Goals for today

- The Stack data structure

Key terms

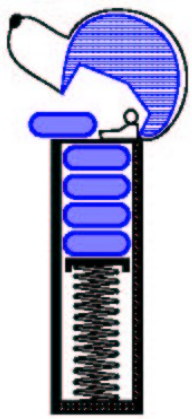
- Key terms
  - Stacks
  - Wrapper class implementation

Stacks

- Two ways of working with data structures
  1. Using a data structure
  2. Implementing a data structure

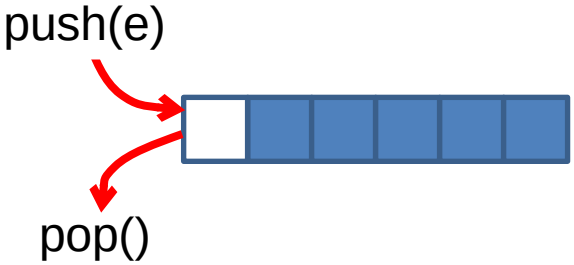
Stacks

- A Stack stores arbitrary objects
- Insertions and deletions follow the last-in first-out scheme
- Think of a spring-loaded plate dispenser



Stack

- Stack (Last-In First-Out)



Stack operations

- Push(x):
  - inserts x on top of stack
- Pop():
  - removes item at top of stack (last inserted element)
- Peek():
  - returns but does not remove item at top of stack
- IsEmpty():
  - returns true if stack has no items
- GetLength():
  - Returns the number of items in the stack

Errors

- In a Stack, operations Pop and Peek cannot be performed if the stack is empty
- Attempting Pop and Peek on an empty stack should be caught
  - In C++, use exceptions

Example

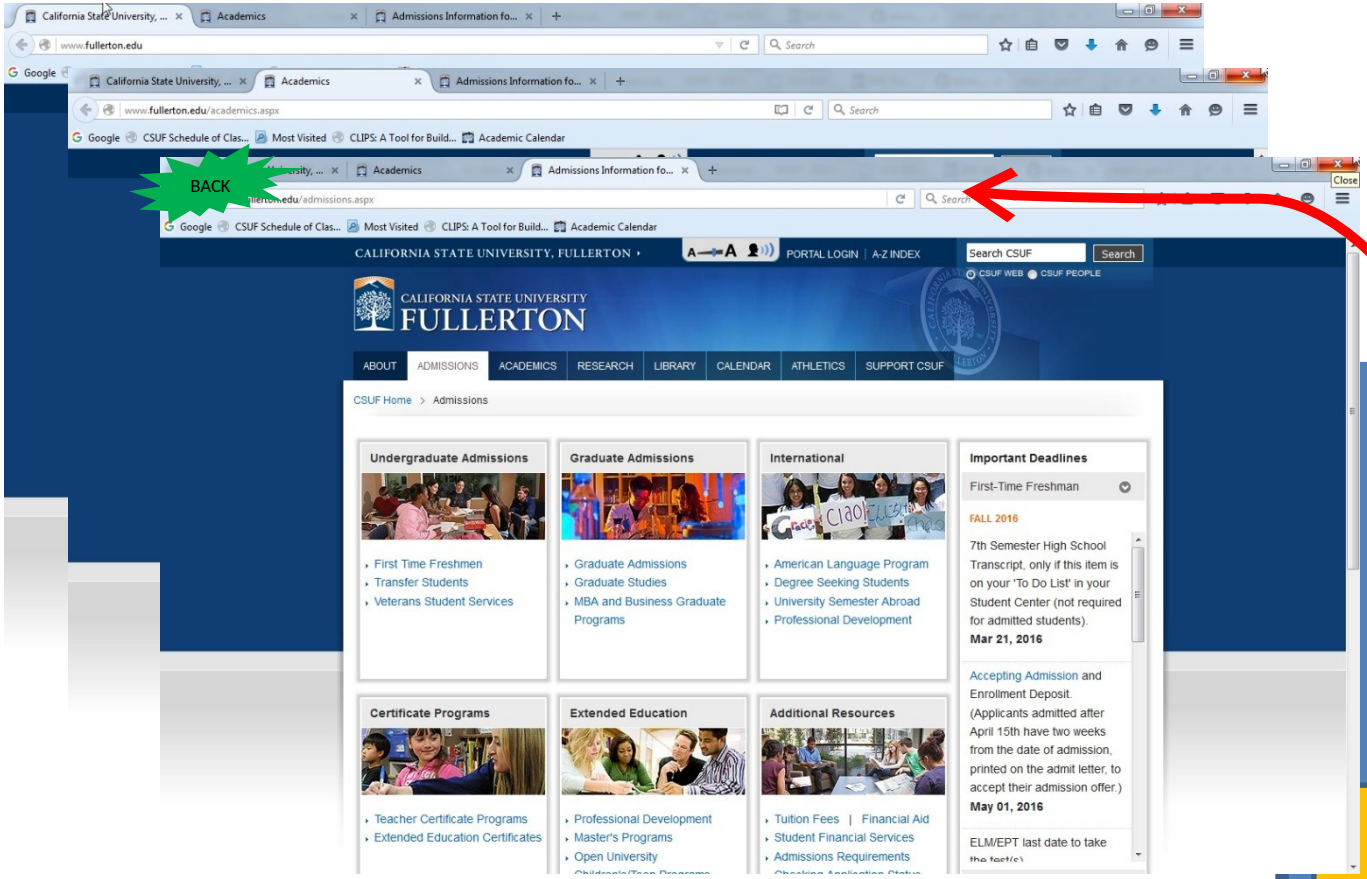
push(5)	--	top: 5]	insert 5 to top of stack
push(3)	--	top: 3, 5]	insert 3 to top of stack
pop()	--	top: 5]	remove top element, 3, from stack
push(7)	--	top: 7, 5]	insert 7 to top of stack
size()	2	top: 7, 5]	return the number of elements in stack
pop()	--	top: 5]	remove top element, 7, from stack
peek()	5	top: 5]	return a reference to top element on stack
pop()	--	top:]	remove top element, 5, from stack
pop()	error	top:]	throw error, nothing to pop
peek()	error	top:]	throw error, nothing to return
IsEmpty()	true	top:]	return true if stack is empty, otherwise, false

