Stack

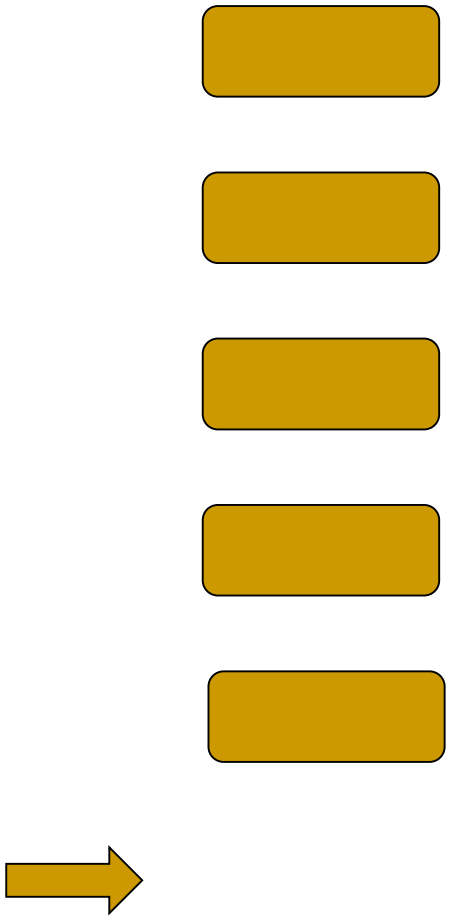
Joy Mukherjee

Stack

- Stack is a linear data structure
- Insertion and deletion take place on same end
- LIFO(Last In First Out)
  - The last one inserted is the first one deleted
- FILO(First In Last Out)
  - The first one inserted is the last one deleted

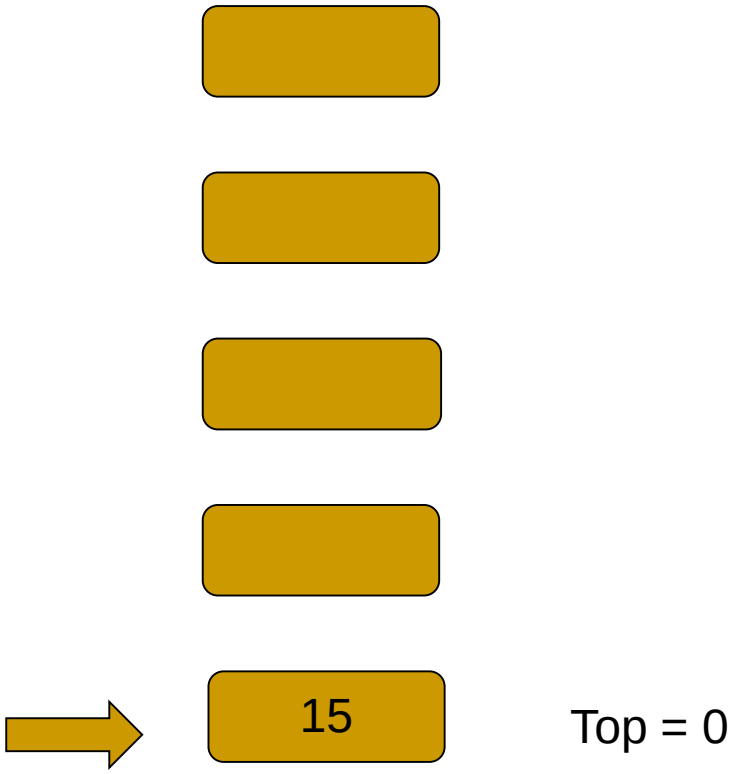
- int A[5], top;
- The maximum size of the stack is 5
- Top points to the topmost element of the stack.

