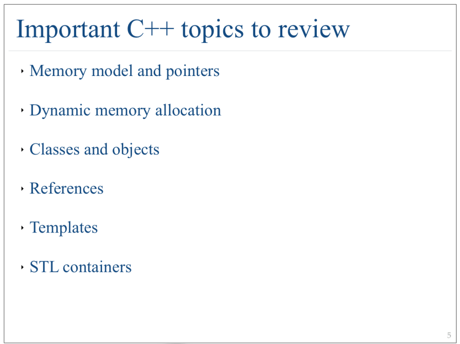
Stacks

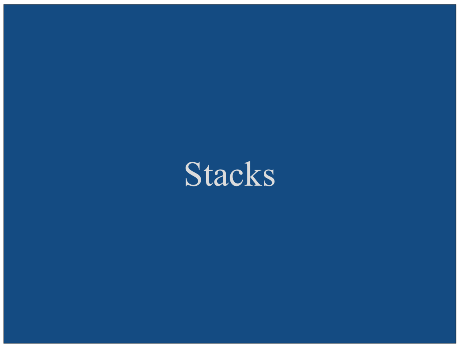
Wednesday, April 2, 2025 12:24 PM

Stacks

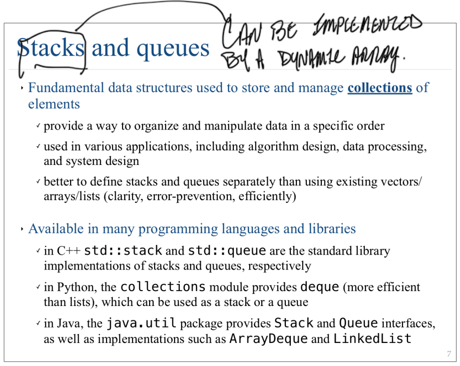


Important C++ topics to review

- Memory model and pointers
- Dynamic memory allocation
- Classes and objects
- References
- Templates
- STL containers



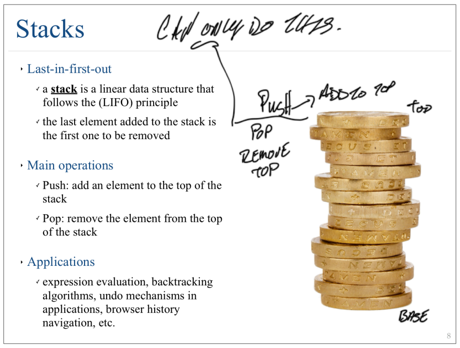
Stacks



Stacks and queues

CAN BE IMPLEMENTED BY A DYNAMIC ARRAY.

- Fundamental data structures used to store and manage **collections** of elements
  - provide a way to organize and manipulate data in a specific order
  - used in various applications, including algorithm design, data processing, and system design
  - better to define stacks and queues separately than using existing vectors/arrays/lists (clarity, error-prevention, efficiently)
- Available in many programming languages and libraries
  - in C++ `std::stack` and `std::queue` are the standard library implementations of stacks and queues, respectively
  - in Python, the `collections` module provides `deque` (more efficient than lists), which can be used as a stack or a queue
  - in Java, the `java.util` package provides `Stack` and `Queue` interfaces, as well as implementations such as `ArrayDeque` and `LinkedList`

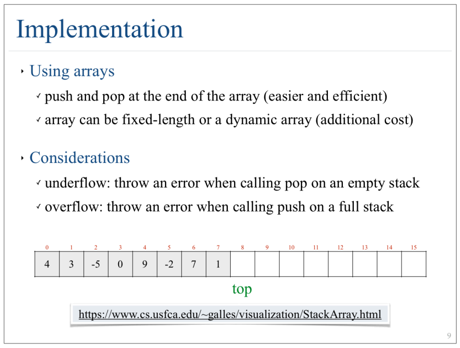


Stacks

CAN ONLY DO THIS.

