- FILO — First-In Last-Out
- Implementation as Array or LinkedList

C++: class Stack

First-In Last-Out (“stack”)

```
bool isEmpty(Stack* stack);
bool push(Stack* stack, char data);
double pop(Stack* stack);
double peek(Stack* stack);
```

C++: `class Stack`

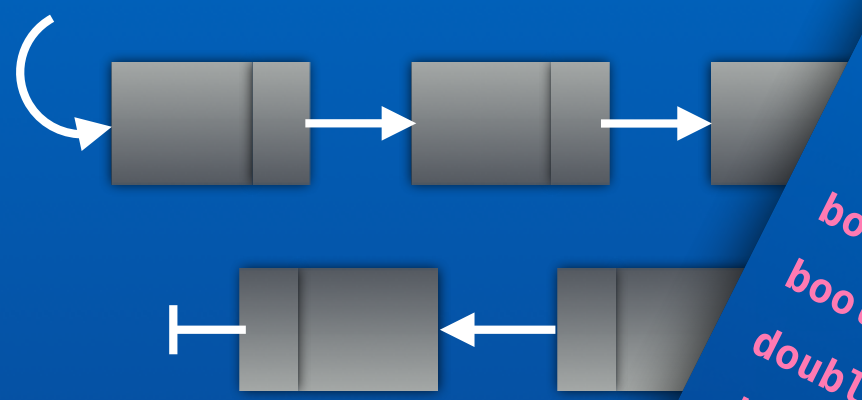
First-In Last-Out (“stack”)

[x,x,x,x,x,x

```
bool isEmpty(Stack* stack);
bool push(Stack* stack, char data);
double pop(Stack* stack);
double peek(Stack* stack);
```

C++: class Stack

First-In Last-Out (“stack”)



```
bool isEmpty(Stack* stack);
bool push(Stack* stack, char data);
double pop(Stack* stack);
double peek(Stack* stack);
```

C++: class Stack

First-In Last-Out (“stack”)

```
bool isEmpty(Stack* stack);
bool push(Stack* stack, char data);
double pop(Stack* stack);
double peek(Stack* stack);
```

C++: class Stack

First-In Last-Out (“stack”)

```
bool isEmpty(Stack* stack);
bool push(Stack* stack, char data);
double pop(Stack* stack);
double peek(Stack* stack);
```

C++: class Stack

First-In Last-Out ("stack")

```
bool isEmpty(Stack* stack);
bool push(Stack* stack, char data);
double pop(Stack* stack);
double peek(Stack* stack);
```

Implemented with Array

C++: class Stack

First-In Last-Out ("stack")

```
bool isEmpty(Stack* stack);
bool push(Stack* stack, char data);
double pop(Stack* stack);
double peek(Stack* stack);
```

Implemented with Array

```
private:
int currentIndex = -1
```

0	1	2	3	4	5

```
isEmpty(stack) == true
peek(stack) == CRASH
```

C++: `class Stack`

First-In Last-Out ("stack")

```
bool isEmpty(Stack* stack);
bool push(Stack* stack, char data);
double pop(Stack* stack);
double peek(Stack* stack);
```

Implemented with Array

```
private:
int currentIndex = -1
```

0	1	2	3	4	5

```
isEmpty(stack) == true
peek(stack) == CRASH
```

```
push(stack, 'g')
```

C++: `class Stack`

First-In Last-Out ("stack")

```
bool isEmpty(Stack* stack);
bool push(Stack* stack, char data);
double pop(Stack* stack);
double peek(Stack* stack);
```

Implemented with Array

```
private:
int currentIndex = -1
```

0	1	2	3	4	5
g					

```
isEmpty(stack) == true
peek(stack) == CRASH
```

```
push(stack, 'g')
```

C++: class Stack

First-In Last-Out ("stack")

```
bool isEmpty(Stack* stack);
bool push(Stack* stack, char data);
double pop(Stack* stack);
double peek(Stack* stack);
```

Implemented with Array

```
private:
int currentIndex = -1
```

0	1	2	3	4	5
g					

```
isEmpty(stack) == true
peek(stack) == CRASH
```

```
push(stack, 'g')
```

C++: `class Stack`

First-In Last-Out ("stack")

```
bool isEmpty(Stack* stack);
bool push(Stack* stack, char data);
double pop(Stack* stack);
double peek(Stack* stack);
```

Implemented with Array

```
private:
int currentIndex = 0
```

0	1	2	3	4	5
g					

```
isEmpty(stack) == false
peek(stack) == 'g'
```

```
push(stack, 'g')
```

C++: `class Stack`

First-In Last-Out ("stack")

```
bool isEmpty(Stack* stack);
bool push(Stack* stack, char data);
double pop(Stack* stack);
double peek(Stack* stack);
```

Implemented with Array

```
private:
int currentIndex = 0
```

0	1	2	3	4	5
g					

```
isEmpty(stack) == false
peek(stack) == 'g'
```

```
push(stack, 'g')
```

C++: class Stack

First-In Last-Out ("stack")

```
bool isEmpty(Stack* stack);
bool push(Stack* stack, char data);
double pop(Stack* stack);
double peek(Stack* stack);
```