Operations: Initially Stack is Empty

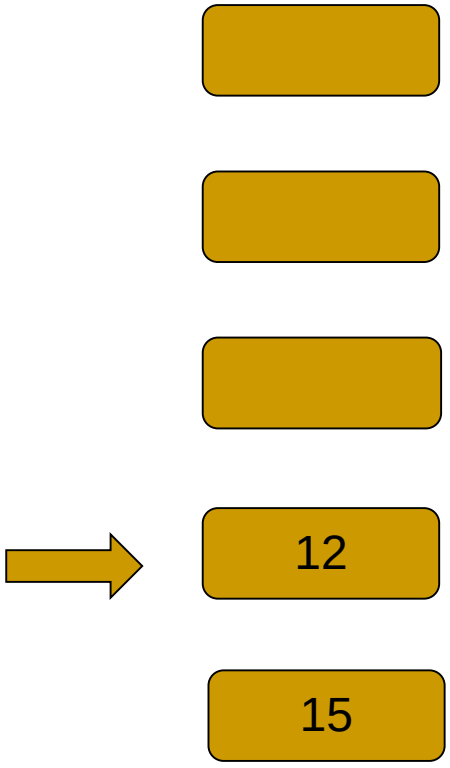


Top = -1

Operations: Push 15

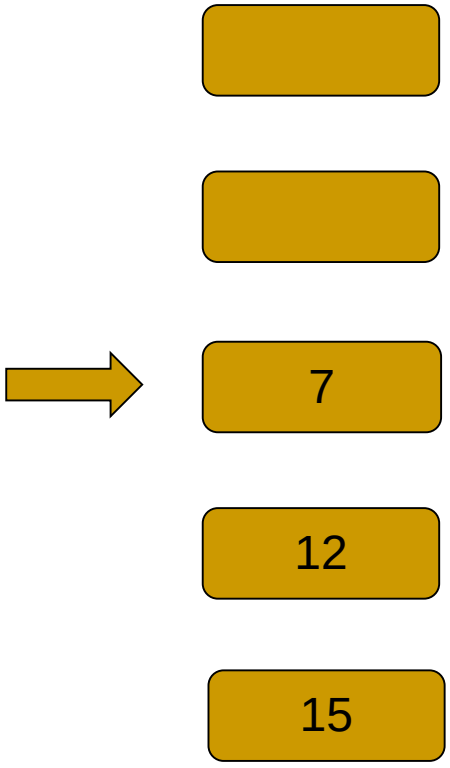


Operations: Push 12



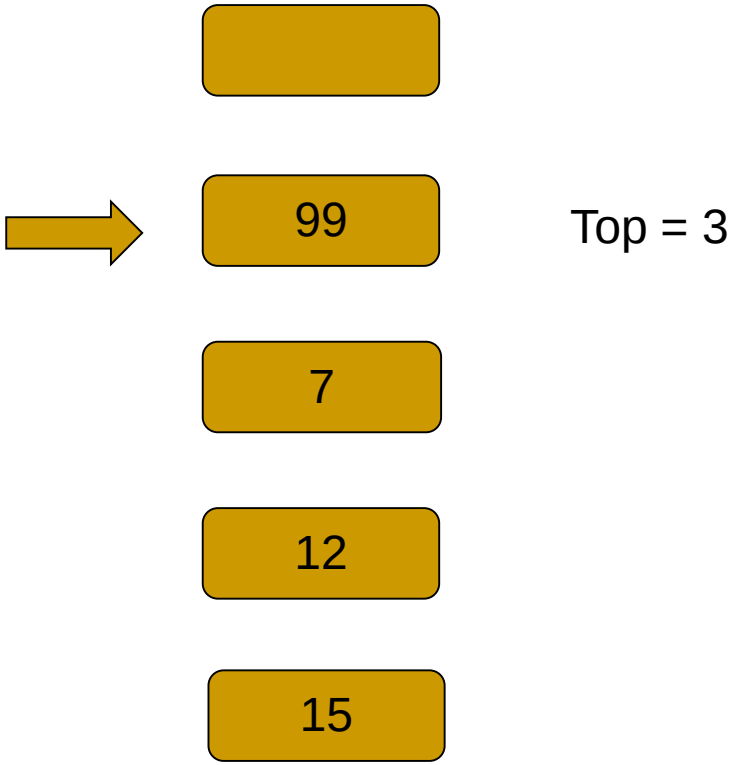
Top = 1

Operations: Push 7

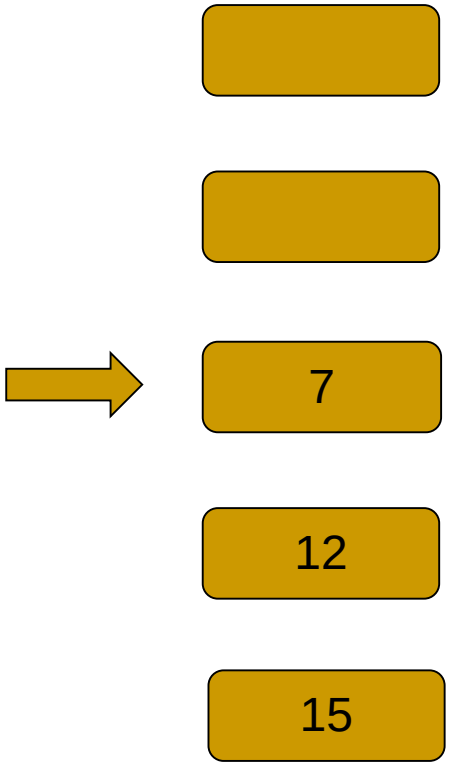


Top = 2

Operations: Push 99

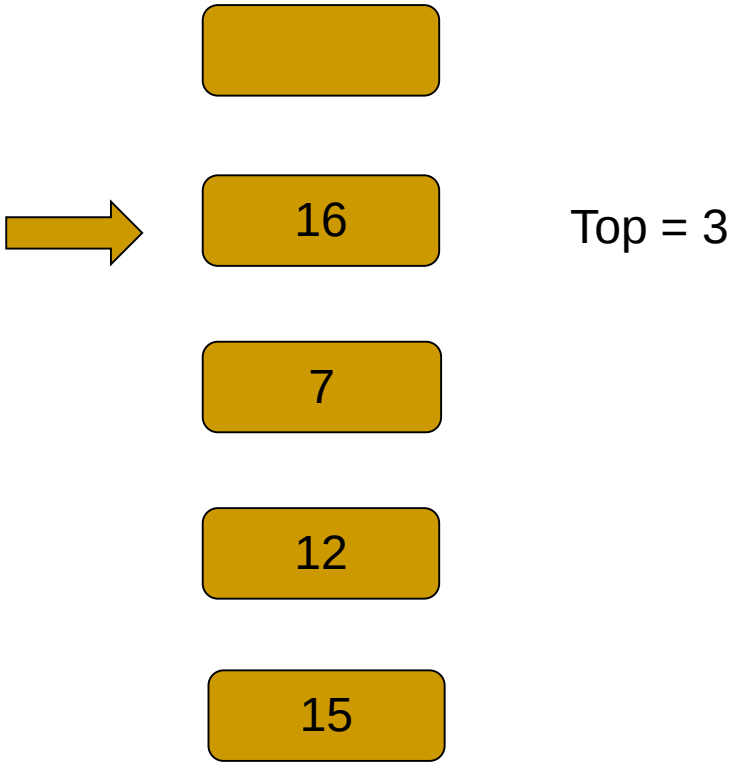


Operations: Pop

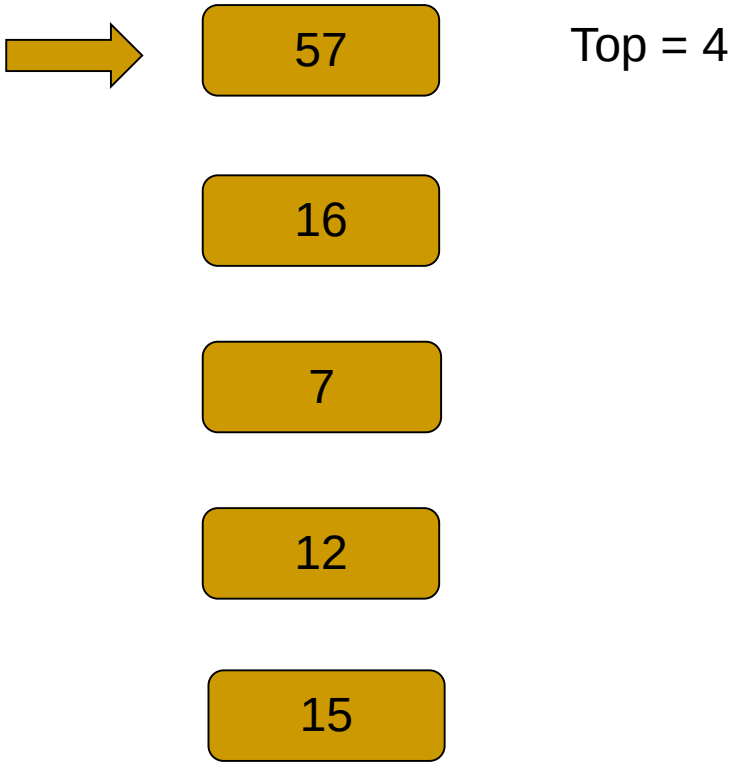


Top = 2

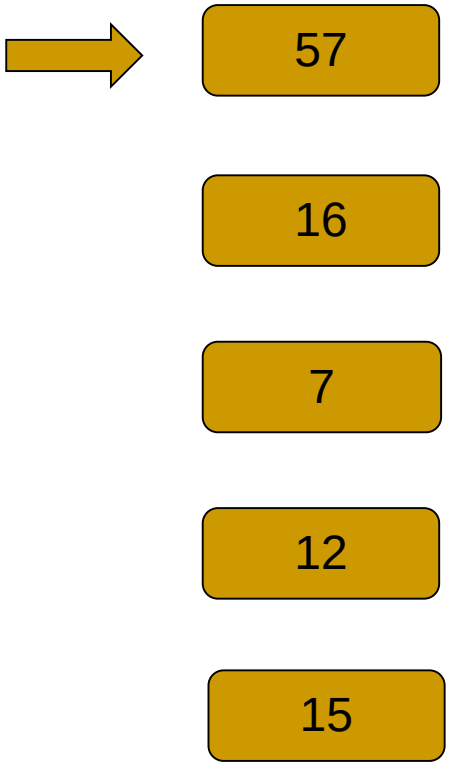
Operations: Push 16



Operations: Push 57

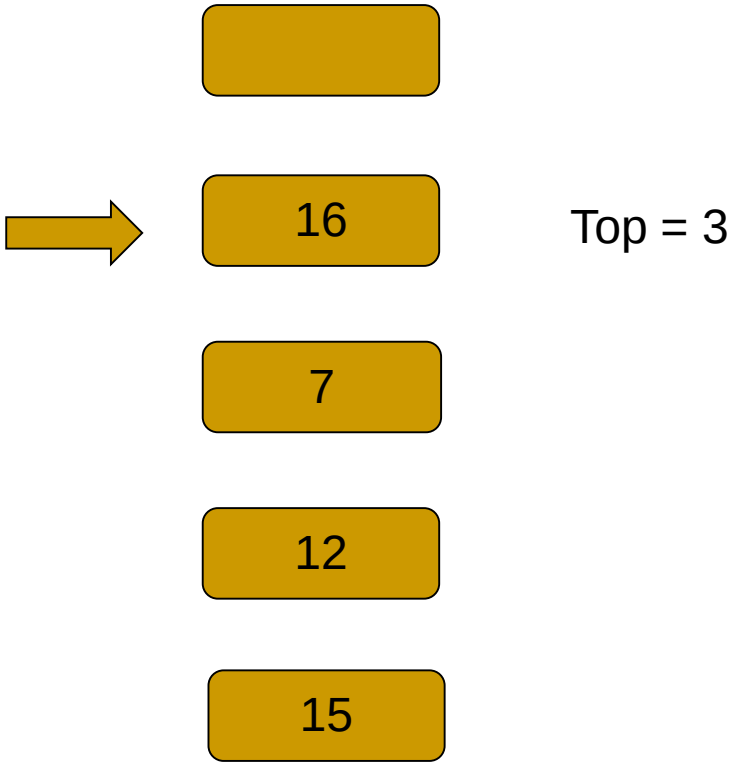


Operations: Push 33

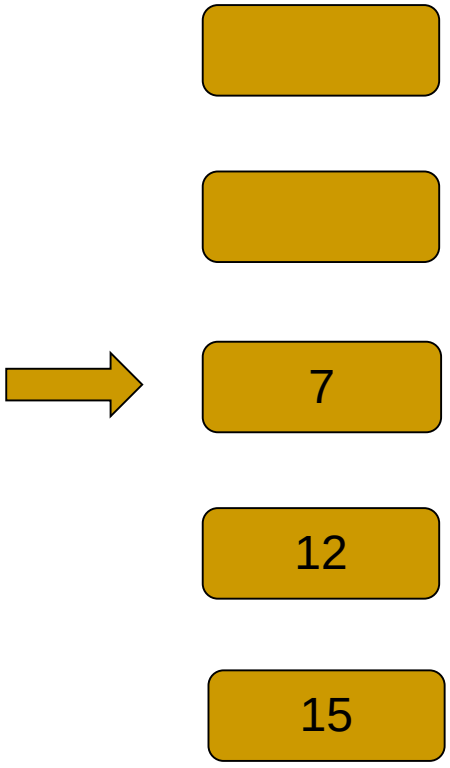


Top = 4

Operations: Pop

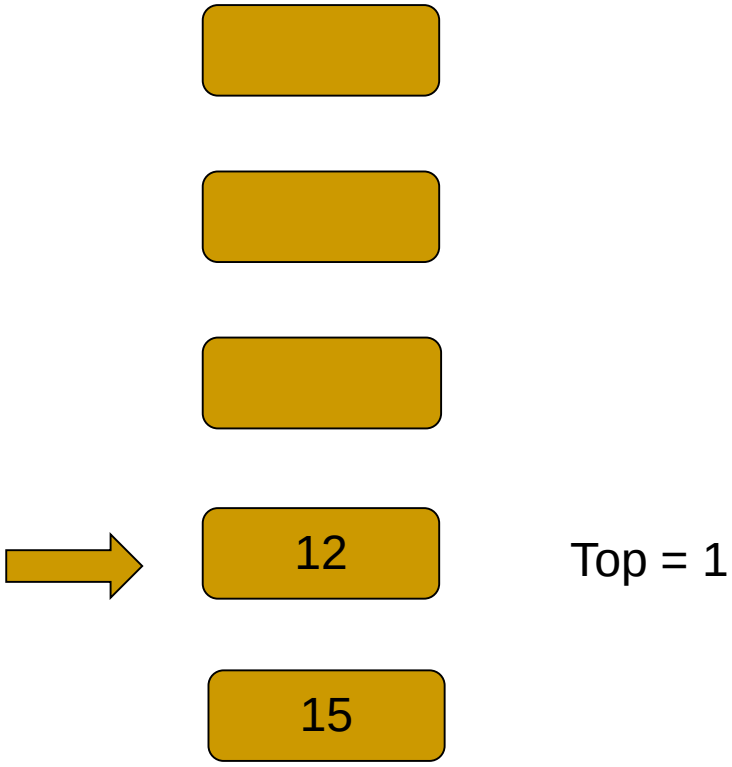


Operations: Pop

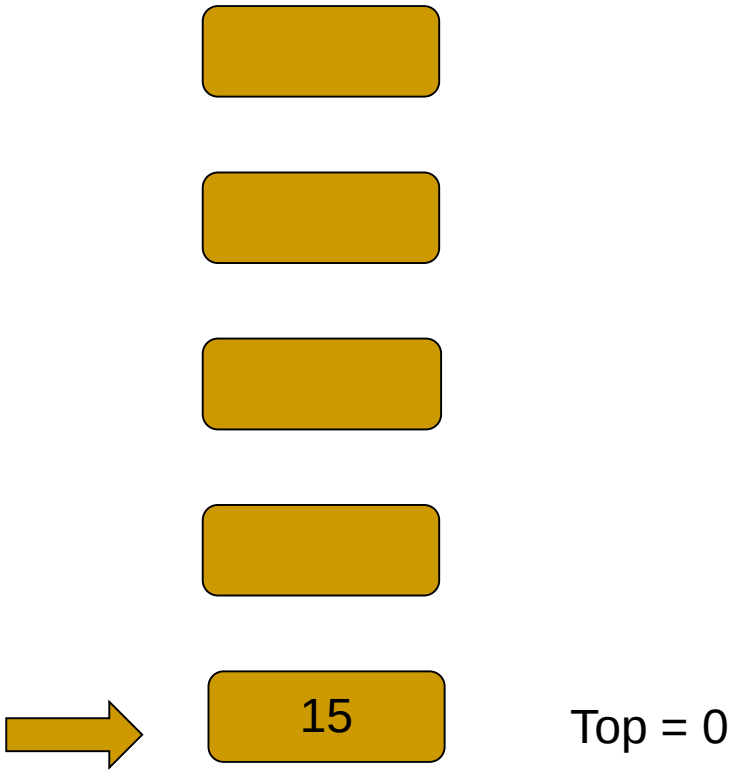


Top = 2

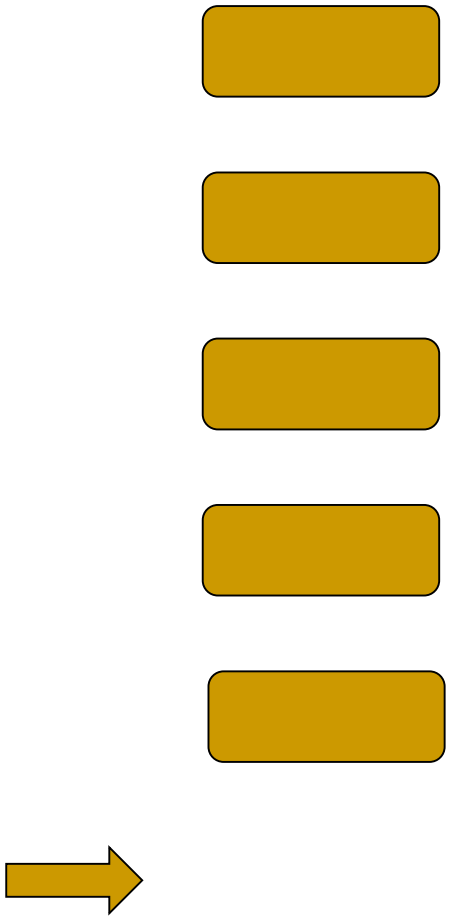
Operations: Pop



Operations: Pop

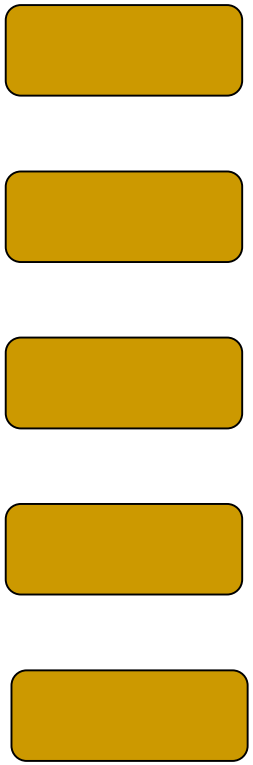


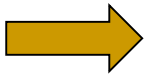
Operations: Pop



Top = -1

Operations: Pop





Top = -1

Stack

```
#define MAXLEN 5

struct st{
        int A[MAXLEN];
        int top;
};  // User-defined data type struct st is created only

typedef struct st stack; // struct st and stack
represent the same data type
```

Stack Operations

```
stack init ()
{
    stack S;
    S.top = -1;
    return S;
}
```

```
int isEmpty (stack S)
{
   if(S.top == -1)
        return 1;
    return 0;
}
```

```
int isFull (stack S)
{
   if (S.top == MAXLEN - 1)
        return 1;
   return 0;
}
```

Stack Operations

```
stack push (stack S , int k)
{
    if (isFull(S) == 1) {
        printf("Stack is Full");
        return S;
    }
    S.top = S.top + 1;
    S.A[S.top] = k;
    return S;
}
```

```
stack pop (stack S)
{
   if (isEmpty(S) == 1) {
            printf("Stack is Empty");
            return S;
    }
    S.top = S.top - 1;
    return S;
}
```

Stack: Applications

Infix to Postfix Conversion

Postfix Evaluation

Share Span Problem

Assignment - 1

- Subject Line: A1_ROLLNO
- File Name: A1_ROLLNO.c/cpp
- Email: pds2016autumn@gmail.com
- Submission: 08-August-2021 11:59 P.M.
- Write a C program that takes a mathematical expression as input (infix), and output its value.
- Input: 4+(8 – 3)*2-(3+4)
- Output: 7
- Input: 40/8 – 30*2
- Output: -55

Infix to Postfix Conversion

Infix to Postfix Conversion

- Write a C program that takes a mathematical expression as input (infix), and output its value.
- Input: 4+(8 – 3)*2-(3+4)
- Output: 7
- Input: 40/8 – 30*2
- Output: -55

- Convert Infix to Postfix (Stack)
- Evaluate the Postfix (Stack)

Example

- Infix: A - B + C (Operator is present in between operands)
- Prefix: +-ABC
- Postfix: AB-C+

- Infix: A - B + C * D(Operator is present in between operands)
- Prefix: +-AB*CD
- Postfix: AB-CD*+

Infix to Postfix Conversion

- 4+(8 – 3)*2-(3+4)
- 4 + 8 3 –* 2-(3+4)
- 4+8 3 – 2*-(3+4)
- 4 + 8 3 – 2 * - 3 4+
- 4 8 3 – 2*+ - 3 4+
- 4 8 3 – 2 * + 3 4+ -
- Input: Infix
- Output: Postfix
- Assumption:
  - Each operator is a binary operator
  - Associativity of an operator is from left to right
    - Example: A+B+C-D = (((A+B)+C)-D)

Postfix Evaluation

- 4 8 3 – 2 * + 3 4+ -
- 7
- Input: Infix
- Output: Postfix
- Assumption:
  - Each operator is a binary operator
  - Associativity of an operator is from left to right
    - Example: A+B+C-D = (((A+B)+C)-D)

Infix to postfix conversion

Any mathematical expression is written as an infix expression

It consists of tokens: (,), operators, and operands

Operator: + – * /
Operand: Variable, or Value

Infix expression: An operator exists between two operands

A + B, A – 5 / C, (A – 5) / C

Postfix expression: Operands followed by an operator

A B +, A 5 C / –, A 5 – C /

Infix to postfix conversion: Algorithm

For each token from an infix expression from left to right :

- If (*token* = operand)
  - Push it to postfix expression
- If (*token* = ‘(‘)
  - Push (to the stack
- If (*token* = ‘)‘)
  - Repeatedly pop a token y from stack and push y to postfix expression until ‘(‘ is encountered in stack.
  - Pop ‘(‘ from stack.
  - If stack is already empty before finding a ‘(‘
    - Expression is invalid

Infix to postfix conversion: Algorithm

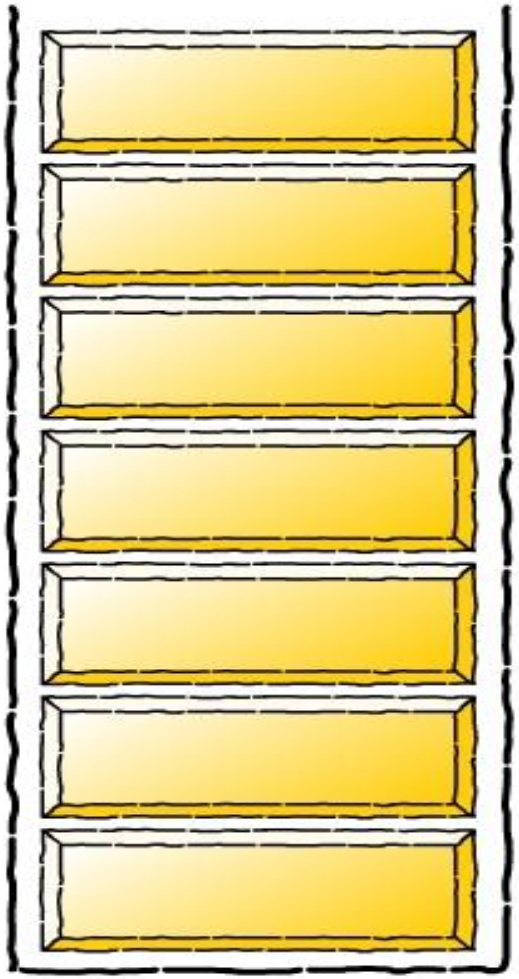
- If (token x = operator)
  - While(the following three conditions are true)
    1. The stack is not empty
    2. The token y is not a ‘(‘ .
    3. The token y is an operator && Precedence(y) >= Precedence (x),
       - Push the token y into postfix expression
       - Pop the token y from stack
  - Push the token x into stack.

Infix to postfix conversion: Algorithm

- After the loop has processed all the tokens in the infix expression, use another loop to repeatedly do the following as long as the stack is not empty
  - Pop token y from the stack.
  - Push the token y into postfix expression.

Infix to postfix conversion

stack



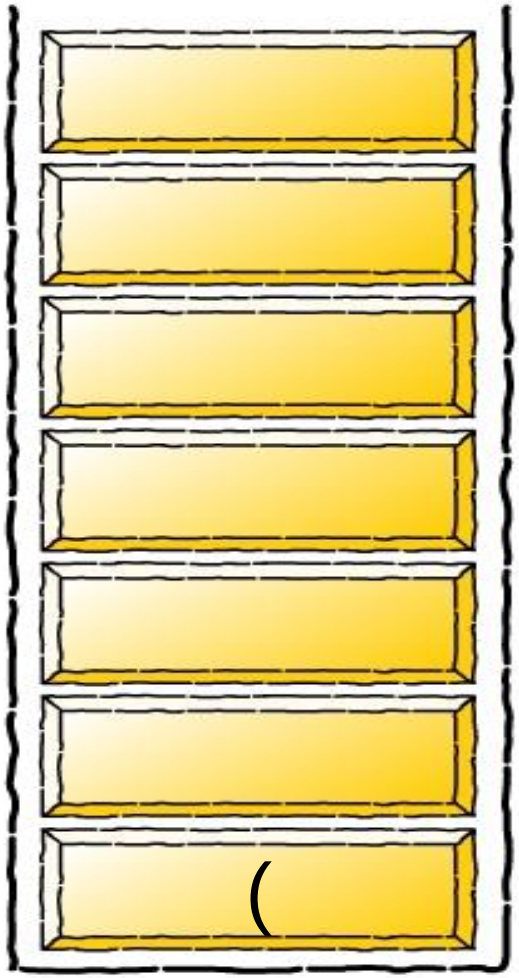
infixVect

(a + b - c) * d - (e + f)

postfixVect

Infix to postfix conversion

stack



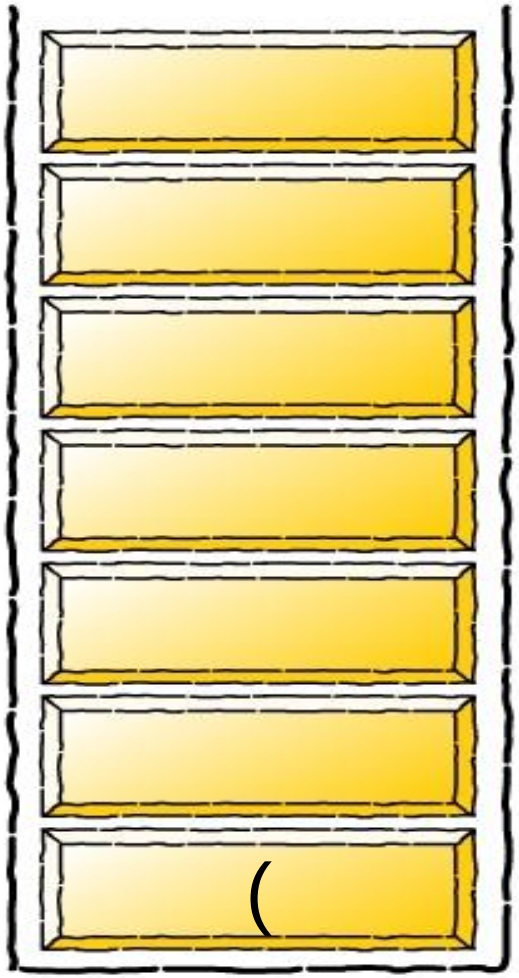
infixVect

a + b - c) * d - (e + f)

postfixVect

Infix to postfix conversion

stack



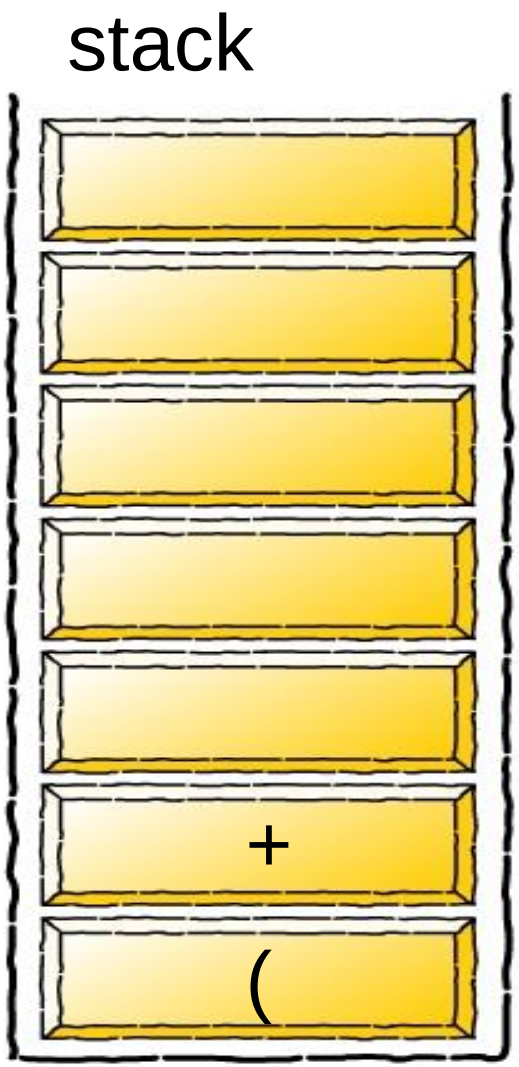
infixVect

+ b - c) * d - (e + f)

postfixVect

a

Infix to postfix conversion



infixVect

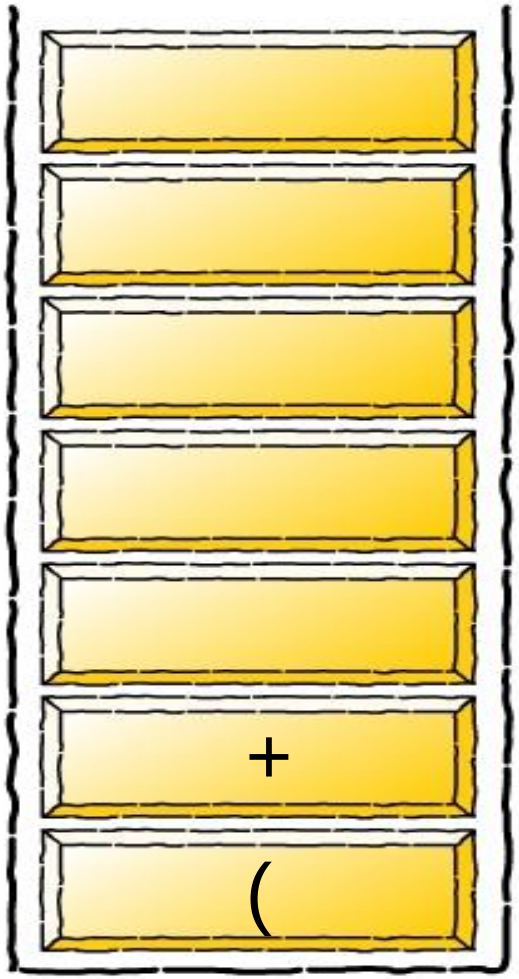
b - c) * d - (e + f)

