



Task 06 - Lab Summary Report



Spike: Task_06

Title: Data Structure Basics

Author: Thomas Horsley, 103071494

Goals & Deliverables

Aim: Learn to utilize standard collection types and basic algorithms to aid in games programming.

Deliverables:

- Lab report containing questions, answers and code snippets
- Git commit history
- Document containing updated project source code

Technology, Tools and Resources

Tech and Tools



The project was scripted in Python C++ 17 using the VSCode IDE version 1.76.

UML's and charts are made with www.Lucidchart.com

Optionally (though recommended), source control is handled using Git.

VSCode Plugins/Extensions

- C/C++
Author: Microsoft
Version: 2023.4.1
- Colorful Comments (I always recommend)
Author: Parth Rastogi
Version: 1.0
- Code Runner
Author: Jun Han

Resources

- Echo360 Lectures “Topic 3.2 - Data Structures”

Version: 0.12.0

- TODO Tree (Great for finding the TODO's)
 - Author: Gruntfuggly
 - Version: 0.0.226

Tasks Undertaken

Q & A

array_demo_1()

1. What do the < and > mean or indicate?
 - a. Indicates that the array is of type value (T) int and size value (n) 3.

```
array<int, 3> a1 = {8, 77, -50}; // == array<T, n> arrayName = {define, define, ...}
```

2. Why don't we need to write std::array here? (Is this good?)
 - a. As the program is using the standard namespace, we don't have to prefix types from the library with std::type. Using the standard namespace however is not considered good practice as it can cause naming ambiguity resulting in compile-time errors.
3. Explain what the int and 3 indicate in this case?
 - a. The int indicates the type value (T) of the array (this array will store integer values) and the 3 represents the size value (n) of the array (how many elements of type T the array can store).
4. In the code above, what is the type of itr2?
 - a. itr2 is an integer pointer. When it's 'couted' it will print an integer as the pointer is dereferenced.

```
201 // ... using auto to get iterator (whew - much easier)
202 cout << "a" int *itr2 using auto provided template iterator: ";
203 for (auto itr2 = a1.begin(); itr2 < a1.end(); itr2++) {
204     cout << *itr2 << " ";
205 }
```

5. In the code above, what is the type of v?
 - a. v is of type int

6. In the code above, what does the & mean in (auto &v : a1)

- a. As the & operator prefixes the v this expression represents a temporary range expression. In temporary range expressions memory assigned to values is only valid during the lifetime of the function. Once the function has concluded, the memory is out of scope and cleared.

```
210 cout << "a1 contents using auto & for-each iterator: " << endl;
211 for (auto &v : a1)
212     cout << v << " ";
213 cout << endl;
```

7. Try this. Why does a1[3] work but at(3) does not?

- a. C++ arrays are essentially ancient leftover C code which didn't implement any value checking what-so-ever as all C arrays were was a grouping of contiguous memory and nothing else. The at() operator is part of the std::vector template and includes bounds checking.

8. auto is awesome. What is the actual type of v that it works out for us?

- a. v is set as an integer pointer

```
236 // #TODO: Q.1.10 auto is awesome. What is the actual type of v that it works out for us?
237 cout << "U int *v with iterator ... " << endl;
238 for (auto v = a1.begin(); v != a1.end(); v++)
239     cout << " " << *v;
240 cout << endl;
```

9. auto is still awesome. What is the actual type of v here?

- a. v kinda looks like a plain old integer to me

10. How would you do a forward (not reverse) sort?

- a. As there was already a reverse sort applied to the array, all I needed to do was reverse this sort using the reverse() method and we have a forward sort.

```
249 // sort?
250 sort(a1.rbegin(), a1.rend());
251 cout << "Reverse Sort() on a1, now ..." << endl;
252 showIntArray(a1);
253 // #TODO: Q.1.10 How would you do a forward (not reverse) sort?
254
255 reverse(a1.begin(), a1.end());
256 cout << "Forward Sort() on a1, now..." << endl;
257 showIntArray(a1);
258
```

array_demo_2()

