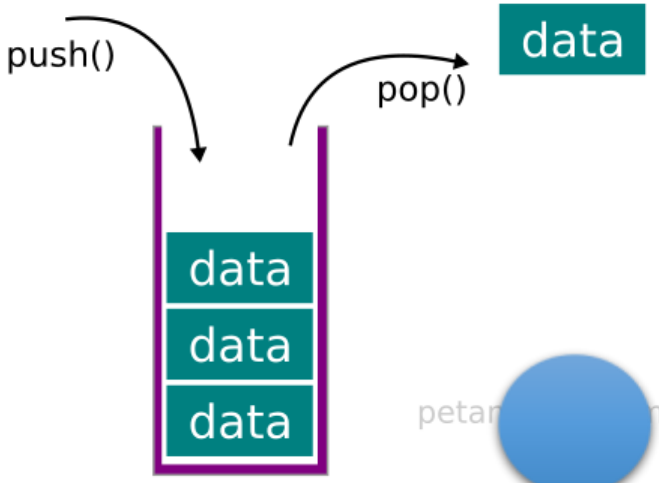
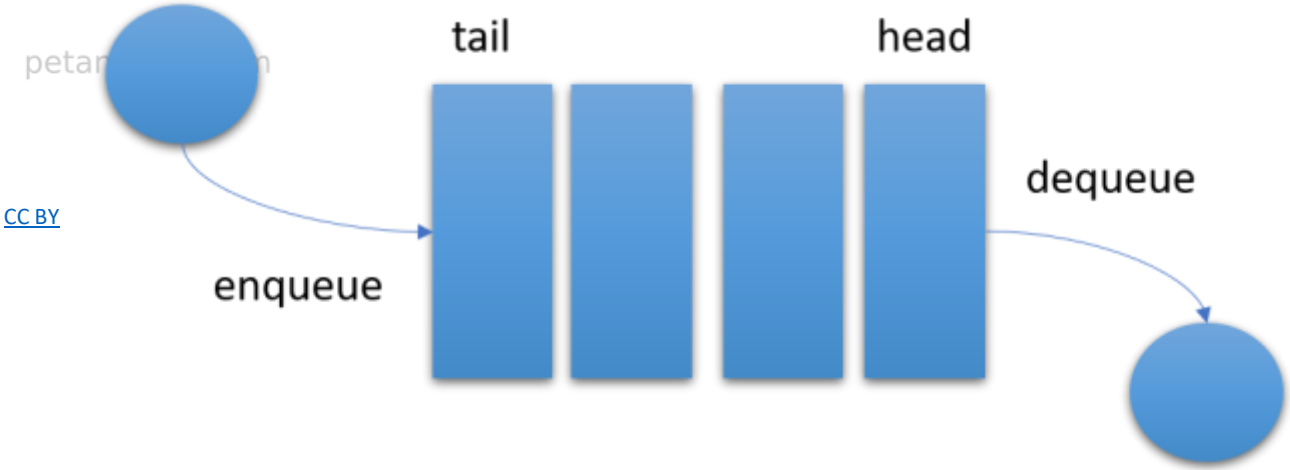
Chapter 9: Stacks





Topics

18.1 Introduction to the Stack ADT

18.2 Dynamic Stacks

18.3 The STL **`stack`** Container

18.4 Introduction to the Queue ADT

18.5 Dynamic Queues

18.6 The STL **`deque`** and **`queue`** Containers

18.7 Eliminating Recursion

18.1 Introduction to the Stack ADT

- Stack: a LIFO (last in, first out) data structure
- Examples:
  - plates in a cafeteria serving area
  - return addresses for function calls

Stack Basics

- Stack is usually implemented as a list, with additions and removals taking place at one end of the list
- The active end of the list implementing the stack is the top of the stack
- Stack types:
  - Static – fixed size, often implemented using an array
  - Dynamic – size varies as needed, often implemented using a linked list

Stack Operations and Functions

Operations:

- push: add a value at the top of the stack
- pop: remove a value from the top of the stack

Boolean function:

