CS 202 - Computer Science II

Project 11

Due date (FIXED): Tuesday, 5/8/2020, 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 conduct the following assessments:

a) **Unit Testing of `vector_resort` and `vector_research`**

Task - Create a unit test program file **proj11_test.cpp** that performs the following:

- Creates a **`vecInt`**, which will be **`std::vector`** of **`int`**s, and fills 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.
  (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).
- Apply **`vector_resort`** on **`vecInt`** and print out (to terminal) the resulting vector.
- Print out whether the Test results on **`vector_resort`** indicate **Success** or **Failure** (you have to discern this via the state of **`vecInt`**).
- Apply **`vector_research`** on **`vecInt`** (for multiple test entries which correspond to meaningful tests –e.g. numbers that you know to be in **`vecInt`**, numbers that you know to be excluded from it, numbers that are outside of the upper/lower range, etc) and print out (to terminal) the corresponding results.
- Print out whether the Test results on **`vector_research`** indicate **Success** or **Failure** (you have to discern this via the previous tests).

b) **Critical assessment of `vector_resort` and `vector_research`**

You are a Software Engineer for a company that develops products to address the needs of operational stakeholders in one or multiple critical socio-economic domains, where dealing with vast amounts of data and conducting routine operations (such as search, sort, etc.) with maximum effectiveness is crucial. Such a context pertains to numerous domains including a) safety and security (police database searches, mass security surveillance, etc.), b) the economy (e-commerce, high-frequency trading applications, etc.), as well as c) directly affected ones such as the environment (emissions footprint of data servers, cryptocurrency mining arrays, etc.).

You are in charge of conducting high-level processing on data that are selected based on conducting 1000s of searches, but your so far toolset has relied on C++ built-in implementations of

- **`std::sort`** (http://en.cppreference.com/w/cpp/algorithm/sort)
- **`std::binary_search`** (http://en.cppreference.com/w/cpp/algorithm/binary_search)

A new proprietary algorithm developed by one of your company's branches is being pushed to adopt across all departments. Your responsibility is to analyze the effects of this proprietary tool (in effect analyze the **`vector_resort`** and the **`vector_research`** performance) in your own pipeline, so you have to design a program that performs a conclusive evaluation.

Task - Create a unit test program file **proj11_assess.cpp** that:

- Allows you to make an assessment for adopting or rejecting this new tool (versus the legacy **`std::sort`**/**`std::binary_search`** one). Your primary criterion w.r.t. your application domain (safety & security / economy / environment, as previously mentioned) should be the runtime efficiency for sort & search operations.

  Hint: Using C++ one 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();
//some tasks to time here …
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;
```

  To ensure a conclusive evaluation of efficiency, you should conduct tests on varying container sizes (e.g. for a vector of 100 elements, for one of 10,000 elements, for one of

1,000,000, etc.) and for multiple trials (average results over 100 runs -per test case- are much more reliable than a single-shot test).

- Report in your Documentation file:
  a) what your **proj11_assess** does,
  b) what findings you derived based on it,
  c) your assessment of whether to adopt your company's new tool.

You will create the necessary VectorRecursion.hpp header file that will contain the **function template declarations and implementations**. You should then #include that in both proj11_test.cpp and proj11_assess.cpp.

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. Then move on to write std::vector and template-based generic versions for your two functions.

The completed project should have the following properties:

- Your code is required to follow the **file organization structure** demonstrated in your Labs, with subfolders for headers, source files, and a final build products location (generated during build).
- Your project's build should be based on a **CMakeLists.txt** script, which will be included with your deliverables.
- It must compile successfully using the g++ compiler on department machines or the provided Xubuntu VM image.
- 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 file structure (with .hpp, .cpp, and .h files, and your CMakeLists.txt). Also, your project documentation file.

REMEMBER: Use debugging tools, GDB and Valgrind, as you were instructed to detect the origin of such memory access violation errors and memory leaks!

IMPORTANT: Creating a build configuration for Debugging:

- When requiring a build with debug symbols, usually you would specify this using the g++ command with the appropriate flag:
  `g++ -g ...`

  But **CMake** provides a convenient functionality for configuring a "standardized" debug build of your project with the required compiler flags and settings automatically handled by CMake. It does so via a configuration option called: `CMAKE_BUILD_TYPE`.
  If you want to enforce configuring your project build to have debug symbols enabled (because you intend to debug it using `gdb` for instance), you may run the cmake configuration command and specify this option as follows:

  ```
  cmake -D CMAKE_BUILD_TYPE=Debug ..
  ```

  Extra: Other possible values are:
  `-D CMAKE_BUILD_TYPE=Release` which enables recommended compiler optimizations to refactor the compiled code to make it more efficient,
  `-D CMAKE_BUILD_TYPE=RelWithDebInfo` which generates a "Release" (optimized) build but retains debug symbols for debugging (be careful with "Release" builds your code is optimized by the compiler and thus refactored).
- It is recommended to use such a configuration together with `gdb` (as demonstrated in your Labs) to trace any dynamic memory management bugs of your code.

The following are a list of restrictions:

- Your code may use the C++11 standard (or any standard higher or lower).

  Note: Usually, you would specify using the g++ command with some flags:
  `g++ -std=c++11 ...`

  But **CMake** provides the functionality of *autodetecting* your system's C++ compiler and generating the Makefiles to invoke the appropriate commands to be used when you eventually `make` your project.
  If you want to enforce configuring your project build to use a particular standard, you either do so everytime you run the cmake configuration command by:

  ```
  cmake -D CMAKE_CXX_STARDARD=11 ..
  ```

  Or you could put a line like the following inside your **CMakeLists.txt** script:

  ```
  set(CMAKE_CXX_STANDARD 11)
  ```

  You do not need to worry about either of these however, and for now just running the usual `cmake ..` for configuration should do the trick.
- No libraries except **`<iostream>`** and **`<fstream>`** and **`<string>`** or **`<cstring>`** / **`<string.h>`** allowed.
- No global variables except `const` ones.
- You are expected to employ code abstraction and reuse by implementing and using functions. The already provided code structure in the project description will be considered sufficient.
- You are expected to implement **`const`** correctness in your program design. This refers to class method qualifications, function parameter qualifications, etc.

Submission Instructions:

- You will submit your work via WebCampus
- Compress your:
  1. Code file structure (containing Source code files, Header files, CMakeLists.txt)
  2. Documentation

  Do not include executable or library files, nor any build, devel, or other non-required folders.
- Name the compressed folder:
  PA#_Lastname_Firstname.zip
  ([PA] stands for [ProjectAssignment], [#] is the Project number)
  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 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.

Instructions to remotely test your project configuration and build on the ECC systems:

a) Download your Webcampus submission on your local computer. Let's say you submitted a file named **PAx_Smith_John.zip**, and you now downloaded it into your **Downloads** folder.

b) Navigate to that directory using your terminal, and check the file you downloaded is there.
   a. On **Ubuntu** you can open a terminal with Ctrl+Alt+T and then do:
      ```
      cd Downloads
      ls -al
      ```
   b. On a **Mac** you can open Spotlight and type "Terminal" and hit Enter, then do:
      ```
      cd Downloads
      ls -al
      ```
   c. On **Windows** in your Start Menu type "cmd" and click on Command Prompt, then:
      ```
      cd Downloads
      dir
      ```

c) Remotely copy your submission file from you local Downloads folder to your CSE Ubuntu user account inside its home/$USER folder.
   ```
   scp FILENAME NETID@ubuntu.cse.unr.edu:/nfs/home/NETID/FILENAME
   ```
   For example if the user NetID is **jsmith** and the submission file **PA7_Smith_John.zip**:
```
scp PAx_Smith_John.zip jsmith@ubuntu.cse.unr.edu:/nfs/home/jsmith/PAx_Smith_John.zip
```

d) Login to your CSE Ubuntu user account.
   ```
   ssh NETID@ubuntu.cse.unr.edu
   ```
   For example if the user NetID is **jsmith**:
   ```
   ssh jsmith@ubuntu.cse.unr.edu
   ```

e) Once you are in, check the contents to verify that you have successfully transferred the file:
   ```
   ls -al
   ```

f) Unzip the file into a folder with the same name:

a. If it is a **.zip** file then:
   unzip -o FILENAME.zip -d FILENAME
   Example:
   unzip -o PAx_Smith_John.zip -d PAx_Smith_John
b. If it is a **tar.gz** file then:
   mkdir FILENAME
   tar -xzvf FILENAME.tar.gz -C FILENAME
   Example:
   mkdir PAx_Smith_John
   tar -xzvf PAx_Smith_John.tar.gz -C PAx_Smith_John

g) The above will create a folder with the same name as your submission file, which will contain the unzipped content. Enter the directory and execute the known configuration and build sequence:

```
cd PAx_Smith_John
mkdir build
cmake ..
make
```