1. In `array_demo_2`, explain what `a4(a1)` does
 - a. This syntax creates a new copy of `a1` in a variable called `a4`. These two arrays exist in separate spaces in memory.

```

268 void array_demo_2() {
269     if (true) {
270         // array of 5 ints, must state size
271         array<int, 5> a1;
272         array<int, 4> a2 = {-4, 2, 7, -100};
273
274         cout << "a1 " << hex << &a1 << " " << a1.size() << endl;
275         cout << "a2 " << hex << &a2 << " " << a2.size() << endl;
276
277         // new array via copy
278         auto a3 = a2; // this is a copy
279         // if auto doesn't work (C++11 extension) either configure your compiler
280         // or state the type explicitly. (VS2010+ should support it, etc)
281         // - array<int, 4> a3 = a2; // equivalent to auto
282         // - array<int, 4> z1 = a1; // compile error - different length
283
284         cout << "a3 " << hex << &a3 << " " << a3.size() << endl;
285         auto a4(a1); // this works too
286         cout << "a4 " << hex << &a4 << " " << a4.size() << endl;
287     }
288 }

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

Widat as at [] (OUT OF BOUNDS) 0x44332

[Done] exited with code=0 in 0.786 seconds

[Running] cd "f:\Uni\COS30031-2023-103071494\06 - Lab - Data Structure Basics\" && g++ ta

```

a1 0x61fefc 5
a2 0x61feec 4
a3 0x61fedc 4
a4 0x61fec8 5

```

stack_demo()

1. How do we (what methods) add and remove items to a stack?
 - a. A stack is a chunk of memory where values are assigned addresses on a LIFO (Last in first out) basis. Stacks are linear and we can add to a stack with the `.push()` method and remove items from a stack with the `.pop()` method.
2. A stack has no `[]` or `at()` method - why?
 - a. As items on the stack can only be accessed one-at-a-time in a LIFO manner, it's left to the developer to track the order in which stack memory is being allocated throughout the programs lifetime. The stack is not flexible memory and can't be manipulated in the same manner the heap can (hence no `[]` or `at()` method). Whilst this is a limitation, working with stack memory is incredibly fast and not as strenuous on a machine and has use for caching and OS creation amongst other speed intensive processes.

queue_demo()

1. What is the difference between `stack.pop()` and `queue.pop()` ?
 - a. Stacks work in a LIFO manner (Last in First Out) where-as a Queue works via FIFO (First in first out). When popping an item off the que it will remove the first item added to the queue. This is the opposite for a stack where the last item will be removed from the stack.

list_demo()

1. Can we access a list value using an int index? Explain.

- a. No, there is no way in C++ to use an integer index to access a list value. This is because `std::lists` don't contain random access operations. As a result an iterator type is constructed to step through a list.

2. Is there a reason to use a list instead of a vector?

- a. Lists can be used if there's a need to insert something in the middle of a sequential container. Other than this however, vector is advised.

vector_demo()

1. Was `max_size` and `size` the same? (Can they be different?)

- a. the size of a vector and it's `max_size` can (and most often will) be different values. The `max_size` of the vector represents how many values of the specified type the vector can hold, this will be some absurdly large value. Where-as the size of the vector is the number of elements contained within the vectors set.

2. Which `ParticleClass` constructor was called?

- a. As we're passing the parameters through the `.pushback()` method, we'll be calling the constructor which takes the two integer parameters.

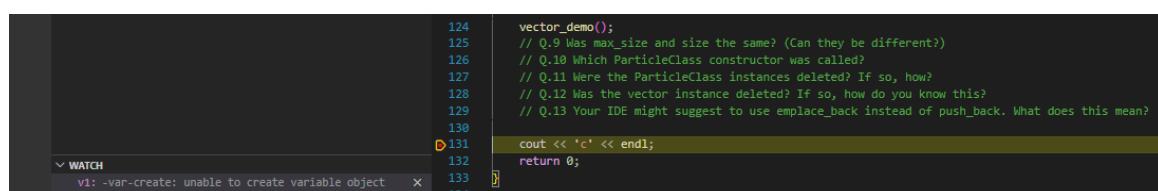
```
71 ParticleClass(int x, int y) {  
72     this->x = x;  
73     this->y = y;  
74     cout << " - ParticleClass(x,y) constructor called" << endl;  
75 }
```

3. Were the `ParticleClass` instances deleted? If so, how?

- a. Yes the `ParticleClasses` were deleted as when the `vector_demo()` method finished the vector fell out of scope and was each `ParticleClass` destructor was called.

4. Was the vector instance deleted? If so, how do you know this?

- a. Yes, it's destructor was automatically called when the vector went out of scope. This is validated by the VSCode Debugger.



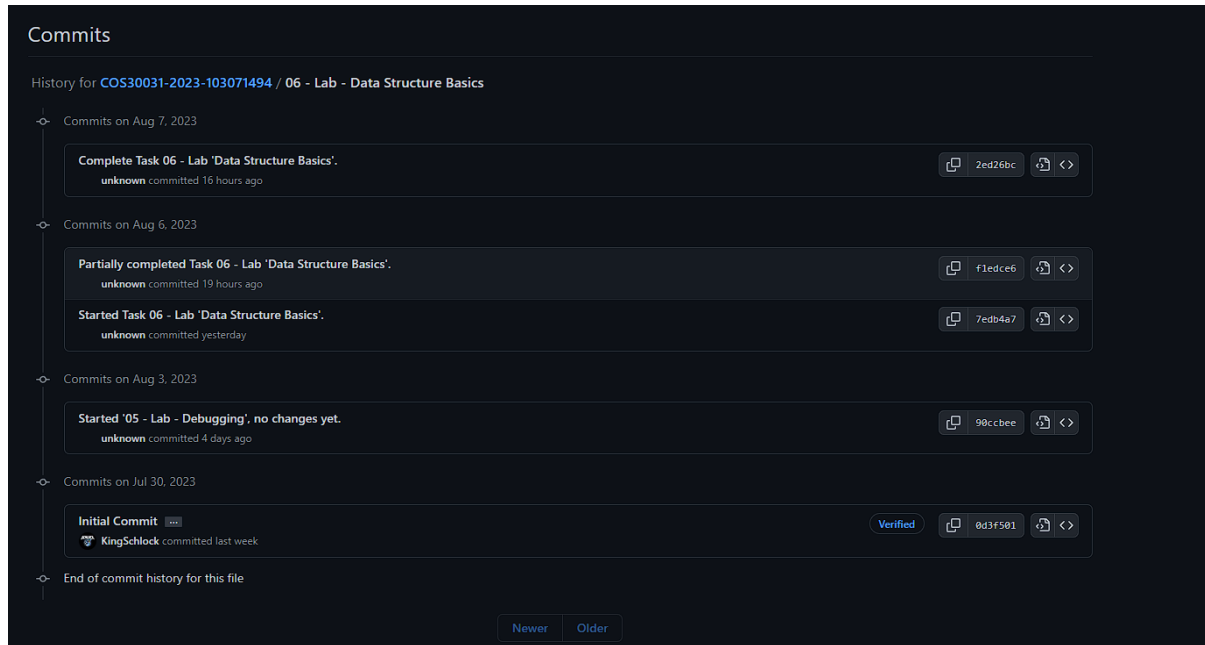
The screenshot shows the VSCode editor with the `vector_demo()` function defined on lines 124-134. The function calls `vector_demo()` recursively, then prints 'c' and returns 0. A debugger watch window is open at the bottom left, showing a message: `v1: -var-create: unable to create variable object`. The code in the editor is as follows:

