STACK

Abstract Data Type

STACK ADT

A List with a restriction:

STACK ADT

`insert` and `delete` can be performed in only one position:
end of the list called TOP.

STACK ADT

LIFO

Last
In,
First
Out

NOW
CRISPIER
Pringles
CHEDDAR CHEESE
Artificially Flavored
Super Stack

STACK ADT

TOS

Top of Stack

STACK ADT Operations

push

equivalent to

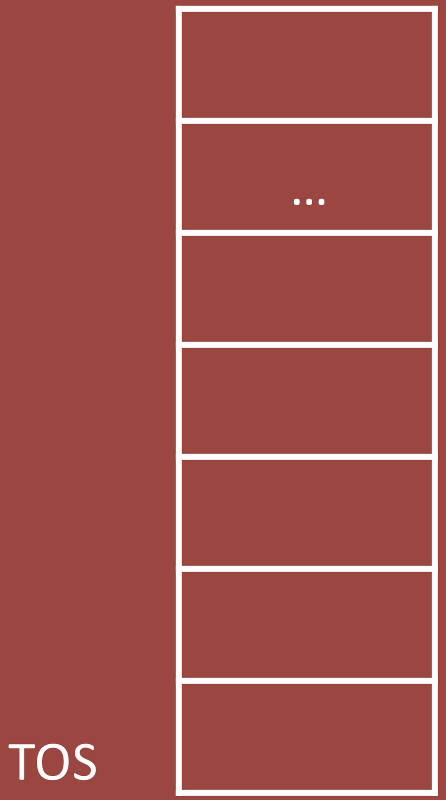
insert

STACK ADT Operations

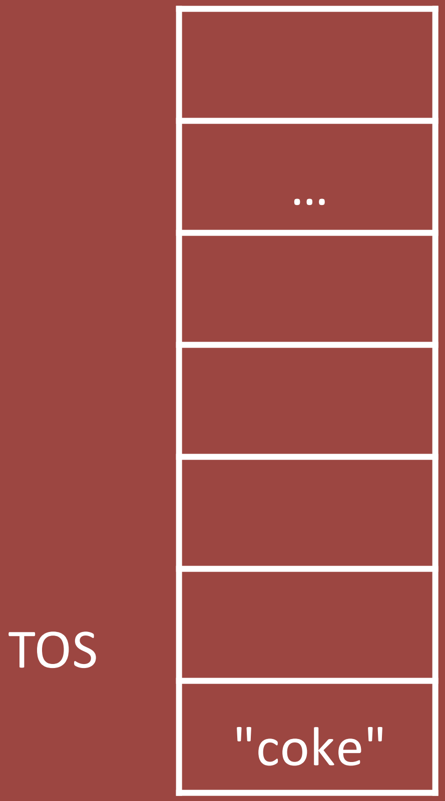
pop

equivalent to

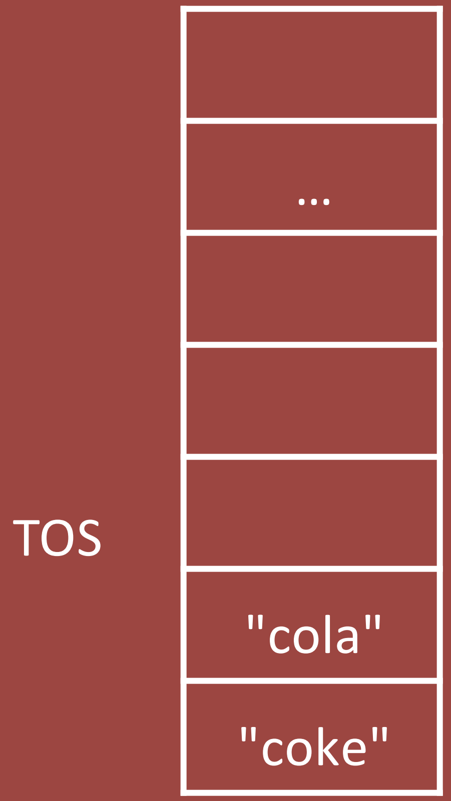
delete



```
push("coke");
push("cola");
x = pop();
push("pepsi");
x = pop();
x = pop();
```



```
push("coke");
push("cola");
x = pop();
push("pepsi");
x = pop();
x = pop();
```



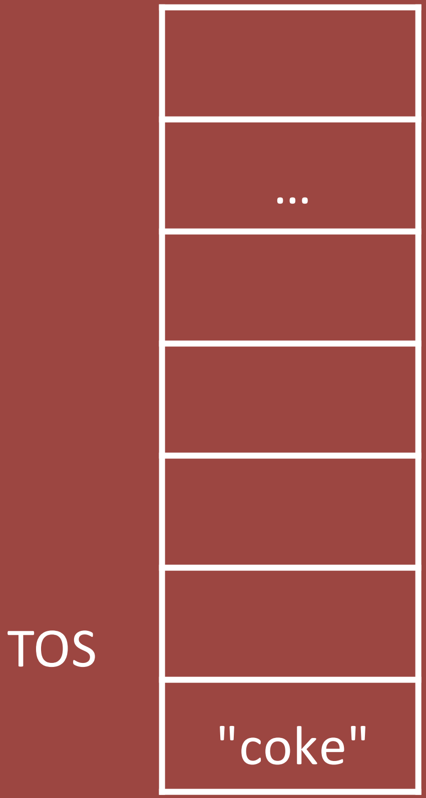
TOS

...

"cola"

"coke"

```
push("coke");
push("cola");
x = pop();
push("pepsi");
x = pop();
x = pop();
```

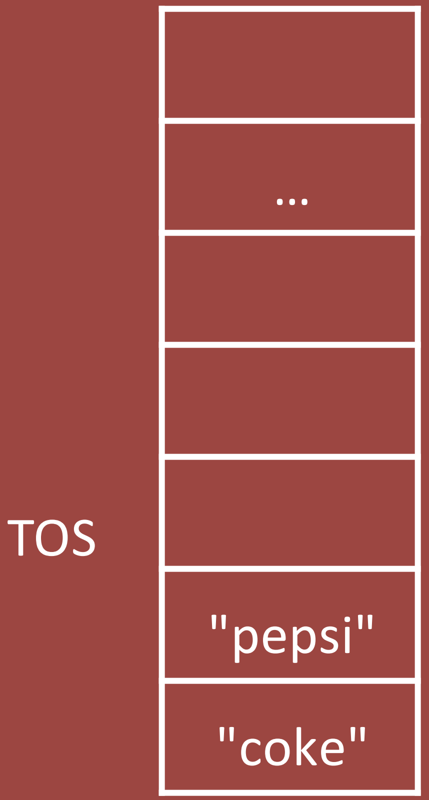


…

TOS

"coke"

```
push("coke");
push("cola");
x = pop();
push("pepsi");
x = pop();
x = pop();
```



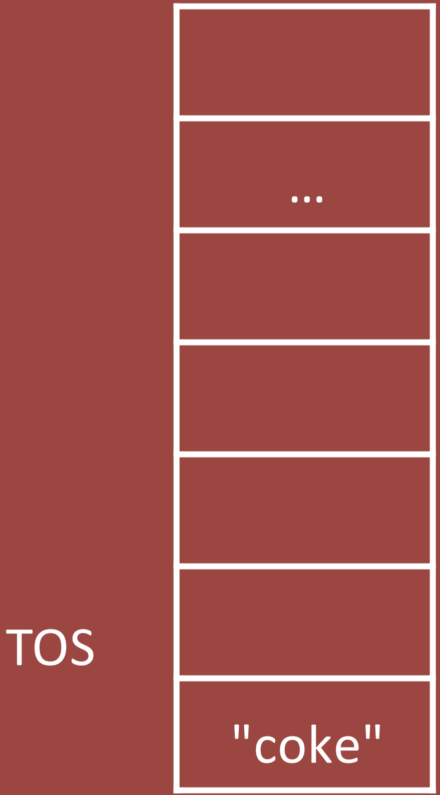
TOS

…

"pepsi"

"coke"

```
push("coke");
push("cola");
x = pop();
push("pepsi");
x = pop();
x = pop();
```

