



CS 225

Data Structures

Feb. 8 – Stacks and Queues
Wade Fagen-Ulmschneider, Craig Zilles

List.h

```
1 #pragma once
2
3 template <typename T>
4 class List {
5     public:
6         /* ... */
28     private:
29
30
31
32
33 };
```

Array Implementation

C	S	2	2	5
[0]	[1]	[2]	[3]	[4]

Array Implementation

insertAtFront:

C	S	2	2	5
[0]	[1]	[2]	[3]	[4]

Resize Strategy: +2 elements every time





Resize Strategy: +2 elements every time

Resize Strategy: x2 elements every time





Resize Strategy: x2 elements every time

Array Implementation

	Singly Linked List	Array
Insert/Remove at front		
Insert at given element		
Remove at given element		
Insert at arbitrary location		
Remove at arbitrary location		



`std::vector`



std::vector

Defined in header `<vector>`

```
template<
    class T,
    class Allocator = std::allocator<T>           (1)
> class vector;

namespace pmr {
    template <class T>
        using vector = std::vector<T, std::pmr::polymorphic_allocator<T>>; (2) (since C++17)
}
```

- 1) `std::vector` is a sequence container that encapsulates dynamic size arrays.
- 2) `std::pmr::vector` is an alias template that uses a [polymorphic allocator](#)

The elements are stored contiguously, which means that elements can be accessed not only through iterators, but also using offsets to regular pointers to elements. This means that a pointer to an element (since C++03) of a vector may be passed to any function that expects a pointer to an element of an array.

The storage of the vector is handled automatically, being expanded and contracted as needed. Vectors usually occupy more space than static arrays, because more memory is allocated to handle future growth. This way a vector does not need to reallocate each time an element is inserted, but only when the additional memory is exhausted. The total amount of allocated memory can be queried using `capacity()` function. Extra memory can be returned to the system via a call to `shrink_to_fit()`. (since C++11)

Element access

<code>at</code>	access specified element with bounds checking (public member function)
<code>operator[]</code>	access specified element (public member function)
<code>front</code>	access the first element (public member function)
<code>back</code>	access the last element (public member function)
<code>data</code> (C++11)	direct access to the underlying array (public member function)

Capacity

<code>empty</code>	checks whether the container is empty (public member function)
<code>size</code>	returns the number of elements (public member function)
<code>max_size</code>	returns the maximum possible number of elements (public member function)
<code>reserve</code>	reserves storage (public member function)
<code>capacity</code>	returns the number of elements that can be held in currently allocated storage (public member function)
<code>shrink_to_fit</code> (C++11)	reduces memory usage by freeing unused memory (public member function)

Modifiers

<code>clear</code>	clears the contents (public member function)
<code>insert</code>	inserts elements (public member function)
<code>emplace</code> (C++11)	constructs element in-place (public member function)
<code>erase</code>	erases elements (public member function)
<code>push_back</code>	adds an element to the end (public member function)
<code>emplace_back</code> (C++11)	constructs an element in-place at the end (public member function)
<code>pop_back</code>	removes the last element (public member function)
<code>resize</code>	changes the number of elements stored (public member function)
<code>swap</code>	swaps the contents (public member function)



Stack ADT



Queue ADT