- Last-in-first-out
  - a **stack** is a linear data structure that follows the (LIFO) principle
  - the last element added to the stack is the first one to be removed
- Main operations
  - Push: add an element to the top of the stack
  - Pop: remove the element from the top of the stack
- Applications
  - expression evaluation, backtracking algorithms, undo mechanisms in applications, browser history navigation, etc.

PUSH → ADD TO TOP
POP
REMOVE TOP
TOP
BASE



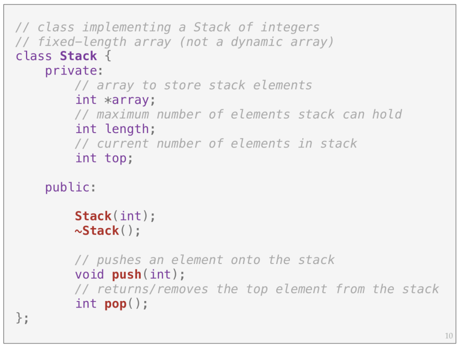
Implementation

- Using arrays
  - push and pop at the end of the array (easier and efficient)
  - array can be fixed-length or a dynamic array (additional cost)
- Considerations
  - underflow: throw an error when calling pop on an empty stack
  - overflow: throw an error when calling push on a full stack

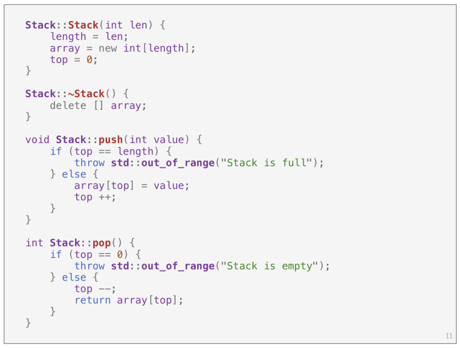
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	3	-5	0	9	-2	7	1								

top

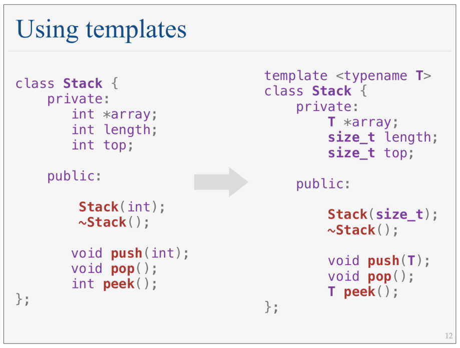
https://www.cs.usfca.edu/~galles/visualization/StackArray.html



```
// class implementing a Stack of integers
// fixed-length array (not a dynamic array)
class Stack {
    private:
        // array to store stack elements
        int *array;
        // maximum number of elements stack can hold
        int length;
        // current number of elements in stack
        int top;

    public:

        Stack(int);
        ~Stack();

        // pushes an element onto the stack
        void push(int);
        // returns/removes the top element from the stack
        int pop();
};
```



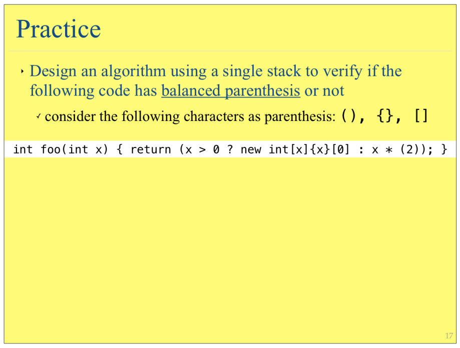
```
Stack::Stack(int len) {
    length = len;
    array = new int[length];
    top = 0;
}

Stack::~Stack() {
    delete [] array;
}

void Stack::push(int value) {
    if (top == length) {
        throw std::out_of_range("Stack is full");
    } else {
        array[top] = value;
        top ++;
    }
}

int Stack::pop() {
    if (top == 0) {
        throw std::out_of_range("Stack is empty");
    } else {
        top --;
        return array[top];
    }
}
```



Using templates

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

    public:

