A+ Computer Science

STACKS

What is a stack?

What is a stack?

A stack is a group of items all of the same type where items are added to the top of the stack and removed from the top.

Stacks work in a LIFO manner.

What is a stack?

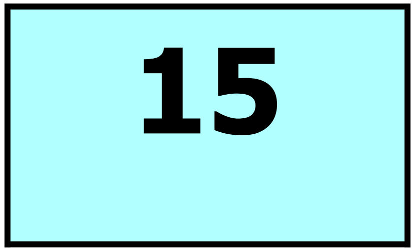
```
Stack<Integer> stack;
stack = new Stack<Integer>();
```

stack will only store Integer values.

What is a stack?

```
stack.push(15);
```

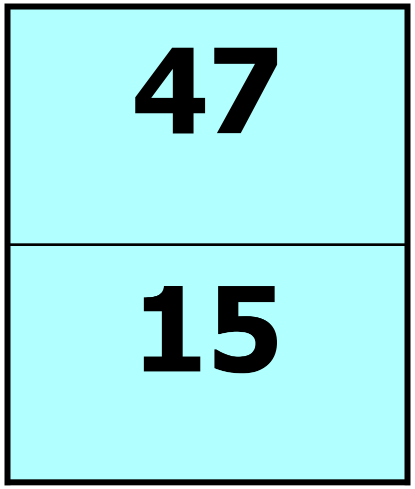
push adds an item to the stack.



What is a stack?

```
stack.push(47);
```

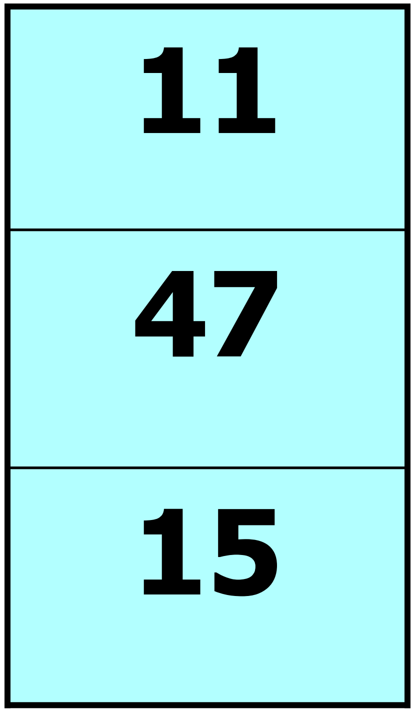
push adds an item to the stack.



What is a stack?

```
stack.push(11);
```

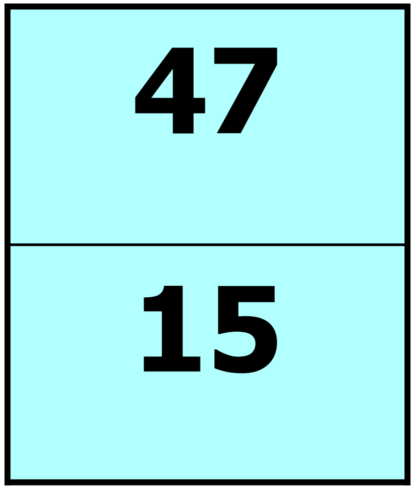
push adds an item to the stack.



What is a stack?

```
stack.pop();
```

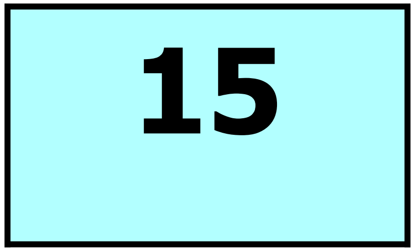
pop removes an item from the stack.



What is a stack?

```
stack.pop();
```

pop removes an item from the stack.



