CSC 212: Data Structures and Abstractions

Stacks and Queues

Marco Alvarez

Department of Computer Science and Statistics
University of Rhode Island

Fall 2020

Stacks

LIFO: Last In First Out

Basic Operations

- **Push**
  - inserts one element onto the stack
- **Pop**
  - returns the element at the top of the stack (and removes it)
- IsEmpty
  - not necessary, but sometimes useful

std::stack

Defined in header <stack>

```
template<
    class T,
    class Container = std::deque<T>
> class stack;
```

The std::stack class is a container adapter 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)	constructs the stack (public member function)
(destructor)	destructs the stack (public member function)
operator=	assigns values to the container adaptor (public member function)
Element access	
top	accesses the top element (public member function)
Capacity	
empty	checks whether the underlying container is empty (public member function)
size	returns the number of elements (public member function)
Modifiers	
push	inserts element at the top (public member function)
emplace (C++11)	constructs element in-place at the top (public member function)
pop	removes the top element (public member function)
swap	swaps the contents (public member function)

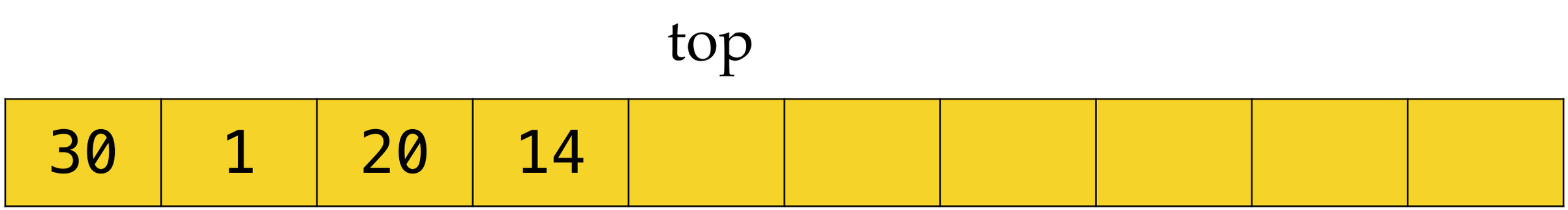
Member objects

Container c	the underlying container (protected member object)

```
#include <stack>
#include <iostream>

int main()
{
    std::stack<int>   s;

    s.push( 2 );
    s.push( 6 );
    s.push( 51 );

    std::cout << s.size() << " elements on stack\n";
    std::cout << "Top element: "
              << s.top()          // Leaves element on stack
              << "\n";
    std::cout << s.size() << " elements on stack\n";
    s.pop();
    std::cout << s.size() << " elements on stack\n";
    std::cout << "Top element: " << s.top() << "\n";

    return 0;
}
```

Implementation

- Arrays
  - **push** and **pop** at the end of the array (easier and efficient)
  - can be fixed-length
  - can also use a dynamic array (grows over time)
    - additional cost for dynamic arrays