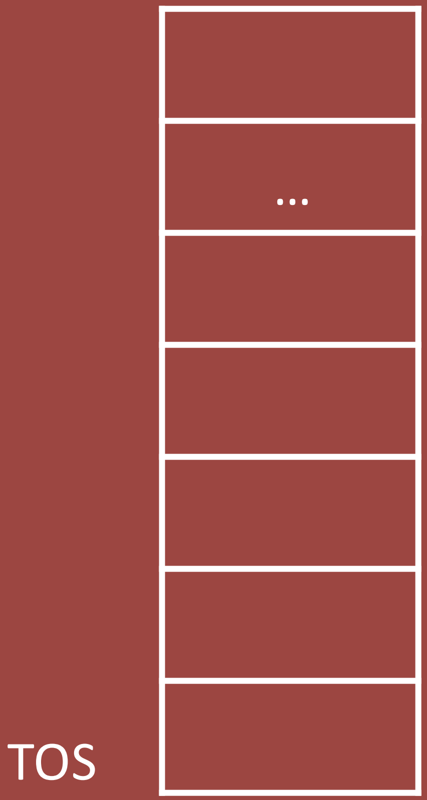


TOS

…

"coke"

```
push("coke");
push("cola");
x = pop();
push("pepsi");
x = pop();
x = pop();
```



```
push("coke");
push("cola");
x = pop();
push("pepsi");
x = pop();
x = pop();
```

STACK

POSSIBLE ERRORS

Stack Underflow

attempt to **pop** a value from an **empty** stack.

Stack Overflow

attempt to **push** a value into a **full** stack.

STACK

Array IMPLEMENTATION

```
def create_stack():
    stack = []
    return stack
```

Array IMPLEMENTATION

```
def push(stack, item):
    stack.append(item)
    print("pushed item: " + item)
```

Array IMPLEMENTATION

```
def pop(stack):
    if (check_empty(stack)):
        return "stack is empty"

    return stack.pop()
```

Array IMPLEMENTATION

STACK

Singly-Linked List

IMPLEMENTATION

```
# node class
class Node:
    def __init__(self, value):
        self.value = value
        self.next = None
```

Singly-Linked List

IMPLEMENTATION

```
def push(self, value):
    node = Node(value)
    node.next = self.head.next
    self.head.next = node
    self.size += 1
```

Singly-Linked List

IMPLEMENTATION

```
def pop(self):
    if self.isEmpty():
        raise Exception("Popping from an empty stack")

    remove = self.head.next
    self.head.next = self.head.next.next
    self.size -= 1
    return remove.value
```

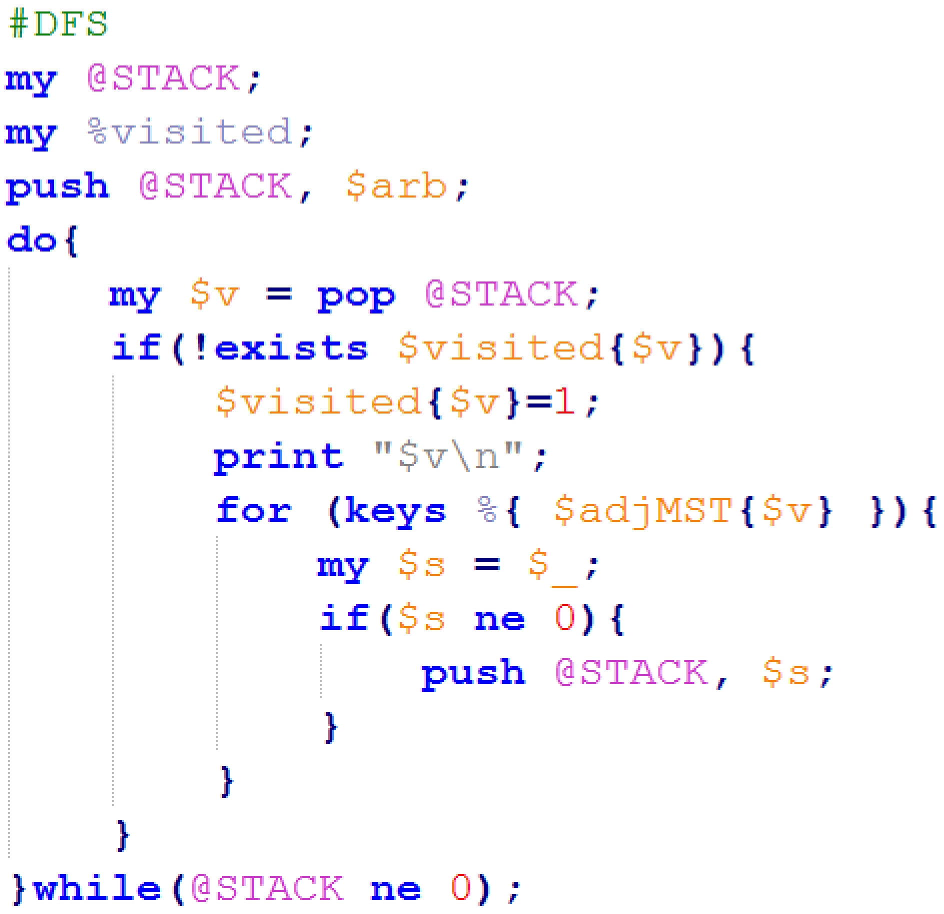
Singly-Linked List
IMPLEMENTATION

STACK

ADT

Applications

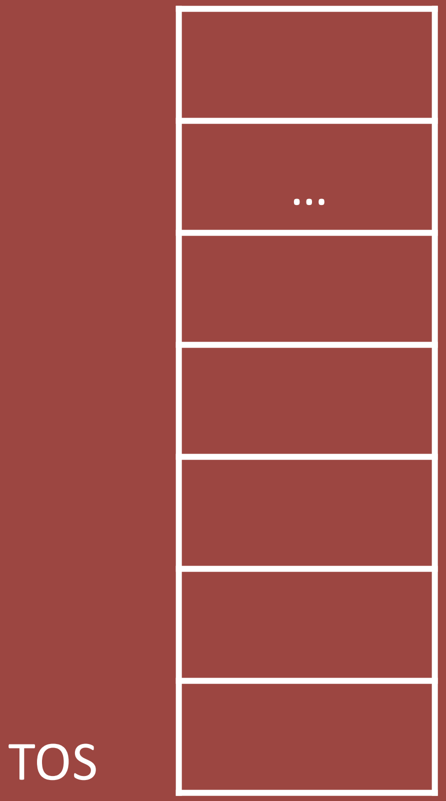
Balancing Symbols



```
#DFS
my @STACK;
my %visited;
push @STACK, $arb;
do{
    my $v = pop @STACK;
    if(!exists $visited{$v}){
        $visited{$v}=1;
        print "$v\n";
        for (keys %{ $adjMST{$v} }){
            my $s = $_;
            if($s ne 0){
                push @STACK, $s;
            }
        }
    }
}while(@STACK ne 0);
```

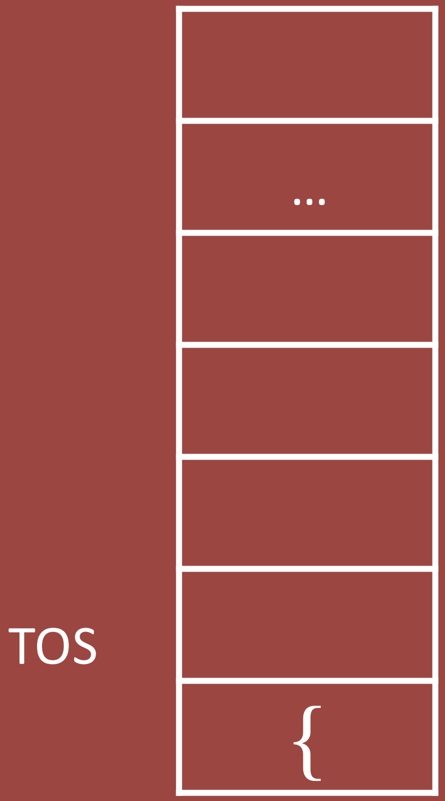
```
make an empty stack
read characters until end of file:
    if the character is an open symbol
        push it onto the stack.
    if it is a close symbol
        if the stack is empty
            report error
        else
            pop the stack
            if the symbol does not
            correspond to the
            opening symbol
                report error
if the stack is not empty
    report error
```