Stack methods

Stack
frequently used methods

Name	Use
push(x)	adds item x to the stack
add(x)	adds item x to the stack
pop()	removes and returns an item
peek()	returns the top item with no remove
size()	returns the # of items in the stack
isEmpty()	checks to see if the stack is empty

```
import  java.util.Stack;
```

push() method

```
Stack<Integer> s;
s = new Stack<Integer>();
s.push(88);
s.push(23);
s.push(11);
out.println(s);
```

OUTPUT

```
[88, 23, 11]
```

stackpush.java

pop() method

```
Stack<Integer> s;
s = new Stack<Integer>();
s.push(88);
s.push(23);
s.push(11);
s.pop();
out.println(s);
```

OUTPUT

```
[88, 23]
```

stockpop.java

push/pop method

```
Stack<Integer> s;
s = new Stack<Integer>();
s.push(88);
s.push(23);
s.push(11);
s.pop();
s.pop();
out.println(s);
```

OUTPUT

```
[88]
```

stackpushpop.java

peek() method

```
Stack<Integer> s;
s = new Stack<Integer>();
s.push(88);
s.push(23);
s.push(11);
out.println(s.peek());
out.println(s);
```

OUTPUT

```
11
[88, 23, 11]
```

stackpeep.java

isEmpty() method

```
Stack<Integer> s;
s = new Stack<Integer>();
s.push(88);
s.push(23);
s.push(11);
while(!s.isEmpty())
{
  out.println(s.pop());
}
```

OUTPUT

```
11
23
88
```

stackisempty.java

Stack Algorithms

Expression Tester

Expressions are made up of values and symbols. Many symbols come in pairs.

( )

{ }

A stack can be used to match up opening and closing symbols.

Expression Tester

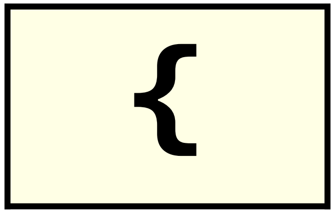
(( ))   is a valid expression

{ (})   is an invalid expression

Open and closing symbol pairs have to occur in the proper sequence.

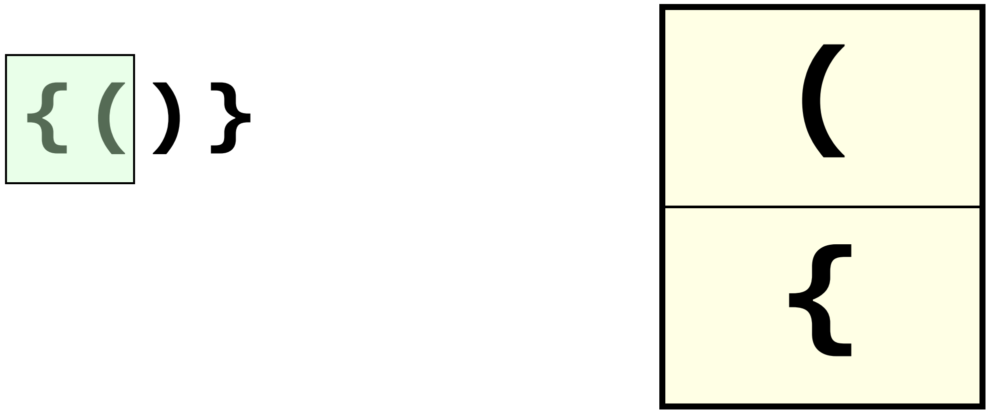
Expression Tester

{ ( ) }



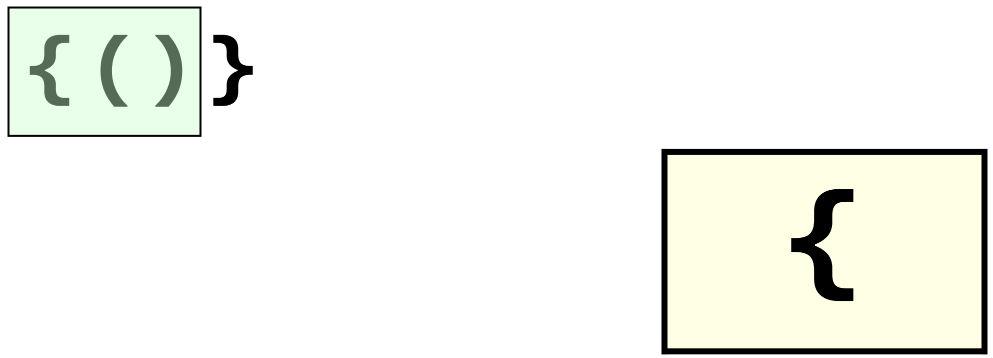
Push { onto the stack.

