CSC 212: Data Structures and Abstractions

08: Queues

Prof. Marco Alvarez

Department of Computer Science and Statistics University of Rhode Island

Spring 2025



Stacks

- Consider a stack implemented by a dynamic array (insertion and deletion at the end)
 - ✓ what is the computational cost?

Push	O(1) amortized
Рор	O(1)

- Consider a stack implemented by a dynamic array (insertion and deletion at the beginning)
 - ✓ what is the computational cost?

both operations require shifting elements

Push	O(n)
Рор	O(n)

Solution to lab problem

```
int eval(const std::string& exp) {
    std::stack<int> operands;
    std::stack<char> operators:
    for (size_t i = 0 ; i < exp.length() ; ++i) {
   if (exp[i] == ' ' || exp[i] == '(') {</pre>
             continue:
         } else if (isdigit(exp[i])) {
        operands.push(exp[i] - '0');
} else if (exp[i] == '+' || exp[i] == '-' || exp[i] == '*' || exp[i] == '/') {
             operators.push(exp[i]);
         } else if (exp[i] == ')')
             int right = operands.top();
             operands.pop();
             int left = operands.top();
             operands.pop();
             char op = operators.top();
             operators.pop();
             switch (op) {
                 case '+': operands.push(left + right); break;
                 case '-': operands.push(left - right); break;
                 case '*': operands.push(left * right); break;
                 case '/': operands.push(left / right); break;
    return operands.top();
```

std::Stack

The std::stack class is a container adaptor that gives the programmer the functionality of a stack @ - specifically, a LIFO (last-in, first-out) data structure.

The class template acts as a wrapper to the underlying container - only a specific set of functions is provided. The stack pushes and pops the element from the back of the underlying container, known as the top of the stack.

Member functions

constructor)	(public member function)
destructor)	destructs the stack (public member function)
perator=	assigns values to the container adaptor (public member function)
ement access	
ор	accesses the top element (public member function)
pacity	
empty	checks whether the container adaptor is empty (public member function)
ize	returns the number of elements (public member function)
odifiers	
ush	inserts element at the top (public member function)
ush_range (C++23)	inserts a range of elements at the top (public member function)
mplace (C++11)	constructs element in-place at the top (public member function)
ion	removes the top element

swaps the contents

```
#include <cassert>
#include <stack>
int main()
{
    std::stack<int> stack;
    assert(stack.size() == 0);

    const int count = 8;
    for (int i = 0 ; i != count ; ++i) {
        stack.push(i);
    }
    assert(stack.size() == count);
}
```

Queues

Queues

- · First-in-first-out
 - ✓ a <u>queue</u> is a linear data structure that follows the (FIFO) principle
 - the first element added to the queue is the first one to be removed
 - analogous to a real-world queue, such as a line of people waiting for service
- · Main operations
 - Enqueue: add an element to the end of the queue
 - Dequeue: remove an element from the front of the queue



 scheduling tasks in operating systems, managing requests in web servers, implementing breadth-first search (BFS) in graph algorithms, etc.



6

Practice

• What is the output of this code?

```
Queue<int> s1, s2;

s1.enqueue(100);
s2.enqueue(s1.dequeue());
s1.enqueue(200);
s1.enqueue(300);
s2.enqueue(s1.dequeue());
s2.enqueue(s1.dequeue());
s1.enqueue(s2.dequeue());
s1.enqueue(s2.dequeue());
while (!s1.empty()) {
    std::cout << s1.dequeue() << std::endl;
}
while (!s2.empty()) {
    std::cout << s2.dequeue() << std::endl;
}</pre>
```

Practice

- Write a function that modifies a queue of elements by replacing every element with two copies of itself
 - ✓ for example: [a, b, c] becomes [a, a, b, b, c, c]

Practice

- Design an algorithm to:
 - ✓ load a number of audio files (songs)
 - ✓ play them in a continuous loop

Practice

- Write an algorithm to reverse the order of elements of a queue (hint: can use a separate stack)
- Write an algorithm that accepts a queue of elements and appends the queue's contents to itself in reverse order (hint: can use a separate stack)
 - for example: [a, b, c] becomes [a, b, c, c, b, a]

10 l

Implementation

- Using arrays
 - ✓ ensure **enqueue** and **dequeue** work at <u>different ends</u> of the array
 - ✓ array can be fixed-length or a dynamic array (additional cost)
- Considerations
 - underflow: throw an error when calling dequeue on an empty queue
 - verflow: throw an error when calling enqueue on a full queue

Implementation

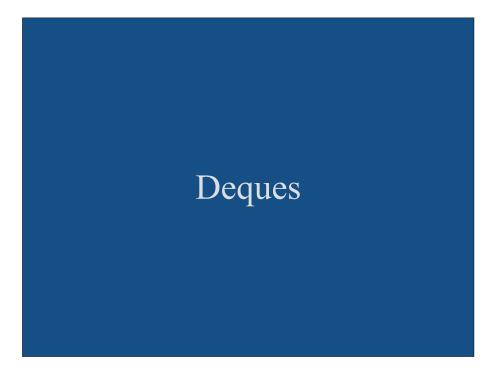
- Array-based (standard)
 - \checkmark enqueue at the end O(1) cost (amortized cost if using a dynamic array)
 - \checkmark dequeue from the beginning O(n) cost
 - requires shifting elements
- Array-based (alternative)
 - \checkmark enqueue at the beginning O(n) cost
 - requires shifting elements
 - \checkmark dequeue from the end O(1) cost
- · Circular array
 - \checkmark enqueue at the end O(1) cost (amortized cost if using a dynamic array)
 - \checkmark dequeue from the beginning O(1) cost
 - who more efficient approach, as it eliminates the need for shifting elements
 - √ requires handling wrap-around at array boundaries