postfixVect

a

Infix to postfix conversion

stack



infixVect

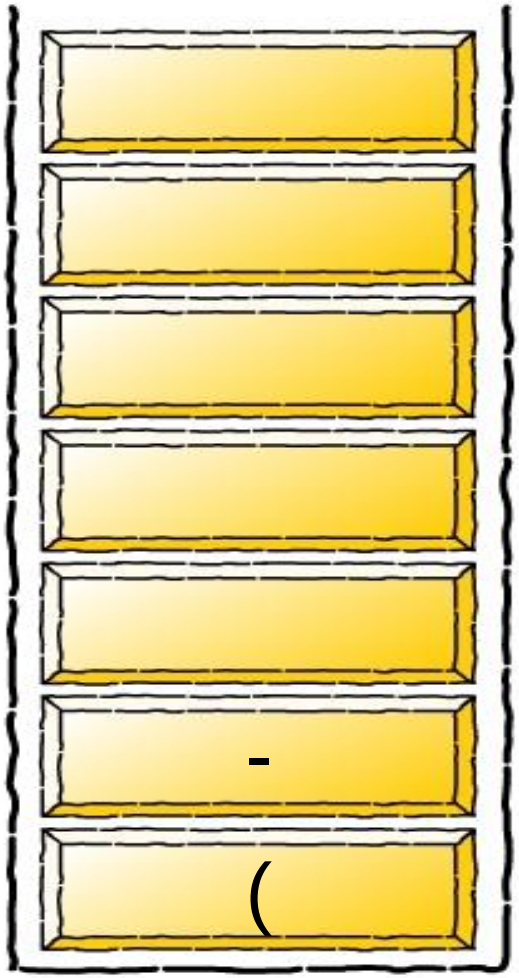
- c) * d - (e + f)

postfixVect

a b

Infix to postfix conversion

stack



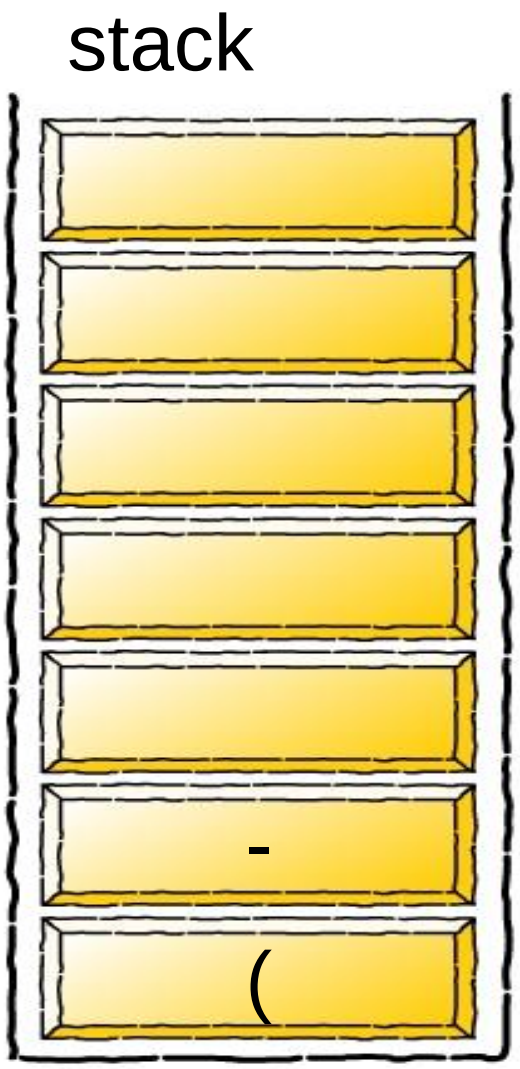
infixVect

c) * d - (e + f)

postfixVect

a b +

Infix to postfix conversion



infixVect

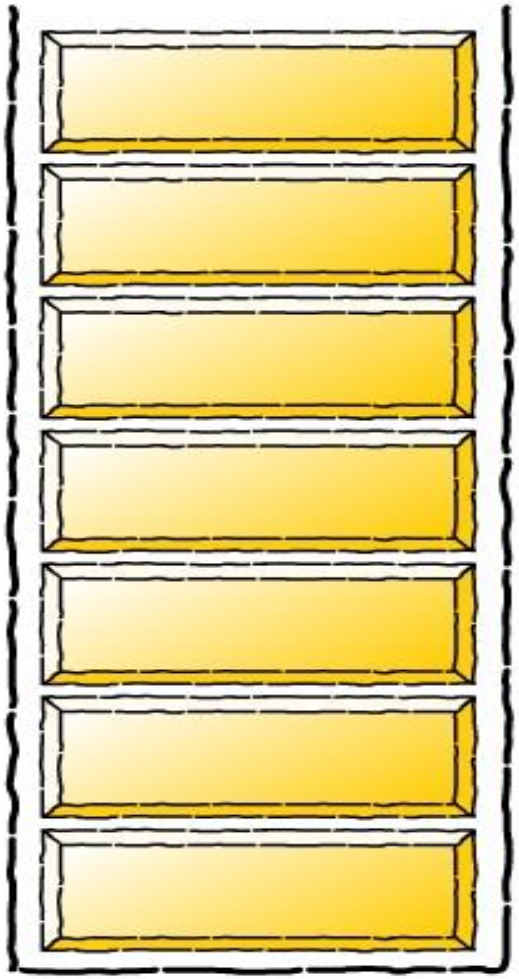
) * d - (e + f)

postfixVect

a b + c

Infix to postfix conversion

stack



infixVect

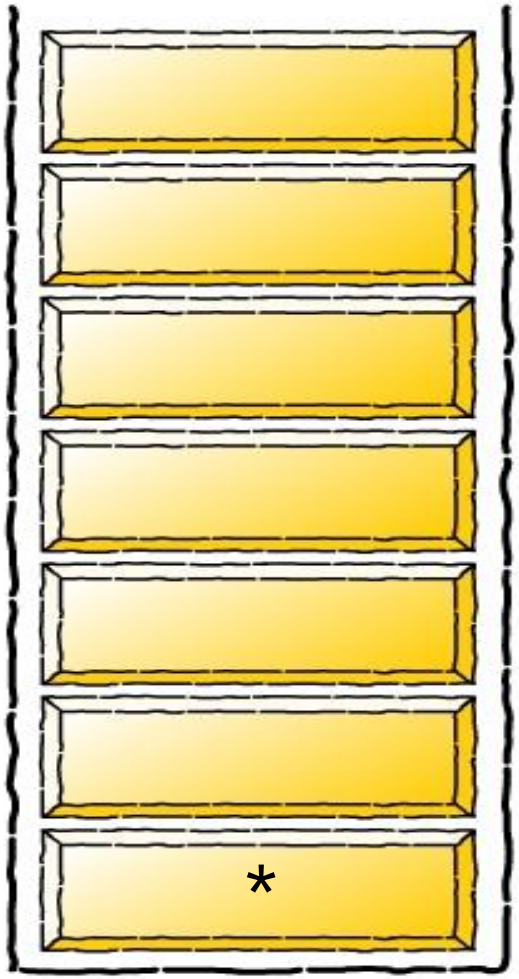
* d - (e + f)

postfixVect

a b + c -

Infix to postfix conversion

stack



infixVect

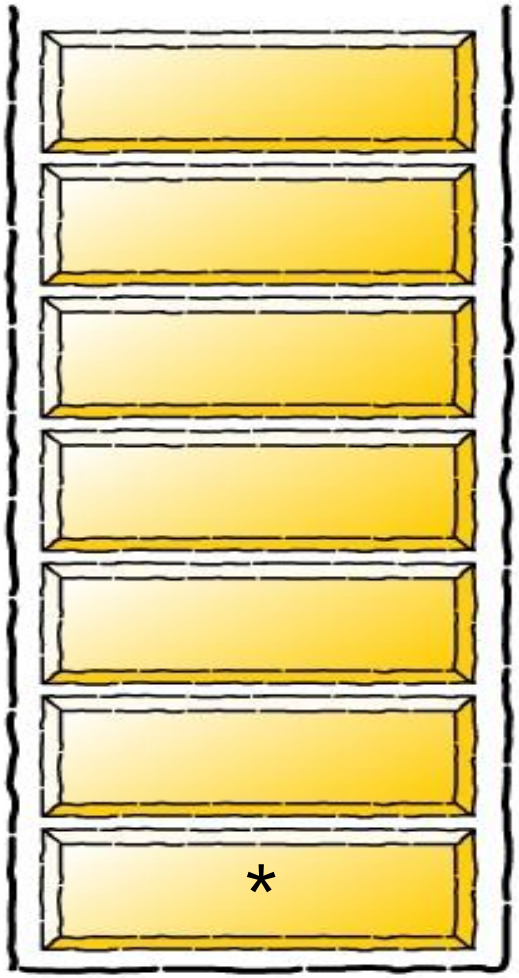
d - (e + f)

postfixVect

a b + c -

Infix to postfix conversion

stack



infixVect

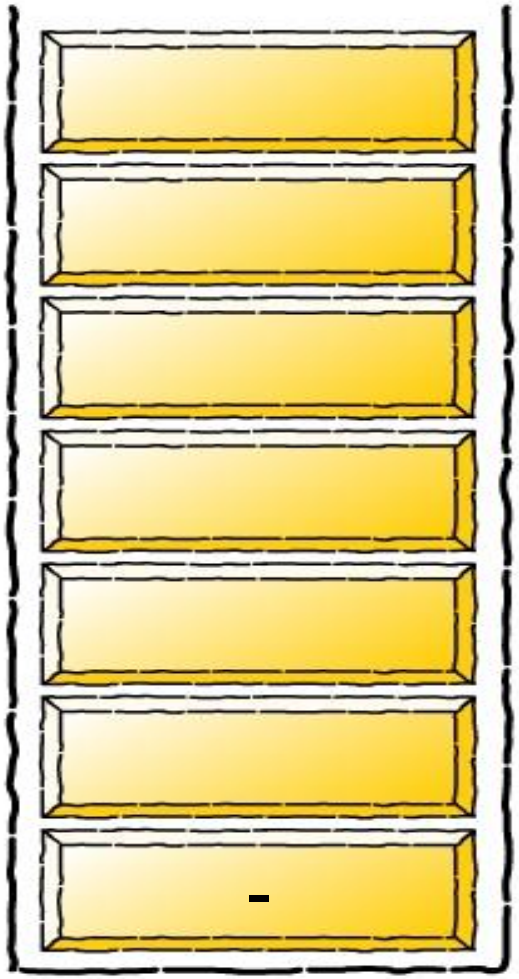
- (e + f)

postfixVect

a b + c - d

Infix to postfix conversion

stack



infixVect

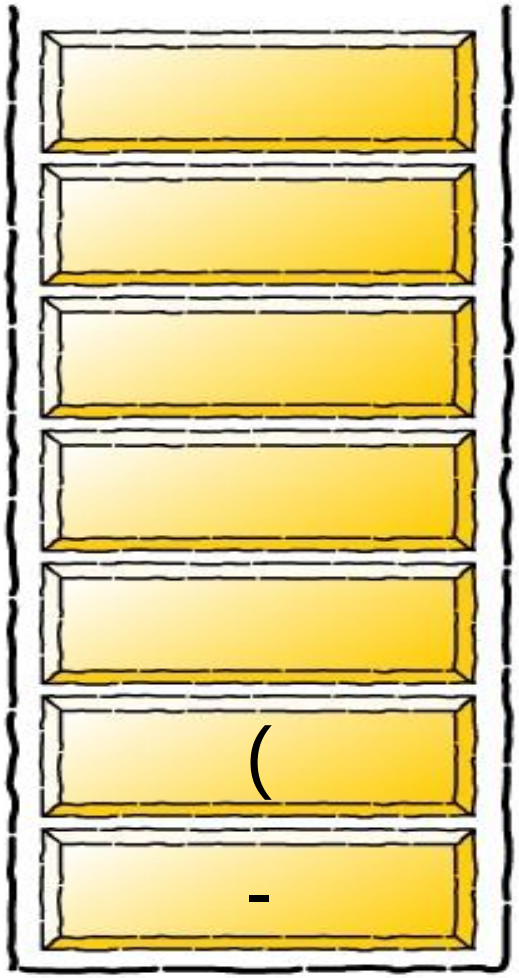
(e + f)

postfixVect

a b + c - d *

Infix to postfix conversion

stack



infixVect

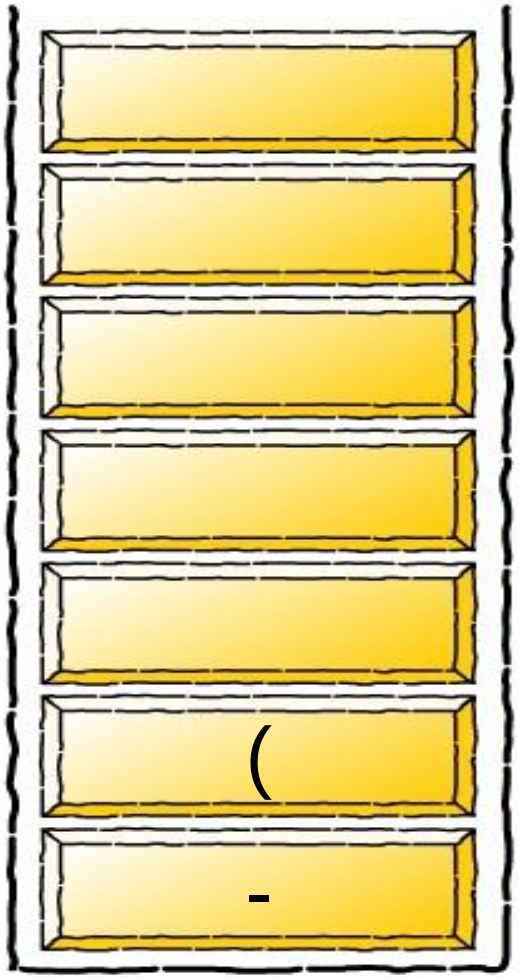
e + f)

postfixVect

a b + c - d *

Infix to postfix conversion

stack



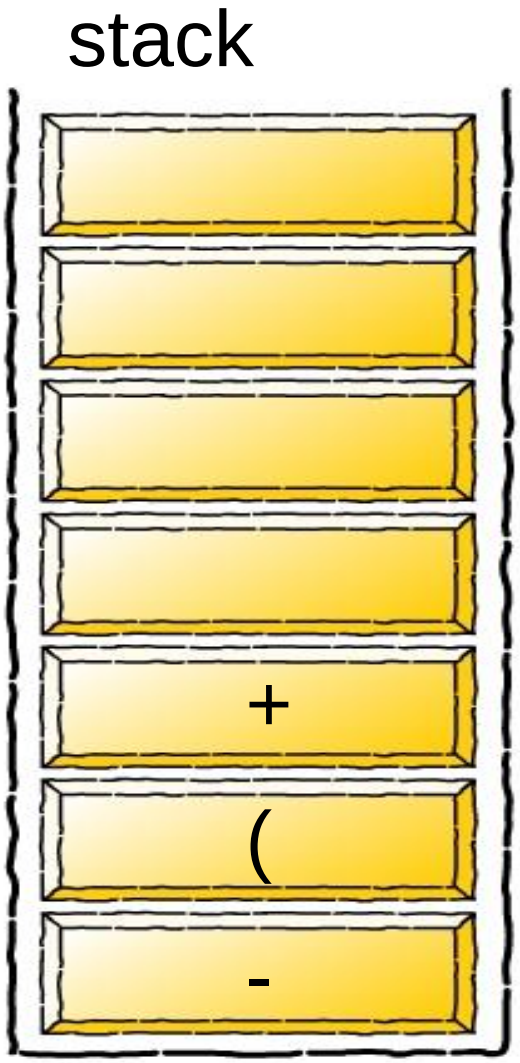
infixVect

+ f)

postfixVect

a b + c - d * e

Infix to postfix conversion



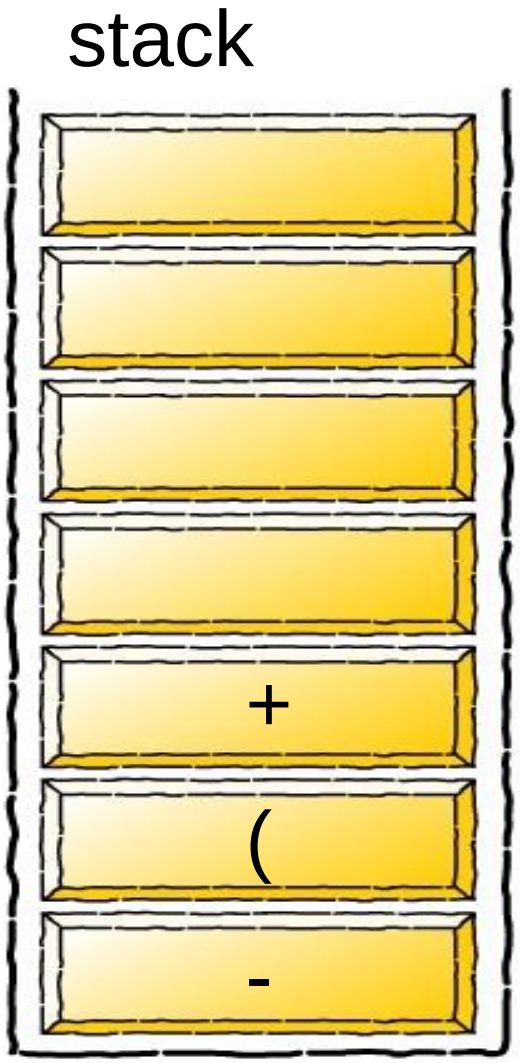
infixVect

f)

