Bible Map Assignment (3 parts)

Part 1 - (30 points)

Technical Goals:

- To understand and make practical use of
  - Class templates
  - Exception handling
  - Operator overloading
  - Conversion operator
  - Nested classes (aka inner classes)
  - Namespaces

Project Goal:

- Create a **Map** template can be used to create map classes given any combination of key value types. This will be done by completing the **Map** template and nested **Wrapper** class methods in the provided code.
- Write a test program to prove that a **Map** template specialization works.

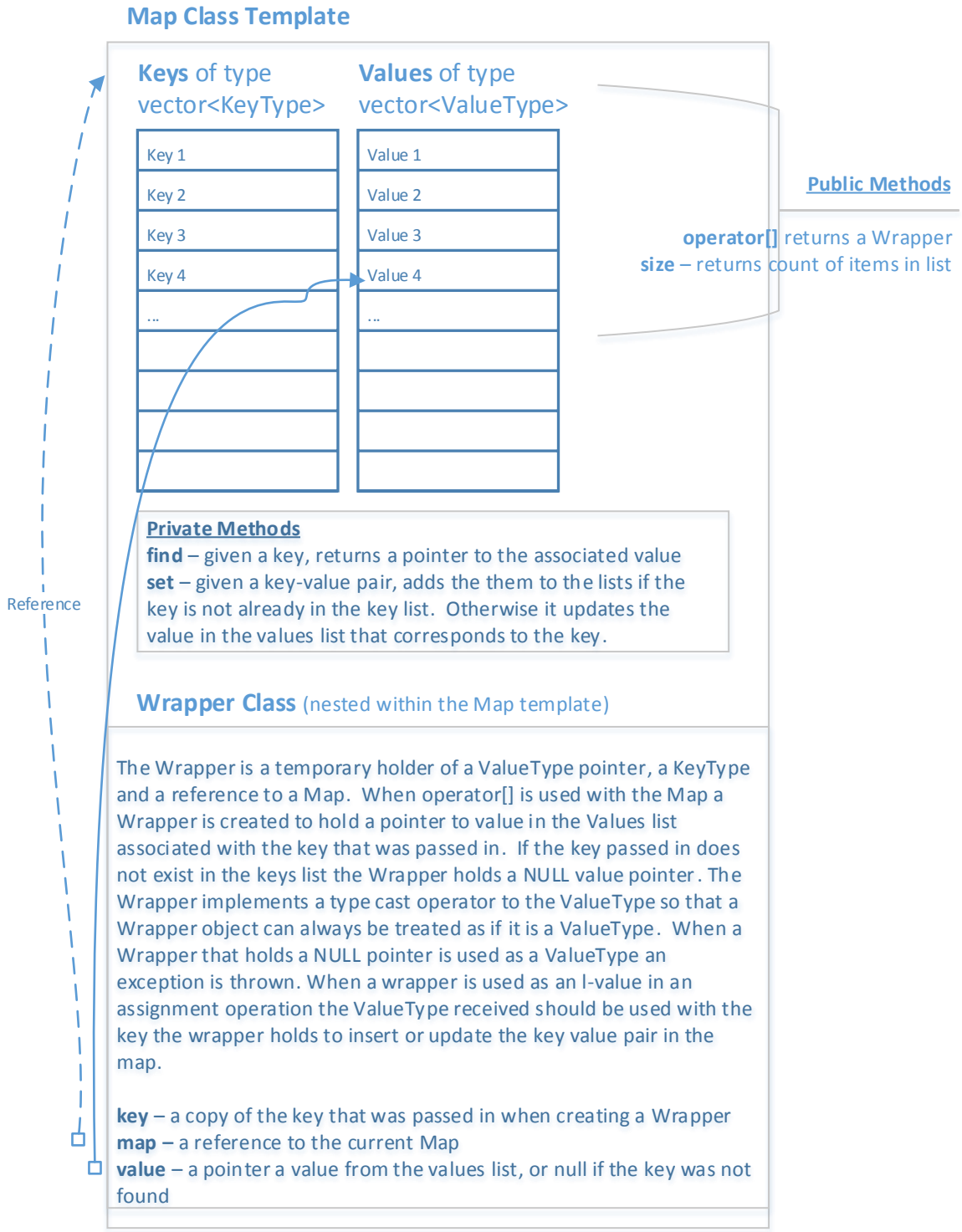
Tasks:

1. Implement the methods in the Map.hpp file by completing the instructions in the comments.
2. Test the Map template by creating a template specialization using simple data types like integers. The test program should show that:
   a. Key/value pairs can be added to the map using the [] operator.
   b. Values can retrieved from the map using the [] operator and a key.
   c. Values can be updated in the map using the [] operator.
   d. An exception is thrown when the result of the []operator is used with a key that is not in the map and
      i. the result is being used an r-Value
      ii. the address of the result is requested

   (Be sure to catch this exception in your test program and display an error message. The exception should NOT be caught in the Map or the Wrapper.)

