Project 8 Documentation

Purpose of the Program

==================

The core objectives of this project are to introduce us to the concept of recursions in program schemes as well as test our ability to work and implement STL containers and their corresponding functionalities. In the process, an extensive knowledge of templates and dynamic data structures will be used to inform on the mechanics of the program for review purposes. In addition, the project will refresh knowledge on the manipulation of dynamic memory, classes, pointers, and iostream to all extents. In particular, the project will make extensive use of STL std::vectors for data manipulation through the successful implementation of two different general templated recursive functions designed to handle std::vectors.

Program Design

=============

The program works to implement two recursive functions of a unique templated class that is designed to both sorting and binary search std::vector objects. These sort and search functions will be translated as template-based generic versions for the purposes of the program, both derived from a base version of the source code meant for data manipulation of simple int arrays. The final source code makes use of vector object manipulation by making use of the standard-provided functionalities of the vector libraries. The two functions are called vector\_resort (recursive sort) and vector\_research (recursive binary search).

The vector\_resort function takes in a std::vector by-Reference, along with the indices of the first and last elements of the std::vector object. Since the std::vector as part of the STL is a templated construct, the function is also templated for generic use. The function is meant to perform recursive sorting of the vector elements through the Quicksort variant algorithm. Specifically, this entails the selection of any random element in the vector as a pivot value, with which every other element of the vector is compared to for reference in terms of their sorting location. For this particular implementation, the first value of the vector in its current state is selected as the pivot. The two indices passed in as parameters serve as initialized values for two markers called leftside and rightside that move toward each other until each detect a pair of elements, with one greater than or equal and another lesser than or equal to the pivot, that are in the wrong order relative to each other. These inverted elements are swapped, and the process repeats with the indices moving toward each other until they meet, after which the algorithm stops and terminates. The pivot is then swapped with the element at the rightside marker index. The function then has two recursive calls for the two sub-vectors made up of the elements before and after the initial pivot value respectively, repeating the entire sorting algorithm until all vector elements are sorted.

The vector\_research function takes in an already sorted std::vector by-Reference, along with the indices of both the first and last elements of the std::vector and a const T& value, which is the one searched for. The function is also templated sicne the std::vector as part of the STL is a templated contruct. Most of the functionality involves performing a recursive binary search for the provided target value in the std::vector. The binary search is performed by first picking a value in the middle of the container to act as the pivot; if the search element is less than the pivot the function narrows the search to the bottom half of the vector container, otherwise it searches the top half of the container. This process is then done recursively until the target value is found, and the index where it is found is returned. If the item is not found in the vector container, then the function will return -1 as its return value.

The second part of the project involved creating a vecInt, a std::vector of ints filled with 100 random int numbers which are meant to be read from the provided file RandomData.txt, as well as a vecIntCpy, another std::vector of ints which is a copy of the previously created vecInt. Both vector\_research and vector\_resort were then applied on vecInt, resulting in the print out to the terminal of the resulting vecInt. In addition, the index of the target value searched for by the vector\_research is printed out to the terminal after all the elements of the vecInt and vecIntCpy containers have been printed out to the terminal.

Problems of Program

================

During the process of coding for the program, there were some issues that turned up concerning exact details about the implementation of the vector\_resort function for certain arrays of values and later some vector STL containers filled with random values. Specifically, the vector\_resort function initially proved to be unable to handle the proper sorting of int arrays and vector containers with sets of elements that had either repeated values, duplicate values that were equal to the pivot at any point during sorting, or both. This oversight resulted in a program that would become stuck in an infinite loop without moving either of the index markers as they would swap identical values or elements with each other without termination, causing the failure of the terminal to print out properly the sorted elements of an array or vector. This was later fixed by adjusting a conditional for the leftside marker that allows the marker to move if it would ever encounter an element in a std::vector that was equal to the current pivot value being used for reference.

Changes to Program

================

Given the flexible functionalities of the finished project source code and its successful implementation in various situations and cases with regards to the elements that it uses to populate the vectors, I am rather proud with the current source code such that I would not desire to change it too much. If I really had some more time to fiddle with the source code, I would probably have liked to add extra functionalities that gave the source code the ability to sort and search the vecIntCpy container using the std::binary\_search and std::sort STL methods. I would also have tried to add to the implementation of vector\_research such that it returns std::vector<T>::iterator to the element in question, being sure to return std::vector<T>::end() if the item was not found. I would also have considered changing the implementation of the vector\_resort function slightly to see if I could successfully implement its functionalities using middle values as pivots.