{()}[]


...
TOS

{()}[]

push("{");

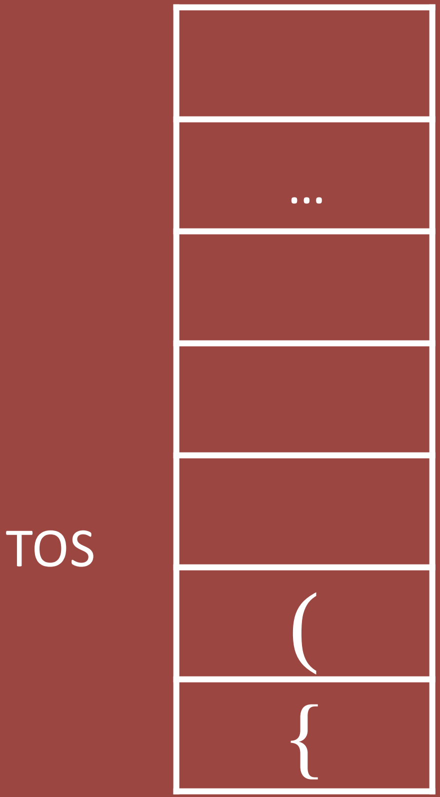


…

TOS

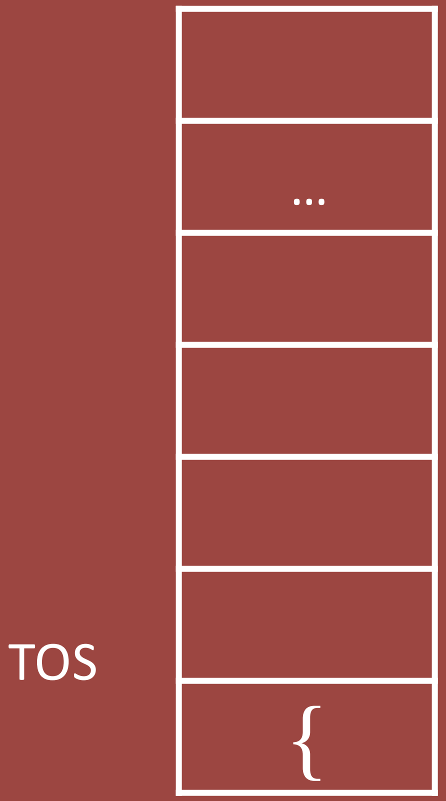
{

{()}[]



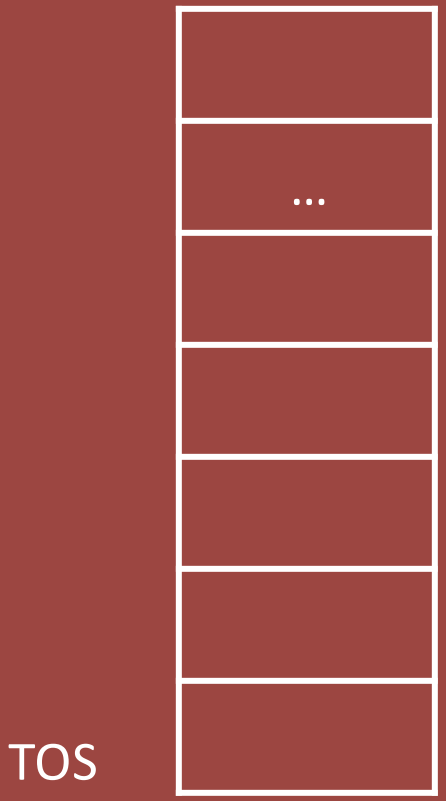
```
push("{");
push("(");
```

{()}[]



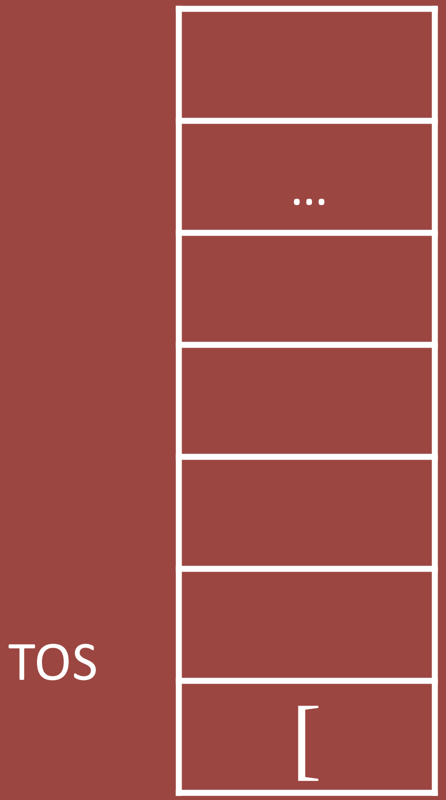
```
push("{");
push("(");
x = pop();
```

{()}[]



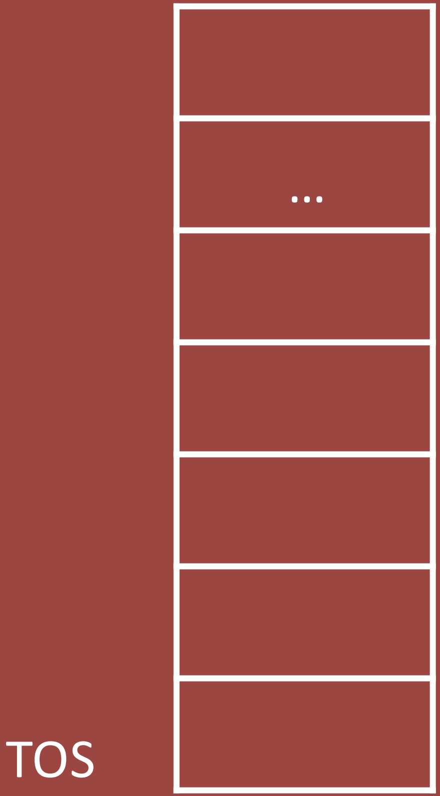
```
push(“{”);
push(“(”);
x = pop();
x = pop();
```

{()}[]



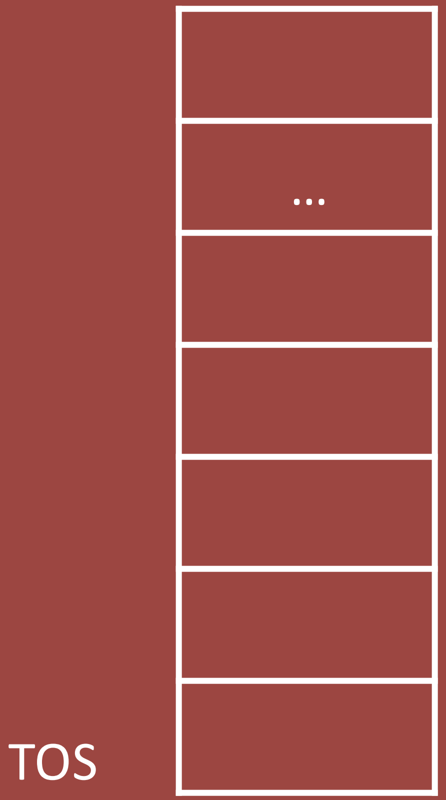
```
push("{");
push("(");
x = pop();
x = pop();
push("[");
```

{()}[]



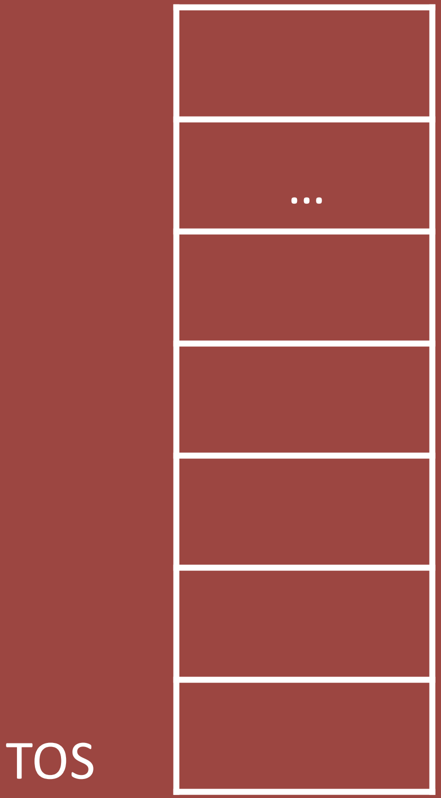
```
push("{");
push("(");
x = pop();
x = pop();
push("[");
x = pop();
```