postfixVect

a b + c - d * e

Infix to postfix conversion



infixVect

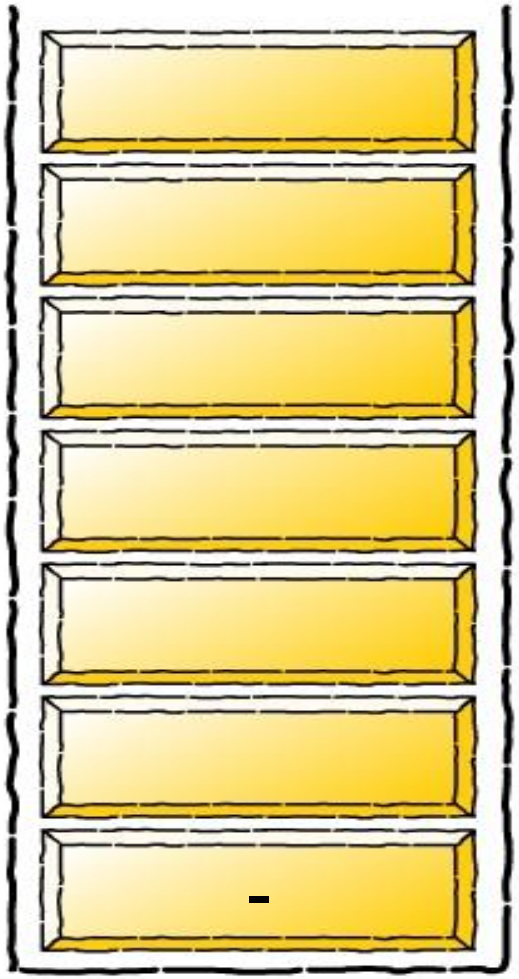
)

postfixVect

a b + c - d * e f

Infix to postfix conversion

stack



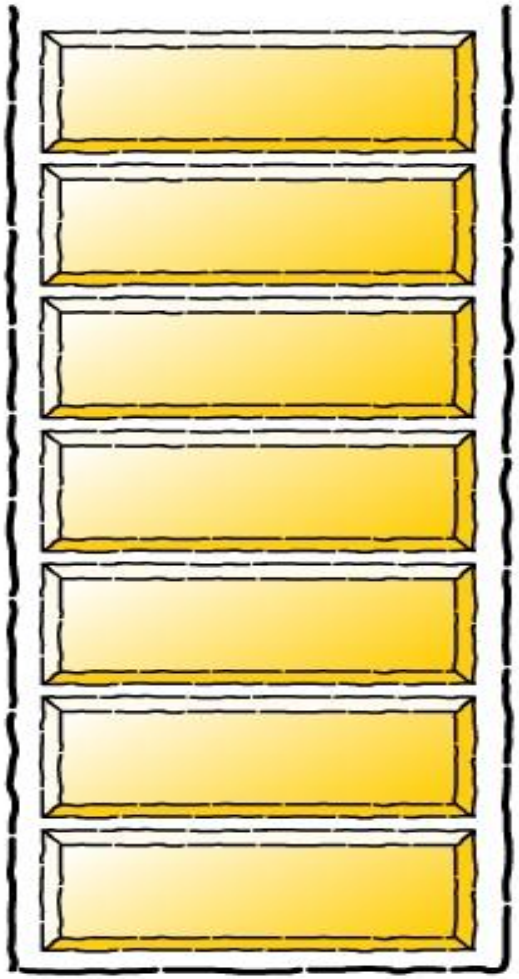
infixVect

postfixVect

a b + c - d * e f +

Infix to postfix conversion

stack



infixVect

postfixVect

a b + c - d * e f + -

Infix to postfix conversion

infixVect

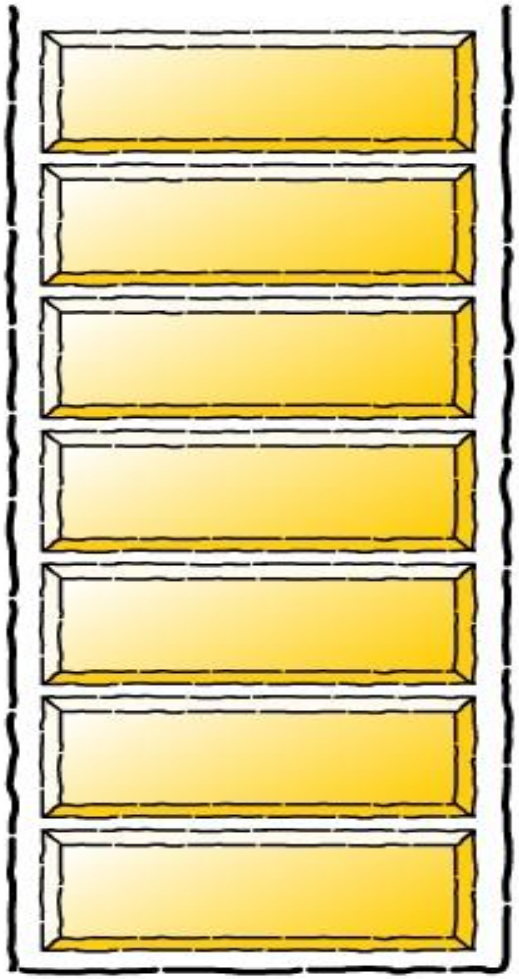
(a + b - c) * d - (e + f)

postfixVect

a b + c - d * e f + -

Infix to postfix conversion

stack



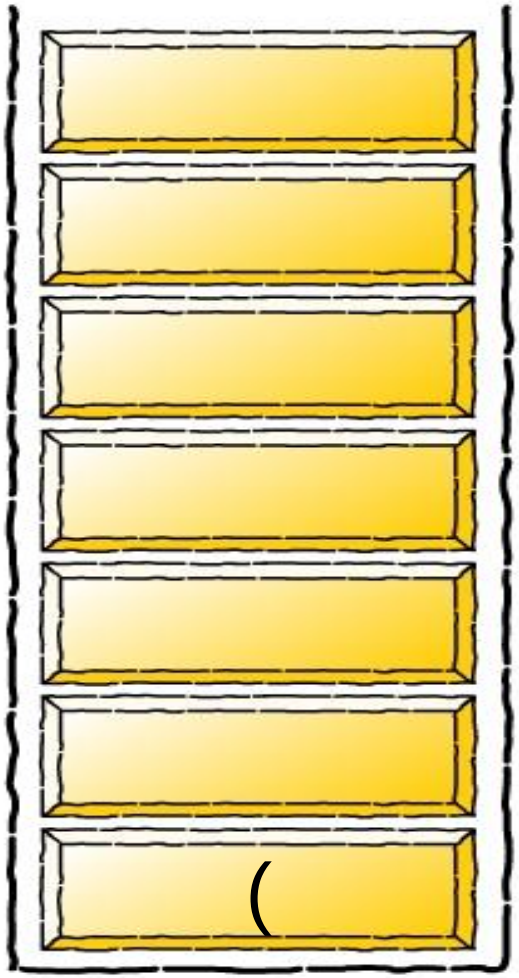
infixVect

(a + b * c)

postfixVect

Infix to postfix conversion

stack



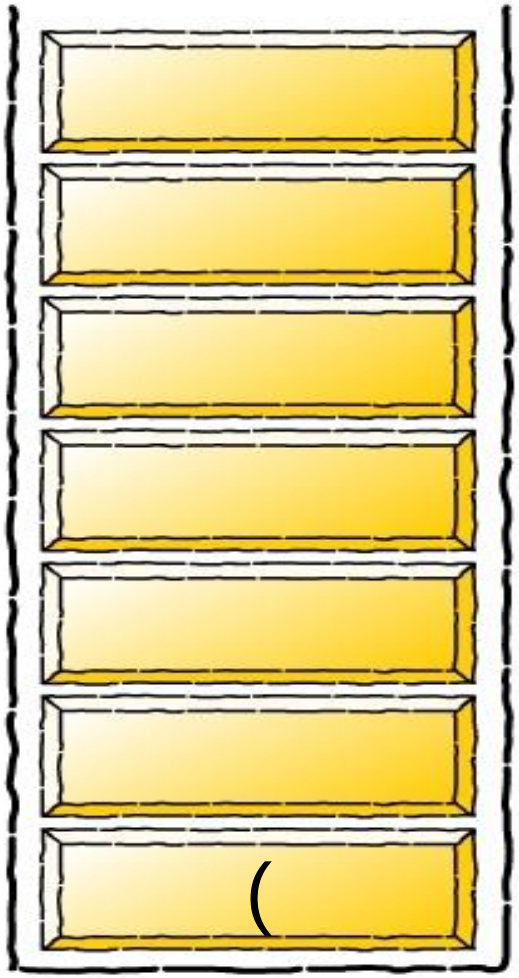
infixVect

a + b * c)

postfixVect

Infix to postfix conversion

stack



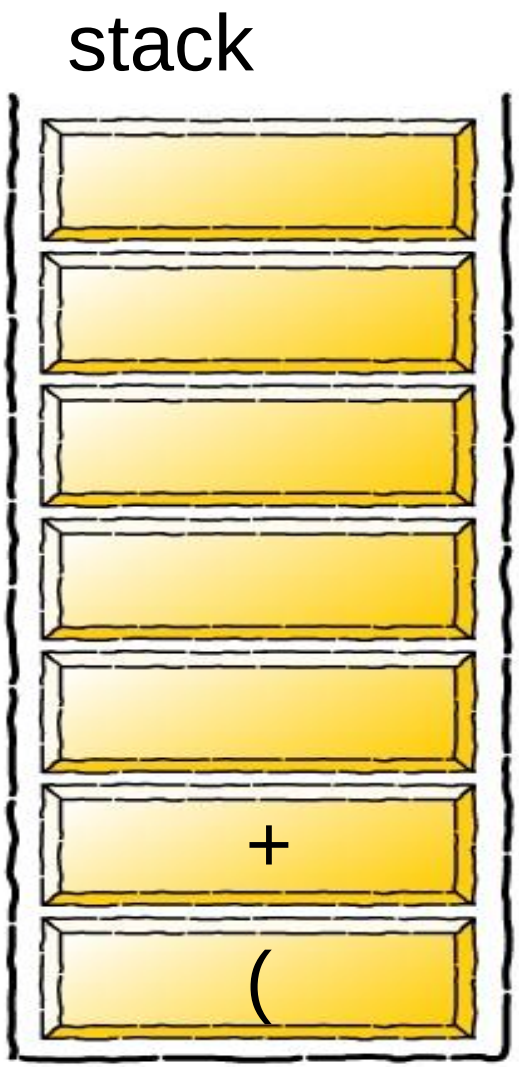
infixVect

+ b * c)

postfixVect

a

Infix to postfix conversion



infixVect

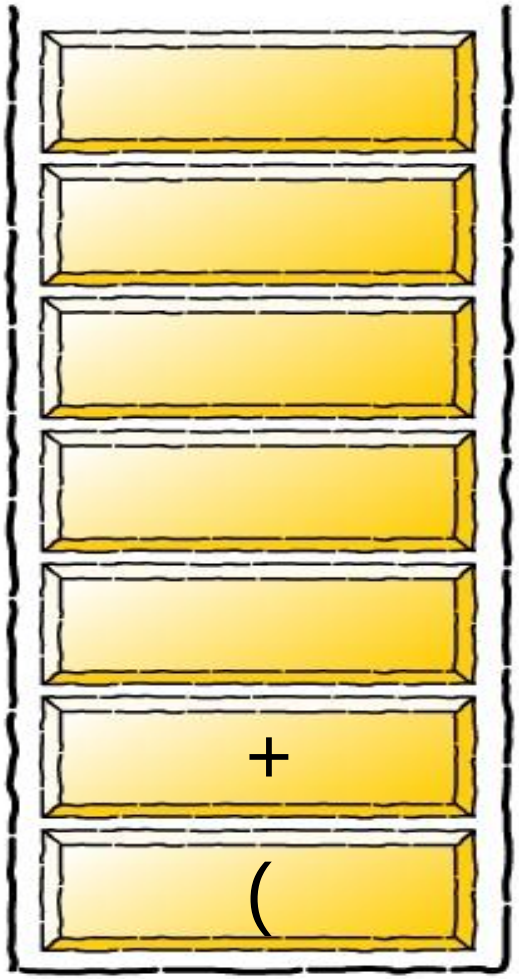
b * c)

postfixVect

a

Infix to postfix conversion

stack



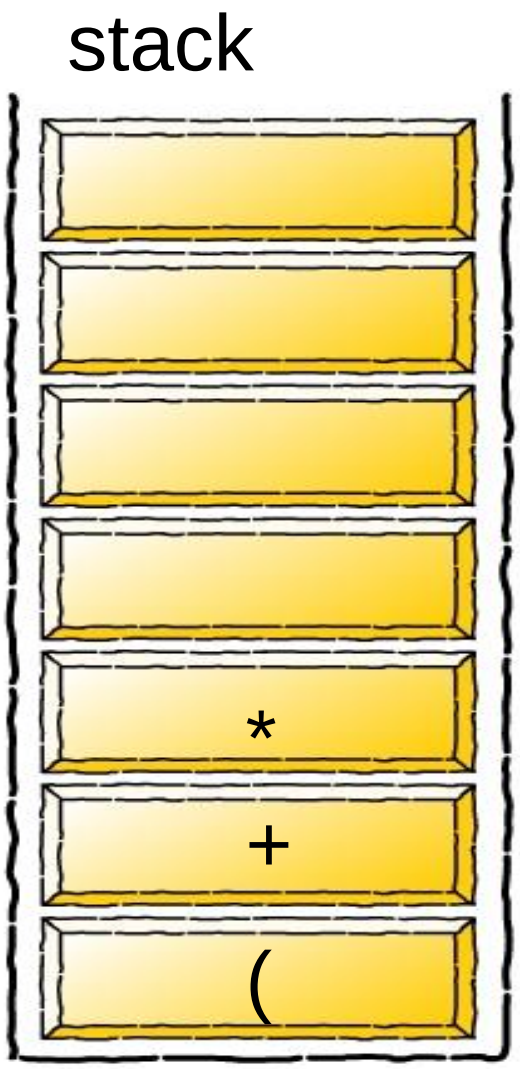
infixVect

* c)

postfixVect

a b

Infix to postfix conversion



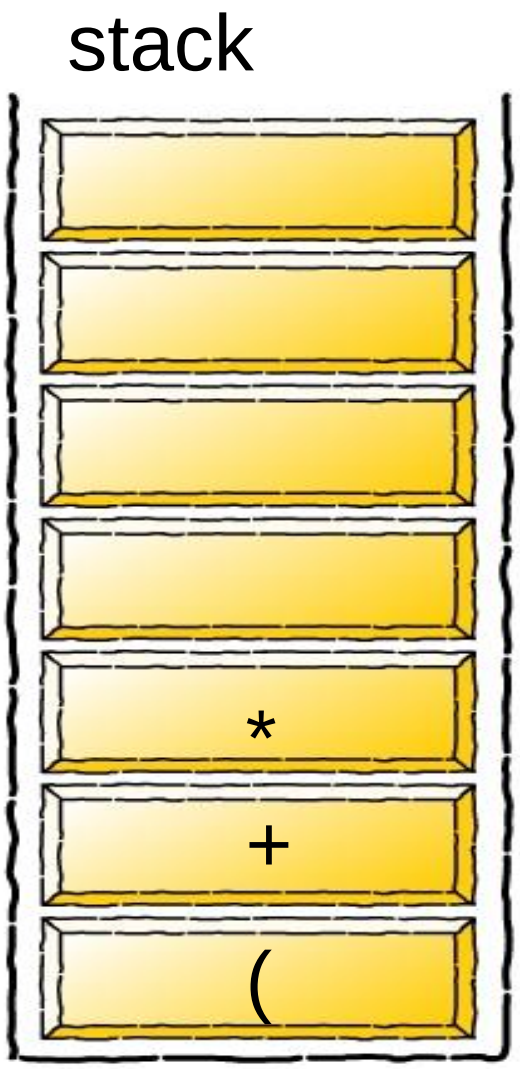
infixVect

c)

postfixVect

a b

Infix to postfix conversion



infixVect

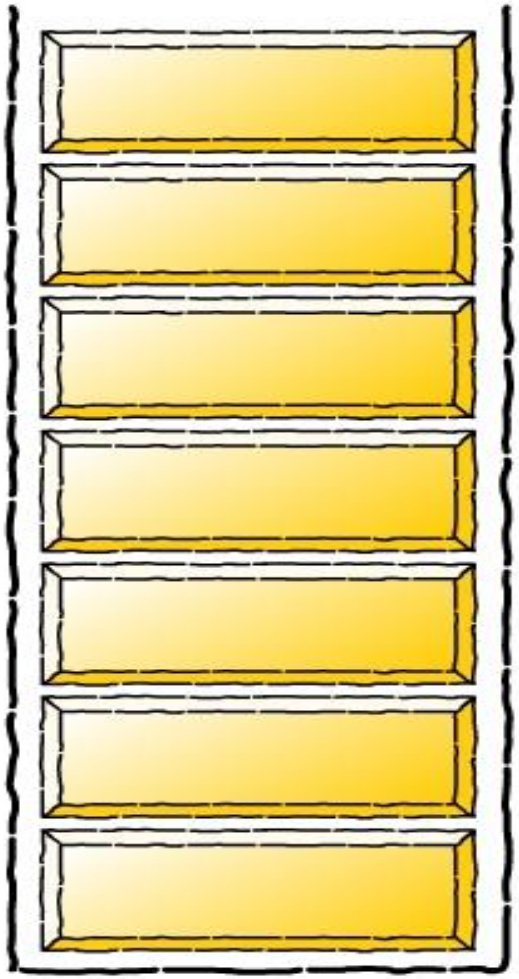
)

postfixVect

a b c

Infix to postfix conversion

stack



infixVect

postfixVect

a b c * +

Postfix Evaluation

Postfix evaluation

For each token in the postfix from left to right

    if (token = Operand)