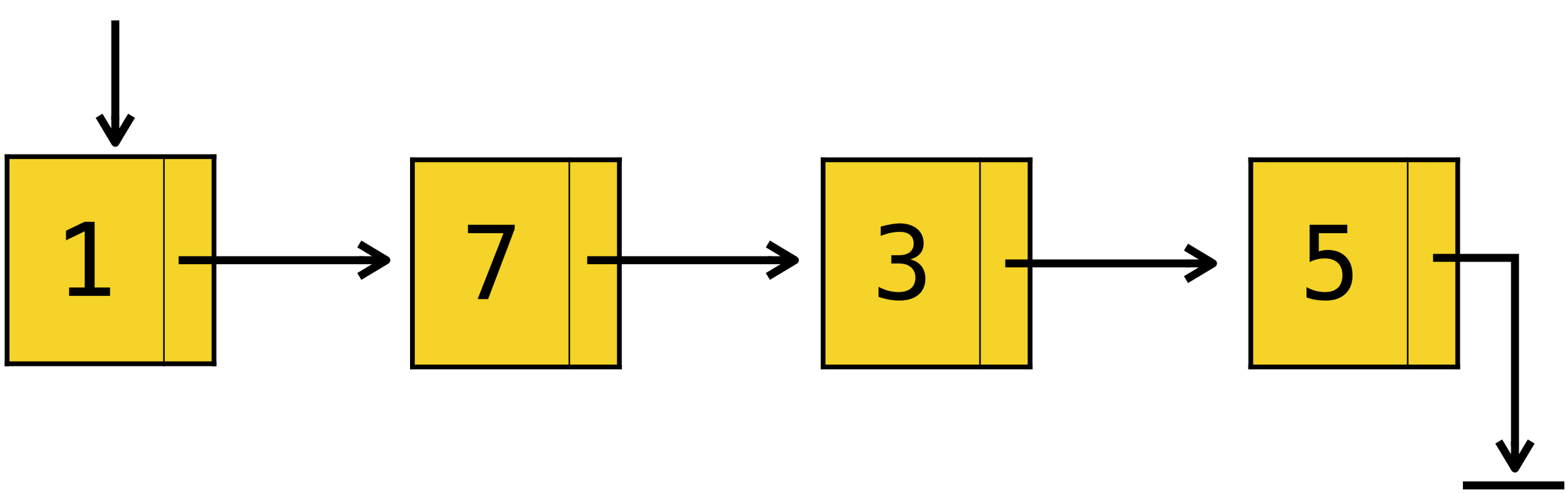


https://www.cs.usfca.edu/~galles/visualization/StackArray.html

```
class Stack {
    private:
        int *array;
        int length;
        int top_idx;

    public:
        Stack();
        ~Stack();

        void push(int);
        int peek(); // returns top
        void pop(); // removes top
};
```

Implementation

- Linked Lists
  - **push** and **pop** at front (could use the other end as well)



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

Considerations

- Underflow
  - error can be thrown when calling **pop** on an empty stack
- Overflow
  - error can be thrown when calling **push** on a full stack (especially in fixed-length implementations)

Applications

- Undo in software applications
- Stack in compilers / programming languages
- Parsing expressions
- . . .

Queues

FIFO: First In First Out

Basic Operations

- **Enqueue**
  - inserts one element onto the queue
- **Dequeue**
  - returns the next element from the queue (and removes it)
- IsEmpty
  - not necessary, but sometimes useful

std::queue

Defined in header <queue>

```
template<
    class T,
    class Container = std::deque<T>
> class queue;
```

The std::queue class is a container adapter that gives the programmer the functionality of a queue - specifically, a FIFO (first-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 queue pushes the elements on the back of the underlying container and pops them from the front.

Member functions

(constructor)	constructs the queue (public member function)
(destructor)	destructs the queue (public member function)
operator=	assigns values to the container adaptor (public member function)
Element access	
front	access the first element (public member function)
back	access the last element (public member function)
Capacity	
empty	checks whether the underlying container is empty (public member function)
size	returns the number of elements (public member function)
Modifiers	
push	inserts element at the end (public member function)
emplace (C++11)	constructs element in-place at the end (public member function)
pop	removes the first element (public member function)
swap	swaps the contents (public member function)

Member objects

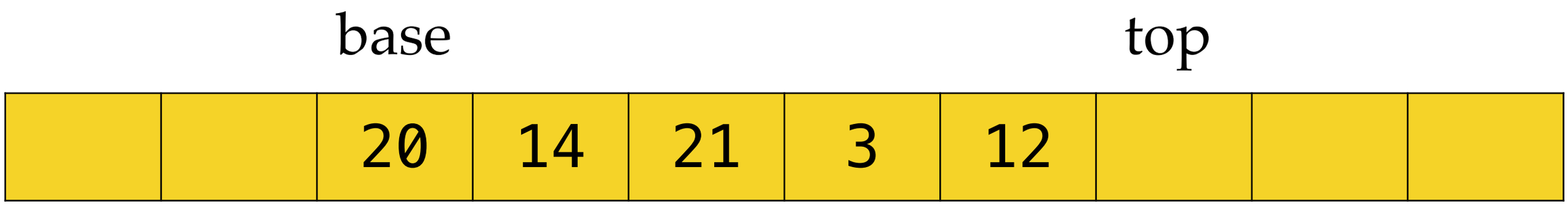
Container c	the underlying container (protected member object)

```
#include <queue>
#include <deque>
#include <iostream>

int main()
{
    std::queue<int> c1;
    c1.push(5);
    std::cout << c1.size() << '\n';

    std::queue<int> c2(c1);
    std::cout << c2.size() << '\n';

    std::deque<int> deq {3, 1, 4, 1, 5};
    std::queue<int> c3(deq);
    std::cout << c3.size() << '\n';
}
```

