

## Lesson 7: Iterate



*Array iteration, iterators*

## Iterators

- One common task you want to do on a collection of objects is to iterate through each component
- If we have a standardised method for all collections then it is much easier to remember what to do
- But we can also write code that works for any collection that follows this pattern
- This pattern is known as the **iterator pattern**
- The pattern was first developed in C++, but is commonly used in many other languages

1. **Iterators**
2. The C++ Iterator Pattern
3. Linked-List Iterators
4. Generic Programming



## Iterating Over C Arrays

- In C we would typically use a for-loop to iterate over an array
 

```
int n = 10; // size of array
int* begin = malloc(n*sizeof(10)); // malloc returns beginning of array
int* end = begin + n; // address past end of array

int sum = 0;
for(int* pt = begin; pt != end; pt++) {
    sum += *pt; // need to dereference pointer
}
```
- Ugly, but efficient

1. Iterators
2. The C++ Iterator Pattern
3. Linked-List Iterators
4. Generic Programming



- The C++ iterator pattern says for every `container<T>` we create a nested class called `container::iterator` which acts as a pointer (for arrays this could just be a pointer to the array)
- The class should implement
  - ★ a dereferencing operator `T operator* ()`
  - ★ an increment operator `operator++ ()`
  - ★ a not equal function `bool operator!=(const ITER&, const ITER&)` where `ITER` is `container::iterator`

## A Beginning and an Ending

- In addition the container should have two methods
  - ★ `begin()`
  - ★ `end()`
 that return iterators representing the first element and an iterator representing one position past the last element
- Wow! That seems awfully complicated
- Don't panic! We can hack this

## Array-based iterators

- For array based containers such as vector we don't actually need to create an iterator class as we can just use the normal pointer
 

```
template <typename T>
class Array {
private:
    T *data;
    unsigned length;
    unsigned capacity;
public:
    ...
    T* begin() {return data;}
    T* end() {return data+length;}
};
```
- That's all we need