Two Special List Structures (in C)

Stack
▶ Last-in First-out (lifo)
▶ Key operations are push, pop, top and empty

Queue
▶ First-in First-out (fifo)
▶ Key operations are enqueue, dequeue, front, empty

Both Stack and Queue are simpler to implement than a general linked list.

Struct for Stacks

Each “cell” of the structure carries the following two things:
- A piece of data
- A pointer to the “cell” below it in the stack

We can “package” this as a recursive struct declaration.

```c
struct elem { /* structure of an element on the stack */
  data d;  /* stack: just one piece of data, and */
  struct elem *next; /* pointer to 'next' cell */
};
typedef struct elem elem;
```
```c
#define EMPTY 0
#define FULL 10000
typedef char data;
typedef enum {false, true} boolean;
struct elem { /* structure of an element on the stack: */
    data d; /* just one piece of data, and pointer */
    struct elem *next; /* to 'next' cell */
};
typedef struct elem elem;
typedef struct { /* Stack is just one element (the 'top' */
    int count; /* one) plus also a count of items in */
    elem *top; /* entire Stack. */
} stack;
```

```c
void initialize(stack *stk);
void push(data d, stack *stk);
data pop(stack *stk);
data top(const stack *stk);
boolean empty(const stack *stk);
boolean full(const stack *stk);
```

```c
void push(data d, stack *stk) {
    elem *p = malloc(sizeof(elem));
    (p).d = d;
    (p).next = (*stk).top;
    (*stk).top = p;
    (*stk).count++;
}
```
Precedence of * vs .

Our push function (and other functions) takes as parameter a pointer to a stack structure. The . operator (used to access a part of a struct) has higher precedence than the * operator (used to de-reference a pointer). For this reason, in the code above,

we need the parentheses in statements

\((p).d = d;\)
\((p).next = (stk).top;\)

So we usually use C’s abbreviation \(-\rightarrow:\)

\(x\rightarrow y\) means \((x).y\)
And if you’re doing ‘pointer chasing’ (often considered bad style),
\(x\rightarrow y\rightarrow z\) means \((x\rightarrow y)\rightarrow z\) means \(((x).y).z\)

push again

```c
void push(data d, stack *stk) {
    elem *p;
    p = malloc(sizeof(elem));
    p->d = d;
    p->next = stk->top;
    stk->top = p;
    stk->count++;
}
```

initialize

```c
void initialize(stack *stk) {
    stk->count = 0;
    stk->top = NULL;
}
```

pop - stack is shrinking

What happens if we pop an empty stack? Chaos and despair! So we’ll use assert to crash the program right now.

```c
data pop(stack *stk) {
    data d;
    elem *p;
    assert(stk->count > 0);
    p = stk->top;
    d = p->d;
    stk->top = p->next;
    stk->count--;
    free(p);
    return d;
}
```

pop again
Example of applying Stack

Reverse a string
- Go through the string, push-ing each character onto the stack.
- Now pop each item off the stack, direct onto standard output.
DEMO!!!