Reference: 190916_CXXList_Challenge

Dual-Indexed List Optimisation Challenge

Candidate Assessment Challenge

Context

Data structures are fundamental to any significant C++ program. The STL (Standard Template Library) provides a range of commonly used data structures, such as vector, map, or list. However, particular programs often have more bespoke requirements for data structures, such as given in this challenge. Implementing them without errors and optimizing their performance is paramount. This challenge takes one pattern found in real-world code and requires the candidate to optimize its performance, as well as implement correctness tests and build project management.

Problem Description

The data structure to be optimised is termed `CustomList`, given in the separate file `custom_list.hpp`, with the following characteristics:

- It allows inserting elements in the end (but not removal)
- Elements can be indexed by a string-valued key as well as by their integer index into the list
- The structure is at template, similar to the STL containers, so that it can be used with any data type

The class interface is given in Listing 1 below.

A naïve implementation of this class is provided, which can be found in typical real-world code bases. The performance of accessing values by string key or index is far from optimal and should be improved.

```
template <class T>
class CustomList
{
public:
    bool empty();                                          // check if empty
    size_t size();                                         // get size of list
    const T& get_by_key(const std::string& key) const;     // get element by string key
    const T& get_by_index(int i) const;                    // get element by its index
    void push_back(const std::string& key, const T& value); // insert element at end
};
```

Listing 1: Interface of the CustomList to be optimized (full code given in custom_list.hpp).

Reference: 190916_CXXList_Challenge

Challenge

The candidate should first implement a set of automated unit tests for the given data structure, to ensure correctness while optimising. The C++ testing framework to be used shall be Google Test. It is sufficient to test the data structure for holding integer values.

The C++ project and build settings shall be implemented using the CMake tool, implemented in a cross-platform fashion. That is, the resulting project should be compileable on at least Windows and Linux. Guidelines for modern CMake can be found here, which should serve as a starting point.

In the next stage, the performance of string-based random access and index-based random access should be optimized. A list containing 100,000 values shall be constructed for this purpose, and later accessed randomly at least 100,000 times. The list values shall be integers, and a string version of the integer index can be used as key for testing purposes. The candidate shall establish a baseline performance using the implementation given, and then attempt to optimise the list to improve the performance as much as possible (without failing the correctness tests). The overall speedup will be evaluated using the CMake Release configuration (i.e., with full compiler optimisations enabled).

The CMake configuration should abstract the underlying compiler as much as possible. The candidate is free to choose the implementation platform / operating system. For testing purposes, using one of the following compilers is recommended:

- Linux: any GCC > 5.0, or Clang/LLVM > 5.0
- Windows: Visual Studio > 2013, or MinGW GCC > 5.0
- Mac: Clang/LLVM > 5.0 (the compiler bundled with XCode is typically ok)

Apart from Google Test and the C++ standard library, no other external libraries are permissible for this challenge. The code should be standard compliant C++17 (or earlier) and not use any proprietary compiler extensions or pragmas.

Solution

The submitted solution should contain:

- The full code base, including CMake files, correctness unit tests, and performance tests.
- A README file describing the build process, so that evaluators can follow these instructions to build and test the submitted solution
- A document describing which speedup was achieved, stating clearly the platform, hardware configuration, operating system (with version), and compiler version used.

Evaluation

The candidate will be assessed on the following aspects:

- Quality of the code, CMake files, and unit tests
- Performance of the final solution, which will be verified by the evaluators

Further, a 15 minute presentation of the code and solution should be prepared, to be given to the evaluators after submission. This should include a code walk-through, an explanation of the approach taken, including the reasons, and an explanation of the process how the candidate arrived at the final best-performing solution. After this, the candidate will pair with the evaluator for 45 minutes, where additional tasks will be given and solved using pair programming.