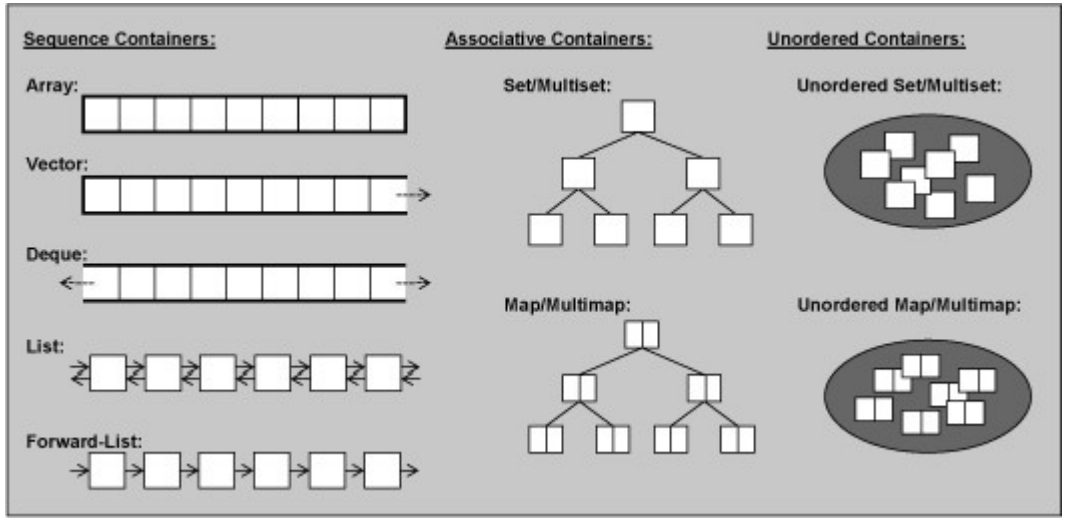
CALIFORNIA STATE UNIVERSITY FULLERTON

CPSC 131 – Data Structures

Array & Vector Abstract Data Types



Professor T. L. Bettens

Fall 2020

Concepts & Interface

- Jusuttis,
  [The C++ Standard Library]
  - [6.2. Containers]
  - [7.1. Common Container Abilities and Operations]
  - [7.2. Arrays]
  - [7.3. Vectors]
- [CPPReference.com]
  - [Containers library]
  - [std::array]
  - [std::vector]
- zyBook
  - tbd

Array Abstract Data Type

Definitions:

Capacity - max number of elements that can be stored
Size - another name for Capacity – an array's size does not (can not) change

Fixed Capacity Array

- Capacity is constant
  - Set at container definition at design (compile) time

Two flavors

- Standard arrays
  - Smart wrapper around native array
  - std::array from the <array> library
  - Ex: `std::array<Student, 10> myArray;`
- Native arrays
  - aka C-Style or raw array
  - AVOID using these
  - Ex: `Student myArray[10];`

Vector Abstract Data Type
Common Implementation choices

Definitions:

Capacity - max number of elements that can be stored
Size - number of elements that are stored