- isEmpty: true if the stack currently contains no elements

Static Stack Implementation

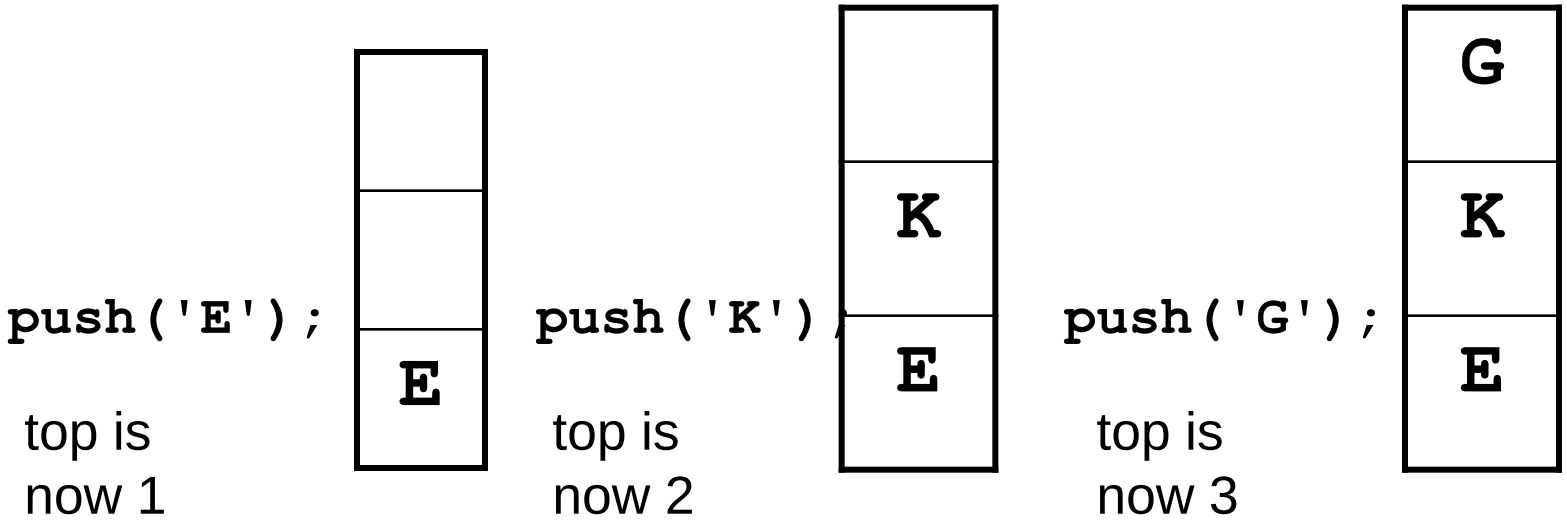
- Uses an array of a fixed size
- Bottom of stack is at index 0. A variable called top tracks the current top of the stack

```
const int STACK_SIZE = 3;
char s[STACK_SIZE];
int top = 0;
```

