in Context (KWIC) format, and a list of the words with contexts is known as a KWIC index. Format consists of a sorted list of key words, with before and after contexts, one key word per line, with the key words aligned on their left margins in the center of the page. Context words to the left of the key word are right aligned with the key word, and context words on the right are left aligned. If there is more than one context entry for a key word, that word is output again with the additional context, in the same order as the corpus.

Sample output (from the Gettysburg address). Note the two contexts for key word 'conceived'.

```
take increased devotion to that cause for which they gave the
        are engaged in a great civil war, testing whether that nation,
          of that war. We have come to dedicate a portion of
  this continent, a new nation, conceived in Liberty, and dedicated to
       nation, or any nation so conceived and so dedicated, can long
```

Capitalization and punctuation are maintained (see sample, above) (listed as optional in the assignment). The key words, however, are used as index values in the program after trailing punctuation is removed and the entire word is converted to lower case; internal punctuation, such as hyphens and apostrophes, is preserved. When the key word is displayed with the adjacent context words, however, the original, non-converted format is presented. Numbers are not included in the key words, but are included in the context.

Examples of key word transformations

| | |
|---|---|
| Frank | frank |
| however, | however |
| examples: | examples |
| it's | it's |
| http://home.com | http://home.com |
| its' | its |
| carry-out | carry-out |

## User Scenario

| **Creating a concordance** |
|---|
| **Initial assumption**: A text file (corpus) is present in the program directory, with name format <file_name.txt>. Excluded words (stop words) are in a list in the same directory with name format <stopwords.txt>. More than one corpus may be present, but each must have an associated stop word list. |
| **Normal:** User executes program from command line. A single argument passed to the program is the name of the text file that is analyzed. Program outputs the KWIC index to stdout, which the user can redirect to an output file. |
| **What can go wrong:** User fails to enter a file name. User enters the incorrect file name. Input file is not in same directory as program. Excluded words file does not exist, or is in the wrong directory. File is corrupted, or is not in ASCII format. |
| **Other activities:** None. |
| **System state on completion:** The KWIC concordance, minus stop words, has been outputted to the computer screen, or was redirected to an output file. Input files are not changed. |

## System Architecture

Program consists of 2 binary search trees (BST): one for the stop words and the other for the KWIC index (concordance). The BSTs are instantiated from a BST template as BST<string> and BST<CData> objects. For both BSTs, each BST node contains a single, non-duplicated instance of the stop word or key word. The stop words are contained in the BST<string> object.

The concordance BST (BST<CData>) stores the before and after context strings together in a LinkedList (ListListC<string>). For each entry of the key word, the two context strings are stored in a new LinkedList node (LNode<string>). Thus, each BST node in the BST<CData> object contains a linked list of one or more context strings (the 5 words before and after the key word), in the order they appeared in the text.

When reading data from the corpus input file, the key word (converted), before context string and after context string are placed into a DataIn object using a 3 item constructor. The DataIn object is passed to the insert() BST method. For the stop word BST, the DataIn object is created with a 1 item constructor, likewise passed to the insert() method.

The maximum character length of the before context string is recorded. When the KWIC index is output, that length is used to center the key words near the center of the page using the

```
cout << setw(len) << right << (before context)
```

sequence.

UML Diagrams

| LNode<T> |
| --- |
| - before : T<br>- after : T<br>- next : LNode* |
| + LNode(before : T, after : T)<br>+ getBeforeContext() : T<br>+ getAfterContext() : T |

| LinkedListC<T> |
| --- |
| - head : LNode*<br>- tail: LNode*<br>- next : LNode*<br>- currNode: LNode* |
| + LinkedListC()<br>+ addNode(leftC : T, rightC : T)<br>+ clear()<br>+ resetCounter()<br>+ advanceCounter() : boolean |

| BST<N> // string or CData |
| --- |
| - root : BNode<N>* |
| + BST()<br>+ find( item : T) : boolean<br>+ insert(item : DataIn*) : boolean<br>+ print()<br>- find(curr : BNode*, item : T) : boolean<br>- insert(curr : BNode*, item : DataIn*) : boolean<br>- inOrder(curr : BNode*) |

| BNode<N> // N => string or CData |
| --- |
| - item : N<br>- left : BNode*<br>- right : BNode* |
| + BNode(item : N) // string or CData<br>+ getItem() : T<br>+ insertItem(item : N) |

| CData<T> |
| --- |
| - keyWord : T<br>- context : LinkedListC<T> |
| + CData(data : DataIn)<br>+ update(data : DataIn)<br>+ operator=(in: DataIn&) : CData&<br>+ operator<(CData& rhs) : boolean<br>+ operator==(CData& rhs) : boolean |

| DataIn<T> |
| --- |
| - keyWord : T<br>- before : T<br>- after : T |
| DataIn(keyWord : T)<br>DataIn(keyWord : T, before : T, after : T) |

| string |
| --- |
| |
| operator=(sin : string&, data: DataIn&) : string& |

UML Sequence Diagrams (see attached)

## Test Plan

### LinkedList<T> with LNode<T>

Create a test program that instantiates a LinkedListC<string> and LinkedListC<int> with LNode<T>.

Add a series of integers and strings to the objects, then do an inorder traverse and print the nodes contents. Try with one entry, two entries and 5 entries. Ensure the output matches the input in content and correct sequence. Invoke the clear() method and check that all allocated memory has been deleted.