Applications of Stacks

- Simple data structure but many applications
  - Page-visited history in a Web browser
  - Undo sequence in a text editor
  - Chain of function calls in the C++ run-time system
  - Computing mathematical expressions
  - Matching tags/brackets in code

Page history in a web-browser

- Clicking a link:
  - Push current page onto stack
- Clicking the Back button:
  - Pop the page from the stack and load to the web-browser



www.fullerton.edu/academics.aspx

www.fullerton.edu/

Abstract Data Types (ADTs)

- An abstract data type (ADT) is an abstraction of a data structure that specifies
  - Data stored
  - Operations on the data
  - Error conditions associated with operations
- Does not talk about implementation

Stacks

- ***Stack*** is a container that stores objects
  - Formally, stack is an ADT
- C++ Stack:

```
template <typename E>
class Stack {
public:
  int getlength() const;   // number of items in stack
  bool isempty() const;   // is the stack empty?
  void push(const E& e); // push element onto stack
  void pop(); // remove the top element
  E& peek(); // return the top element (but don't remove it)
  // must implement these methods!
};
```

Implementing stacks

1. Array
2. Linked list

Code on GitHub:

https://github.com/CSUF-CPSC-131-Spring2019/Data-Structures-Code/

Has both implementations (.h files) and simple main programs to test them (.cpp files)

Wrapper class implementation

- Linked lists
  - "Wrap a stack around a singly linked list"
- Wrapper class
  - Stores a linked list as a private member
  - Translates public stack methods to linked list methods

Stacks Implemented with Singly Linked List

Stack as a Linked List

- Stack can be implemented with a singly linked list
- The top element is stored at the first node of the list
- The space used is $O(n)$ and each operation of the Stack ADT takes $O(1)$ time

C++ STL implementation

- C++ Language Library
- Contains highly optimized implementations of commonly used data structures
  - Including stacks

- http://www.cplusplus.com/reference/stack/stack/
- http://www.cplusplus.com/reference/stack/stack/pop/

std::stack

ZyBook	std::stack
Push	push()
Pop	pop()
Peek	top()
GetLength	size()
IsEmpty	empty()