        Push Operand to the stack

    if (token = Operator)

        Second = Pop from the stack

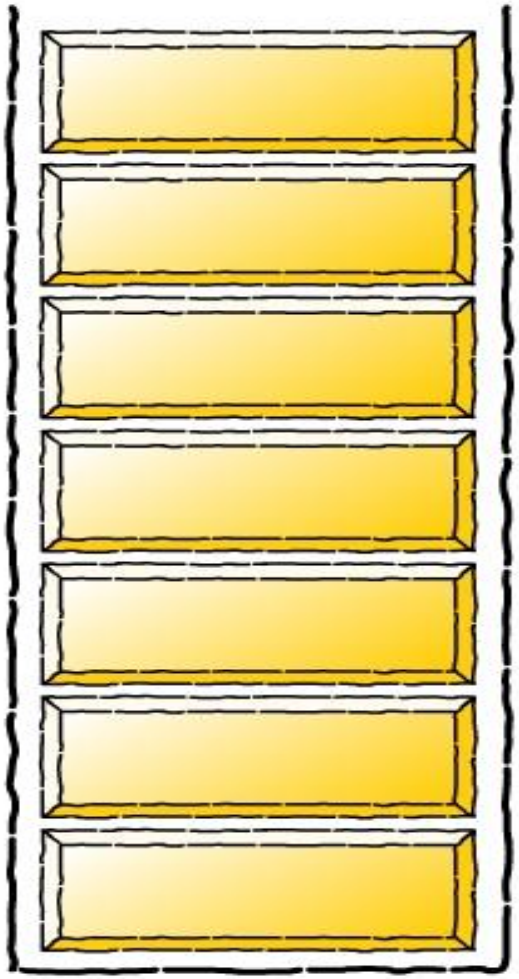
        First = Pop from the stack

        Push First Operator Second to the stack

Return Stack[top]

Postfix evaluation

stack



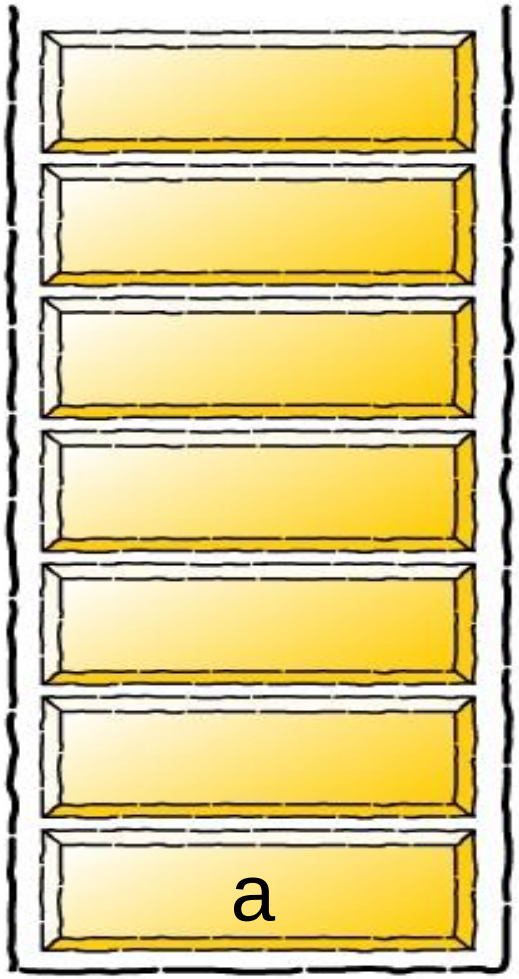
(a + b - c) * d - (e + f)

postfixVect

a b + c - d * e f + -

Postfix evaluation

stack



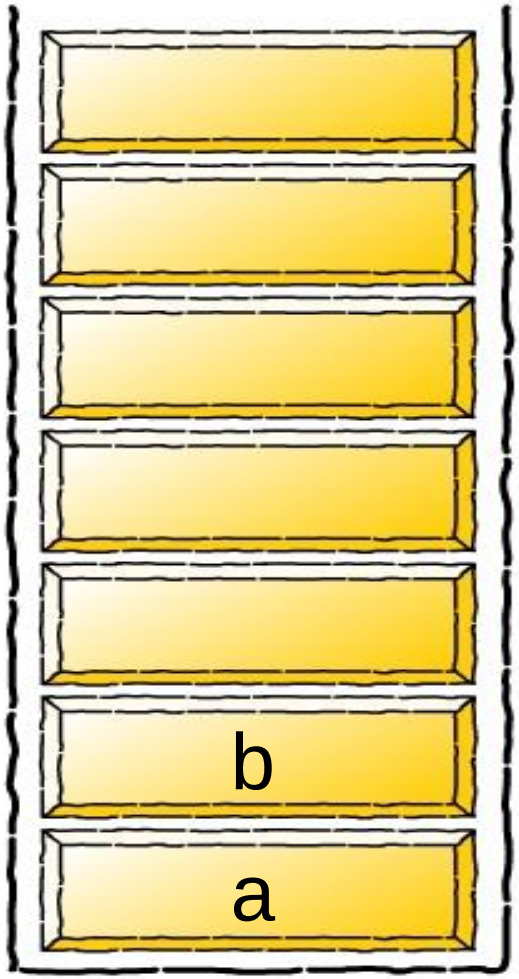
(a + b - c) * d - (e + f)

postfixVect

b + c - d * e f + -

Postfix evaluation

stack



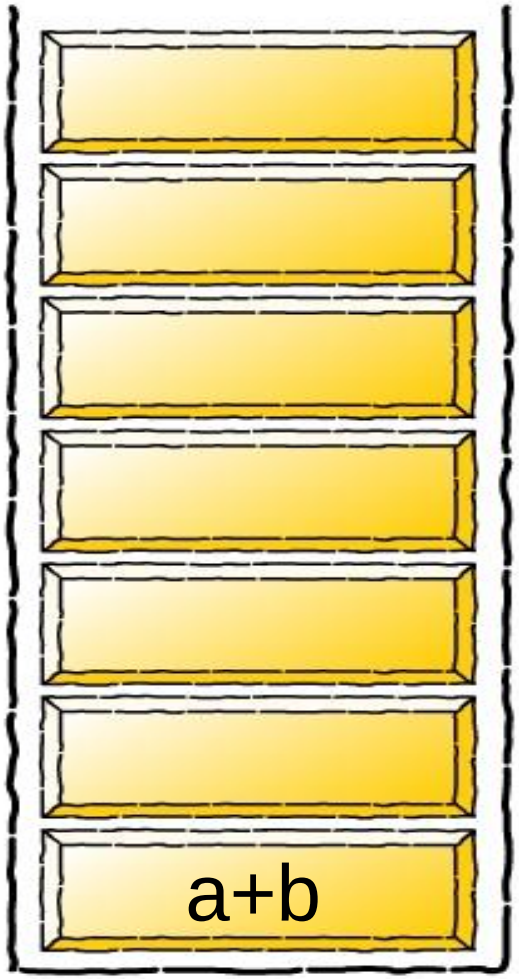
(a + b - c) * d - (e + f)

postfixVect

+ c - d * e f + -

Postfix evaluation

stack



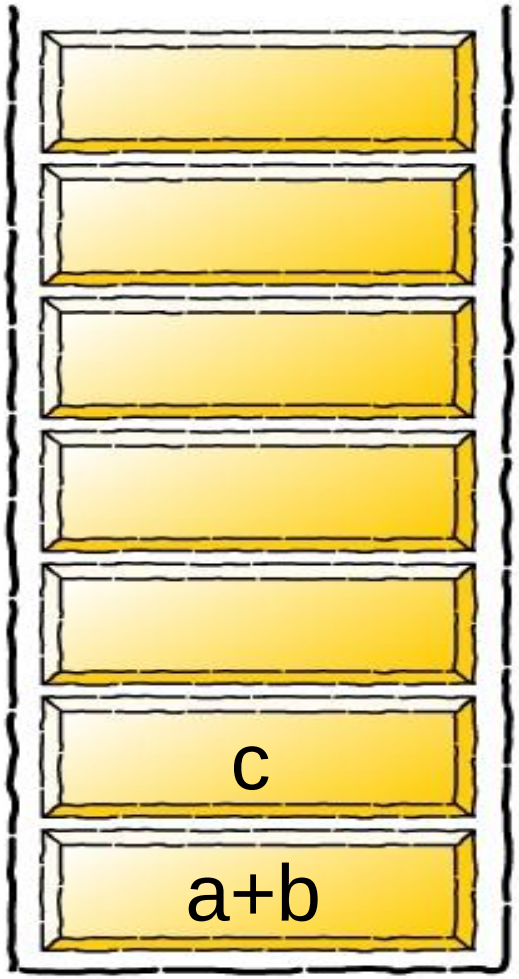
(a + b - c) * d - (e + f)

postfixVect

c - d * e f + -

Postfix evaluation

stack



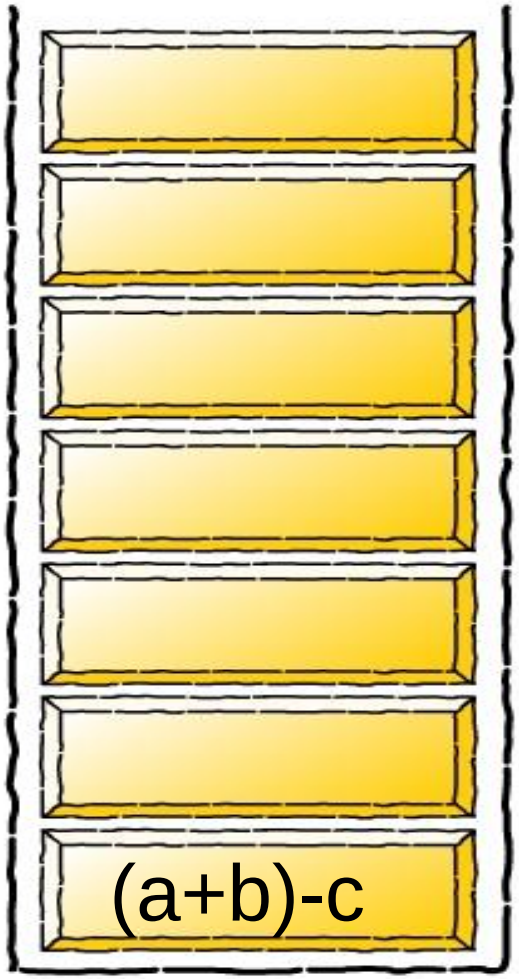
(a + b - c) * d - (e + f)

postfixVect

- d * e f + -

Postfix evaluation

stack



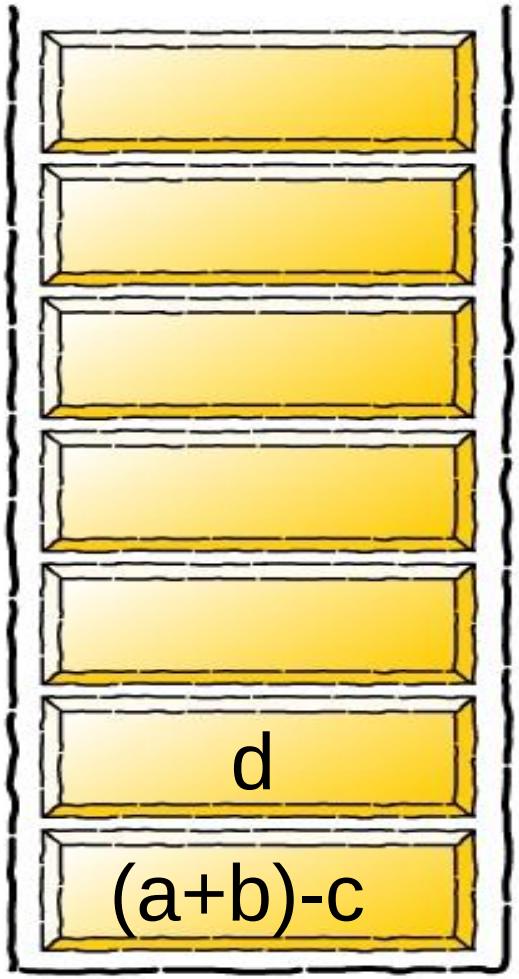
(a + b - c) * d - (e + f)

postfixVect

d * e f + -

Postfix evaluation

stack



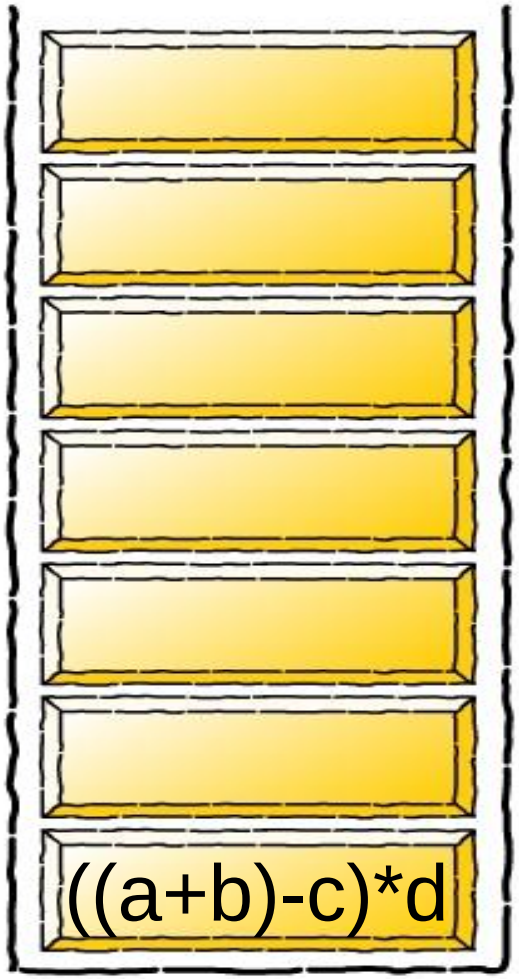
(a + b - c) * d - (e + f)

postfixVect

* e f + -

Postfix evaluation

stack

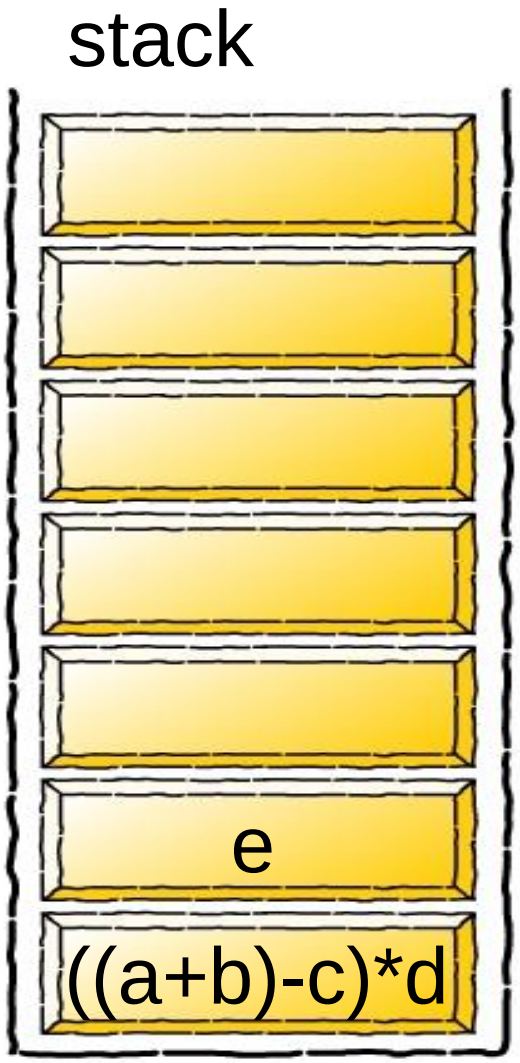


(a + b - c) * d - (e + f)

postfixVect

e f + -

Postfix evaluation



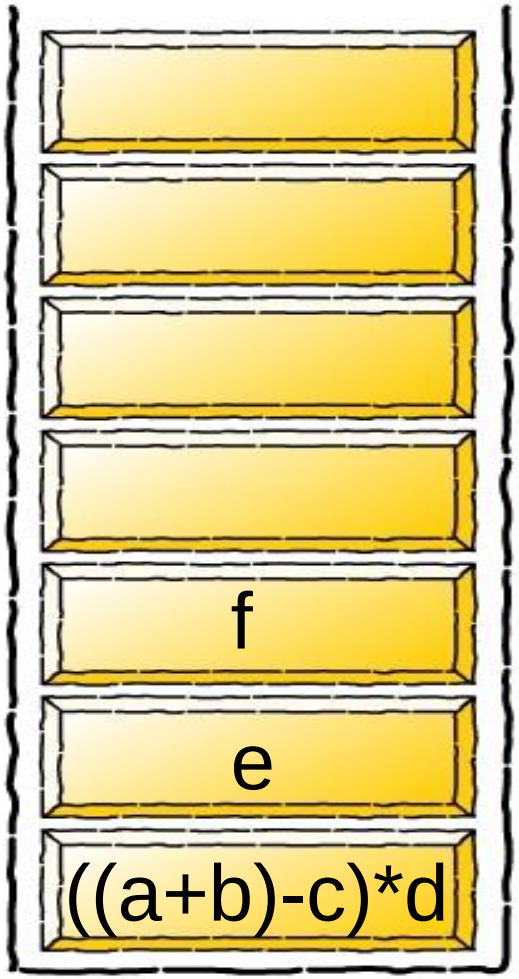
(a + b - c) * d - (e + f)