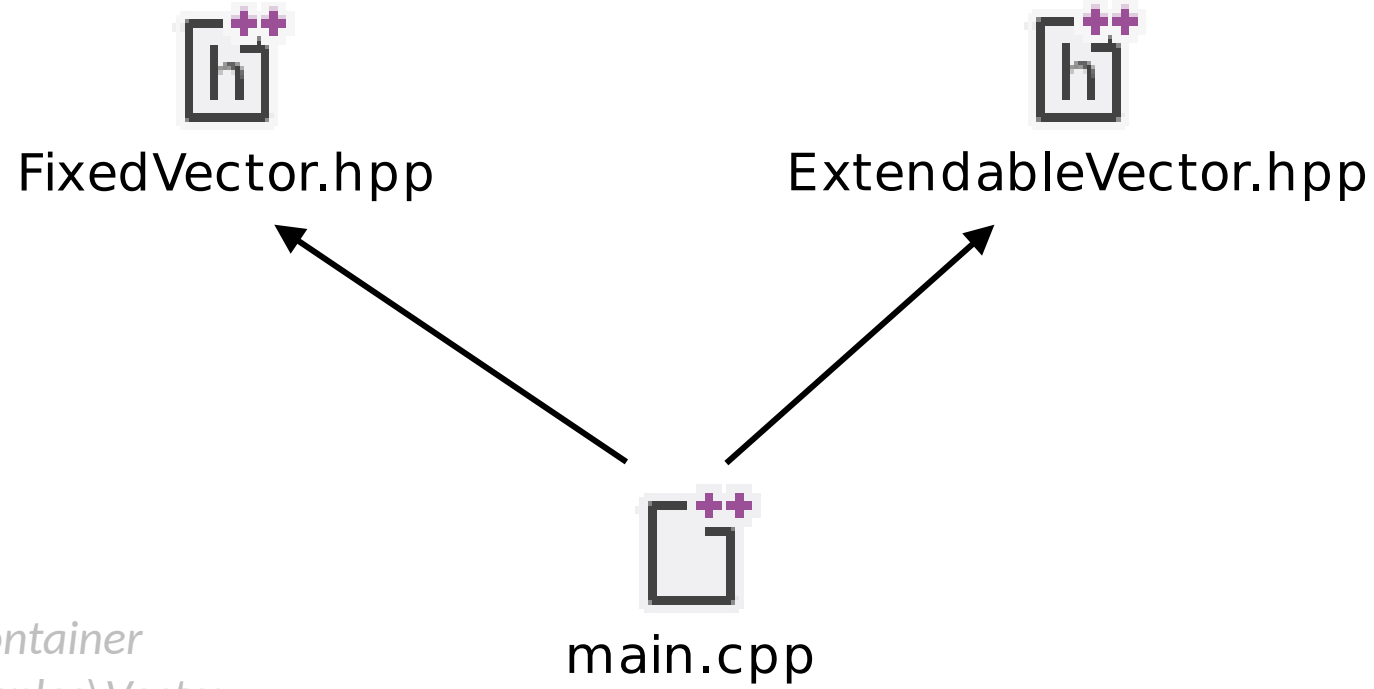
Fixed Capacity Vector

- Capacity is constant
  - Set at container construction during runtime, or
  - Set at container definition at design (compile) time

Extendable Capacity Vector

- Capacity is dynamic and changes during runtime
  - Initialized at container construction during runtime
  - Grows and shrinks during runtime

Vector Implementation Example



See Sequence Container Implementation Examples\Vector

Construction Examples

https://en.cppreference.com/w/cpp/container/vector/vector

```
int main()
{
 // c++11 initializer list syntax:
 std::vector<std::string> words1 {"the", "frogurt", "is", "also", "cursed"};

 // words2 == words1
 std::vector<std::string> words2(words1.begin(), words1.end());

 // words3 == words1
 std::vector<std::string> words3(words1);

 // words4 is {"Mo", "Mo", "Mo", "Mo", "Mo"}
 std::vector<std::string> words4(5, "Mo");
```

NxN Matrix via Vectors

```
// Create the game board and initialize it to be empty
unsigned BOARD_SIZE = 3U; // Could ask the user for this value ...


// define the NxN matrix and initialize every cell to the empty cell
GameBoardType theBoard(BOARD_SIZE, std::vector<char>(BOARD_SIZE, EMPTY_CELL));
```