Expression Tester



Push (onto the stack.

Expression Tester



A close) was encountered. Pop the top symbol off the stack and see if it matches.

Expression Tester

{ ( ) }

{

A close } was encountered. Pop the top symbol off the stack and see if it matches.

Expression Tester

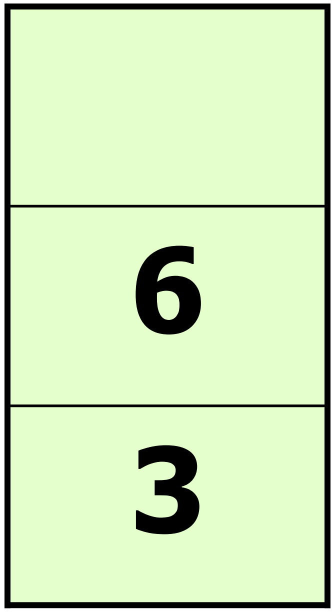
```
{ ( ) }
```

All symbols have been processed. All symbols matched up and the stack is empty. The expression is valid.

Stack Algorithms

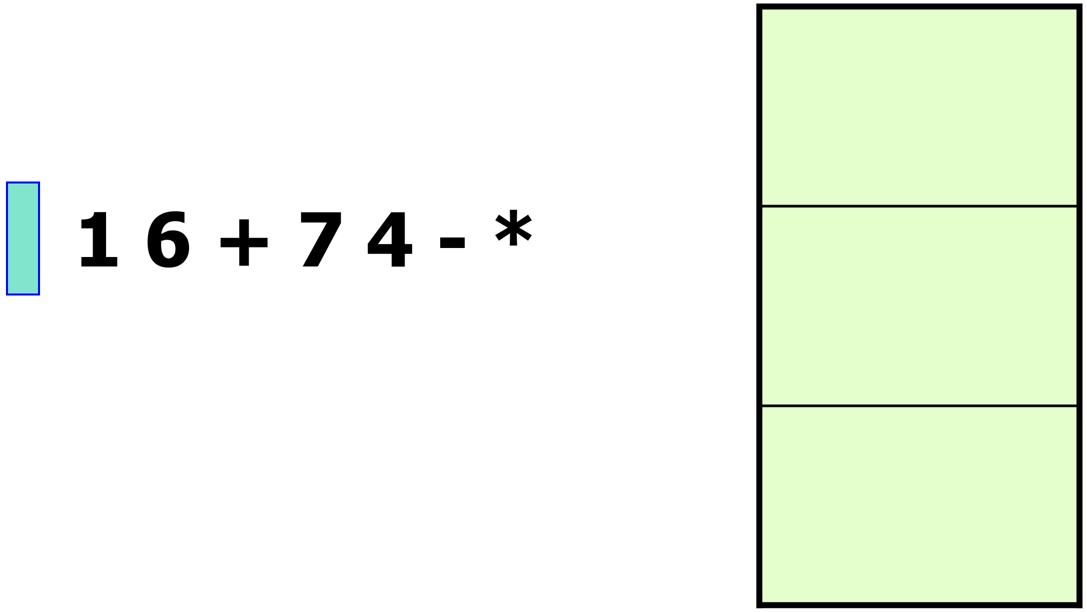
Postfix Solver

3 6 + = 9



Stacks work great for solving many types of expressions. Postfix expressions are well suited for solutions using stacks.

Postfix Solver



You would never need more than one stack!

Postfix Solver

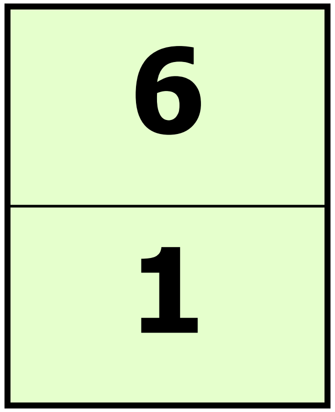
1 6 + 7 4 - *

$\boxed{1}$

Get the 1. 1 is a digit and is pushed on the stack.

Postfix Solver

1 6 + 7 4 - *



Next, get the 6. 6 is a digit and is pushed on the stack.

