Containers classified into:

- Sequence containers-
  - Holder object that stores a collection of objects (elements) in a STRICT LINEAR SEQUENCE
  - Include: array, vector, deque, and list
- Associative containers-
  - Ordered
  - Include: sets, multisets, maps, multimaps
- Container adapters-
  - [To add]

*Going through every single element will always be O(n) regardless data structure

Vector [Sequence containers]- (REFER TO PAGE 1465/1473 ON HOW TO LOOP THROUGH VECTOR) [1/3]

Use when we want to access known [i] location + ALWAYS Iterate over EVERY element and do something

Property:

- Vectors are sequence containers representing *arrays that can change size*
- Contain collection of same type objects (elements) that are ordered in a strict linear sequence
- Item access is VERY FAST

Disadvantages:

- Not ideal searching (Have to go through all elements. Unlike tree) O(n)
  - Sorted array search O(log n)
- Not ideal insertion + deletion of elements (Has to go through all elements)

Advantages:

- Can change size and resize unlike array
- Ideal for accessing single element at [i] location O(1)
- Iterate over every element and do something with EVERY element (since all contingous + no chasing)

Binary search tree-

Use when we want the data as the key (so key) + Order

Property:

- Contains key (non-duplicate or duplicate) and right + left pointer

Disadvantages:

- Insertion of ordered data turns it into linked list thus searching is slow

Advantages:

- Ideal for searching if height is low
- Easier to code compared to balanced tree

Maps [Associative containers]- (red black tree)

Use when we want key-value + we care about order + no duplicates + Key as Struct

Property:

- Contain key (non-duplicate) and value pair [NODE]
- Usually implemented as a red black tree (balanced tree = small height)
- Key is not some random description it has meaning

Advantages:

- Searching is fast. Given it's a red black tree insertion of ordered data doesn't turn it to a linked list due to rebalancing thus height is low
- Insertion is fast (Doesn't need to go through all the elements + height is guaranteed low)

MultiMaps [Associative containers]-

Use when we want key-value + we care about order + duplicates allowed

Property:

- Contain key (duplicate) and value pair [NODE]

Advantages:

-

Unordered map – (Hash table) #1 FOR SPEED

Use when we don't care about key order + Will Randomize key order inserted + !!!

Property:

- Contain key (non-duplicate) and value pair
- The key is not ordered and is converted to some random number identifier key (via hash)
    - Key "dog" = 3ehp
    - A,B,C inserted → C A B
- So think of it as an a group of buckets (arrays). Each bucket contains group of elements

Advantages:

- Searching/Access, Insertion, Deletion O(1)

Disadvantages:

- Finite number of hash codes. So possible for elements to share same index thus COLLISION = ISSUE

Unordered multimap – (Hash table) #1 FOR SPEED

Hashing is used @1 hr 22 min lecture 10/05/2021

Stack [Sequence containers]- (Array, linked lists, vector) [FILO]

Use for last in first out (putting books on the ground) + Instances where you want to

Use for first in last out (putting books on the ground)

Property:

- Behavior dictates only thing we have access to is top of stack so DON'T iterate for printing because we are violating behavior! Only see what is on top and CAN'T see inside of stack
- Minimal and complete- Push (put on top), Pop (remove from top + give back), Empty()
- STL- Pop (remove from top), Top(return element), Push(put on top)

Advantages:

- Insertion O(1) because we just dump it to ground
- Deletion O(1) because we always just remove from top

Disadvantages:

- Searching for element O(n) because we need to iterate/pop every element to find
- Access is O(n) because we can't just access [i] we need to we need to iterate/pop every element to access

Queue- (Array or Linked list or vector) [FIFO]

Use for first in the queue first out of the queue (lining up in coles lane)

Property:

- Behavior dictates that transversal/search/iteration is NOT required or else not minimal
- Minimal and complete behaviour (abstract)- Enqueue (enter a queue), Dequeue (leaving a queue), Empty(), Full()
- STL- Push(enter a queue), Front(return element front of queue), Pop(Remove from front of queue)

Advantages:

- Insertion O(1) because we just come from (enqueue) end
- Deletion O(1) because we just take from (dequeue) front

Circular queue: allows for resource management of fixed data structure size. For advance queue movement of front and back elements. Depends on the modules operation for management of fixed resources

Linked list-

Property:

- Contains Key (data) + pointer [Node]
- Last node contains nullptr
- Store different data types

Advantages:

- Insertion O(1) (think if we need to search sure it will be O(n) but the insertion itself will always be O(1))!!
- Delete O(1) (think if we need to search sure it will be O(n) but the insertion itself will always be O(1)

Disadvantages:

- Searching/Access is not ideal since with both you need go through all elements to do both operations O(n)

Sets [Associative containers]- [3/4] USE MULTISET, MULTIMAPS FOR REPEATED

Sets are containers* that store unique (no duplicates) elements (we make it that way)

Think of it like our BST but balanced (red black tree).