{()}[]



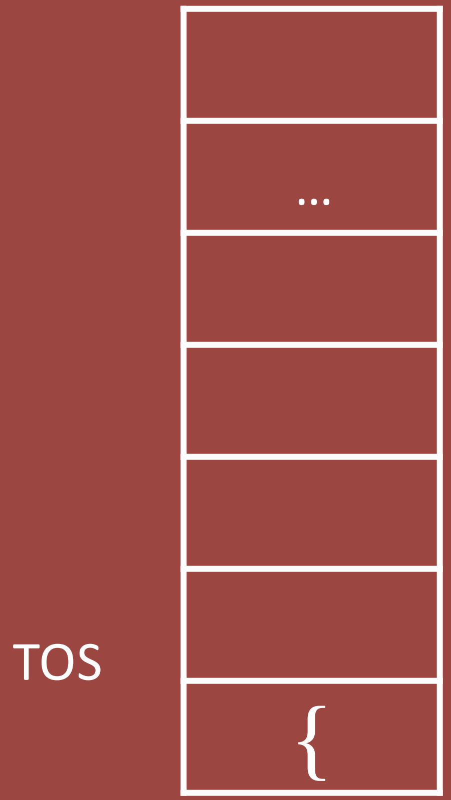
BALANCED!

{()}[[]


…
TOS

{()}[[]

```
push(“{”);
```



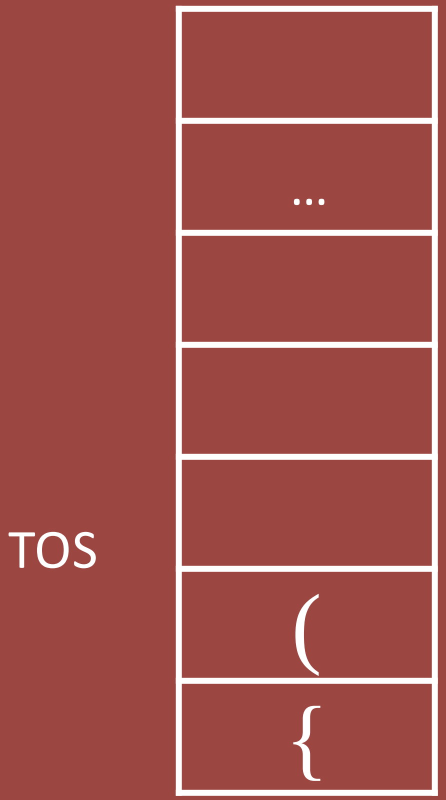
…

TOS

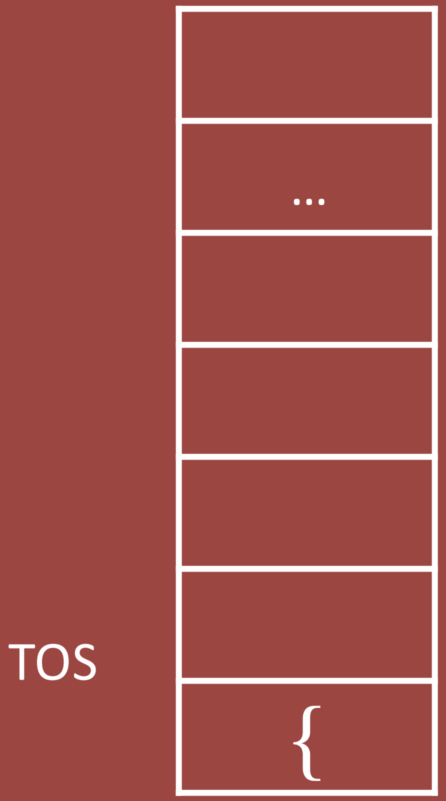
{

{()}[[]

```
push("{");
push("(");
```

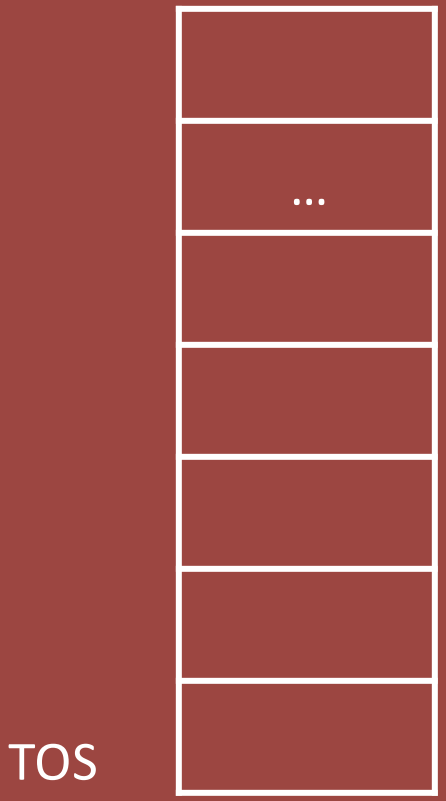


{()}[[]



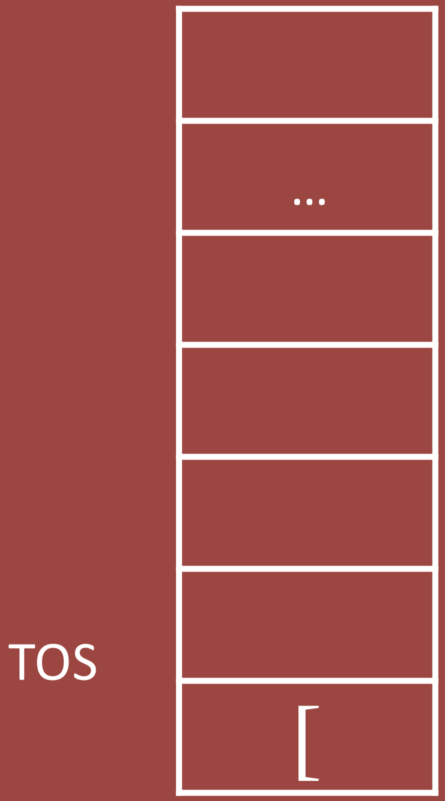
```
push("{");
push("(");
x = pop();
```

{()}[[]



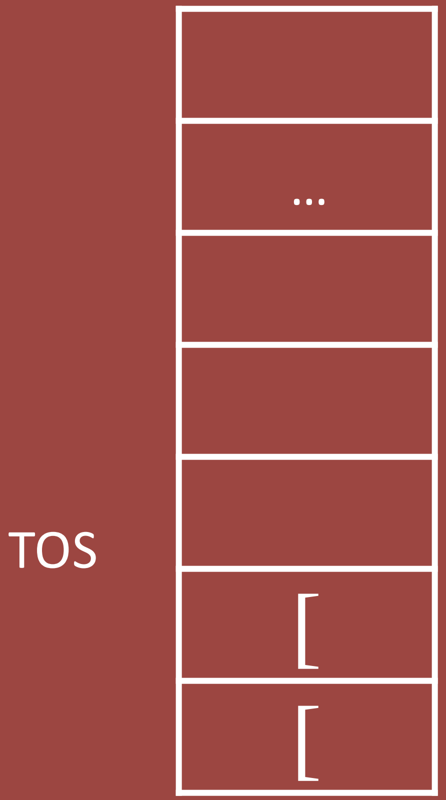
```
push(“{”);
push(“(”);
x = pop();
x = pop();
```

{()}[[]