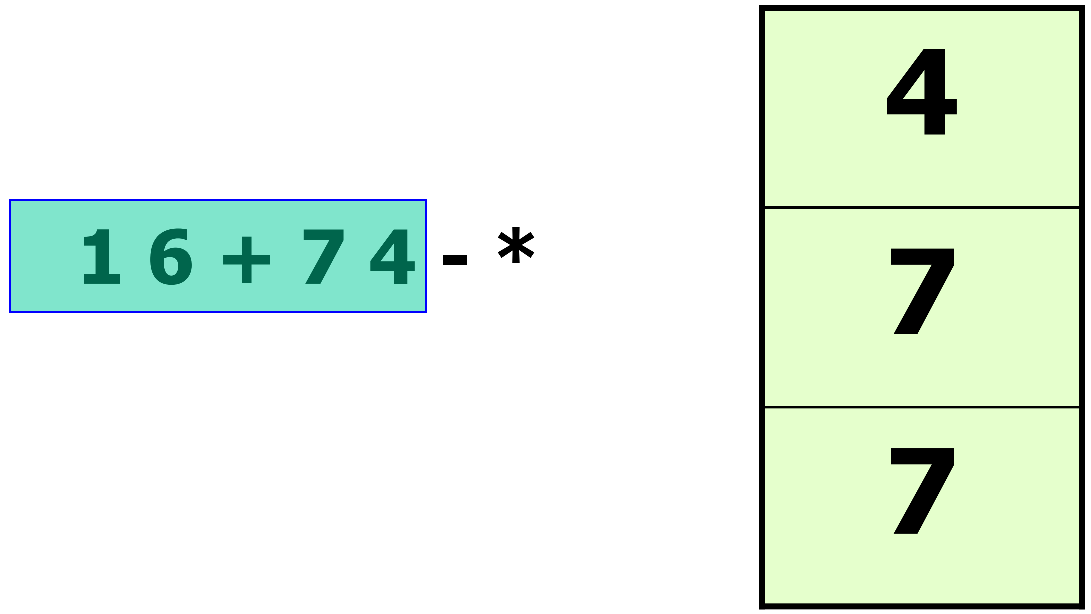
Postfix Solver

1 6 + 7 4 - *

7

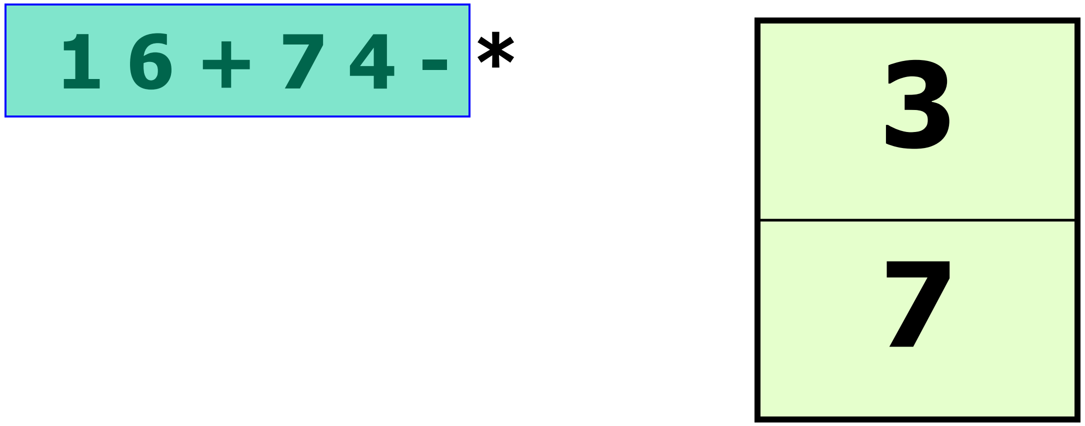
Get the +. + is an operator.
Pop 2 digits off the stack. Push 1 + 6 onto the stack.

Postfix Solver



Get 7 and 4 and push both on the stack.

Postfix Solver



Get the -. Pop 2 digits. Push 7 – 4 onto the stack.

Postfix Solver

1 6 + 7 4 - *

21

Get the *. Pop 2 digits and push the result.

Work on Programs!

Crank

Some Code!

A+ Computer Science

STACKS

A+ Computer Science
Computer Science Curriculum Solutions