COMP6771
Advanced C++ Programming

Week 2.1
STL Containers

Libraries

Most of us are quite familiar with libraries in software. For example, in COMP1511, we've used <stdio.h> and <stdlib.h>.

Being an effective programmer often consists of the effective use of libraries. In some ways, this becomes more important than being a genius at writing code from scratch (Don't reinvent the wheel!).

While there are many libraries that can be used with C++, the Standard Template Library is the one we will focus on.

STL: Standard Template Library

- STL is an architecture and design philosophy for managing generic and abstract collections of data with algorithms
- All components of the STL are templates
- Containers store data, but don't know about algorithms
- Iterators are an API to access items within a container in a particular order, agnostic of the container used
  - Each container has its own iterator types
- Algorithms manipulate values referenced by iterators, but don't know about containers
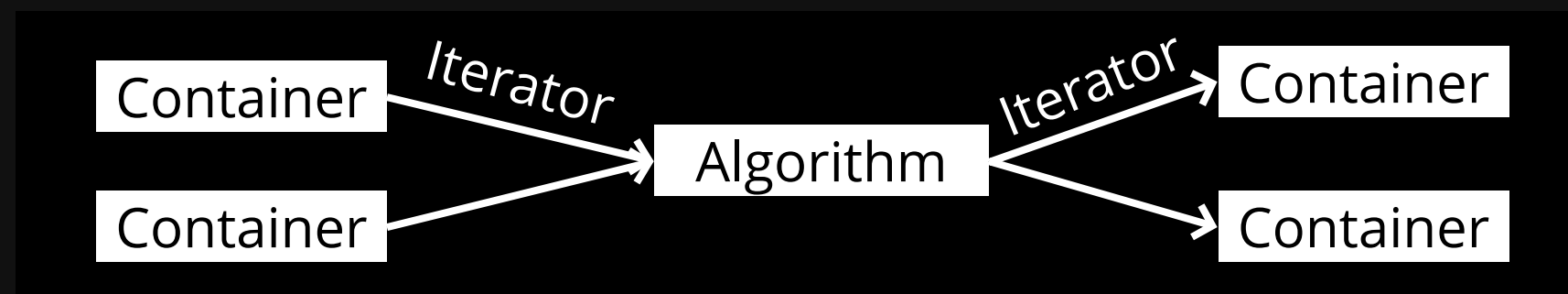
Iterating through a basic container

```cpp
1  #include <array>
2  #include <iostream>
3
4  int main() {
5    // C-style. Don't do this
6    // int ages[3] = { 18, 19, 20 };
7    // for (int i = 0; i < 3; ++i) {
8    //   std::cout << ages[i] << "\n";
9    // }
10
11   // C++ style. This can be used like any other C++ container.
12   // It has iterators, safe accesses, and it doesn't act like a pointer.
13   std::array<int, 3> ages{ 18, 19, 20 };
14
15   for (unsigned int i = 0; i < ages.size(); ++i) {
16     std::cout << ages[i] << "\n";
17   }
18   for (auto it = ages.begin(); it != ages.end(); ++it) {
19     std::cout << *it << "\n";
20   }
21   for (const auto& age : ages) {
22     std::cout << age << "\n";
23   }
24 }
```

demo201-vec-iter.cpp

Sequential Containers

Organises a finite set of objects into a strict linear arrangement.

`std::vector`   Dynamically-sized array.

`std::array`   Fixed-sized array.

`std::deque`   Double-ended queue.

`std::forward_list`   Singly-linked list.

`std::list`   Doubly-linked list.

We will explore these in greater detail in Week 10.

It won't be necessary to use anything other than std::vector in COMP6771.

Another look at Vector

- Array-like container most used is <vector>
  - Abstract, dynamically resizable array
  - In later weeks we will learn about various ways to construct a vector

```cpp
1  #include <iostream>
2  #include <vector>
3
4  // Begin with numbers 1, 2, 3 in the list already
5  int main() {
6    // In C++17 we can omit the int if the compiler can determine the type.
7    std::vector<int> numbers {1, 2, 3};
8    int input;
9    while (std::cin >> input) {
10     numbers.push_back(input);
11   }
12   std::cout << "1st element: " << numbers.at(0) << "\n"; // slower, safer
13   std::cout << "2nd element: " << numbers[1] << "\n"; // faster, less safe
14   std::cout << "Max size before realloc: " << numbers.capacity() << "\n";
15   for (int n : numbers) {
16     std::cout << n << "\n";
17   }
18 }
```

demo202-vec-obj.cpp

Ordered Associative Containers

Provide fast retrieval of data based on keys. The keys are sorted. A **value** is accessed via a **key**.

`std::set`   A collection of unique keys.

`std::multiset`   A collection of keys.

`std::map`   Associative array that map a unique keys to values.

`std::multimap`   Associative array where one key may map to many values.

They are mostly interface-compatible with the unordered associative containers.

std::map example

```cpp
1  #include <iostream>
2  #include <map>
3  #include <string>
4
5  int main() {
6    std::map<std::string, double> m;
7    // The insert function takes in a key-value pair.
8    std::pair<std::string, double> p1{"bat", 14.75};
9    m.insert(p1);
10   // The compiler will automatically construct values as
11   // required when it knows the required type.
12   m.insert({"cat", 10.157});
13   // This is the preferred way of using a map
14   m.emplace("cat", 10.157);
15
16   // This is very dangerous, and one of the most common causes of mistakes in C++.
17   std::cout << m["bat"] << '\n';
18
19   auto it = m.find("bat"); // Iterator to bat if present, otherwise m.end()
20
21   // This is a great example of when to use auto, but we want to show you what type it is.
22   for (const std::pair<const std::string, double>& kv : m) {
23     std::cout << kv.first << ' ' << kv.second << '\n';
24   }
25 }
```

demo203-map.cpp

Unordered Associative Containers

Provide fast retrieval of data based on keys. The keys are hashed.

`std::unordered_set`   A collection of unique keys.

`std::unordered_map`   Associative array that map unique keys to a values.

Container Performance

- Performance still matters
- STL containers are abstractions of common data structures
- cppreference has a summary of them here.
- Different containers have different time complexity of the same operation (see right)

Operation	vector	list	queue
container()	O(1)	O(1)	O(1)
container(size)	O(1)	O(N)	O(1)
operator[]()	O(1)	-	O(1)
operator=(container)	O(N)	O(N)	O(N)
at(int)	O(1)	-	O(1)
size()	O(1)	O(1)	O(1)
resize()	O(N)	-	O(N)
capacity()	O(1)		
erase(iterator)	O(N)	O(1)	O(N)
front()	O(1)	O(1)	O(1)
insert(iterator, value)	O(N)	O(1)	O(N)
pop_back()	O(1)	O(1)	O(1)
pop_front()		O(1)	O(1)
push_back(value)	O(1)+	O(1)	O(1)+
push_front(value)		O(1)	O(1)+
begin()	O(1)	O(1)	O(1)
end()	O(1)	O(1)	O(1)

O(1)+ means amortised constant time

Feedback