Lab: Task 7
Title: Data Structure Basics
Author: Sam Huffer, 101633177

STD: Array Demos

- *Q:*

```
void showParticleArray(const array<int, 3> &arr)
{
    // #TODO: apparently const prevents a copy - quicker performance. true? ref/url?
    cout << " - array<int, 3> contents: ";
    for (int i = 0; i < arr.size(); i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
}
```

A: Googled the answer, which was either “No, declaring something const, by itself, doesn’t help performance” (http://www.cplusplus.com/forum/general/13775/, https://www.bfilipek.com/2016/12/please-declare-your-variables-as-const.html) or no consensus (https://stackoverflow.com/questions/20693136/do-const-declarations-help-the-compiler-gcc-produce-faster-code).

- *Q:*

```
// access of array by [index] is not range protected (BAD)
cout << "What is at [3]? (out of bounds) " << a1[3] << endl;
// the at(index) is range protected (but slower due to getter)
// #TODO - try this and note what happens.
if (true)
    cout << "What is at(3)? (out of range exception) " << a1.at( Pos: 3) << endl;
```

A: attempting to access array element [3] throws an exception (“An invalid parameter was passed to a function that considers invalid parameters fatal”). Trying to access it using array.at(3) throws an out_of_range exception (as noted in the string attempted to be output).

- *Q:*

```
// let's use some other container methods
cout << "front() == " << a1.front() << endl;
cout << "back() == " << a1.back() << endl;
cout << "empty() == " << a1.empty() << endl; // #TODO empty() work? try
```

A: Apparently it works (when spelt correctly), as it output “empty() == 0”.

- Q:

```
// #TODO: (optional). create examples of swap() and fill()
// a1.swap(s2) and a1.fill(value) also.
array<int, 3> s2 = { _Elems[0]:5, _Elems[1]:10, _Elems[2]:32 };
a1.swap(s2);
cout << "swap()" << endl;
cout << "a1 contents (originally 42, 77, -50):";

for (int i = 0; i < a1.size(); i++)
{
    cout << " " << a1[i];
}

cout << endl;
cout << "s2 contents (originally 5, 10, 32):";

for (int i = 0; i < s2.size(); i++)
{
    cout << " " << s2[i];
}

cout << endl;
cout << "fill()" << endl;
s2.fill( _Value: 394);
cout << "s2 contents:";

for (int i = 0; i < s2.size(); i++)
{
    cout << " " << s2[i];
}

cout << endl;
```

```
swap()
a1 contents (originally 42, 77, -50): 5 10 32
s2 contents (originally 5, 10, 32): 42 77 -50
fill()
s2 contents: 394 394 394
```

A: swap() exchanges the contents of two arrays, fill() sets all elements in an array to the same value.

- *Q:*
  A: begin() returns an iterator pointing to the first element of the array (from http://www.cplusplus.com/reference/array/array/begin/). (Evidently, that iterator can be incremented, as evidenced by the v++.) Presumably, "auto v" adopts the data type of whatever is assigned to it, akin to "var v" in C#.

```
// #TODO: auto is awesome. What is the actual type of v that it works out for us?
cout << "Using for with iterator ... " << endl;
for (auto v = a1.begin(); v != a1.end(); v++)
    cout << " " << *v;
cout << endl;
```

- *Q:*
  A: "auto &v" here, given a1 is an array of ints, would have to be int.

```
// iterator for-each loop
// #TODO: auto is still awesome. What is the actual type of v here?
cout << "Using for-each (ranged) iterator ... " << endl;
for (auto &v : a1)
    cout << " " << v;
cout << endl;
```

- *Q:*
  A: sort using begin() and end(), rather than rbegin() and rend()
- *Q: vote – are multi-dimensional arrays pretty to create?*
  A: I wouldn't say pretty, but they're pretty organised. Admittedly, the more dimensions the array has and the more nested content in its declaration, the uglier it becomes.

```
// sort?
sort(_First: a1.rbegin(), _Last: a1.rend());
cout << "Reverse Sort() on a1, now ..." << endl;
showParticleArray(a1);
// #TODO: do a forward (not reverse) sort?
sort(_First: a1.begin(), _Last: a1.end());
cout << "Forward Sort() on a1, now ..." << endl;
showParticleArray(a1);
```

- *Q: a1 etc will be cleaned up (deleted) when out of scope . . . how could you confirm this?*
  A: Uhh . . . I don't know. You wouldn't be able to access the variable a1 outside the scope it was declared in to try a cout statement or something to confirm it was cleaned up, as this would just produce an error or exception. Googling this question didn't turn up anything useful either. Unless there's something I'm missing, I wouldn't think that you would be able to. Not unless you assigned a1's memory address to a variable outside the scope of the method it's called in, and then tried to access that memory address or print it with cout. But that would be overkill, I'd think.