```
124 vector_demo();  
125 // Q.9 Was max_size and size the same? (Can they be different?)  
126 // Q.10 Which ParticleClass constructor was called?  
127 // Q.11 Were the ParticleClass instances deleted? If so, how?  
128 // Q.12 Was the vector instance deleted? If so, how do you know this?  
129 // Q.13 Your IDE might suggest to use emplace_back instead of push_back. What does this mean?  
130  
131 cout << 'c' << endl;  
132 return 0;  
133  
134
```

5. Your IDE might suggest to use `emplace_back` instead of `push_back`. What does this mean?

- a. The `push_back()` method inserts elements of a set by copying that set to a new space in memory and inserting the value during this copying process. However, in the case of `emplace_back()`, the value is taken and directly inserted into the set without the copying process. This results in a much faster method.

Git Commit History



The screenshot displays the Git commit history for a file named `COS30031-2023-103071494 / 06 - Lab - Data Structure Basics`. The history is organized by date, showing commits from August 7, 2023, back to July 30, 2023.

- Commits on Aug 7, 2023:**
 - Complete Task 06 - Lab 'Data Structure Basics'.** (Commit hash: `2ed26bc`)
unknown committed 16 hours ago
- Commits on Aug 6, 2023:**
 - Partially completed Task 06 - Lab 'Data Structure Basics'.** (Commit hash: `f1edce6`)
unknown committed 19 hours ago
 - Started Task 06 - Lab 'Data Structure Basics'.** (Commit hash: `7ed04a7`)
unknown committed yesterday
- Commits on Aug 3, 2023:**
 - Started '05 - Lab - Debugging', no changes yet.** (Commit hash: `90ccbee`)
unknown committed 4 days ago
- Commits on Jul 30, 2023:**
 - Initial Commit** (Commit hash: `8d3f501`)
KingSchlock committed last week (Verified)

The interface includes expand/collapse icons for each date group and navigation buttons for 'Newer' and 'Older' commits at the bottom.

What was Learned?



Throughout this lab, I learnt the applications and some specialties of the common C++ container types.