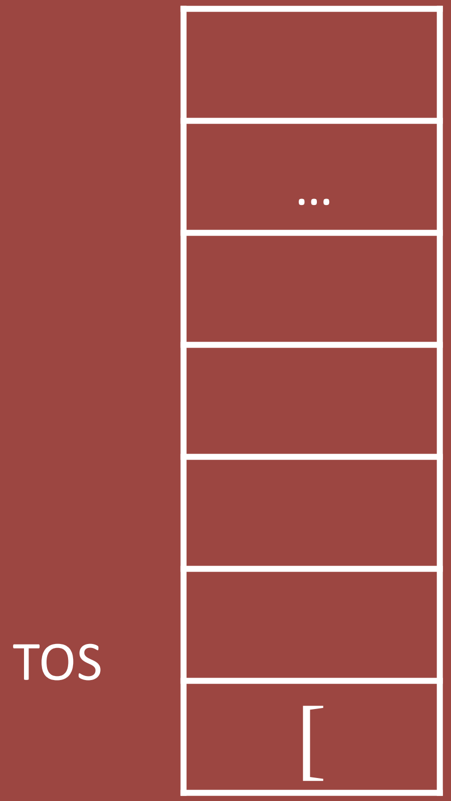
```
push("{");
push("(");
x = pop();
x = pop();
push("[");
```

{()}[[]



```
push("{");
push("(");
x = pop();
x = pop();
push("[");
push("[");
```

{()}[[]



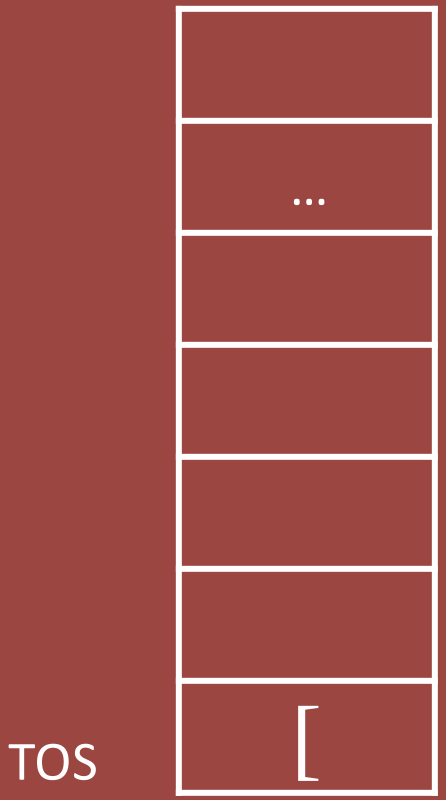
…

TOS

[

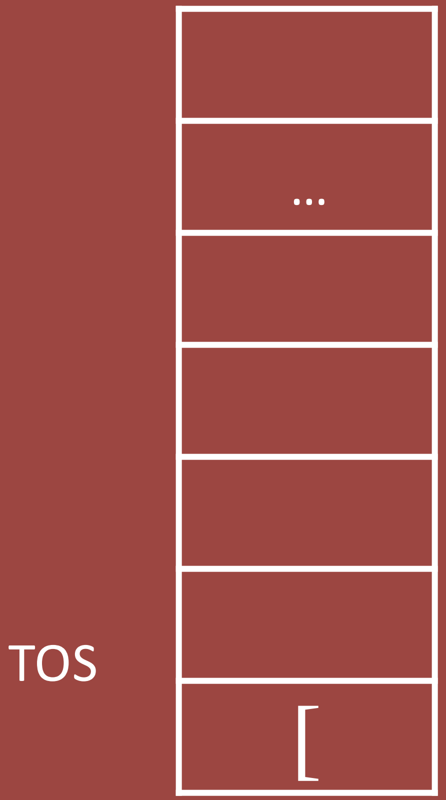
```
push("{");
push("(");
x = pop();
x = pop();
push("[");
push("[");
x = pop();
```

{()}[[]



```
push(“{”);
push(“(”);
x = pop();
x = pop();
push(“[”);
push(“[”);
x = pop();
```

{()}[[]



NOT BALANCED!

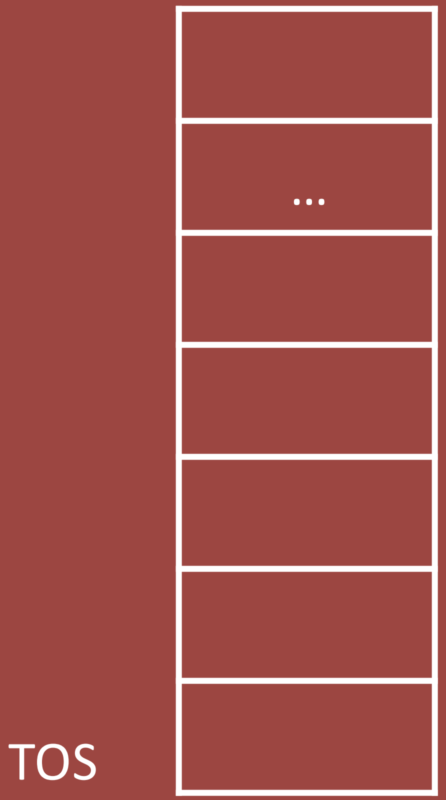
The stack still contain an element "[" after comparing all symbols.

4 6 + 3 5 + * 2 *

Postfix Expressions

```
make an empty stack
read characters until end of file:
    if a number is encountered
        push it onto the stack.
    if it is an operator symbol
        apply it to the two numbers that
        are popped.
        push the result onto the stack
```

4 6 + 3 5 + * 2 *

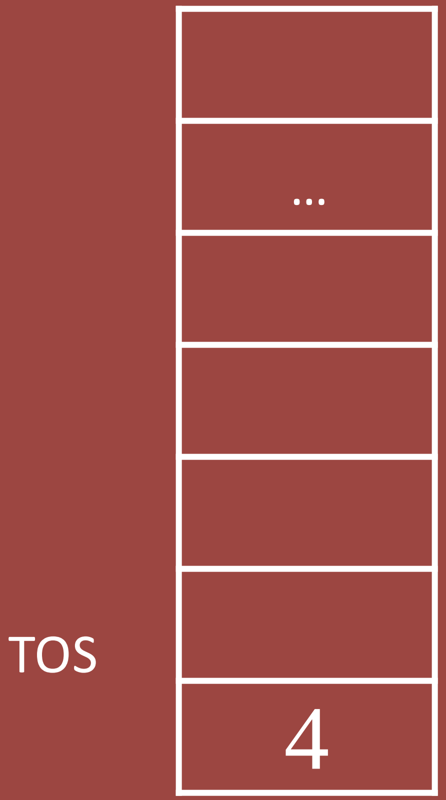


…

TOS

4 6 + 3 5 + * 2 *

push(4);



TOS

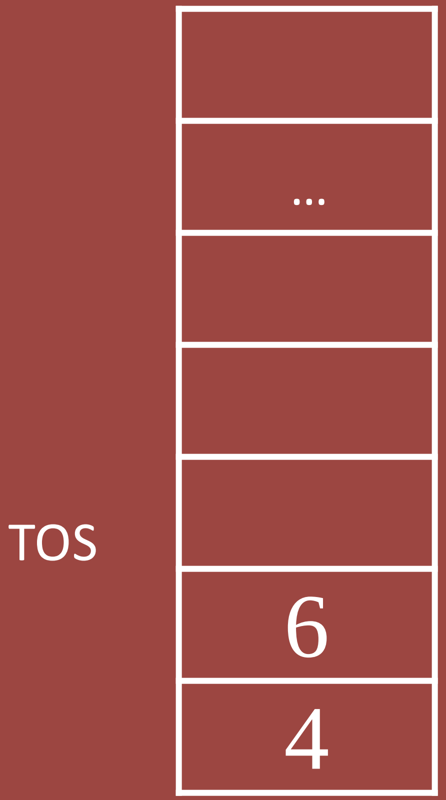
…

4

4 6 + 3 5 + * 2 *

push(4);

push(6);