### stop words BST

Create a BST<string> object to handle the stop words. Read a stop words file, and each entry into a DataIn object. Insert into BST with insert(DataIn). Test find() with words that are and are not in the tree. Print the output of an inorder traverse and check that the output is in correct ascending alphabetic order. Test with stop words file that is empty, has one entry, and has multiple entries. On program termination, check for memory leaks.

### concordance BST

Create a BST<CData> object to handle a corpus text. Read the corpus file, putting the words into an array of 11 strings. From that array, extract a key word and 2 context strings. Insert the strings into the object with a DataStruct object. Print the object with an inorder traversal. Check that the key words are printed in ascending alphabetic order. Key words with multiple context strings should be reprinted with the different context. Check that words at the start and end of the corpus are printed with correct left and right context strings. Check that the output format matches the sample shown above. Test with corpus files that are empty, has one word, and has multiple words (the Gettysburg address). On program termination, check for memory leaks.

### CData

CData overloaded =, == and < operators are checked by program that assigns values to a CData and DataIn objects, and validates correct function of the operations.

### string

string overloaded = operator is checked by a test program that assigns values to a string and DataIn object, and ensures that previous content in string is replace by the first attribute of DataIn.

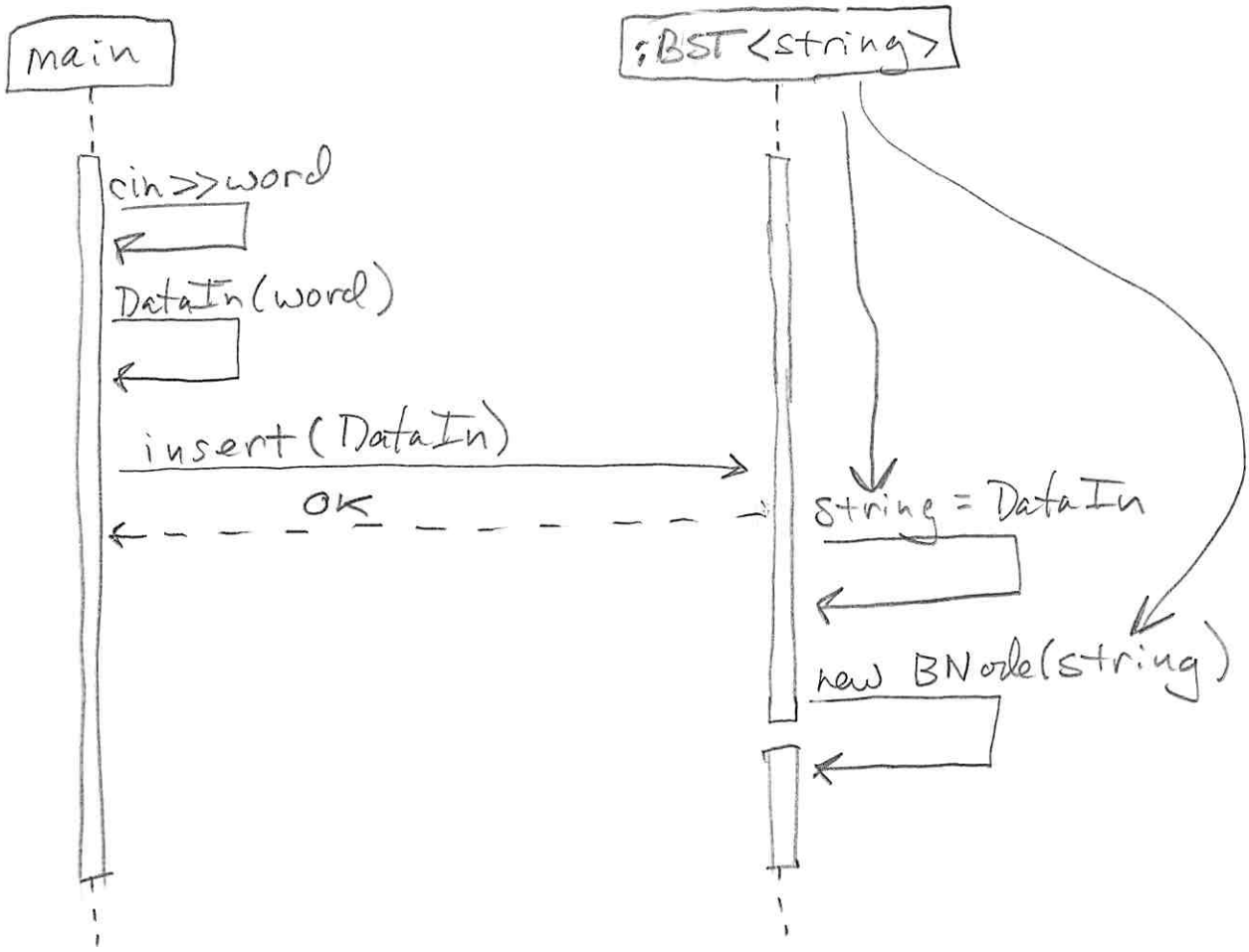
### DataIn

DataIn is checked with a simple program that checks the contents of the object after using the single parameter and three parameter contructors.

User testing

| User action | Result |
|---|---|
| no parameters entered on command line | Error message, abort |
| incorrect file name | Error message, abort |
| file not present, or different directory | Error message, abort |
| stopwords.txt not present or wrong directory | KWIC index printed with all words in correct format |
| correct file name, stopwords.txt present | KWIC index printed in currect format without stop words |
| sample corpus with examples of punctuation to be excluded and included (see key word transformation table, above) | correct storing and output of key words |
| not tested: non ASCII file, corrupted ASCII file | |

(1.) stop word BST



(2.) concordance BST