CS 261 Lab #3

RPN calculators and writing test harnesses
Add 5 to 10, then multiply by 2
Add 5 to 10, then multiply by 2

(5 + 10) * 2
Add 5 to 10, then multiply by 2

\[(5 + 10) \times 2\]
Add 5 to 10, then multiply by 2

(5 + 10) * 2
Add 5 to 10, then multiply by 2

\[(5 + 10) \times 2\]

**infix**

**operator**

**operands**
Add 5 to 10, then multiply by 2

\[(5 + 10) \times 2\]

\[5 10 + 2 *\]
Add 5 to 10, then multiply by 2
Add 5 to 10, then multiply by 2
Add 5 to 10, then multiply by 2

Infix: 
\[(5 + 10) \times 2\]

Postfix: 
\[5\;10\;+\;2\;\star\]
RPN calculators use **postfix** notation
RPN calculators use *postfix* notation

The *stack* data type is perfectly suited to postfix notation
RPN calculators use **postfix** notation

The **stack** data type is perfectly suited to postfix notation

Operands are **pushed** onto the stack
RPN calculators use **postfix** notation

The **stack** data type is perfectly suited to postfix notation

Operands are **pushed** onto the stack

Operators **pop** the operands off, then **push** the result back onto the stack
5 \times (10 + 2)

Stack
5 \times (10 + 2)

Push 5

Top of the stack

Stack

5
5 \times 10 + 2

Stack

Push 5
Push 10

Top of the stack
5 10 + 2 *

Stack

Top of the stack

Push 5
Push 10
Push 5
Push 10
Pop

5 10 + 2 *
Push 5
Push 10
Pop
Pop

5 10 + 2 *
Push 5
Push 10
Pop
Pop
$5 + 10 = 15$
5 10 + 2 *

Stack

Top of the stack

15

Push 5
Push 10
Pop
Pop

5 + 10 = 15
Push 15
\[5 + 10 = 15\]
(5 + 10) + 2

Stack:

- Top of the stack
- 15
- 2

Operations:
- Push 5
- Push 10
- Pop
- Pop
- Push 15
- Push 2
- 5 + 10 = 15
5 10 + 2

Top of the stack

Stack

2
15

Push 5
Push 10
Pop
Pop
5 + 10 = 15
Push 15
Push 2
5 10 + 2

Stack

Top of the stack

15

5 + 10 = 15

Push 5
Push 10
Pop
Pop
Push 15
Push 2
Pop
5 10 + 2

Stack

Top of the stack

Push 5
Push 10
Pop
Pop

5 + 10 = 15
Push 15
Push 2
Pop
Pop
5 10 + 2 *

Stack

Push 5
Push 10
Pop
Pop
5 + 10 = 15
Push 15
Push 2
Pop
Pop
15 * 2 = 30

Top of the stack
5 10 + 2 *

Top of the stack

Stack

Push 5
Push 10
Pop
Pop
5 + 10 = 15
Push 15
Push 2
Pop
Pop
15 * 2 = 30
Push 30
5 10 + 2 *

Stack

Top of the stack

Push 5
Push 10
Pop
Pop
5 + 10 = 15
Push 15
Push 2
Pop
Pop
15 * 2 = 30
Push 30

30
Push 5
Push 10
Pop
Pop

5 + 10 = 15
Push 15
Push 2
Pop
Pop

15 * 2 = 30
Push 30
Top
Push 5
Push 10
Pop
Pop
Push 15
Push 2
Push 30
Top

We know the correct output of these commands (and what the stack should look like at each step)
We know the correct output of these commands (and what the stack should look like at each step)

This is what we need to build a test harness
Push 5
Push 10
Pop
Pop
Pop
Push 15
Push 2
Push 2
Pop
Pop
Pop
Pop
Push 30
Top
printf(“Creating a stack\n”);
struct IntStack *s = _createStack();
printf("Creating a stack\n");
struct IntStack *s = _createStack();

printf("Pushing 5 onto the stack: ");
push(s, 5);

Push 5
Push 10
Pop
Pop
Push 15
Push 2
Pop
Pop
Push 30
Top

Push 5 onto the stack: 
printf(“Creating a stack\n”);
struct IntStack *s = _createStack();

printf(“Pushing 5 onto the stack: ”);
push(s, 5);
if (top(s) == 5)
    printf(“success.\n”);
else
    printf(“FAIL.\n”);
printf("Creating a stack\n");
struct IntStack *s = _createStack();
printf("Pushing 5 onto the stack: ");
push(s, 5);
if (top(s) == 5)
    printf("success.\n");
else
    printf("FAIL.\n");
printf("Creating a stack\n");
struct IntStack *s = _createStack();

printf("Pushing 5 onto the stack: ");
push(s, 5);
if (top(s) == 5)
    printf("success.\n");
else
    printf("FAIL.\n");

printf("Pushing 10 onto the stack: ");
push(s, 10);
if (top(s) == 10)
    printf("success.\n");
else
    printf("FAIL.\n");

Push 5
Push 10
Pop
Pop
Push 15
Push 2
Pop
Pop
Push 30
Top
printf("Creating a stack\n");
struct IntStack *s = _createStack();

printf("Pushing 5 onto the stack: ");
push(s, 5);
if (top(s) == 5)
    printf("success.\n");
else
    printf("FAIL.\n");

push(s, 10);
if (top(s) == 10)
    printf("success.\n");
else
    printf("FAIL.\n");
When you implement a data type, be sure to test **each method** with a range of inputs and assumptions.
When you implement a data type, be sure to test **each method** with a range of inputs and assumptions

What happens when you…
When you implement a data type, be sure to test **each method** with a range of inputs and assumptions

What happens when you...

... push() three values onto a stack, pop() twice, then push() another value?
When you implement a data type, be sure to test **each method** with a range of inputs and assumptions.

What happens when you…

... push() three values onto a stack, pop() twice, then push() another value?

... call top() on an empty stack?
When you implement a data type, be sure to test each method with a range of inputs and assumptions.

What happens when you...

... push() three values onto a stack, pop() twice, then push() another value?

... call top() on an empty stack?

... call pop() on an empty stack?
1) Practice using an RPN calculator
   http://www.kronenberg.org/rpn/

2) Finish writing a test harness we started for you.
   http://dropline.net/cs261/lab3