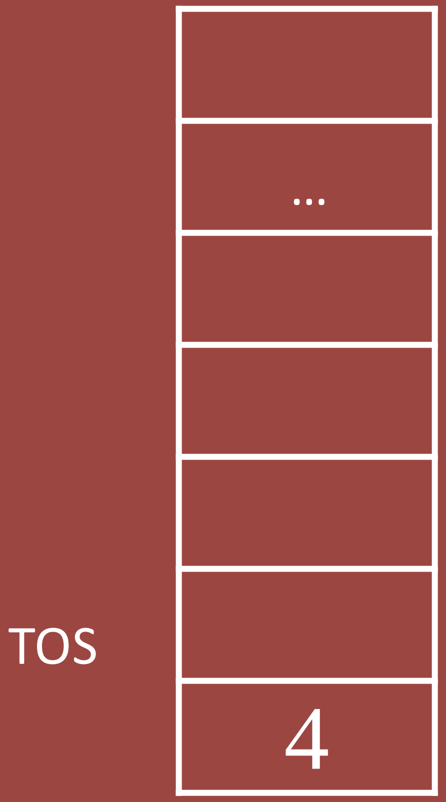
…

TOS

6

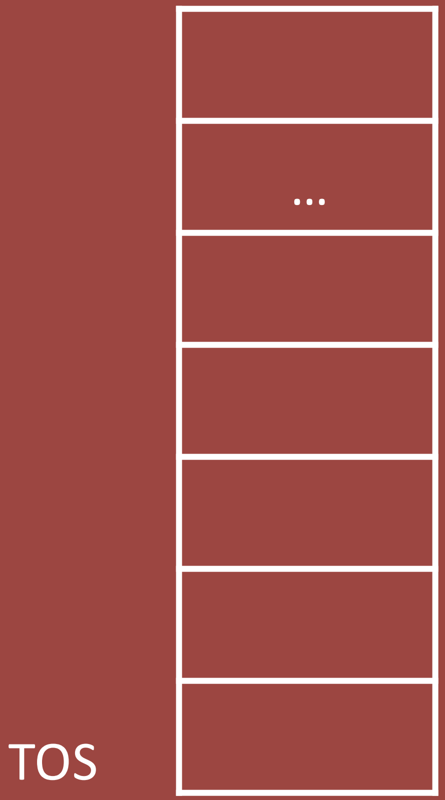
4

4 6 + 3 5 + * 2 *



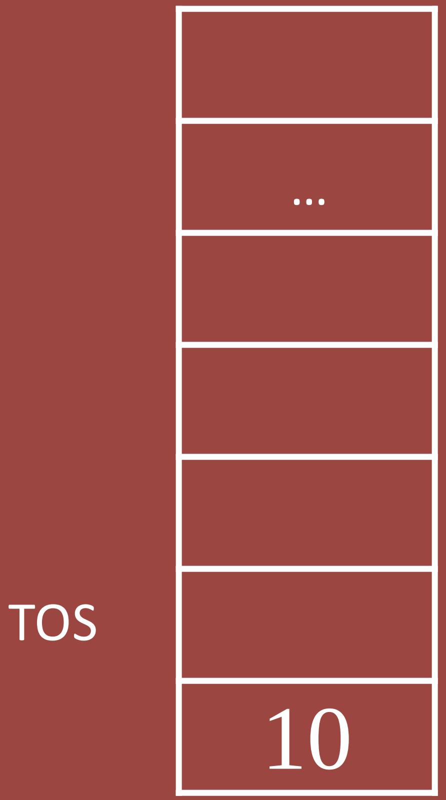
```
push(4);
push(6);
x = pop(); //6
```

4 6 + 3 5 + * 2 *



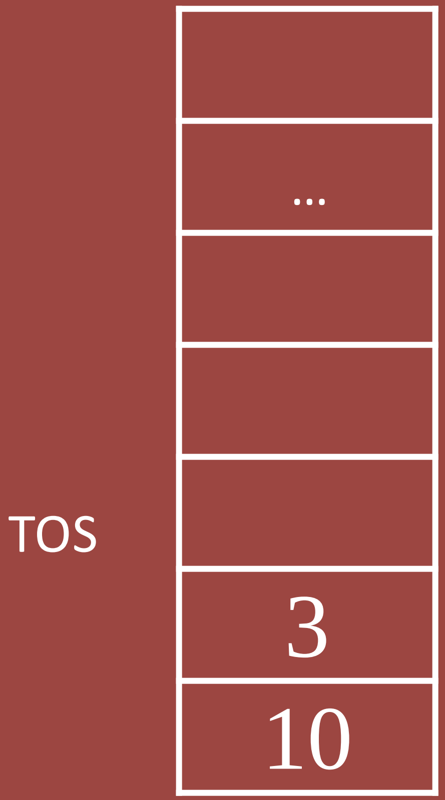
```
push(4);
push(6);
x = pop(); // 6
x = pop(); // 4
```

4 6 + 3 5 + * 2 *



```
push(4);
push(6);
x = pop(); // 6
x = pop(); // 4
push(10); //6+4
```

4 6 + 3 5 + * 2 *



…

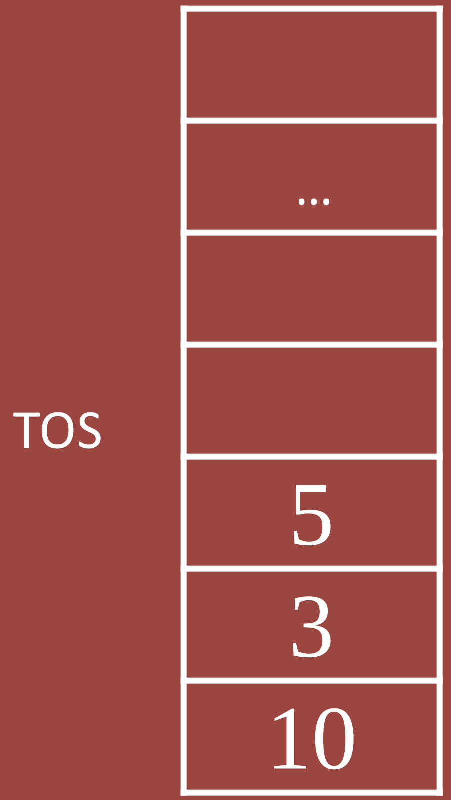
TOS

3

10

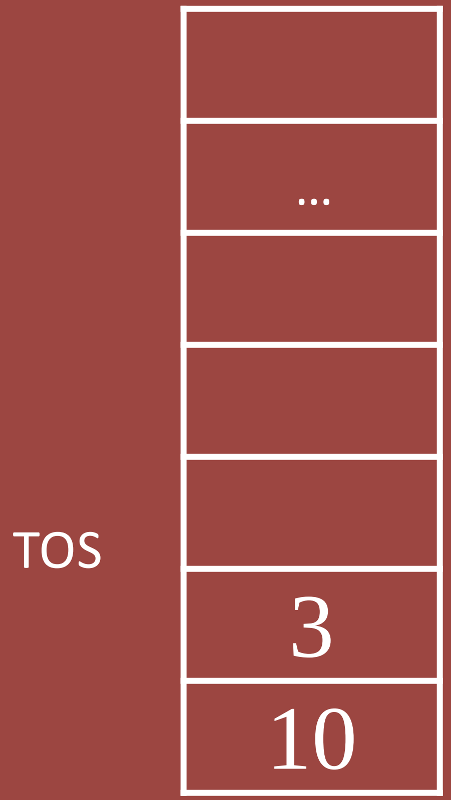
```
push(4);
push(6);
x = pop(); // 6
x = pop(); // 4
push(10); //6+4
push(3);
```

4 6 + 3 5 + * 2 *



```
push(4);
push(6);
x = pop(); // 6
x = pop(); // 4
push(10); //6+4
push(3);
push(5);
```

4 6 + 3 5 + * 2 *



TOS

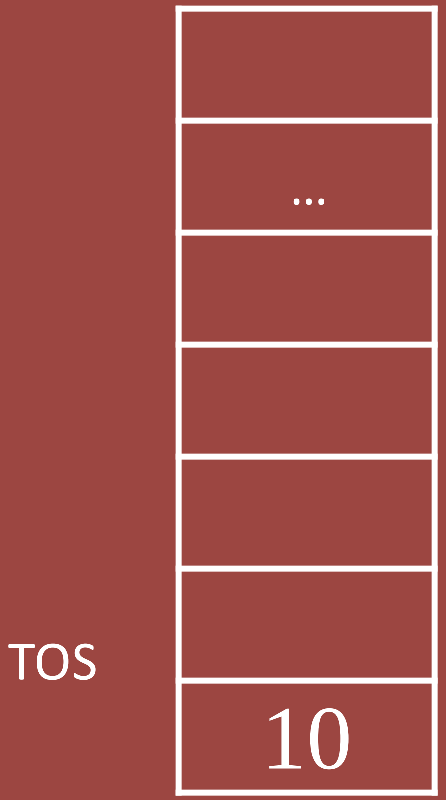
...