Only has a single key (data) and no value

Property:

- Contain key (non-duplicate) + Ordered for better efficiency
- Ordered unique collection of homogenous Key (data)

Advantages:

- Searching is fast o(log n) through the find() since it is a red black tree

Disadvantages:

- Does not allow item insertion/removal at a position (only by value. The value is the key/identifier)

STL deque [Sequence containers]- [2/3]

Deques are sequence containers representing *arrays that can change size*. Elements inserted both ends so front or back. Elements are scattered in different storage locations with container keeping information on where all elements are

Properties of deque:

- Sequence- Contain collection of objects (elements) that are ordered in a strict linear sequence. Objects same datatype
- Generally Dynamic array- Allows direct access to any element in sequence and provides FAST addition/removal of elements at start + end of the sequence
- Allocator-aware- Container uses allocator object to dynamically handle storage needs

Advantages of deque:

- Item insertion/removal at END is FAST (compared to arrays)
- Item insertion/removal at BEGINNING is FAST (compared to arrays)

Disadvantage of deque:

- Item insertion/removal at MIDDLE is SLOW

Constructors of deque: (refer page1476 for more)

```
deque<[DataType]> dequeName;
```

Declare iterator of deque: (refer page1458 for more)

```
std::deque<[DataType]>::iterator IteratorName;

*iteratorName //Returns the object at current iteratorPosition
```

Comparision

Stack vs Array

- Don't iterate all through stack! Use array

Struct vs Class

- Struct only public + protected
- Struct contains no invariants properties maintained? (*Think one member affects another member OR one member has restrictions)
    - So does any member in struct depend on each other? Day, Month, Year?
        - Month determines amount of days. So in class
        - Windspeed has restriction so in class
        - Struct can't contain month and day variable as one determines another so put in a class (encapsulate) then put in struct
- Important: Class use
    - Protecting/providing controlled access to data members (Encapsulation) is valid ONLY
        - Setters and Getters check for something in regards to data members
            - Having no setters checking anything is stupid and roadblock
        - And invariant is maintained (think as a collection of data members like day,month,year or as a single data member)
- Struct controls access vs long list of pass by parameter
    - OutputResult(int avgWindSpeed, int sdWindspeed, int avgTemp) Vs Struct

Array vs Linked list

- Array when we want to iterate through them all
- Array for fast access O(1)
- Linked list O(1) insertion [O(n) searching] vs Array O(n) insertion [O(n) searching] because we need essentially move all the elements to the right of the insertion spot that requires copying since in array all elements are joined. For linked list just change pointers

Linked list vs BST

- They are the same but BST contains two pointers left and right to make sure bst can be easily ordered and searched fast. A linked list contains only one pointer
- Use a linked list when we can't find a logical ordering for the key

unordered map (hash table) vs Array

- Arrays you can't identify with strings
- Searching/Access unordered map O(1) == O(1) Access
- Insertion unordered map O(1)

Checklist:

- Do you need to iterate through every single element (unsorted) and do something each iteration (like add element to stack)
    - MUST Use an array
    - NEVER ITERATE THROUGH EVERY ELEMENT WITH stack/BST/Map

- Do you need find something but don't want to go through all elements?
    - Use BST/Map

https://stackoverflow.com/questions/23103690/why-is-accessing-any-single-element-in-an-array-done-in-constant-time-o1

Accessing a single element is NOT finding an element whose value is x.

Accessing an element i means getting the element at the i'th position of the array.

This is done in O(1) because it is pretty simple (constant number of math calculations) where the element is located given the index, the beginning of the array and the size of each element.

RAM memory offers a constant time (or to be more exact, a bounded time) to access each address in the RAM, and since finding the address is O(1), and retrieving the element in it is also O(1), it gives you total of O(1).

Finding if an element whose value is x is actually Omega(n) problem, unless there is some more information on the array (sorted, for example).

Share

If you're going to iterate every single element use a array