Week 6

Standard Library & Container Classes

Agenda

- Week 6-1 – Standard Library
  - string, string_view
  - Standard Template Library
- Week 6-2 – Container Classes
  - Vector
  - Deque
  - List
  - Stack
  - Queue
  - Iterators

Week 6-1

Standard Library

- The C++ standard library has a abundance of functionality to that support the core language
- Some of these libraries are the C++ equivalent of familiar C libraries
- Others expand the language beyond that of C
- Examples:
  - <iostream>, <fstream>, <string>, <exception>, <typeinfo>, <cstring> … etc
- All of the libraries in the standard typically make use of the std namespace

<string>

- The <string> library offers an alternative to the C-Style strings we're familiar with from C but also has some support for them as well
- This library offers support for three types of strings:
  - The std::string classes
  - The string_view classes
  - C-Style string functions

<string>

- The string classes store and manipulate sequences of characters but compared to C-style strings they do the memory management portion internally
- The strings classes are based off an specialization of the std::basic_string template for specific character types
  - std::string – std::basic_string<char>
  - std::wstring – std::basic_string<wchar_t>

<string>

- The string class has some public members that make it quite useful to use over C-style strings:
  - operator= - assign a string to the current string
  - operator[] – accesses a specified character in the string
  - size() – returns the number of characters in the string
  - find(char c) – find the first occurrence of c in the string
  - operator== - equality comparison

<string_view>

- The string_view class was introduced in C++17 as an alternative to strings for when there isn't any need to particularly manipulate a string (ie to just view it or a part of it)
- The use of string_view is more efficient than string in these cases as it doesn't do any allocation of memory when copying strings, it instead refers to other strings
- Generally it has all the public members and functionality a std::string has

Standard Template Library

- Also known as the STL, the Standard Template Library is one of the most prominent parts of the standard library
- It contains code for managing data structures in a generic form, encapsulating the details and allow for reuse of code
- The STL contains:
  - Container classes
  - Iterators
  - Algorithms

Week 6-2

Container Classes

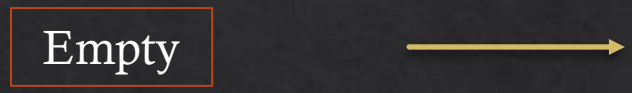
- The STL provides some container classes whose purpose is to allow for collections of types with complimenting functionality for managing said collections
- Some of the container classes we'll look at in this course are:
  - Vector
  - Deque
  - List
  - Stack
  - Queue

<vector>

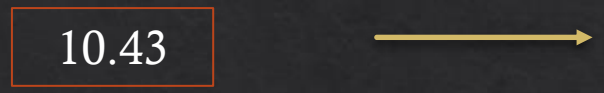
- The vector class template is similar to a built in array of some user defined type
- They store elements in a contiguous blocks of memory and are nearly as efficient as arrays
- They store the elements on the free store/heap and can internally adjust their size as required without manual allocations and deallocations from the user
- Elements are added from the back of the sequence like an array

<vector>

```
std::vector<double> v()
```

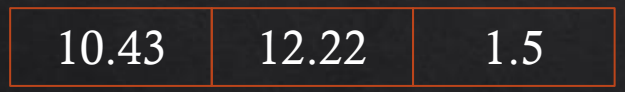



```
v.push_back(10.43);
```




```
v.push_back(12.22);
```


```
v.push_back(1.5);
```




```
v.popback();
```


```
v.back();
```

```
v.front();
```

<deque>

- A deque is short for a “doubly ended queue”
- It’s a container class that can change in size by having elements inserted from either end (front or back), the elements are then ordered in sequence
- The elements themselves however are scattered throughout memory and contiguous storage isn’t guaranteed
- Insertions from both the front and back are efficient operations

<deque>


std::deque<double> dq;
Empty
10.43
dq.push_back(10.43);
dq.push_front(1.25);
1.25
10.43
3.45
1.25
10.43
dq.push_front(3.45);
dq.pop_back();
1.25
10.43
1.25
dq.pop_back();

<list>

- The list container in C++ is a doubly linked list
- This container can like the deque insert elements from both the front and the back in efficient time
- However random element access can be slower than that of vector and deque due to how the container is laid out.

<list>

Container Adapters

- Along with the previous containers there are also adapters that use those containers in a specific context:
  - Stack – last in, first out context (LIFO)
  - Queue – first in, first out context (FIFO)
  - Priority_Queue – First element is always the greatest

<stack>

```
std::stack<double> s;
```

Empty

```
s.push(10.43);
```

10.43

```
s.push(25.25);
```

25.25
10.43

```
s.push(1.25);
```

1.25
25.25
10.43

```
s.pop();
```

25.25
10.43

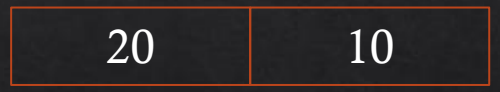
```
s.top();
```

<queue>

```
std::queue<int> q;
```


```
q.push(10);
```


```
q.push(20);
```




```
q.push(30);
```


```
q.pop();
```


```
q.front();
```

Iterators

- Iterators are objects that point to an element in a sequence
- They allow for the simulation of sequential access to elements of container classes. This is particularly useful for those classes that don't use contiguous memory for the storage of their elements.
- Iterators are used to insert and remove elements from a sequence
- The syntax takes the form of:
  - Container<type>::iterator identifier;
  - std::vector<double>::iterator iter;

Iterators

- Each container class has the following member functions that return iterators:
  - iterator begin() – returns an iterator pointing to the first element in a sequence
    - Has a variant called cbegin() which is a const version of begin()
  - iterator end() – returns an iterator pointing to the element one past the end of a sequence
    - Has a variant called cend() which is a const version of end()

Iterators

- Inserting and removing items in a container class can be done through the use these functions that incorporate iterators:
  - iterator insert(iterator position, const T& t) - inserts t at position p and returns an iterator pointing to the inserted element
  - void insert(iterator position, size_t n, const T& t) - inserts t n times at position p
  - void insert(iterator position, InIter f, InIter l) - inserts the range [f,l) at position p
  - iterator erase(iterator p) - removes the element at position p and returns an iterator to the next element
  - iterator erase(iterator f, iterator l) - removes the elements in the range [f,l) and returns an iterator to the next element