# Release Notes – COS30031 Task 6

## Data Structure Basics

Author: Luke Valentino

Student ID: 103024456

Unit: COS30031, Games Programming

GitHub Repository: [https://github.com/LukeValentino138/COS30031-2023-103024456](https://github.com/LukeValentino138/COS30031-2023-103024456)

Date: 25/08/2023

## Summary:

In this lab I have gone through the given cpp file and answered all questions and modified any relevant code. Questions 1.1 to 1.10 are in the cpp file and in this document.

## Tasks Completed:

- Analysed cpp file.
- Answered Questions.
- Created Lab report.

## Issues:

- None.

## Answers:

Q.1.1 [line 164] What do the < and > mean or indicate?

The < and > are used because array is a template class. < and > are used to denote template arguments.

Q.1.2 [line 165] Why don't we need to write std:array here? (Is this good?)

We do not need to write std::array here because we have written "using namespace std;" at the top of the file. This is generally good for readability.

Q.1.3 [line 166] Explain what the int and 3 indicate in this case?

int indicates the type of element the array will hold, and 3 indicates how many elements the array will hold.

Q.1.4 [line 204] In the code above, what is the type of itr2?

the type of itr2 is set by auto. auto sets the type to std::array<int, 3>::iterator

Q.1.5 [line 211] In the code above, what is the type of v?

v is of type int as it references the elements of a1.

Q.1.6 [line 212] In the code above, what does the & mean in (auto &v : a1)

It means reference. It means that the loop can directly change the elements in a1 instead of changing copies.

Q.1.7 [line 220] Try this. Why does a1[3] work but at(3) does not?

a1[3] work and at(3) because at() does bounds checking whilst [] does not.

Q.1.8 [line 233] auto is awesome. What is the actual type of v that it works out for us?

the type of v is <std::array<int, 3>::iterator>

Q.1.9 [line 240] auto is still awesome. What is the actual type of v here?

the type of v here is int.

Q.1.10 [line 250] How would you do a forward (not reverse) sort?

A forward (not reverse) sort would look like this:

sort(a1.begin(), a1.end());

Q.2 [line 105] In array_demo_2, explain what a4(a1) does

auto a4(a1) creates a copy of the a1 array.

Q.3 [line 108] No questions for array_demo_3, it's just a demo of Struct/Class use with array.

Q.4 [line 111] How do we (what methods) add and remove items to a stack?

4 items are added to the stack using "push()". Items are removed from the stack using "pop()".

Q.5 [line 112] A stack has no [] or at() method - why?

There is no [] or at() method for the stack because only the most recent item can be interacted with.

Q.6 [line 115] What is the difference between a stack.pop() and a queue.pop() ?

Because the queue works on a FIFO (first in, first out) basis, a queue pop will remove the first item added.

A stack works differently, a stack works on a LIFO (last in, first out) basis. This means a stack pop will remove the last item added.

Q.7 [line 118] Can we access a list value using and int index? Explain.

No we cannot access a list value directly. As lists in c++ are doubly linked lists, they must iterate through them to find a specific element.

Q.8 [line 119] Is there a reason to use a list instead of a vector?

Yes, a list is better suited if you need to insert items frequently as this operation on a vector takes longer.

Q.9 [line 122] Was max_size and size the same? (Can they be different?)

size and max_size are very different. The size of the vector is equal to how many elements are currently stored. The max_size is equal to how many elements the vector can hold based on system limitations.

Q.10 [line 123] Which ParticleClass constructor was called?

The ParticleClass(int x, int y) constructor was called.

Q.11 [line 124] Were the ParticleClass instances deleted? If so, how?

The ParticleClass instances were deleted when v1 went out of scope.

Q.12 [line 125] Was the vector instance deleted? If so, how do you know this?

Yes the vector instance was deleted. This can be seen in the output when running the code.

Q.13 [line 126] Your IDE might suggest to use emplace_back instead of push_back. What does this mean?

emplace_back is a method that inserts a new element at the end of the container. The new element is constructed in place using args as opposed to push_back which copies the content of val to the new element. In some cases, emplace_back is more efficient.