Basic Operations (enqueue / dequeue)

Implementation

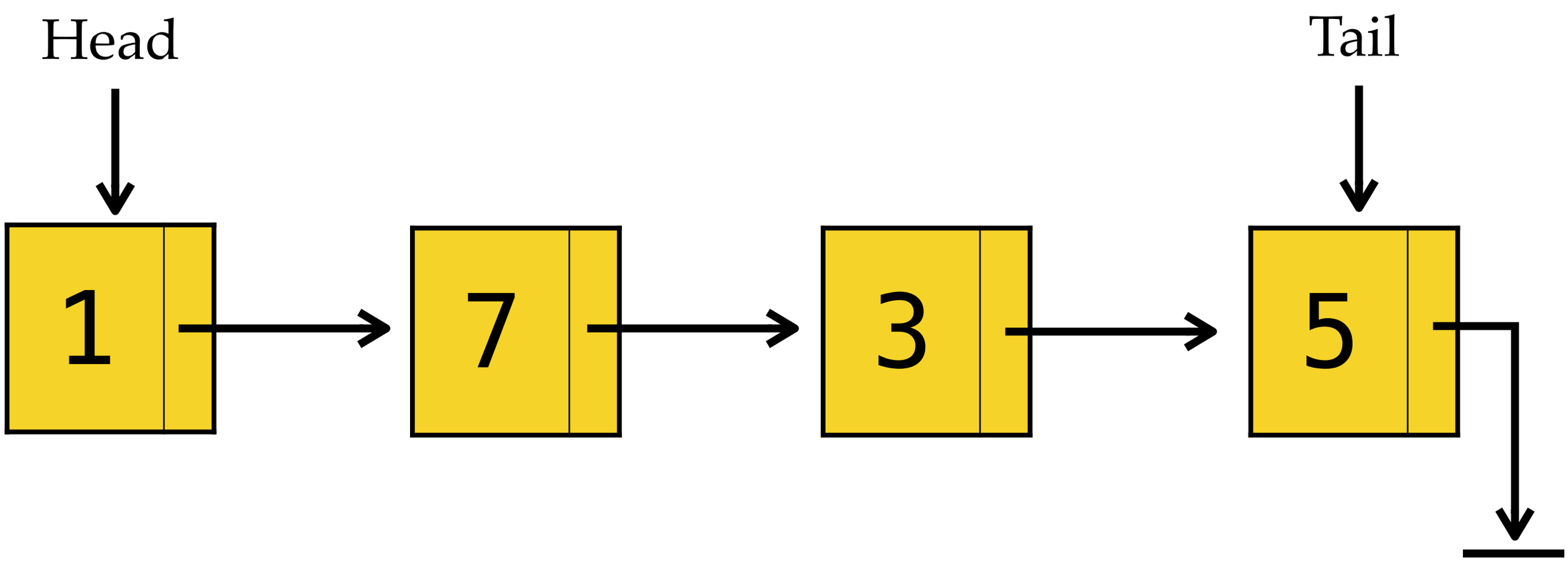
- Arrays
  - **enqueue** and **dequeue** at different ends of the array
  - can be fixed-length
  - can also use a dynamic array (grows over time)
    - additional cost for dynamic arrays



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

Implementation

- Linked Lists
  - **enqueue** and **dequeue** at different ends



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

Considerations

- Underflow
  - error can be thrown when calling **dequeue** on an empty queue
- Overflow
  - error can be thrown when calling **enqueue** on a full queue (especially in fixed-length implementations)

Applications

- Media Playlists (Youtube, Spotify, Music, etc.)
- Process management in Operating Systems
- Simulations
- Used in other algorithms
- …