CS 202 - Computer Science II

Project 11

Due date (FIXED): Tuesday, 5/14/2019, 11:59 pm

Objectives: The main objectives of this project is to introduce you to recursion, and test your ability to work with some STL functionalities. A review of your knowledge on working with templates, dynamic data structures, as well as manipulating dynamic memory, classes, pointers and iostream to all extents, is also included.

Description:
For the entirety of this project, you will work with STL **`std::vectors`**. You are free to implement any of the standard-provided functionalities with the **`<vector>`** libraries.
For the first part of this assignment, you will create three recursive functions:

a) **The `vector_resort` Function (Recursive Sort)**

Parameters List: Takes in a **`std::vector`** by-Reference. Note that a **`std::vector`** as part of the STL is a templated construct, hence the function will itself also need to be templated.
Hint: This is the base parameter requirement. Any other parameters necessary for the function will depend on your specific implementation.

Functionality: The function will have to perform Recursive Sorting of the vector elements. You may implement whichever sorting variant you wish (from the ones introduced in the Lab sections or any other established variant). Note: One of the most powerful naturally-recursive sorting algorithms is Quicksort (https://en.wikipedia.org/wiki/Quicksort)

Output: Nothing, since the **`std::vector`** is passed by reference it should be sorted at the return of the **`vector_resort`** function.

b) **The `vector_research` Function (Recursive Binary Search)**

Parameters List: Takes in a **`std::vector`** by-Reference, which has to be already sorted. Note that a **`std::vector`** as part of the STL is a templated construct, hence the function will itself also need to be templated. The function also takes in a const T& value, which is the one that is searched.
Hint: These are the base parameter requirements. Any other parameters necessary for the function will depend on your specific implementation.

Functionality: The function will have to perform Recursive Binary Search (https://en.wikipedia.org/wiki/Binary_search_algorithm) as introduced during the class Lectures, for the provided value in the **`std::vector`**. A Binary Search is performed as follows: Pick the value in the middle of the container (the pivot); if the search item is less than the pivot narrow the search to the bottom half of the container, otherwise search the top half of the container. This is done recursively until the item is found and the index (since **`std::vectors`** can have index-based access) where it is found is returned. (Return -1 if the item is not found).

Output: The index (since **`std::vectors`** can have index-based access) where the value is found is returned. (Return -1 if the item is not found). Extra: You may also devise an implementation that returns **`std::vector<T>::iterator`** to the element in question (Return **`std::vector<T>::end()`** if the item is not found, not **`NULL`** – iterators are not generally pointers and should not be treated as such).

For the second part of this assignment, you will implement the following functionalities:

- Create a **`vecInt`**, which will be **`std::vector`** of **`int`**s, and fill it with 100 random **`int`** numbers which will be read from the provided file RandomData.txt. User input is not mandatory, and hard-coding the file name is allowed for this Project.
  (Note: If you wish to test with your own randomly ordered numbers, you may use **`std::rand()`** (http://en.cppreference.com/w/cpp/numeric/random/rand) to do this).
- Create a **`vecIntCpy`**, which will be another **`std::vector`** of **`int`**s, which should be a copy of the previously created one.
- Apply **`vector_resort`** and **`vector_research`** on **`vecInt`**.
- Print out (only to terminal) the resulting **`vecInt`**.
- **Extras (not required for 100pt grade):** Try **`std::sort`** and **`std::binary_search`** (http://en.cppreference.com/w/cpp/algorithm/sort http://en.cppreference.com/w/cpp/algorithm/binary_search) to perform the same tasks on the **`vecIntCpy`** container. Time the difference between your implementation and the STL one; you may perform high-resolution time-difference counting with (http://en.cppreference.com/w/cpp/chrono/high_resolution_clock/now):

```
std::chrono::time_point<std::chrono::system_clock> t1 = std::chrono::system_clock::now();
//tasks to time…
std::chrono::time_point<std::chrono::system_clock> t2 = std::chrono::system_clock::now();
std::chrono::duration<double> diff = t2-t1;
std::cout << diff.count() << std::endl;
```

You will create the necessary VectorRecursion.h header file that will contain the **templated functions' declarations and implementations**. You should also create a source file proj11.cpp which will implement the above required functionalities (File Input, populating the vectors, calling the recursive functions, Terminal Output, etc…).

If you cannot get past the `vector_resort` function to proceed to `vector_research`, you may use std::sort to order the elements of `vecInt`. In such a case, only partial credit will be awarded.

Suggestion(s): First try to implement the recursive sorting and binary search function on simple int arrays. Create the test driver for your code and verify that it works. Only then move on to write std::vector and template-based generic versions for your two functions.

The completed project should have the following properties:

- Written, compiled and tested using Linux.
- It must compile successfully on the department machines using Makefile(s), which will be invoking the g++ compiler. Instructions how to remotely connect to department machines are included in the Projects folder in WebCampus.
- The code must be commented and indented properly.
  Header comments are required on all files and recommended for the rest of the program.
  Descriptions of functions commented properly.
- A one page (minimum) typed sheet documenting your code. This should include the overall purpose of the program, your design, problems (if any), and any changes you would make given more time.

Turn in: Compressed Header & Source files, Makefile(s), and project documentation.

Submission Instructions:

- You will submit your work via WebCampus
- Name your code file proj11.cpp
- If you have header file, name it proj11.h
- If you have class header and source files, name them as the respective class (VectorRecursion.h) This source code structure is not mandatory, but advised.
- Compress your:
  1. Source code
  2. Makefile(s)
  3. Documentation

  Do not include executable
- Name the compressed folder:
  PA#_Lastname_Firstname.zip
  Ex: PA11_Smith_John.zip

Verify: After you upload your .zip file, re-download it from WebCampus. Extract it, compile it and verify that it compiles and runs on the NoMachine virtual machines or directly on the ECC systems.

- Code that does not compile will be heavily penalized –may even cost you your *entire* grade–. Executables that do not work 100% will receive partial grade points.
- It is better to hand in code that compiles and performs partial functionality, rather than broken code. You may use your Documentation file to mention what you could not get to work exactly as you wanted in the given timeframe of the Project.

Late Submission:

A project submission is "late" if any of the submitted files are time-stamped after the due date and time. Projects will be accepted up to 24 hours late, with 20% penalty.