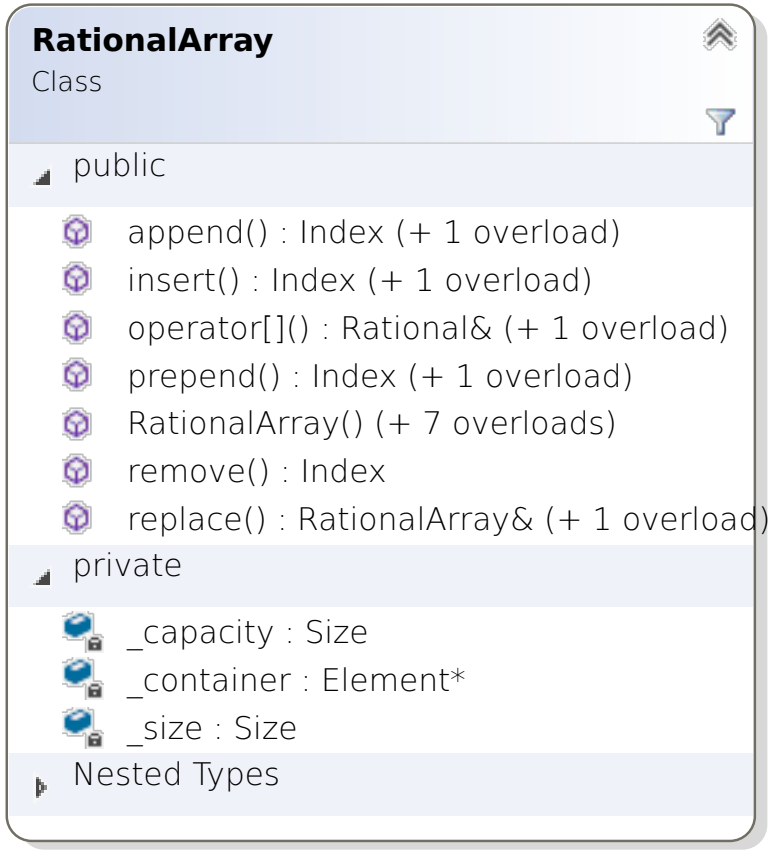
number of elements

value of each element

number of elements

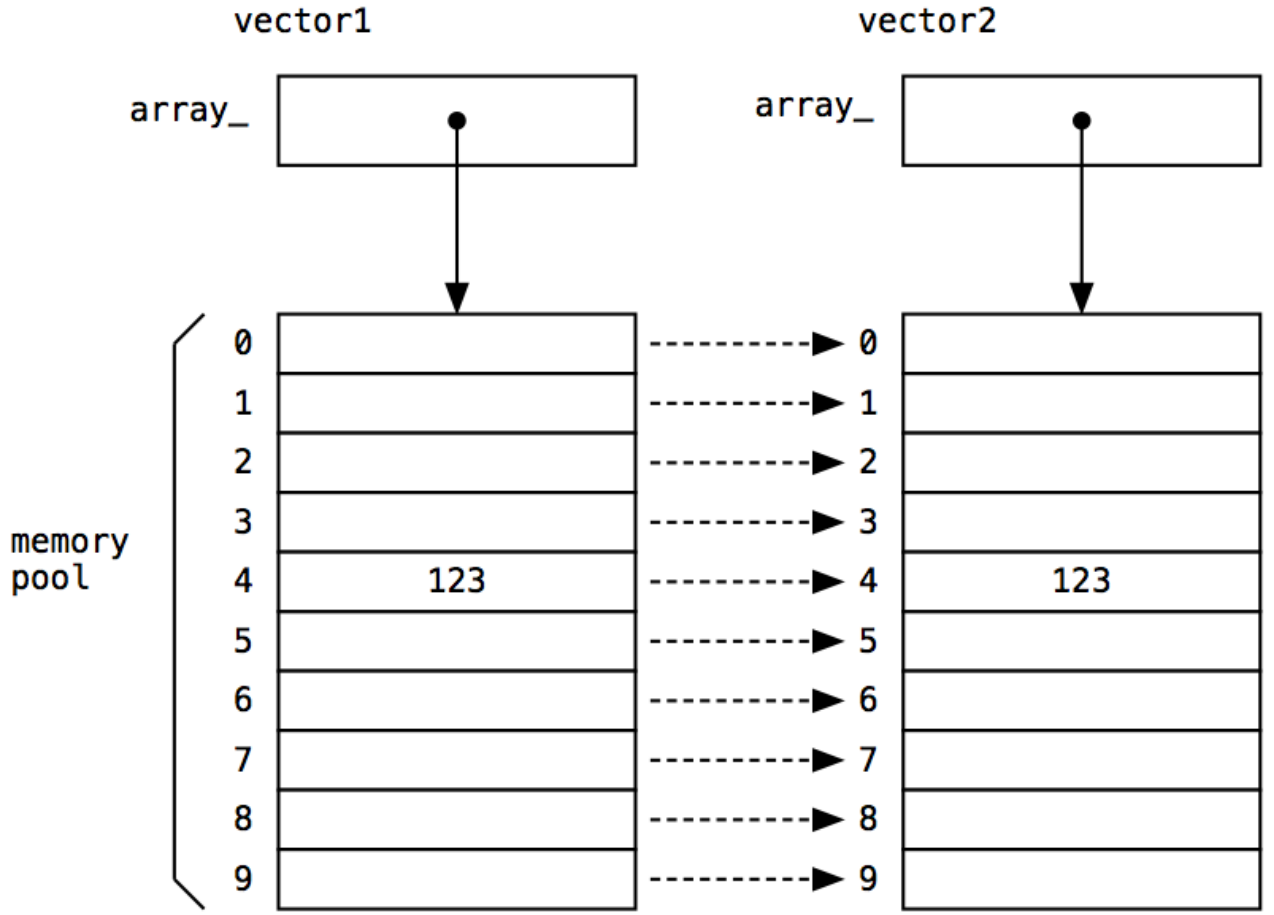
value of each element

RationalArray Case Study



Complexity Analysis
Coping vectors requires O(n) deep copy

- vector2 needs its own copy of vector1
- Shallow copy of only pointer isn't enough
- Need deep copy of data that’s pointed to



Analysis of the Vector Abstract Data Type
Complexity Analysis (1)

Function	Analysis – std::array<T, S>	Analysis – std::vector<T> (Extendable Vector)
at()	O(1) Elements directly indexable	same
size()	O(1) Always returns S, as in std::array<T, S>	O(1) Returns the number of elements held
empty()	O(1)	same
clear()	Not available std::array<T, S> will always have S elements	O(n) All elements are destroyed and size set to zero *O(1) if only size set to zero, as in zyBook*

Analysis of the Vector Abstract Data Type
Complexity Analysis (2)

Function	Analysis – std::array<T, S>	Analysis – std::vector<T> (Extendable Vector)
push_back()	Not available std::array<T, S> will always have S elements	O(n) amortized to O(1) Special case of insert()
erase()	Not available std::array<T, S> will always have S elements	O(n) Have to “close the gap” which means N copies (worst case, N/2 copies average case)
splice	Not available	Not available

Analysis of the Vector Abstract Data Type
Complexity Analysis (3)

Function	Analysis – std::array<T, S>	Analysis – std::vector<T> (Extendable Vector)
insert()	Not available std::array<T,S> will always have S elements	O(n) (worst case) If space is not available, • get more space and copy N elements • Destroy N elements "Open a gap" which means N copies
default construction	O(n) container is never empty	O(1) creates an empty container
Equality $C_1 == C_2$	O(n)	same

Analysis of the Vector Abstract Data Type
Complexity Analysis (4)

Function	Analysis – std::array<T, S>	Analysis – std::vector<T> (Extendable Vector)
push_front	Not available std::array<T, S> will always have S elements	Not available
resize	Not available std::array<T, S> will always have S elements	O(n)
find	O(n) linear search from begin() to end() *(i.e. a[0] to a[size()-1])*	same

Analysis of the Vector Abstract Data Type
Complexity Analysis (4)

Function	Analysis – std::array<T, S>	Analysis – std::vector<T> (Extendable Vector)
Visit every element e.g. print()	O(n) Visiting every node from begin() to end()	same
Visit in reverse e.g. print_reverse()	O(n) Visiting every node from rbegin() to rend() Direction doesn't matter	same