3

10

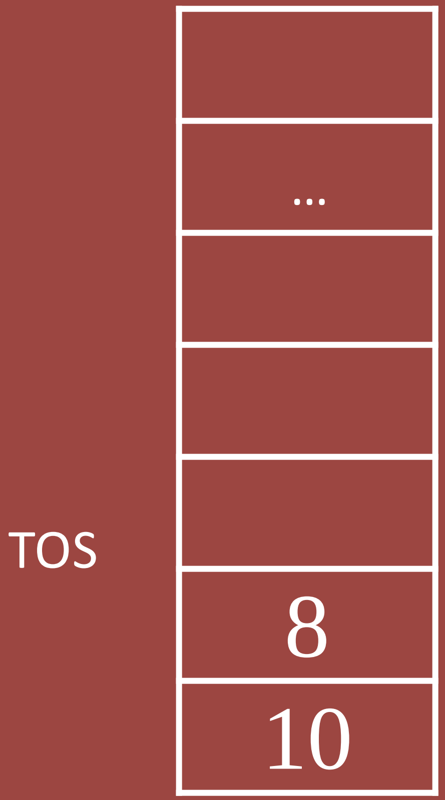
```
push(4);
push(6);
x = pop(); // 6
x = pop(); // 4
push(10); //6+4
push(3);
push(5);
x = pop(); //5
```

4 6 + 3 5 + * 2 *



```
push(4);
push(6);
x = pop(); // 6
x = pop(); // 4
push(10); //6+4
push(3);
push(5);
x = pop(); //5
x = pop(); //3
```

4 6 + 3 5 + * 2 *



TOS

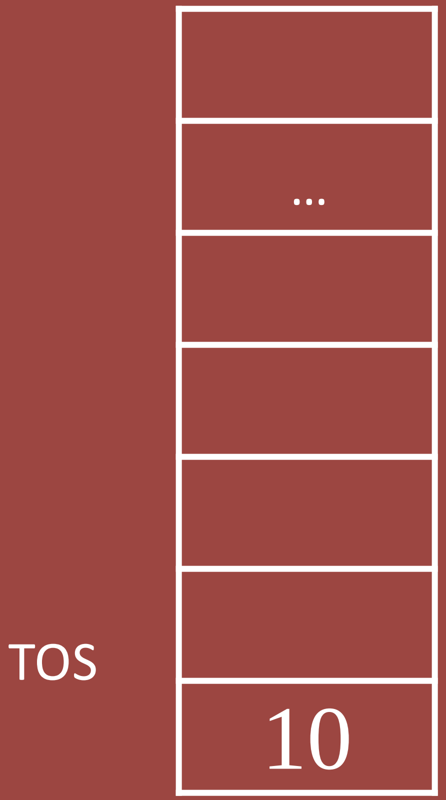
…

8

10

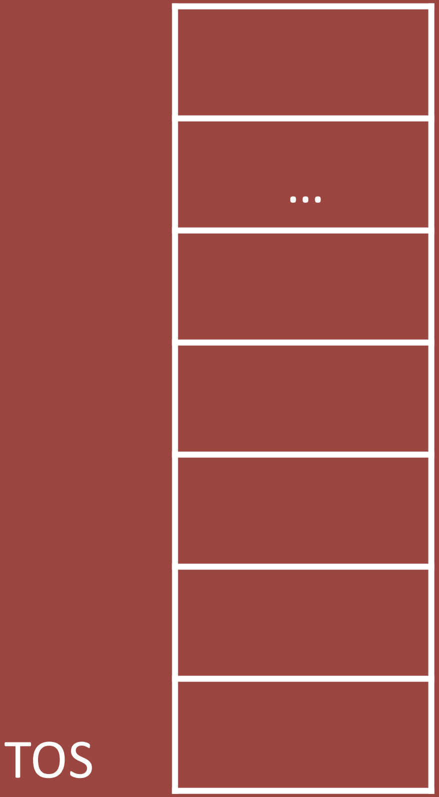
```
push(4);
push(6);
x = pop(); // 6
x = pop(); // 4
push(10); //6+4
push(3);
push(5);
x = pop(); //5
x = pop(); //3
push(8); //5+3
```

4 6 + 3 5 + * 2 *



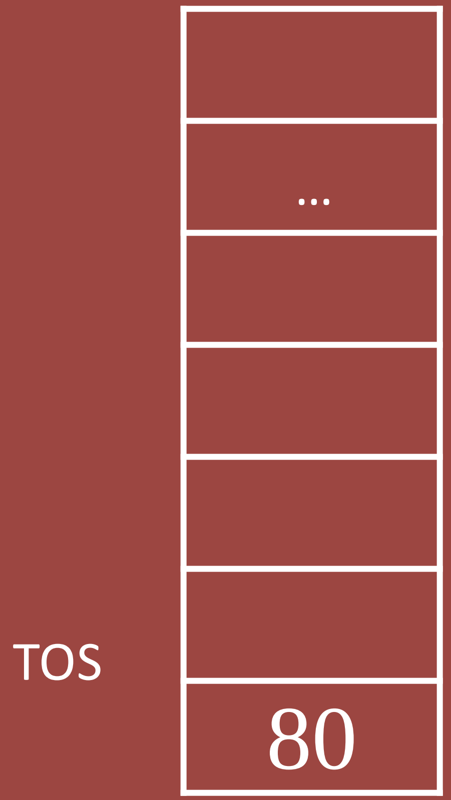
```
push(4);
push(6);
x = pop(); // 6
x = pop(); // 4
push(10); //6+4
push(3);
push(5);
x = pop(); //5
x = pop(); //3
push(8); //5+3
x = pop(); //8
```

4 6 + 3 5 + * 2 *



```
push(4);
push(6);
x = pop(); // 6
x = pop(); // 4
push(10); //6+4
push(3);
push(5);
x = pop(); //5
x = pop(); //3
push(8); //5+3
x = pop(); //8
x = pop(); //10
```

4 6 + 3 5 + * 2 *



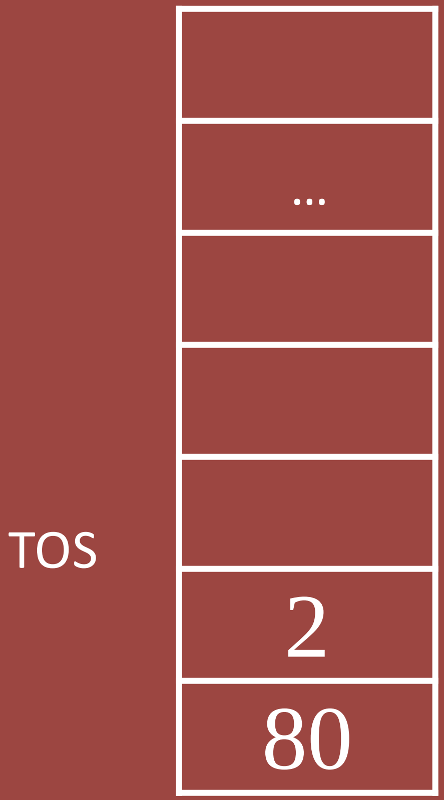
TOS

…

80

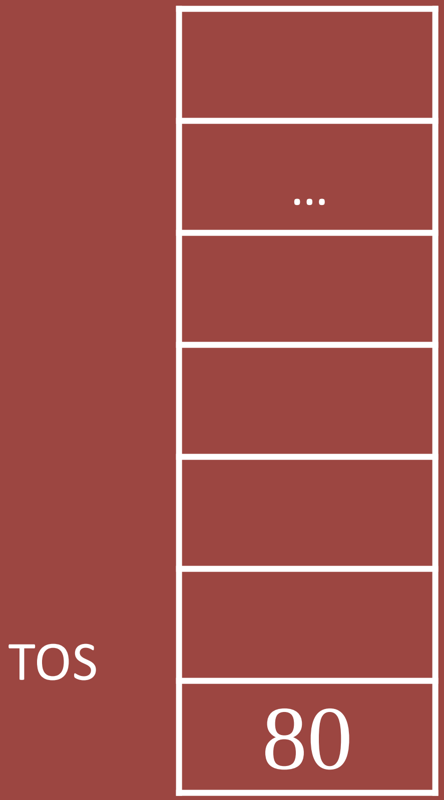
```
push(4);
push(6);
x = pop(); // 6
x = pop(); // 4
push(10); //6+4
push(3);
push(5);
x = pop(); //5
x = pop(); //3
push(8); //5+3
x = pop(); //8
x = pop(); //10
push(80); //8*10
```

