Title: Data Structure Basics

Author: Parth Madhani, 101901858

Tasks undertaken:

- Research different data structures and their features.
- Look at the section of the code given and explain.

What we found out:

We found out about some of the different data structures which are present in the C++ STL library and learned that some features are specific to some data structures based on their container type (associative, sequence, unordered associative etc.).

Notes :

- First thing we learn about is usage of constant in function parameters which is basically used for quicker performance as basically we are passing a parameter by a 'reference call' but in fact we are using it as value call as when we use constant it will pass copy of variable. So in short when you probably want to start using const. when sometimes you want to pass a parameter by value (because you don't want the function being able to change the variable), but if it is a huge variable like a really big array, copying the data can be an overhead so using constant would quicken process as we are not storing / copying data in new address.

Array Demo:

- Coming to array demo we learn about lots thing starting with creating a simple fixed size array.
- Then next we check size of array using size/max_size function which gives same result in our case as it's a fixed size array.
- The next we see the difference between accessing elements of array using [ ] and at() which mainly is that at() supports bound checking while [ ] does not.
- Next we use container methods front / back for getting first and last values respectively. Then I implement swap function and fill function similarly as shown below:

```
array<int, 3> a2 = { 8, 66, 40 };
a1.swap(a2);
cout << "a1 contents after swap: ";
for (int i = 0; i < a1.size(); i++) {
    cout << a1[i] << " ";
}
cout << endl;
a1.fill(5);
cout << "a1 contents after fill: ";
for (int i = 0; i < a1.size(); i++) {
    cout << a1[i] << " ";
}
cout << endl;*/
```

- After that we learn about usage of ‘auto’ which sets the data type automatically. Then we use it for iterators such as begin and end.

```
for (auto v = a1.begin(); v != a1.end(); v++)
    cout << " " << *v;
cout << endl;
```

The data type for auto in above case would be array of integers.

- In the next case shown below when we use for each iterator the datatype would just be integer.

```
cout << "Using for-each (ranged) iterator ... " << endl;
for (auto &v : a1)
    cout << " " << v;
cout << endl;
```

- Then we sort the array using iterators rbegin/rend and then learn reverse sort and learn difference between between rbegin/begin and rend/end.
- Then we move on to multi-dimensional array and learn how to initialize/create them.
- After that we learn about copying array using direct approach and learn about size of array and hex value.
- After that we learn ways to initialize array of struct and learn showing output using for and foreach.

```
a1 array of Particles ...
 - Particle: 0 (cccccccc, cccccccc)
 - Particle: 1 (cccccccc, cccccccc)
 - Particle: 2 (cccccccc, cccccccc)
a2 array of Particles, initialised, using for-each ...
 - Particle: (0, 0)
 - Particle: (0, 0)
 - Particle: (0, 0)
```

- Next, we make array of particle class and then we check how to show output using simple foreach and show function we created in particle class.

```
Show a1 array of ParticleClass instance details ...
 - ParticleClass: (0, 0)
 - ParticleClass: (0, 0)
 - ParticleClass: (0, 0)
Show a1 array of ParticleClass instance details using show() ...
 - ParticleClass: (0, 0)
 - ParticleClass: (0, 0)
 - ParticleClass: (0, 0)
```

StackDemo :

- This brings us to stack with a short and brief explanation of stack mainly learning about stack being LIFO that is last in first out pproach. First we create stack of int and learn how to add/push elements in stack.

- Then we remove elements from stack using pop and learn how to display last added elements using top().

```
Stack (LIFO) ...
Removing stack elements with pop() ... 4 3 2 1 0
```

QueueDemo :

- Queue is opposite of stack following FIFO approach that is First in first out. In this like stack first we add elements to queue.
- Then we display elements using front and remove them using pop but when we display it shows first element or front element instead of top element.

```
Queue (FIFO) ...
Removing queue elements with pop() ... 0 1 2 3 4
```

ListDemo :

- Here we learn about list which is a doubly linked list sequence container. First we learn how to create list and iterator which can be used for inserting/extracting or moving.

- Next we insert in list using pushback() and display it.

```
List (double-linked list) ...
 - list contains: 1 2 3 4 5
```

- Then we learn how to insert using iterators by inserting n second last position.

```
Insert using iterator access (end() - 1)
 - list contains: 1 2 3 4 77 5
```

- Next we learn how to sort our list using sort() function which is simple.

```
Sort list (using default compare) ...
 - list contains: 1 2 3 4 5 77
```

VectorDemo :

- Next we learn about vectors which act as templated class for dynamic size arrays. First we create a vector using initialiser list method and then we display address of vector in hex.

```
v1 address: 012FF9E8
```

- Then we display size of vector we created and maxsize() which is not fixed for vectors.

```
v1 size: 3
v1 max_size: 1073741823
```

- Lastly we create a vector of particle class and initialize it with values and then use show method to display the data.

```
Show v1 vector of ParticleClass instance details using show() ...
 - ParticleClass: (1, 2)
 - ParticleClass: (3, 4)
 - ParticleClass: (5, 6)
```