postfixVect

f + -

Postfix evaluation

stack

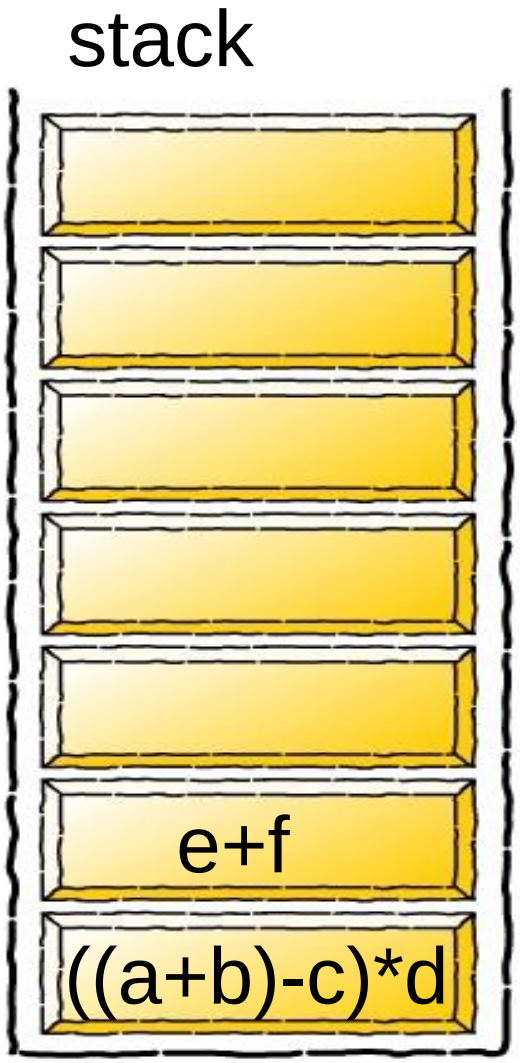


(a + b - c) * d - (e + f)

postfixVect

+ -

Postfix evaluation



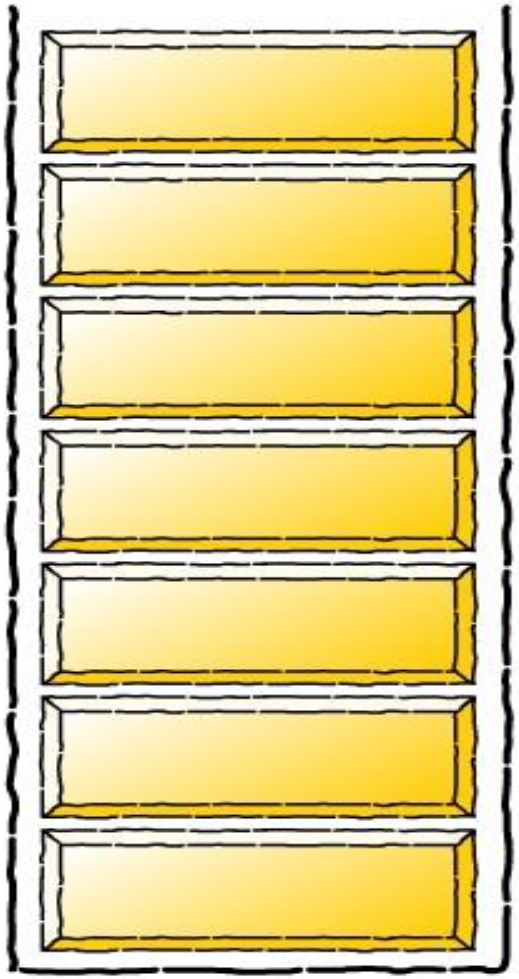
(a + b - c) * d - (e + f)

postfixVect

-

Postfix evaluation

stack



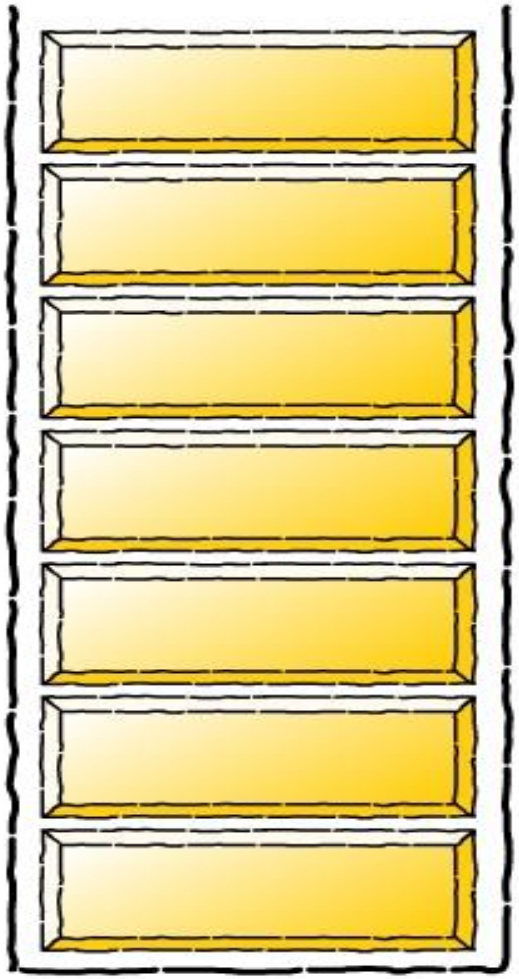
(a + b - c) * d - (e + f)

postfixVect

(((a+b)-c)*d)-(e+f)