4 6 + 3 5 + * 2 *



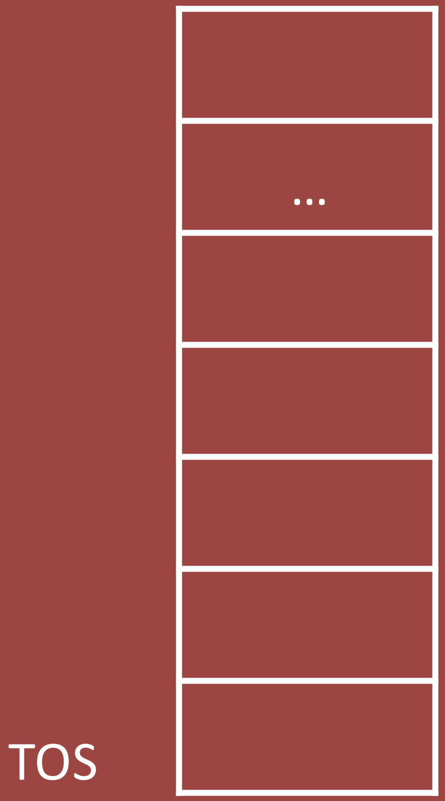
```
push(4);
push(6);
x = pop(); // 6
x = pop(); // 4
push(10); //6+4
push(3);
push(5);
x = pop(); //5
x = pop(); //3
push(8); //5+3
x = pop(); //8
x = pop(); //10
push(80); //8*10
push(2);
```

4 6 + 3 5 + * 2 *



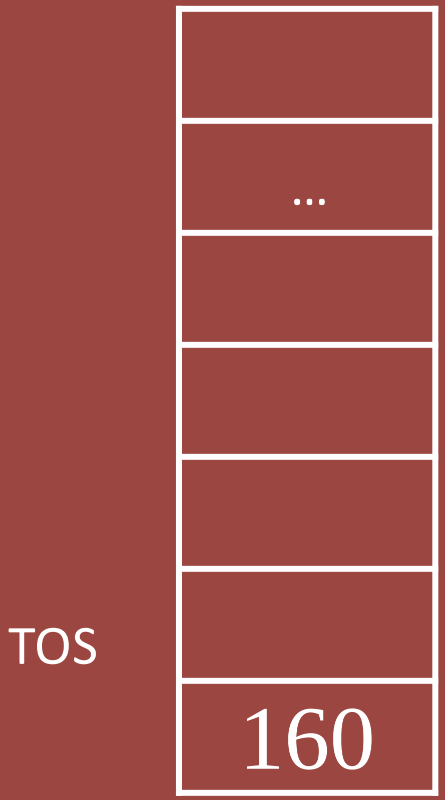
```
push(4);
push(6);
x = pop(); // 6
x = pop(); // 4
push(10); //6+4
push(3);
push(5);
x = pop(); //5
x = pop(); //3
push(8); //5+3
x = pop(); //8
x = pop(); //10
push(80); //8*10
push(2);
x = pop();//2
```

4 6 + 3 5 + * 2 *



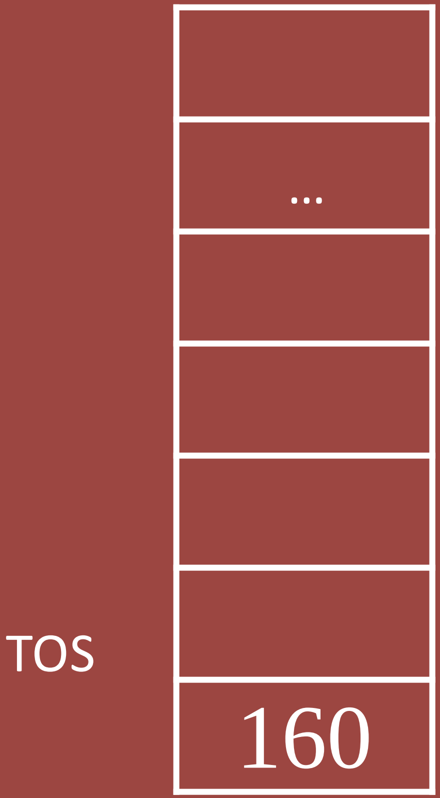
```
push(4);
push(6);
x = pop(); // 6
x = pop(); // 4
push(10); //6+4
push(3);
push(5);
x = pop(); //5
x = pop(); //3
push(8); //5+3
x = pop(); //8
x = pop(); //10
push(80); //8*10
push(2);
x = pop(); //2
x = pop(); //80
```

4 6 + 3 5 + * 2 *



```
push(4);
push(6);
x = pop(); // 6
x = pop(); // 4
push(10); //6+4
push(3);
push(5);
x = pop(); //5
x = pop(); //3
push(8); //5+3
x = pop(); //8
x = pop(); //10
push(80); //8*10
push(2);
x = pop(); //2
x = pop(); //80
push(160); //2*80
```

4 6 + 3 5 + * 2 *



TOS

…

160

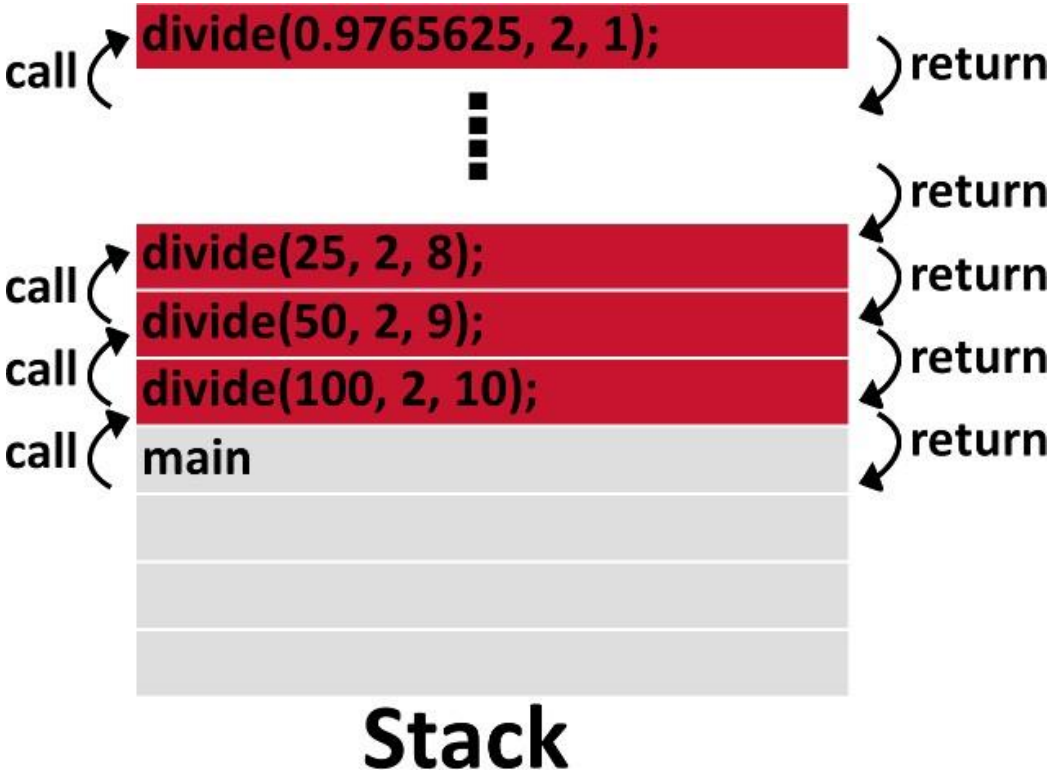
The answer to the expression is ***160****.*

Infix to Postfix Conversion

4 + 6 * 3 + 5 * 2

4 6 + 3 5 + * 2 *

Function Calls



divide(0.9765625, 2, 1);

divide(25, 2, 8);

divide(50, 2, 9);

divide(100, 2, 10);

main

call

call

call

call

return

return

return

return

return

Stack