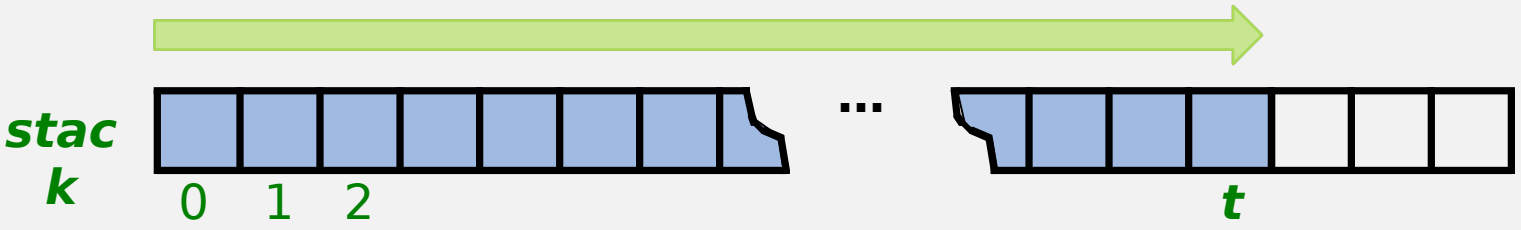
```
#include <stack>

int main() {
   std::stack<int> ds;
   ds.push(10);
   ds.push(20);
   ds.pop();
   cout << ds.top();
}
```

Simple Array-Based Stack Implementation

Array-based Stack – Size & Pop

- A simple way of implementing the Stack ADT uses an array
- We add elements from left to right
- A variable keeps track of the index of the top element

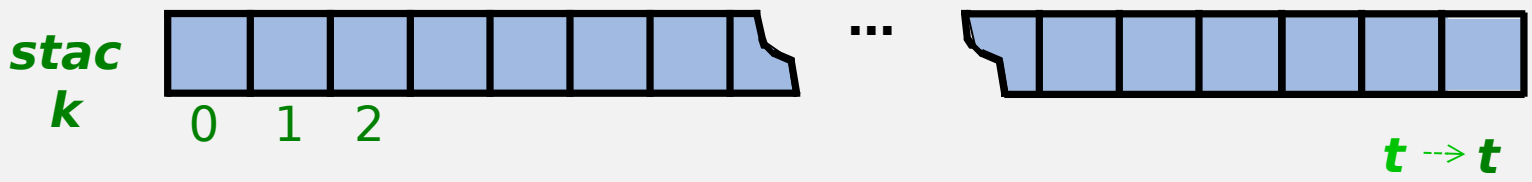
```
Algorithm getlength()
    return t + 1

Algorithm pop()
    if isempty() then
        throw exception
    else
        t ← t - 1
```



Array-based Stack - Push

- The array storing the stack elements may become full
- A push operation will then throw an exception
  - Limitation of the fixed size array implementation
  - Not limitation of the Stack ADT

```
Algorithm push(e)
    if t == capacity then
        throw exception
    else {
        t ← t + 1
        stack[t] ← e
    }
```



Array-based Stack Implementation

```
template <typename E>
class Stack {
enum { DEF_CAPACITY = 100 }; // default stack capacity
public:
   // constructor from capacity (also a default constructor)
   Stack(int cap=DEF_CAPACITY) {
    S = new E[cap];
    capacity = cap;
    t = -1;
   }

   int getlength() const       // number of items in the stack
      { return (t + 1); }
   bool isempty() const      // is the stack empty?
      { return (t < 0); }

   void push(const E& e);  // push element onto stack
   void pop();               // pop the stack
   E& peek() const;     // get the top element

private:
   E* S;           // dynamically allocated array of stack elements
   int capacity;       // stack capacity
   int t;              // index of the top of the stack
};
```

Array-based Stack Implementation (cont)

```
// push element onto the stack
template <typename E>
void Stack<E>::push(const E& e) {
   if (getlength() == capacity)
       throw length_error("Push to full
stack");
   S[++t] = e;
}

// pop the stack
template <typename E>
void Stack<E>::pop() {
   if (isempty())
      throw length_error("Pop from empty
stack");
   --t;
}

// get the top element
template <typename E>
const E& Stack<E>::peek() const {
   if (isempty())
      throw length_error("Top of empty
stack");
   return S[t];
}
```

Performance and Limitations

- Performance
  - Let ***n*** be the number of elements in the stack
  - The space used is ***O***(***n***)
  - Each operation runs in time ***O***(1)
- Limitations
  - The maximum size of the stack must be defined a priori and cannot be changed
  - Trying to push a new element into a full stack causes an implementation-specific exception

Operation	Time
size	O(1)
empty	O(1)
push(e)	O(1)
pop	O(1)
top	O(1)