Postfix evaluation

stack



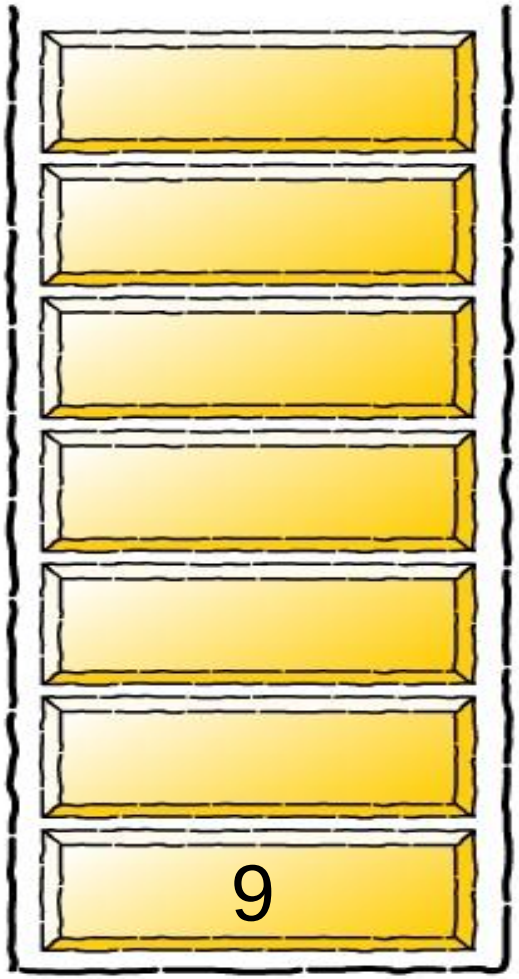
(9 + 4 - 6) * 2 - (3 + 6) = 5

postfixVect

9 4 + 6 - 2 * 3 6 + -

Postfix evaluation

stack



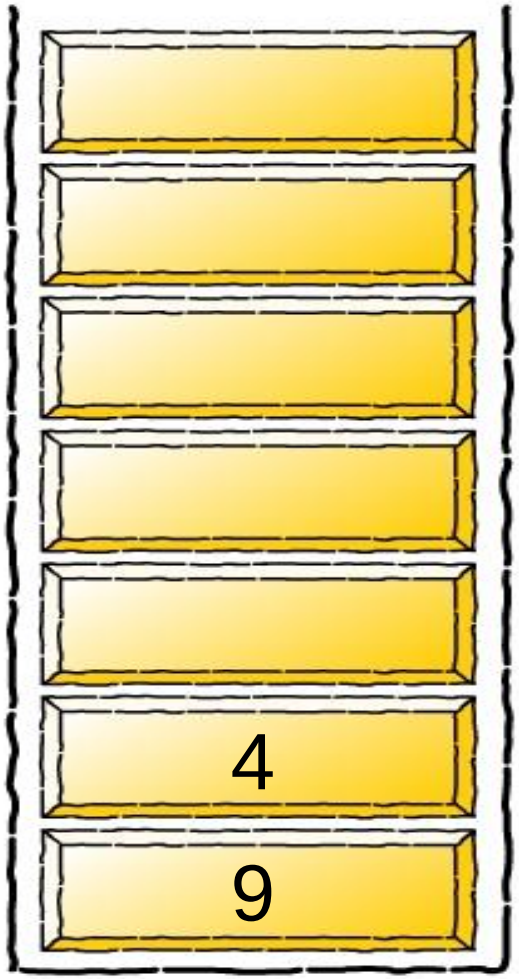
(9 + 4 - 6) * 2 - (3 + 6) = 5

postfixVect

4 + 6 - 2 * 3 6 + -

Postfix evaluation

stack



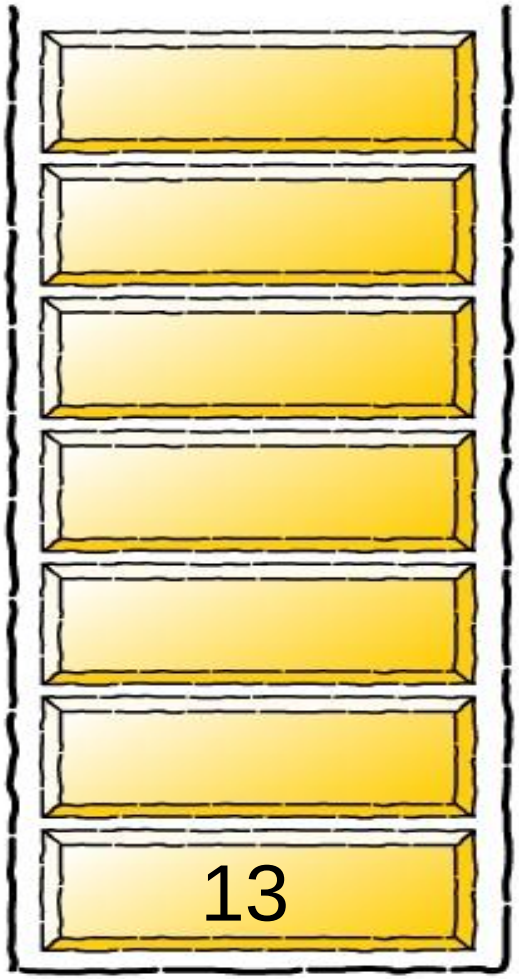
(9 + 4 - 6) * 2 - (3 + 6) = 5

postfixVect

+ 6 - 2 * 3 6 + -

Postfix evaluation

stack



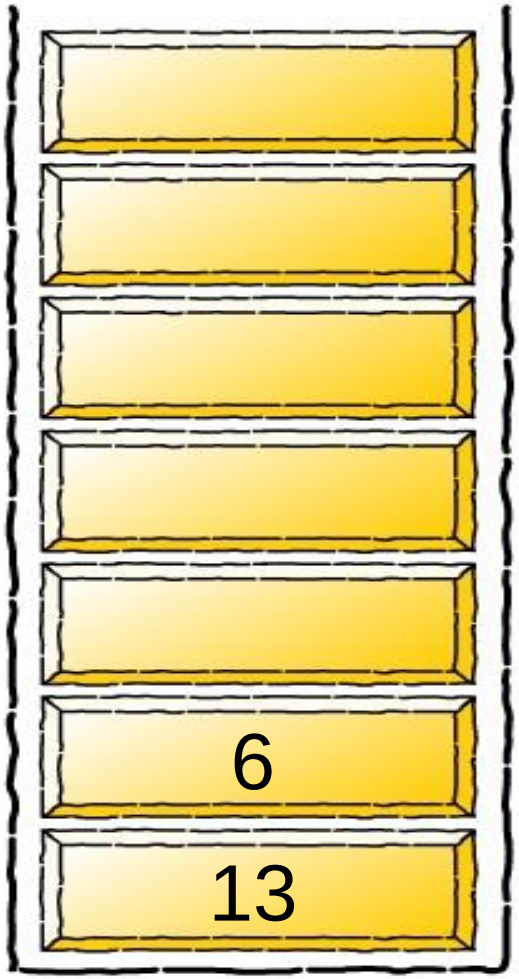
(9 + 4 - 6) * 2 - (3 + 6) = 5

postfixVect

6 - 2 * 3 6 + -

Postfix evaluation

stack



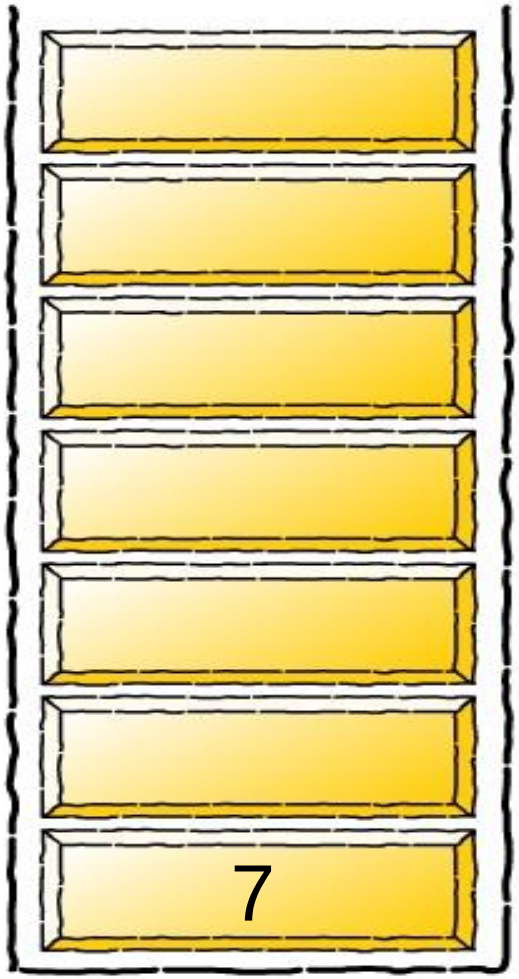
(9 + 4 - 6) * 2 - (3 + 6) = 5

postfixVect

- 2 * 3 6 + -

Postfix evaluation

stack



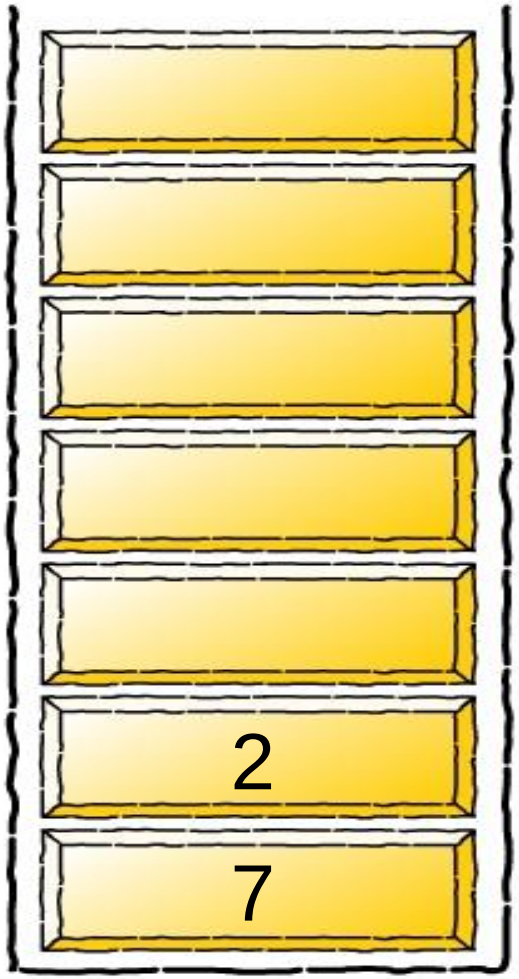
(9 + 4 - 6) * 2 - (3 + 6) = 5

postfixVect

2 * 3 6 + -

Postfix evaluation

stack



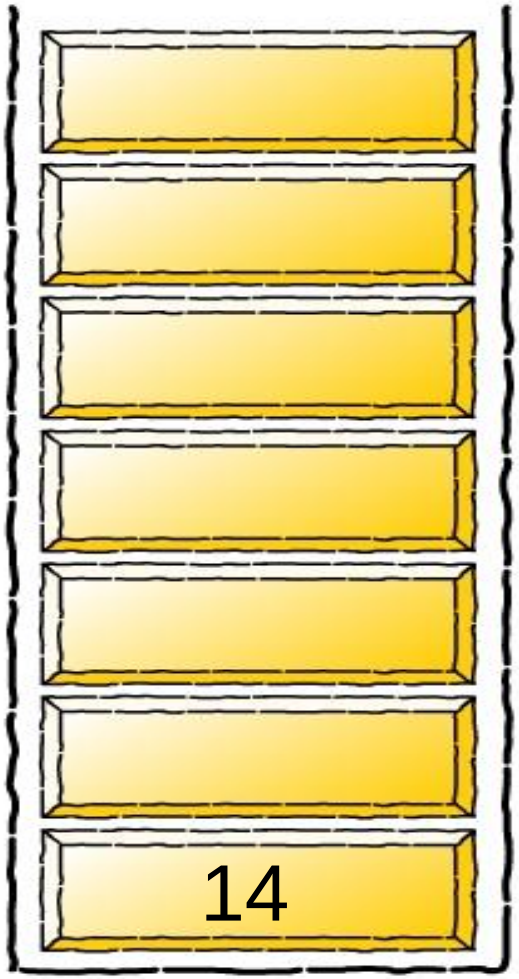
(9 + 4 - 6) * 2 - (3 + 6) = 5

postfixVect

* 3 6 + -

Postfix evaluation

stack



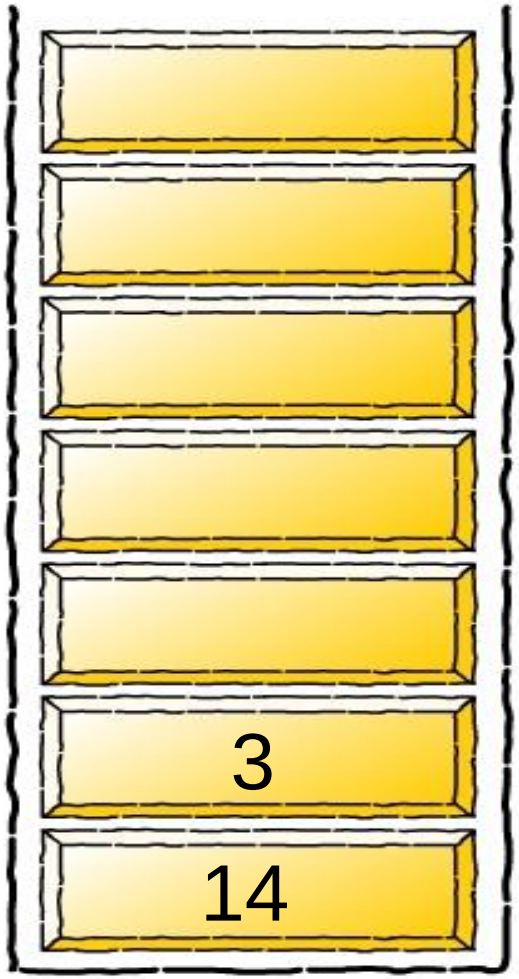
(9 + 4 - 6) * 2 - (3 + 6) = 5

postfixVect

3 6 + -

Postfix evaluation

stack



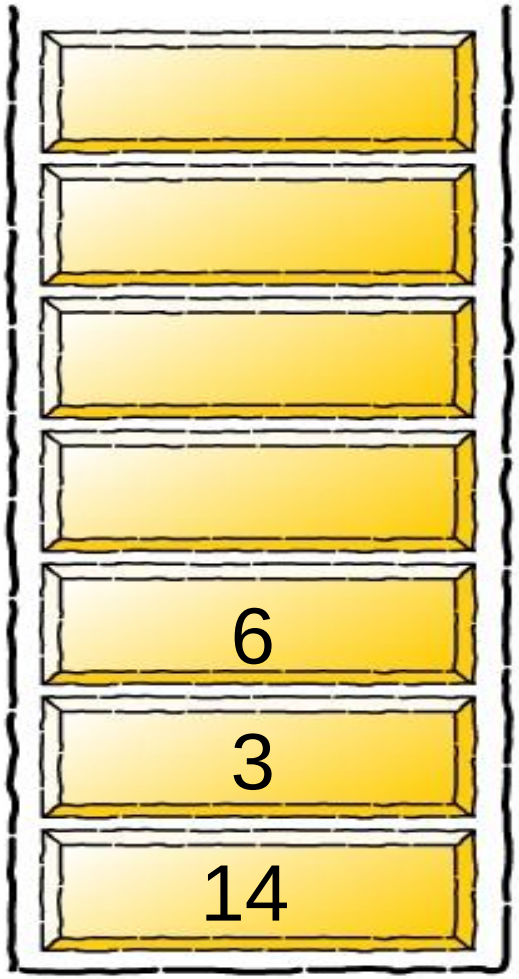
(9 + 4 - 6) * 2 - (3 + 6) = 5

postfixVect

6 + -

Postfix evaluation

stack



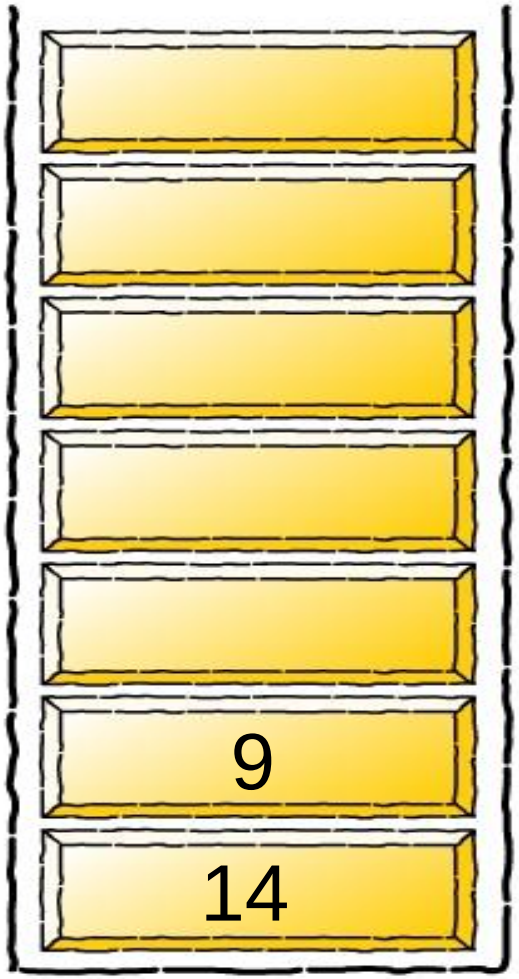
(9 + 4 - 6) * 2 - (3 + 6) = 5

postfixVect

+ -

Postfix evaluation

stack



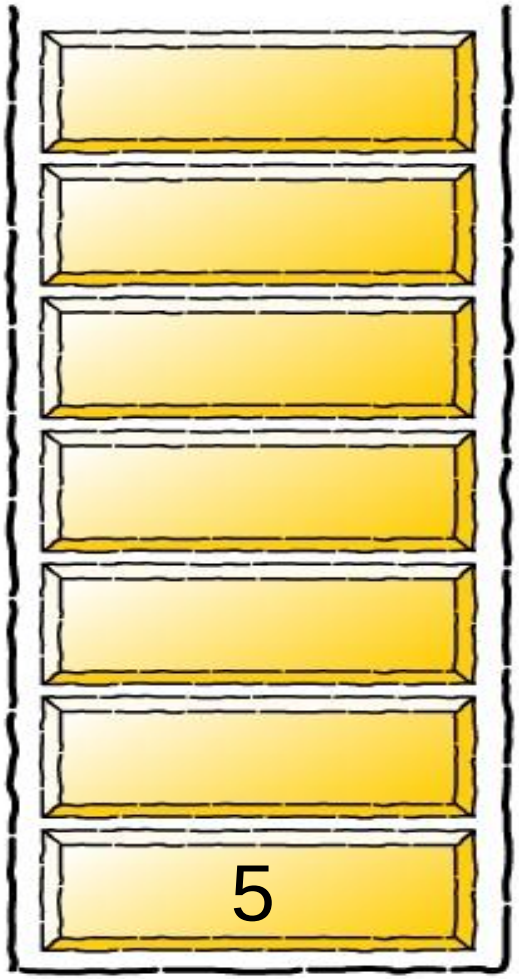
(9 + 4 - 6) * 2 - (3 + 6) = 5

postfixVect

-

Postfix evaluation

stack



(9 + 4 - 6) * 2 - (3 + 6) = 5

postfixVect

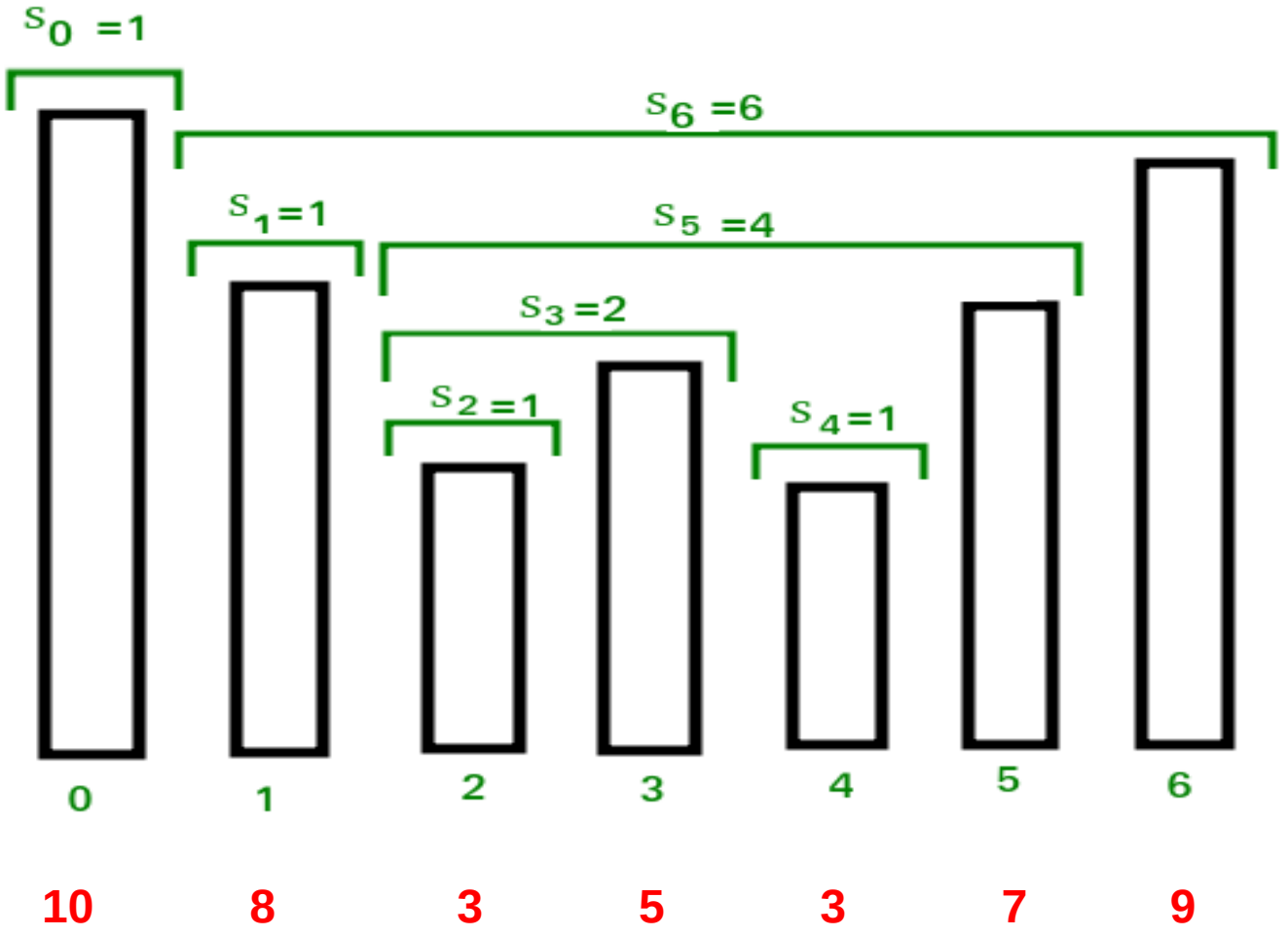
Share Span Problem

Share Span Problem

- **Input**: A sequence of n prices for a share for n consecutive days
- **Output**: Calculate span of share's price for all n days.

- Share Span for Day i
  - Span[i]: Maximum no of consecutive days on or before i-th day such that price(i) >= price(all the previous consecutive days)

Share Span Problem



Share Span Problem

- **Input**: A sequence of n prices for a share for n consecutive days
- **Output**: Calculate span of share's price for all n days.
- **Example**:
- **Input**: n = 7 and

  Price[7] = {10, 9, 6, 7, 6, 7, 8}

  Output: Span[7] = { 1, 1, 1, 2, 1, 4, 5}
- **Input**: n = 7 and

  Price[7] = {5, 9, 6, 7, 6, 7, 10}

  Output: Span[7] = {1, 2, 1, 2, 1, 4, 7}

Naïve Solution

```
Span[ 0 ] = 1;
For(i = 1; i < n; i++) {
        Span[ i ] = 1;
        For(j = i - 1; j >= 0; j--)
                if(A[ j ] <= A[ i ])
                        Span[ i ] = Span[ i ] + 1;
}
```

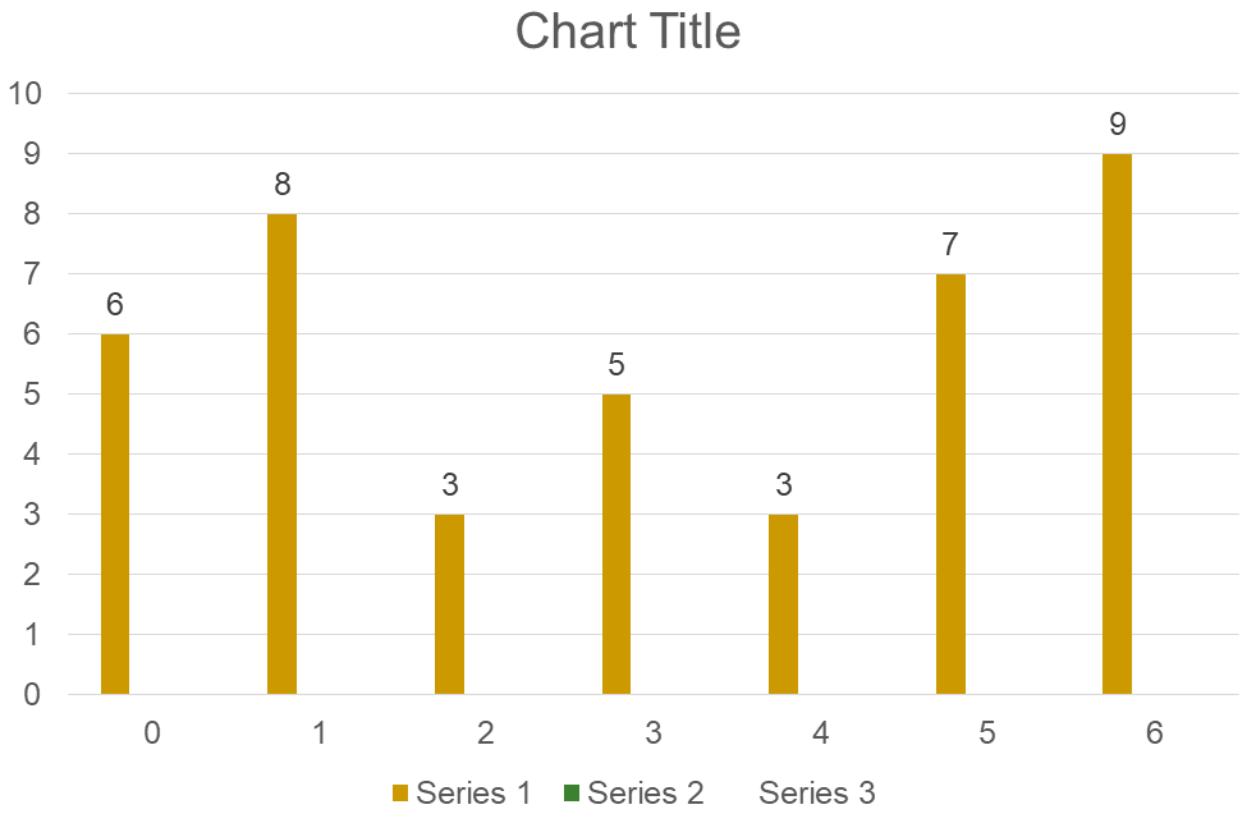
Time complexity = $O(n(n-1)/2) = O(n^2)$

Can we do better? Yes (using stack)

Linear Time Solution

```
stack.push(0);
span[0] = 1;
for (i = 1; i < n; i++) {
    while (stack is not empty and price[top of stack] <= price[i])
        stack.pop();
    If (stack becomes empty) then it implies price[i] is greater than all
    elements on left of it, i.e., price[0], price[1], ..price[i-1].
        span[i] = i + 1;
    Else price[i] is less than the element at top of stack
        span[i] = i - stack.top();
     stack.push(i);
}
```

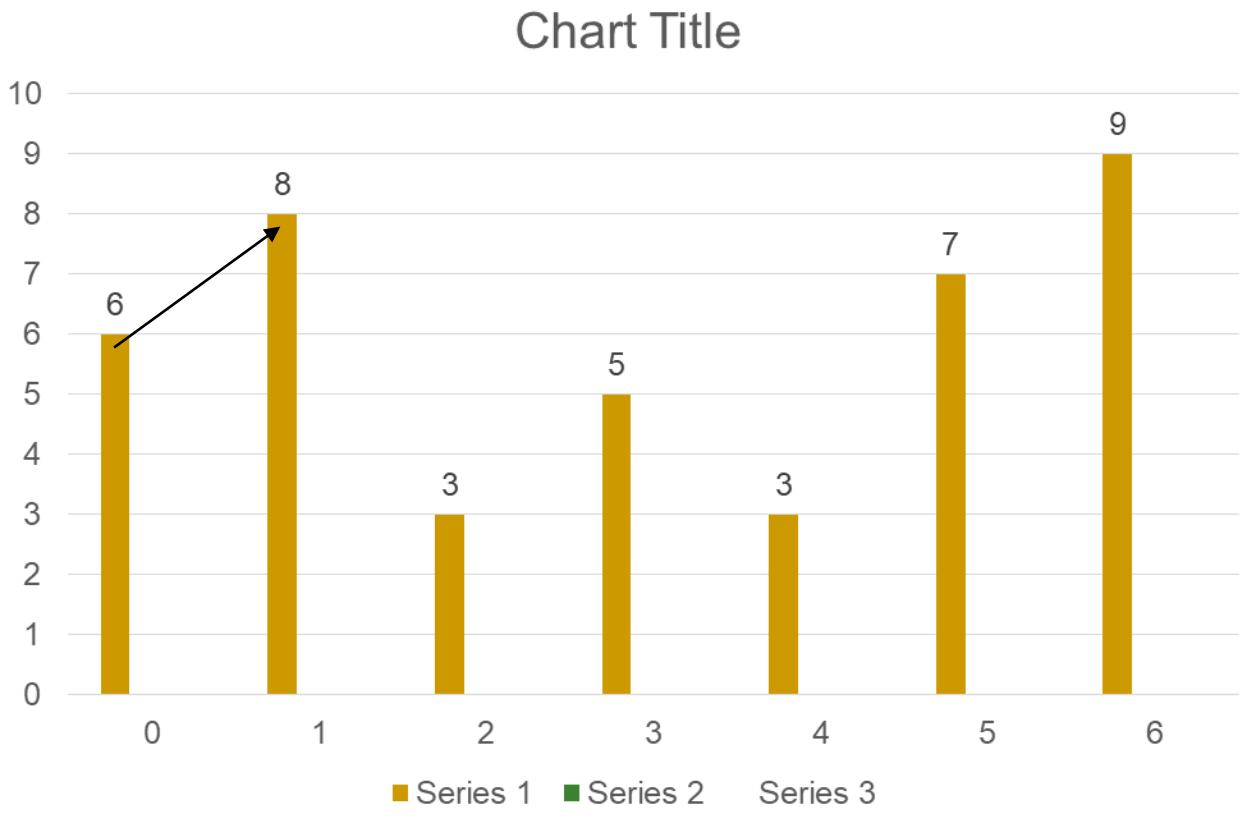
I = 0 Push(0)