top is where the next item will be added

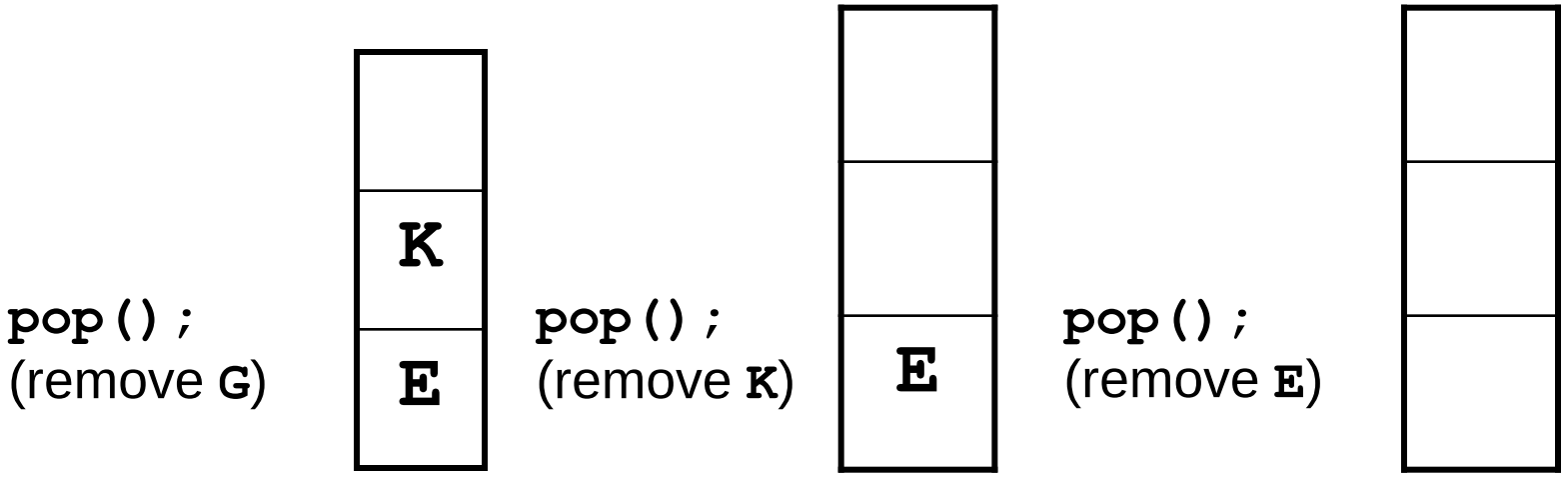
Array Implementation Example

This stack has max capacity 3, initially top = 0 and stack is empty.



Stack Operations Example

After three pops, **`top`** is **`0`** and the stack is empty



Class Implementation Using an Array

```
class STACK
  {
private:
     char *s;  // for the array
     int capacity, top;
 public:
     void push(char x);
     void pop(char &x);
     bool isEmpty();
     STACK(int stackSize);
     ~STACK()
  };
```

Class Implementation Using an Array

Use exception classes as members of the STACK class to signal that an underflow or overflow condition has occurred:

```
class Overflow {};
class Underflow {};
```

Class Implementation Using an Array

To check if the stack is empty:

```
bool isEmpty()
 {
   if (top == 0)
       return true;
  else return false;
 }
```

Class Implementation Using an Array

To add an item to the stack

```
void push(char x)
 {
  if (top==capacity)
    throw STACK::Overflow();
  s[top] = x;
  top++;
 }
```

Class Implementation Using an Array

To remove an item from the stack

```
void pop(char &x)
 {
  if (isEmpty())
   throw STACK::Underflow();
  top--;
  x = s[top];
 }
```

Exceptions from Stack Operations

- The preceding example uses exception classes to handle cases where an attempt is made to push onto a full stack (overflow) or to pop from an empty stack (underflow)
- Programs that use **`push`** and **`pop`** operations should do so from within a **`try`** block.
- **`catch`** block(s) should follow the **`try`** block to interpret what occurred and to inform the user.

18.2 Dynamic Stacks

- The storage for a stack can be implemented as a linked list
- There is no need to indicate the initial capacity of the stack. It can grow and shrink as necessary.
- It can't ever be full as long as memory is available, so there is no need to test for overflow.
- Testing for underflow (empty stack) is still needed.

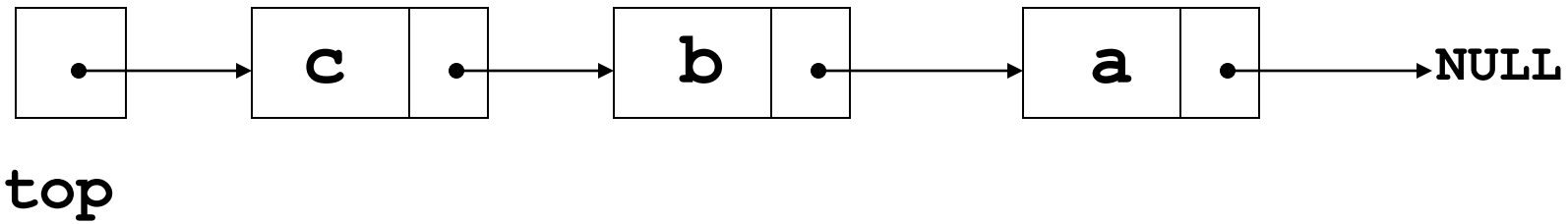
Dynamic Linked List Implementation

- Define a class for a dynamic linked list
- Within the class, define a private member class for dynamic nodes in the list
- Define a node pointer to the beginning of the linked list, which will serve as the top of the stack

Linked List Implementation

A linked stack after three push operations:

```
push('a'); push('b'); push('c');
```



Operations on a Linked Stack

Check if stack is empty:

```
bool isEmpty()
{
  if (top == NULL)
    return true;
  else
    return false;
}
```

Operations on a Linked Stack

Add a new item to the stack

```
void push(char x)
{
  top = new LNode(x, top);
}
```

Operations on a Linked Stack

Remove an item from the stack

```
void pop(char &x)
{
  if (isEmpty())
    throw STACK::Underflow();
  x = top->value;
  LNode *oldTop = top;
  top = top->next;
  delete oldTop;
}
```

.