        Stack(int);
        ~Stack();

        void push(int);
        void pop();
        int peek();
};
```

```
template <typename T>
class Stack {
    private:
        T *array;
        size_t length;
        size_t top;

    public:

        Stack(size_t);
        ~Stack();

        void push(T);
        void pop();
        T peek();
};
```

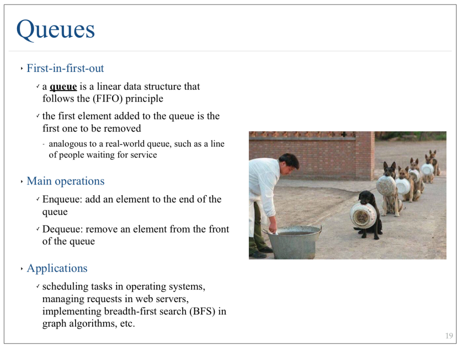


Practice

- Design an algorithm using a single stack to verify if the following code has balanced parenthesis or not
  - consider the following characters as parenthesis: `()`, `{}`, `[]`

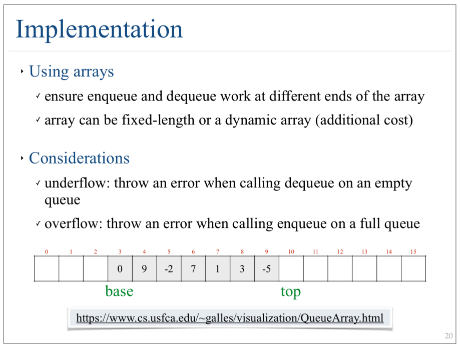
```
int foo(int x) { return (x > 0 ? new int[x]{x}[0] : x * (2)); }
```

Queues



Queues

- First-in-first-out
  - a **queue** 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
- Applications
  - scheduling tasks in operating systems, managing requests in web servers, implementing breadth-first search (BFS) in graph algorithms, etc.

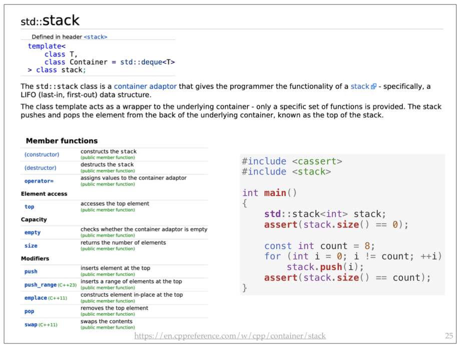


Implementation

- Using arrays
  - ensure enqueue and dequeue work at different ends 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
  - overflow: throw an error when calling enqueue on a full queue

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
			0	9	-2	7	1	3	-5						
			base							top					

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



std::stack

Defined in header `<stack>`

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

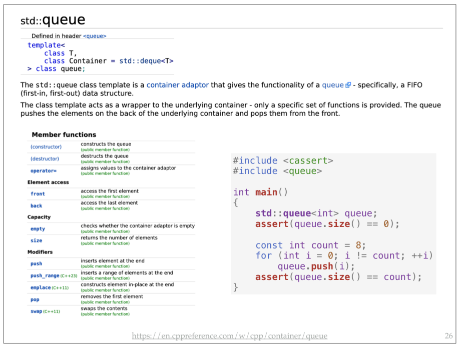
The `std::stack` class is a container adaptor that gives the programmer the functionality of a stack - specifically, a LIFO (last-in, first-out) data structure.

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)	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 container adaptor is empty (public member function)
size	returns the number of elements (public member function)
Modifiers	
push	inserts element at the top (public member function)
push_range (C++23)	inserts a range of elements 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 (C++11)	swaps the contents (public member function)

```
#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);
}
```

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



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 (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 container adaptor is empty (public member function)
size	returns the number of elements (public member function)
Modifiers	
push	inserts element at the end (public member function)
push_range (C++23)	inserts a range of elements 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 (C++11)	swaps the contents (public member function)

```
#include <cassert>
#include <queue>

int main()
{
    std::queue<int> queue;
    assert(queue.size() == 0);

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

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