Chapter 20 Review

Ulysses Carlos

March 23, 2020

1 Questions

1. Why does code written by different people look different? Give examples.

   (a) Code written by different people look different because they may write code to suit their needs, either neglecting or enhancing an aspect in their code. They may be poorly documented/suited for a specific cause.

2. What are simple questions we ask of data?

   (a) Some questions include whether we can apply some operations on the data can be output some result, whether this data can be stored in a format accessible by all.

3. What are a few different ways of storing data?

   (a) We can store data in a file, a container (Such as a vector, list), or a raw array.

4. What basic operations can we do to a collection of data items?

   (a) Some basic operations could be finding the highest value in a collection, sorting the collection in some away, searching for a particular value. We should able to add/remove items in the collections.

5. What are some ideals for the way we store our data?

   (a) Some ideals would be for this data to collected into containers that can organize the data to fit some need, and should be able to be accessed in some away.

6. What is an STL sequence?

   (a) An STL sequence is defined as a collection of data with a beginning and an end. It can be traversed from its beginning to its end, and elements within that range can be read/written to.

7. What is an STL iterator? What operations does it support?

(a) An STL iterator is a object that refers/identifies an element of a sequence. STL iterators support many of the same operations of a standard pointer, such as equality tests, dereferenced, assignment, and be incremented / decremented.

8. How do you move an iterator to the next element?

   (a) You can move an iterator to the next element by doing

   `iterator++.`

9. How do you move an iterator to the previous element?

   (a) You can move an iterator back by doing

   `iterator--.`

10. What happens if you try to move an iterator past the end of a sequence?

    (a) It depends on the iterator implementation. Some iterators implement a form of range checking which prevent moving past the end of a sequence. Some iterators are simply alias for a pointer of that type, which may just allow moving past the end.

11. What kinds of iterators can you move to the previous element?

    (a) Some kind of iterators that can do that are regular pointers, iterators for double-linked lists and for vectors.

12. Why is it useful to separate data from algorithms?

    (a) It is useful because it allows the algorithms to be used by a variety of different types instead of tailoring a algorithm to work with a specific data type.

13. What is the STL?

(a) The STL (Standard Template Library) is a group of containers and algorithms that work on data as sequences of elements. This library is included in the C++ Standard Library and takes advantage of C++ template to allow data from any data type to be contained in sequence and have algorithms that can manipulate that data.

14. What is a linked list? How does it fundamentally differ from a vector?

    (a) A linked list is a data structure where each element is connected by a iterator to another element. Linked Lists do not need data to in a single continuous block of memory (Like in vectors) and so insert/removing an element does not moving the elements.

15. What is a link (in a linked list)?

    (a) The link in a linked list holds a value of type T, alongside Links to the the previous and next link.

16. What does **insert()** do? What does **erase()** do?

    (a) **insert()** allows a element to be inserted at some specified place in a sequence, and **erase()** remove a specified element from the sequence.

17. How do you know if a sequence is empty?

    (a) A sequence is empty when both **begin()** and **end()** point to each other. If the sequence wasn't empty, then they would point to different elements.

18. What operations does an iterator for a list provide?

    (a) Some operations that a list iterator provides include
        - ++ (Move to the next link in the list)
        - – (Move to the previous link in the list)
        - Comparing two iterators (`==`, `!=`)
        - Allowing the data referred by the iterator to be accessed

19. How do you iterate over a container using the STL?

    (a) You can iterate over a container by using the Ranged-For loop. Ranged For loops are defined by the use of **begin()** and **end()**, so implementing them will allowing iteration by a ranged for loop.

20. When would you use **string** rather than a **vector**?

    (a) You would use a string if you are planning to do string concatenation, or reading whitespace-separated words.

21. When would you use a **list** rather than a **vector**?

    (a) You would use a list if you are planning to insert/remove elements in a large data collection. Doing that on a vector with a large amount of elements would cause a performance hit.

22. What is a container?

    (a) A container is a sequence of elements with a begin() and a end that provides copy operations implemented through assignment or a constructor. They have iterators that work with regular and constant values (iterator and const_iterator). They provide the following operations: insert(), erase(), front(), back(), etc. Lastly, they provide comparison operators.

23. What should **begin()** and **end()** do for a container?

    (a) **begin()** and **end()** designate the range of a sequence so that operations can be done using that container.

24. What containers does the STL provide?

    (a) Some example include vector, list, map, deque, multimap, unordered_map/multimap, set/multiset, unordered_set/multi_set and array.

25. What is an iterator category? What kinds of iterators does the STL offer?

(a) An iterator category are a group of advanced iterators that do a specific task. Some examples include

- Input Iterators that can read element values while iterating
- Output Iterators that can write element values while iterating
- Forward Iterators that are both Input and Output Iterators
- Bidirectional Iterators that iterate forward and backwards
- Random-Access Iterators that can iterate forward and backwards and can read and write element values using * or [].

26. What operations are provided by a random-access iterator, but not a bidirectional iterator?

(a) Random-Access Iterators can read and write element values using * or [], much like a array.