Classes:

- Templates and classes related to the map data structure
  - Map – See code provided and diagram
  - Wrapper– See code provided and diagram
    - Be sure an exception is thrown when you try to use a value from the map with a key that does not exist. Remember the exception is not thrown until the Wrapper object is "cast" to the ValueType reference or the (&) address of the ValueType is requested. (Note: this means that exceptions should only be thrown from these two methods.)



What to turn in:

- Map.zip that contains your visual studio project minus the binary files.

Part 2- (30 points)

Technical Goals:

- To make use of the Map Template with user defined data types rather than simple types.

Project Goal:

- Write a bible verse searching application.

Tasks:

1. Download the text of the bible from https://www.harding.edu/dsteil/345/assignments/bible.txt
2. Create the VerseKey, Verse and Bible classes as described below. These classes should all have their own header and implementation files.
3. Create a bibleSearcher.cpp file that:
    a. Creates a Bible object named bible
    b. Repeatedly
        i. Get a VerseKey from the user using the stream extraction operator of the VerseKey.
        ii. Find the verse in the map using the [] operator of the Bible class.
        iii. Display the verse text if it was found.
        iv. Display a message like "Verse not found" if the verse is not found in the bible.
        v. Continue prompting the user for a VerseKey until they enter "Quit" as the book to search.

Classes:

- VerseKey - A class that uniquely identifies a verse from the bible.
    - book – string
    - chapter – int
    - verseNumber – int
    - Overload the comparison operator so that this type can be used as a key in the Map template.
    - Overload the stream extraction operator for the VerseKey. This should prompt the user for a book, chapter, and verseNumber and store the values in the VerseKey data members.
- Verse - A class that "has a" VerseKey and the text of the verse
    - verseKey - VerseKey
    - verseText – string that hold the text of the Bible verse
- Bible - "is a" Map of VerseKey and Verse
    - Inherits from Map<VerseKey,Verse>. So it inherits from a Map specialization.
    - When a Bible is created it should be filled with all of the VerseKey and Verse combinations from bible.txt. This should happen in the Bible constructor.

What to turn in:

- Map.zip that contains your visual studio project minus the binary files.

Part 3- (30 points)

Technical Goals:

- To understand and apply the following design patterns
    - Singleton
    - Iterator
    - Simple Factory (Not really a design pattern)
    - Visitor

Project Goals:

- Guarantee that only one Bible object can be created per execution of the program.
- Provide a means of iterating through a Map like the iterators built into the c++ STL.
- Provide a means of performing an operation on ("visiting") all key-value pairs in a map.

Tasks:

1. Modify the Bible to be a **Singleton** Class.
    a. A private static Bible pointer should be added and initialized to NULL.
    b. A public static getInstance method with no arguments should be added that returns a reference to the only instance of the Bible. The first time this is called the new Bible should be created and the static Bible pointer should be made to point to it.
    c. The Bible default and copy constructors should be made private.
2. Add an **Iterator** type as a nested template class of the map.
    a. The Iterator can be used to iterate through a map template specialization. It should have a reference to the map, and an index.
    b. The interator constructor should take a reference to the map as an argument in its constructor.
    c. The prefix and postfix increment and decrement operators should be overloaded. The iterator should be made to wrap when it reaches the beginning or the end of the map.
    d. The **Simple Factory** design pattern should be used to get an instance of an Iterator.
        i. The constructor of the Iterator should be private.
        ii. A getIterator method should be added to the Map. It should receive a KeyType argument. The method should throw an exception if the key is not found in the map. When possible the method should create and initialize a new Iterator and return it.
    e. The * operator should be overloaded for the Iterator to return a reference to the ValueType from the current index in the values list.
3. Use the **Visitor** design pattern to establish a means of "visiting" all of the key-value pairs in a Map.
    a. Add an IVisitor as a class template nested within the Map. It is to be an interface with one pure virtual method named visit. The visit method should take two arguments; a reference to a KeyType and a reference to a ValueType.
    b. Add a visit method to the map. The visit method should take an IVisitor as an argument. The visit method should loop through each of the keys/values in the map and pass the pairs to the visitor's visit method.
    c. Create a BibleWordCounter class that inherits from IVisitor. BibleWordCounter should have:
        i. Its own header and implementation files.

ii. A string data member for the word that will be counted
iii. A count data member for the number of times the word is found.
iv. A constructor that initializes the count to 0 and the word to a parameter that is passed in.
v. An implementation of the visit method that increments the count if the word is found in the value that is being visited.

d. Create one additional Visitor of your own choosing and illustrate that it works.

4. Write a test program that:
    a. Gets and stores a reference to the Bible using the getInstance method.
    b. Illustrates that the Bible Iterator works. Create an Iterator by calling the buildIterator method of the Map passing in a VerseKey. Show that the iterator can move backward, forward, and wraps at the ends as instructed.
    c. Illustrates that the BibleWordCounter works with a work of your choice.

What to turn in:

- Map.zip that contains your visual studio project minus the binary files.