```
$Id: asg4-listmap-templates.mm,v 1.31 2014-04-24 18:23:49-07 - - $
PWD: /afs/cats.ucsc.edu/courses/cmps109-wm/Assignments/asg3-listmap-templates
URL: http://www2.ucsc.edu/courses/cmps109-wm/:/Assignments/asg3-listmap-templates/
```

1. Overview

In this assignment, you will implement template code and not use any template classes from the STL. You will also write your own code to handle files. Refer to the earlier assignment as to how to open and read files. You may use **<cassert>**, **<cerrno>**, **<cstdlib>**, **<exception>**, **<fstream>**, **<iomanip>**, **<iostream>**, **<stdexcept>**, **<string>**, **<typeinfo>**, and if you think anything else is needed, post a question to the mailing list. Specifically, you may not use any classes that take template parameters, such as **<iterator>**, **<list>**, **<map>**, **<pair>**, **<vector>**, except for those you write yourself.

2. Program Specification

The program is specified in the format of a Unix **man**(1) page.

NAME

keyvalue — manage a list of key and value pairs

SYNOPSIS

keyvalue [**-@** *flags*] [*filename* . . .]

DESCRIPTION

Input is read from each file in turn. Before any processing, each input line is echoed to **cout**, preceded by its filename and line number within the file. The name of **cin** is printed as a minus sign (**-**).

Each non-comment line causes some action on the part of the program, as described below. Before processing a command, leading and trailing white space is trimmed off of the key and off of the value. White space interior to the key or value is not trimmed. When a key and value pair is printed, the equivalent of the format string used is **"%s = %s\n"**. Of course, use **<iostream>**, not **<stdio>**. The newline character is removed from any input line before processing. If there is more than one equal sign on the line, the first separates the key from the value, and the rest are part of the value. Input lines are one of the following:

#

Any input line whose first non-whitespace character is a hash (**#**) is ignored as a comment. This means that no key can begin with a hash. An empty line or a line consisting of nothing but white space is ignored.

key

A line consisting of at least one non-whitespace character and no equal sign causes the key and value pair to be printed. If not found, the message

key: **key not found**

is printed. Note that the characters in italics are not printed exactly. The actual key is printed. This message is printed to **cout**.

key **=**
: If there is only whitespace after the equal sign, the key and value pair is deleted from the map.

key **=** *value*
: If the key is found in the map, its value field is replaced by the new value. If not found, the key and value are inserted in increasing lexicographic order, sorted by key. The new key and value pair is printed.

=
: All of the key and value pairs in the map are printed in lexicographic order.

= *value*
: All of the key and value pairs with the given value are printed in lexicographic order sorted by key.

OPTIONS

The `-@` option is followed by a sequence of flags to enable debugging output, which is written to the standard error. The option flags are only meaningful to the programmer.

OPERANDS

Each operand is the name of a file to be read. If no filenames are specified, `cin` is read. If filenames are specified, a filename consisting of a single minus sign (-) causes `cin` to be read in sequence at that position. Any file that can not be accessed causes a message in proper format to be printed to `cerr`.

EXIT STATUS

0 No errors were found.

1 There were some problems accessing files, and error messages were reported to `cerr`.

3. Implementation Sequence

In this assignment, you will constuct a program from scratch, using some of the code from previous assignments.

(a) Study the behavior of `misc/pkeyvalue.perl`, whose behavior your program should emulate. The Perl version does not support the debug option of your program.

(b) Copy `Makefile` from your previous assignment, and edit it so that it will build and submit your new assignment.

(c) Implement your main program whose name is `main.cpp`, and handle files in the same way as the sample Perl code. Instead of trying to use a map, just print debug statements showing which of the five kinds of statements are recognized, printing out the key and value portion of the statement.

(d) Instead of `<pair>` from the STL, you will use `xpair.*`.

(e) You will be using a linear linked list to implement your data structure. This is obviously unacceptable in terms of a real data structures problem, since unit operations will run in $O(n)$ time instead of the proper $O(\log_2 n)$ time for a

balanced binary search tree. But iteration over a binary search tree is rather complex and will not contribute to your learning about how to implement templates. And balancing a BST is part of CMPS-101, which is not a prerequisite for this course.

(f) Look at `xless.h` and `misc/testxless.cpp`, which show how to create and use an `xless` object to make comparisons. The `listmap` class assumes this has already been declared.

(g) The files `*.tcc` are explicit template instantiations that should not be necessary, since the compiler ought to be able to figure out these requirements themselves and automatically perform the instantiations. `.cpp` file ***after*** the definition of the template function. They are placed in separate files in order to keep the implementation file separate from excessive coupling between implementation and client modules.

4. The main function

Replace the code in the main function to do options analysis. Then for each input line, parse the line by splitting it up into two pieces separated by the first equal (`=`) sign. Then trim leading and trailing white space off each end, and perform the appropriate function.

(a) Use `find_first_of` to locate the first `'='` in the string and then `substr` to pull off the various parts.

(b) Use `find_first_not_of` and `find_last_not_of` to find the first and last non-whitespace characters in the string. Write your own function `trim`.

(c) Use an `if-else` sequence to determine which command is being parsed.

5. Template class `listmap`

We now examine the class `listmap`, which is partially implemented for you. You need not implement functions that are never called.

(a) `template <typename Key, typename Value, class Less=xless<Key>>`
`class listmap`
defines the template class with three arguments. `Key` and `Value` are the elements to be kept in the list. `Less` is the class used to determine the ordering of the list and defaults to `xless<Key>`.

(b) `typedef Key key_type;`
`typedef Value mapped_type;`
`typedef xpair<key_type,mapped_type> value_type;`
are some standard names given to usual STL types. Note that the value type is an `xpair`, not what is normally thought as the value, which here is called the mapped type.
`struct node`
is a private node used to hold a value type along with forward and backword links to form a doubly linked list. A list map keeps track of the head and tail.

(c) `listmap();`
`listmap (const listmap&);`

`listmap& operator= (const listmap&);`
`~listmap();`
The usual four members are overriden and explicitly defined.

(d) `void insert (const value_type&);` Note that insertion takes a pair as a single argument. If the key is already there, the value is replaced, otherwise there is a new entry inserted into the list.

(e) `iterator find (const key_type&) const;` Searches and returns an iterator.

(f) `iterator begin();`
`iterator end();`
The usual iterator generators. We don't bother here with a constant iterator.

6. Template class `listmap::iterator`

Although the iterator is nested inside the list map, it is easier to read when specified separately.

(a) `class listmap<Key,Value,Less>::iterator` specifies precisely which class the iterator belongs to.

(b) `friend class listmap<Key,Value>;` Only a listmap is permitted to construct a valid iterator.

(c) `iterator (listmap* map, node* where);` The iterator keeps track of both the node and the list as a whole, so that `end()` can return an iterator "off the end", which can the find the tail element.

(d) `value_type& operator*();` Returns a reference to some value type (key and value pair) in the list. Selections are then by dot (`.`).

(e) `value_type* operator->();` Returns a pointer to some value type, from which fields can be selected with an arrow (`->`).

(f) `iterator& operator++(); //++itor`
`iterator& operator--(); //--itor`
Move backwards and forwards along the list. Moving off the end with `++` and moving from an end iterator to the last element requires special coding.

(g) `void erase();` Removes the key and value pair from the list.

7. What to Submit

`Makefile`, `README`, and all necessary C++ header and implementation files. And don't forget `checksource`. If you are using pair programming, also submit `PARTNER`.