12


```
// implements a (circular) queue using a fixed-size array
class Oueue {
    private:
        // array to store queue elements
        int *array;
        // maximum number of elements queue can hold
        int length;
        // index of the first element in the queue
        int base:
        // index of the last element in the queue
        int top;
    public:
        Queue(int);
        ~Queue();
        // adds an element to the end of the queue
        void enqueue(int);
        // removes the first element from the queue
        int dequeue();
```

```
std::queue
   Defined in header <queue>
  template<
       class T.
       class Container = std::deque<T>
  > class queue;
The std::queue class template is a container adaptor that gives the functionality of a queue 🗗 - specifically, a FIFO
The class template acts as a wrapper to the underlying container - only a specific set of functions is provided. The queue
pushes the elements on the back of the underlying container and pops them from the front.
  Member functions
                constructs the queue
  (constructor)
                                                        #include <cassert>
                                                       #include <queue>
                assigns values to the container adaptor
  operator=
 Element access
                                                        int main()
                access the first element
                access the last element
                                                              std::queue<int> queue:
 Capacity
                                                              assert(queue.size() == 0);
                checks whether the container adaptor is empty
                 returns the number of elements
                                                              const int count = 8;
  size
                                                              for (int i = 0; i != count ; ++i) {
 Modifiers
                                                                    queue.push(i);
                inserts element at the end
                inserts a range of elements at the end
  push_range (C++23
                                                              assert(queue.size() == count);
                constructs element in-place at the end
  emplace (C++11)
                  emoves the first element
                swaps the contents
  swap (C++11)
```

https://www.cs.usfca.edu/~galles/visualization/QueueArray.html



Deques

- · Double-ended queue
 - a deque (pronounced "deck") is a linear data structure that allows insertion and removal of elements from both ends
 - combines the capabilities of stacks and queues
- Main operations
 - InsertFront, InsertEnd: add an element to the front or to the end of the queue respectively
 - DeleteFront, DeleteEnd: remove an element from the front or from the end of the queue respectively



 task scheduling, undo/redo functionality, web browser history (forward/backward), sliding window problems, palindrome checking, etc.





1

Implementation

- Using arrays
 - ✓ array can be fixed-length or a dynamic array (additional cost)
- Considerations
 - ✓ underflow: throw an error when calling "remove" on an empty queue
 - ✓ overflow: throw an error when calling "insert" on a full queue
- · Circular array
 - ✓ use a circular array to allow efficient operations at both ends
 - $\checkmark O(1)$ cost for all operations
 - "InsertEnd" has an amortized constant time if using a dynamic array

18

std::deque

```
Defined in header <deque>

template<
    class T,
    class Allocator = std::allocator<T>
    class deque;

namespace pmr {
    template< class T >
    using deque = std::deque<T, std::pmr::polymorphic_allocator<T>>;
}

(1)

(2) (since C++17)
```

std::deque (double-ended queue) is an indexed sequence container that allows fast insertion and deletion at both its beginning and its end. In addition, insertion and deletion at either end of a deque never invalidates pointers or references to the rest of the elements.

As opposed to std::vector, the elements of a deque are not stored contiguously: typical implementations use a sequence of individually allocated fixed-size arrays, with additional bookkeeping, which means indexed access to deque must perform two pointer dereferences, compared to vector's indexed access which performs only one.

The storage of a deque is automatically expanded and contracted as needed. Expansion of a deque is cheaper than the expansion of a std::vector because it does not involve copying of the existing elements to a new memory location. On the other hand, deques typically have large minimal memory cost; a deque holding just one element has to allocate its full internal array (e.g. 8 times the object size on 64-bit libstdc++; 16 times the object size or 4096 bytes, whichever is larger, on 64-bit libc++).

The complexity (efficiency) of common operations on deques is as follows:

- Random access constant O(1).
- Insertion or removal of elements at the end or beginning constant O(1).
- Insertion or removal of elements linear O(n).

Member functions constructs the deque (constructor) destructs the deque (destructor assigns values to the container assigns values to the container assign assigns a range of values to the containe assign_range(C++23) returns the associated allocator Element access access specified element with bounds checking access specified element operator[] access the first element front Iterators returns an iterator to the beginning cbegin (C++11) returns an iterator to the end cend (C++11) returns a reverse iterator to the beginning returns a reverse iterator to the end crend (C++11)

Capacity checks whether the container is empty returns the number of elements size returns the maximum possible number of elements max size reduces memory usage by freeing unused memory shrink to fit (DR*) clears the contents clear inserts elements insert inserts a range of elements insert range (C++23) constructs element in-place emplace (C++11) erase adds an element to the end push back constructs an element in-place at the end adds a range of elements to the end append_range (C++23) removes the last element pop back inserts an element to the beginning push front emplace_front (C++11) constructs an element in-place at the beginning adds a range of elements to the beginning prepend_range (C++23 removes the first element pop_front changes the number of elements stored resize swans the contents

```
#include <deque>
#include <iostream>
int main()
{
    // create a deque containing integers
    std::deque<int> d = {7, 5, 16, 8};

    // add an integer to the beginning and end of the deque
    d.push_front(13);
    d.push_back(25);

    // iterate and print values of deque
    for (int n : d) {
        std::cout << n << ' ';
    }
    std::cout << '\n';
}</pre>
```