Implemented with Array

```
private:
int currentIndex = 1
```

0	1	2	3	4	5
g	a				

```
isEmpty(stack) == false
peek(stack) == 'a'
```

```
push(stack, 'a')
```

C++: class Stack

First-In Last-Out ("stack")

```
bool isEmpty(Stack* stack);
bool push(Stack* stack, char data);
double pop(Stack* stack);
double peek(Stack* stack);
```

Implemented with Array

```
private:
int currentIndex = 1
```

0	1	2	3	4	5
g	a				

```
isEmpty(stack) == false
peek(stack) == 'a'
```

```
push(stack, 'a')
```

C++: class Stack

First-In Last-Out ("stack")

```
bool isEmpty(Stack* stack);
bool push(Stack* stack, char data);
double pop(Stack* stack);
double peek(Stack* stack);
```

Implemented with Array

```
private:
int currentIndex = 2
```

0	1	2	3	4	5
g	a	r			

```
isEmpty(stack) == false
peek(stack) == 'r'
```

```
push(stack, 'r')
```

C++: class Stack

First-In Last-Out ("stack")

```
bool isEmpty(Stack* stack);
bool push(Stack* stack, char data);
double pop(Stack* stack);
double peek(Stack* stack);
```

Implemented with Array

```
private:
int currentIndex = 2
```

0	1	2	3	4	5
g	a	r			

```
isEmpty(stack) == false
peek(stack) == 'r'
```

```
push(stack, 'r')
```

C++: `class Stack`

First-In Last-Out ("stack")

```
bool isEmpty(Stack* stack);
bool push(Stack* stack, char data);
double pop(Stack* stack);
double peek(Stack* stack);
```

Implemented with Array

```
private:
int currentIndex = 3
```

0	1	2	3	4	5
g	a	r	r		

```
isEmpty(stack) == false
peek(stack) == 'r'
```

```
push(stack, 'r')
```

C++: `class Stack`

First-In Last-Out ("stack")

```
bool isEmpty(Stack* stack);
bool push(Stack* stack, char data);
double pop(Stack* stack);
double peek(Stack* stack);
```

Implemented with Array

```
private:
int currentIndex = 3
```

0	1	2	3	4	5
g	a	r	r		

```
isEmpty(stack) == false
peek(stack) == 'r'
```

```
push(stack, 'r')
```

C++: class Stack

First-In Last-Out ("stack")

```
bool isEmpty(Stack* stack);
bool push(Stack* stack, char data);
double pop(Stack* stack);
double peek(Stack* stack);
```

Implemented with Array

```
private:
int currentIndex = 4
```

0	1	2	3	4	5
g	a	r	r	e	

```
isEmpty(stack) == false
peek(stack) == 'e'
```

```
push(stack, 'e')
```

C++: class Stack

First-In Last-Out ("stack")

```
bool isEmpty(Stack* stack);
bool push(Stack* stack, char data);
double pop(Stack* stack);
double peek(Stack* stack);
```

Implemented with Array

```
private:
int currentIndex = 4
```

0	1	2	3	4	5
g	a	r	r	e	

```
isEmpty(stack) == false
peek(stack) == 'e'
```

```
push(stack, 'e')
```

C++: class Stack

First-In Last-Out ("stack")

```
bool isEmpty(Stack* stack);
bool push(Stack* stack, char data);
double pop(Stack* stack);
double peek(Stack* stack);
```

Implemented with Array

```
private:
int currentIndex = 5
```

0	1	2	3	4	5
g	a	r	r	e	t

```
isEmpty(stack) == false
peek(stack) == 't'
```

```
push(stack, 't')
```

C++: class Stack

First-In Last-Out ("stack")

```
bool isEmpty(Stack* stack);
bool push(Stack* stack, char data);
double pop(Stack* stack);
double peek(Stack* stack);
```

Implemented with Array

```
private:
int currentIndex = 5
```

0	1	2	3	4	5
g	a	r	r	e	t

```
isEmpty(stack) == false
peek(stack) == 't'
```

```
push(stack, 't')
```

C++: `class Stack`

First-In Last-Out ("stack")

```
bool isEmpty(Stack* stack);
bool push(Stack* stack, char data);
double pop(Stack* stack);
double peek(Stack* stack);
```

Implemented with Array

```
private:
int currentIndex = 5
```

0	1	2	3	4	5
g	a	r	r	e	t

```
isEmpty(stack) == false
peek(stack) == 't'
```

⚠️ This will crash

```
push(stack, 't')
```

C++: class Stack

First-In Last-Out ("stack")

```
bool isEmpty(Stack* stack);
bool push(Stack* stack, char data);
double pop(Stack* stack);
double peek(Stack* stack);
```

Implemented with Array

```
private:
int currentIndex = 5
```

0	1	2	3	4	5
g	a	r	r	e	t

```
isEmpty(stack) == false
peek(stack) == 't'
```

```
push(stack, 't')
```

C++: class Stack

First-In Last-Out ("stack")

```
bool isEmpty(Stack* stack);
bool push(Stack* stack, char data);
double pop(Stack* stack);
double peek(Stack* stack);
```

