Bernardo Fitzmaurice Acevedo 105297603

Class code: COS30031

Start date: 04/09/2024

Task 04 Lab Data Structure Basics

**Array\_demo\_1:**

* Q.1.1 What do the < and > mean or indicate?
  + They specify template parameters, and in this code is used to hold 3 elements in the array.
* Q.1.2 Why don't we need to write std:array here? (Is this good?)
  + Because “using namespace std” is already declared at the beginning of the code. It’s not really good, because it can generate bug due to naming conflicts.
* Q.1.3 Explain what the int and 3 indicate in this case?
  + ‘Int’ is the type of element the array will contain. And ‘3’ is the number of elements the array will contain.
* Q.1.4 In the code above, what is the type of itr2?
  + The type of itr2 is ‘std::array<int, 3>::iterator’.
* Q.1.5 In the code above, what is the type of v?
  + V’s type is ‘int&’.
* Q.1.6 In the code above, what does the & mean in (auto& v : a1)
  + Means that ‘v’ is a reference to the elements in the array ‘a1’.
* Q.1.7 Try this. Why does a1[3] work but at(3) does not?
  + ‘a1[3]’ works because the ‘[ ]’ operator doesn’t perform bound checking. While ‘at(3)’ doesn’t work because it performs bound checking, preventing access outside the valid range of the array.
* Q.1.8 How would you do a forward (not reverse) sort?
  + I’d use the std::sort function with the array's begin() and end() iterators. Like this: “sort(a1.begin(), a1.end());”

**Array\_demo\_2:**

* Q.2 In array\_demo\_2, explain what a4(a1) does
  + It creates a new array by copying the contents of the ‘a1’ array.

**Array\_demo\_3:**

* No questions

**Stack\_demo:**

* Q.4 How do we (what methods) add and remove items to a stack?
  + To add an item we use ‘push()’, and to remove we use ‘pop()’
* Q.5 A stack has no no[ ] or at() method - why?
  + Because is designed to only allow access to the top element through ‘top()’.

**Queue\_demo:**

* Q.6 What is the difference between a stack.pop() and a queue.pop() ?
  + ‘stack.pop()’ removes the element at the top of the stack. While ‘queue.pop()’ removes the element at the front of the queue.

**List\_demo:**

* Q.7 Can we access a list value using an int index? Explain.
  + No. A ‘std::list’ doesn’t support direct access using an integer index because it’s implemented as a doubly-linked list.
* Q.8 Is there a reason to use a list instead of a vector?
  + Yes, because a list is more efficient than a vector.

**Vector\_demo:**

* Q.9 Was max\_size and size the same? (Can they be different?)
  + They are not the same size. One is the max number of elements, and the number of elements in the vector.
* Q.10 Which ParticleClass constructor was called?
  + ‘ParticleClass(int x, int y)’
* Q.11 Were the ParticleClass instances deleted? If so, how?
  + Yes, they were deleted. It was deleted when ‘v2’ was deleted at the end of the scope.
* Q.12 Was the vector instance deleted? If so, how do you know this?
  + Yes, it was deleted. Because its destructor was called automatically when the function ended.
* Q.13 Your IDE might suggest using emplace\_back instead of push\_back. What does this mean?
  + That I have an existing temporary object and needs to be copied or moved into the vector.

**Void showIntArray:**

* Q 14 Apparently const prevents a copy - quicker performance. Is this true? Use evidence or sources to justify your answer.
  + Yes. Using const references doesn’t make a copy of the array, therefore improving the performance, the larger the object the more it improves.
  + Links of evidence:
    - <https://stackoverflow.com/questions/7080256/why-are-references-to-const-necessary-to-avoid-copying-parameters>
    - <https://en.cppreference.com/book/intro/const>