Implemented with Array

```
private:
int currentIndex = 5
```

0	1	2	3	4	5
g	a	r	r	e	t

```
isEmpty(stack) == false
peek(stack) == 't'
```

pop(stack)

C++: class Stack

First-In Last-Out ("stack")

```
bool isEmpty(Stack* stack);
bool push(Stack* stack, char data);
double pop(Stack* stack);
double peek(Stack* stack);
```

Implemented with Array

```
private:
int currentIndex = 4
```

0	1	2	3	4	5
g	a	r	r	e	~~t~~

```
isEmpty(stack) == false
peek(stack) == 'e'
```

pop(stack) t

C++: class Stack

First-In Last-Out ("stack")

```
bool isEmpty(Stack* stack);
bool push(Stack* stack, char data);
double pop(Stack* stack);
double peek(Stack* stack);
```

Implemented with Array

```
private:
int currentIndex = 3
```

0	1	2	3	4	5
g	a	r	r	~~e~~	~~t~~

```
isEmpty(stack) == false
peek(stack) == 'r'
```

pop(stack) e

C++: `class Stack`

First-In Last-Out ("stack")

```
bool isEmpty(Stack* stack);
bool push(Stack* stack, char data);
double pop(Stack* stack);
double peek(Stack* stack);
```

Implemented with Array

Implicitly, we mark these items as "garbage," since the `currentIndex` is less than it.

```
private
int cu
```

		2	3	4	5
g	a	r	r	~~e~~	~~t~~

```
isEmpty(stack) == false
peek(stack) == 'r'
```

`pop(stack)` e

C++: class Stack

First-In Last-Out ("stack")

```
bool isEmpty(Stack* stack);
bool push(Stack* stack, char data);
double pop(Stack* stack);
double peek(Stack* stack);
```

Implemented with Array

Implicitly, we mark these items as "garbage," since the `currentIndex` is less than it.

		2	3	4	5
	a	r	r	~~e~~	~~t~~

A benefit of using an array is we don't need to deallocate data, we mark as deleted and overwrite when needed.

Changing an integer value is faster than deallocation.

C++: class Stack

First-In Last-Out ("stack")

```
bool isEmpty(Stack* stack);
bool push(Stack* stack, char data);
double pop(Stack* stack);
double peek(Stack* stack);
```

Implemented with Array

```
private:
int currentIndex = 3
```

0	1	2	3	4	5
g	a	r	r	~~e~~	~~t~~

```
isEmpty(stack) == false
peek(stack) == 'r'
```

pop(stack) e

C++: class Stack

First-In Last-Out ("stack")

```
bool isEmpty(Stack* stack);
bool push(Stack* stack, char data);
double pop(Stack* stack);
double peek(Stack* stack);
```

Implemented with Array

```
private:
int currentIndex = 4
```

0	1	2	3	4	5
g	a	r	r	~~e~~	~~t~~

```
isEmpty(stack) == false
peek(stack) == 'a'
```

```
push(stack, 'a')
```

C++: class Stack

First-In Last-Out ("stack")

```
bool isEmpty(Stack* stack);
bool push(Stack* stack, char data);
double pop(Stack* stack);
double peek(Stack* stack);
```

Implemented with Array

```
private:
int currentIndex = 4
```

0	1	2	3	4	5
g	a	r	r	a	⊥

```
isEmpty(stack) == false
peek(stack) == 'a'
```

```
push(stack, 'a')
```

C++: class Stack

First-In Last-Out ("stack")

```
bool isEmpty(Stack* stack);
bool push(Stack* stack, char data);
double pop(Stack* stack);
double peek(Stack* stack);
```

Implemented with LinkedList

Do you want to use:

- ☐ singly-linked list
- ☐ doubly-linked list

C++: class Stack

First-In Last-Out ("stack")

```
bool isEmpty(Stack* stack);
bool push(Stack* stack, char data);
double pop(Stack* stack);
double peek(Stack* stack);
```

Implemented with LinkedList

Do you want to use:

☑ singly-linked list

☐ doubly-linked list

First-In Last-Out (“stack”)

```
bool isEmpty(Stack* stack);
bool push(Stack* stack, char data);
double pop(Stack* stack);
double peek(Stack* stack);
```

First-In Last-Out (“stack”)

```
bool isEmpty(Stack* stack);
bool push(Stack* stack, char data);
double pop(Stack* stack);
double peek(Stack* stack);
```

Task 1: Implement a Stack using a singly-linked list in C.

First-In Last-Out (“stack”)

```
bool isEmpty(Stack* stack);
bool push(Stack* stack, char data);
double pop(Stack* stack);
double peek(Stack* stack);
```

Task 1: Implement a Stack using a singly-linked list in C.

Hint: Stacks only care about what’s on top.

First-In Last-Out ("stack")

```
bool isEmpty(Stack* stack);
bool push(Stack* stack, char data);
double pop(Stack* stack);
double peek(Stack* stack);
```

Task 1: Implement a Stack using a singly-linked list in C.

Hint: Stacks only care about what's on top.

Task 2: Write Unit Tests for the stack