Stack 0

Span 1

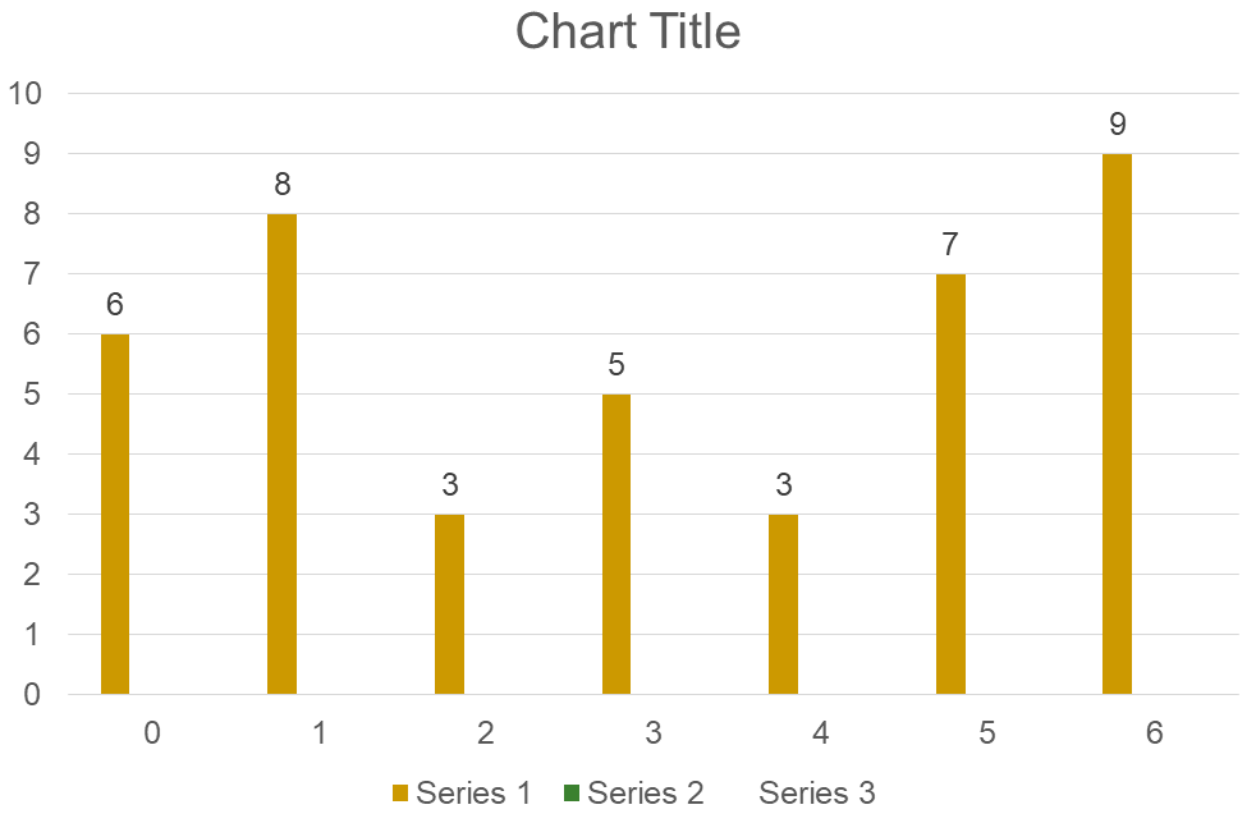
I = 1



Stack 0

Span 1

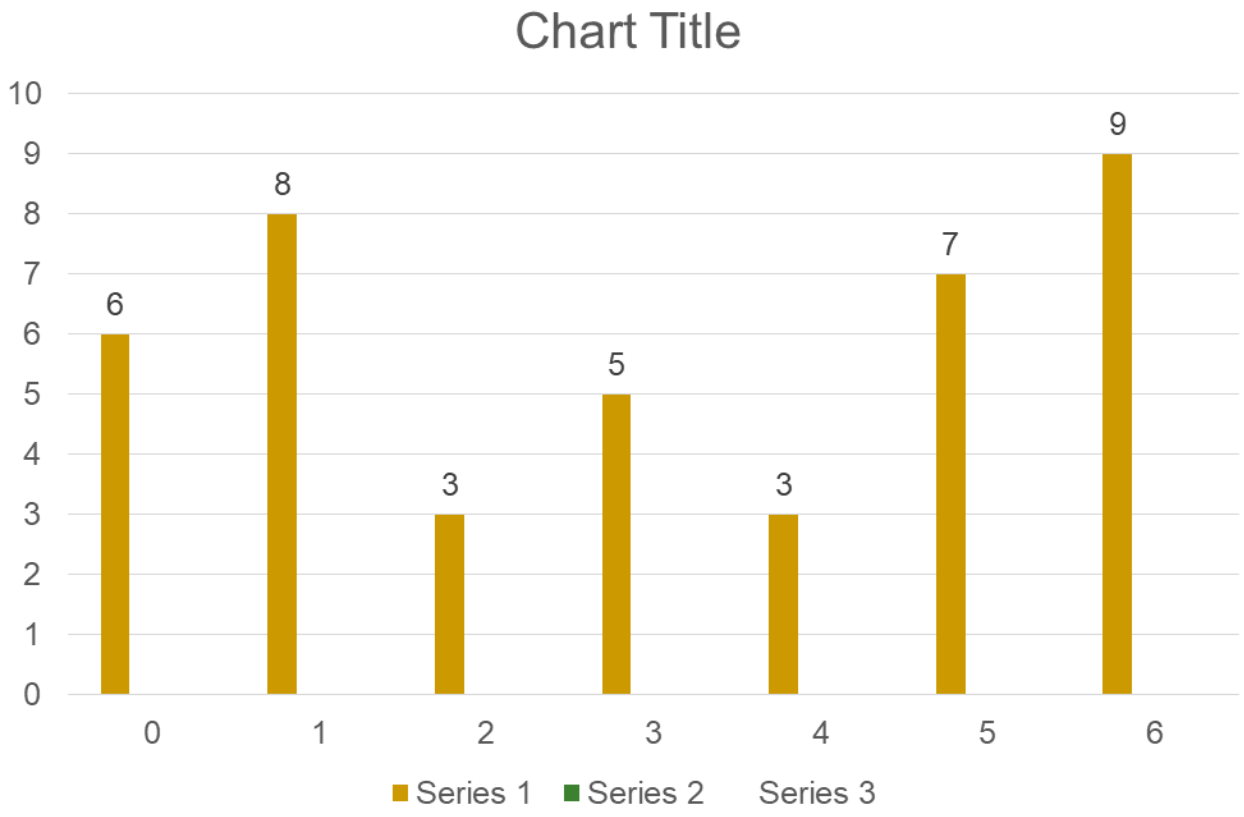
Pop



Stack

Span 1

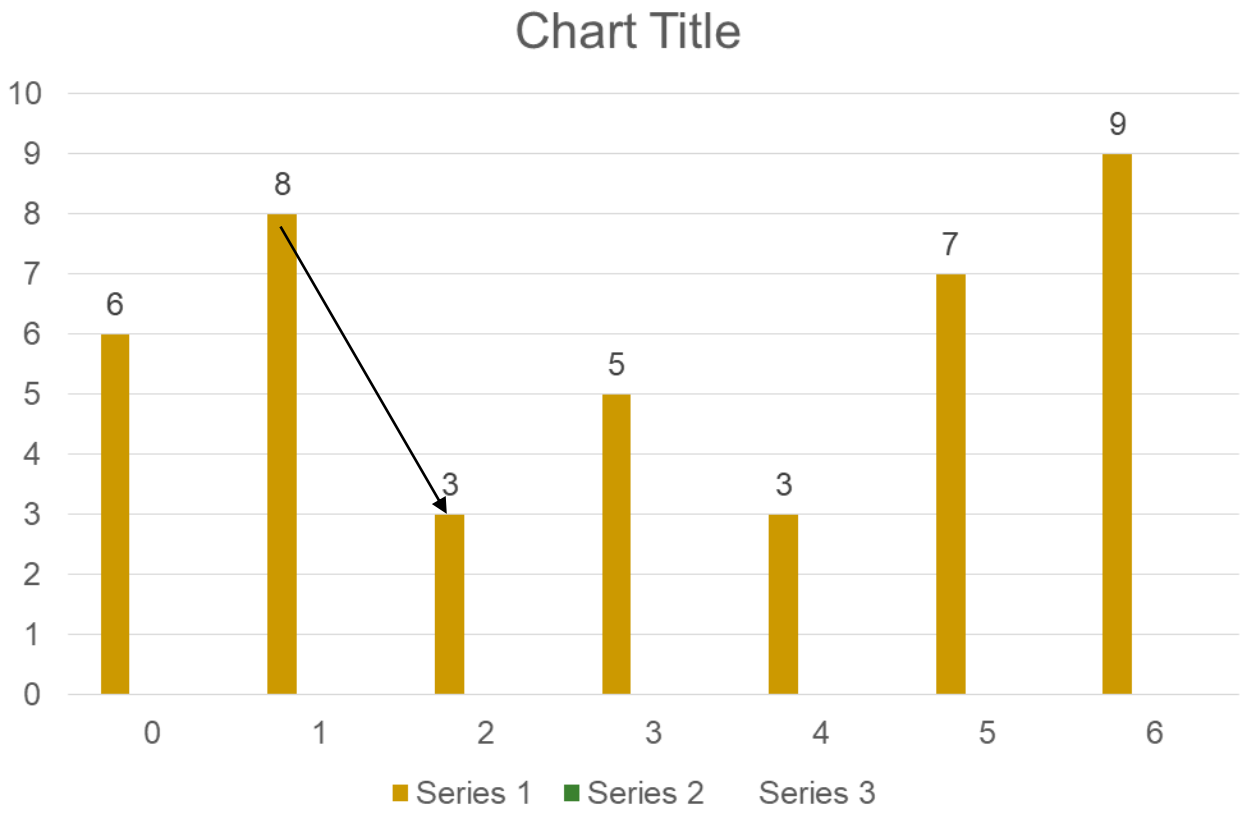
Push(1)



Stack 1

Span **1** **2**

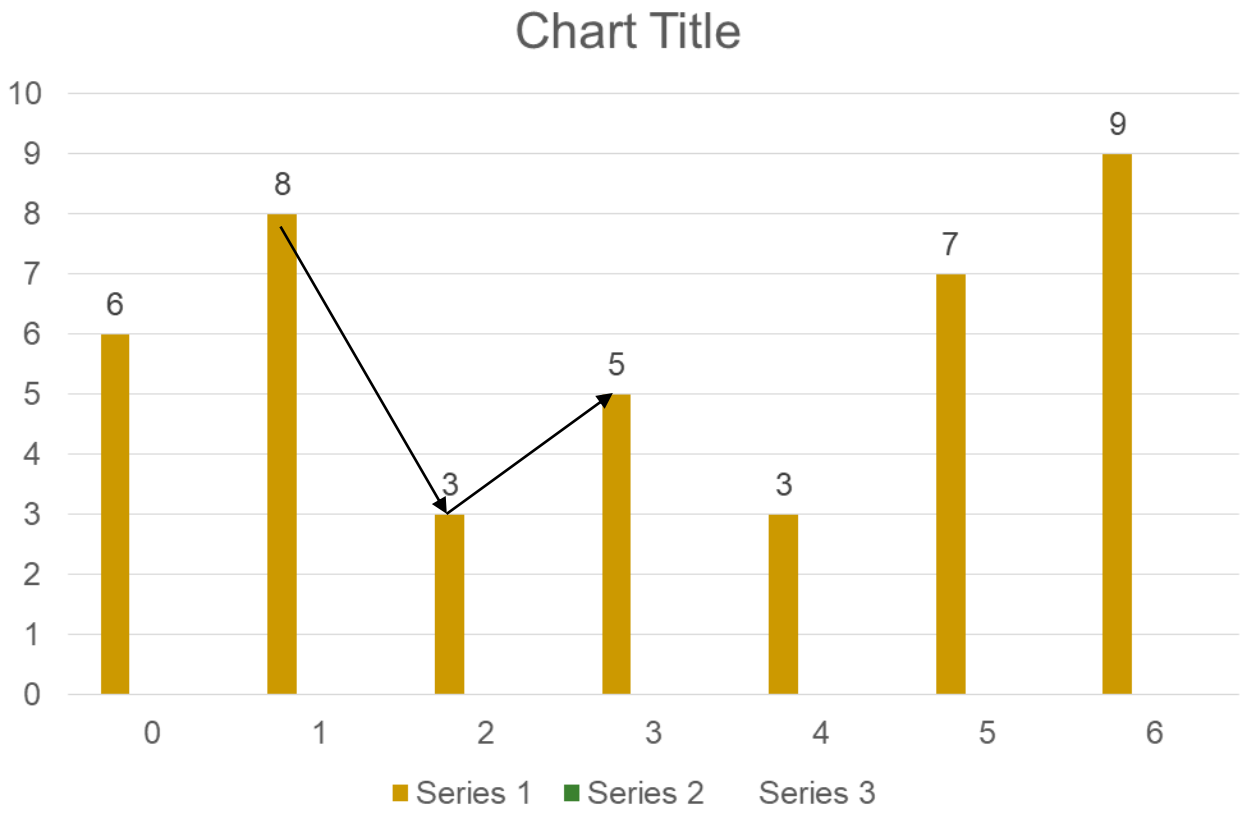
I = 2 Push(2)



Stack 1 2

Span **1** **2** **1**

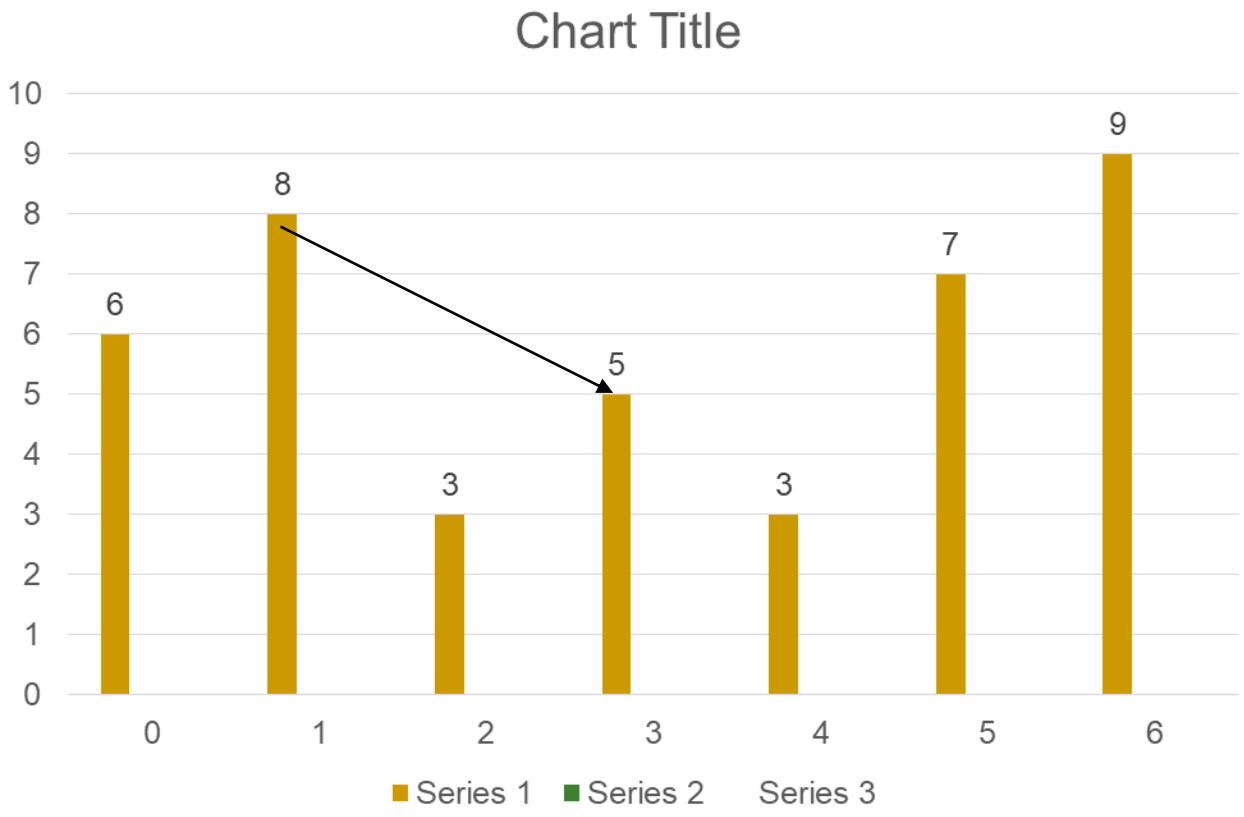
I = 3



Stack 1 2

Span **1** **2** **1**

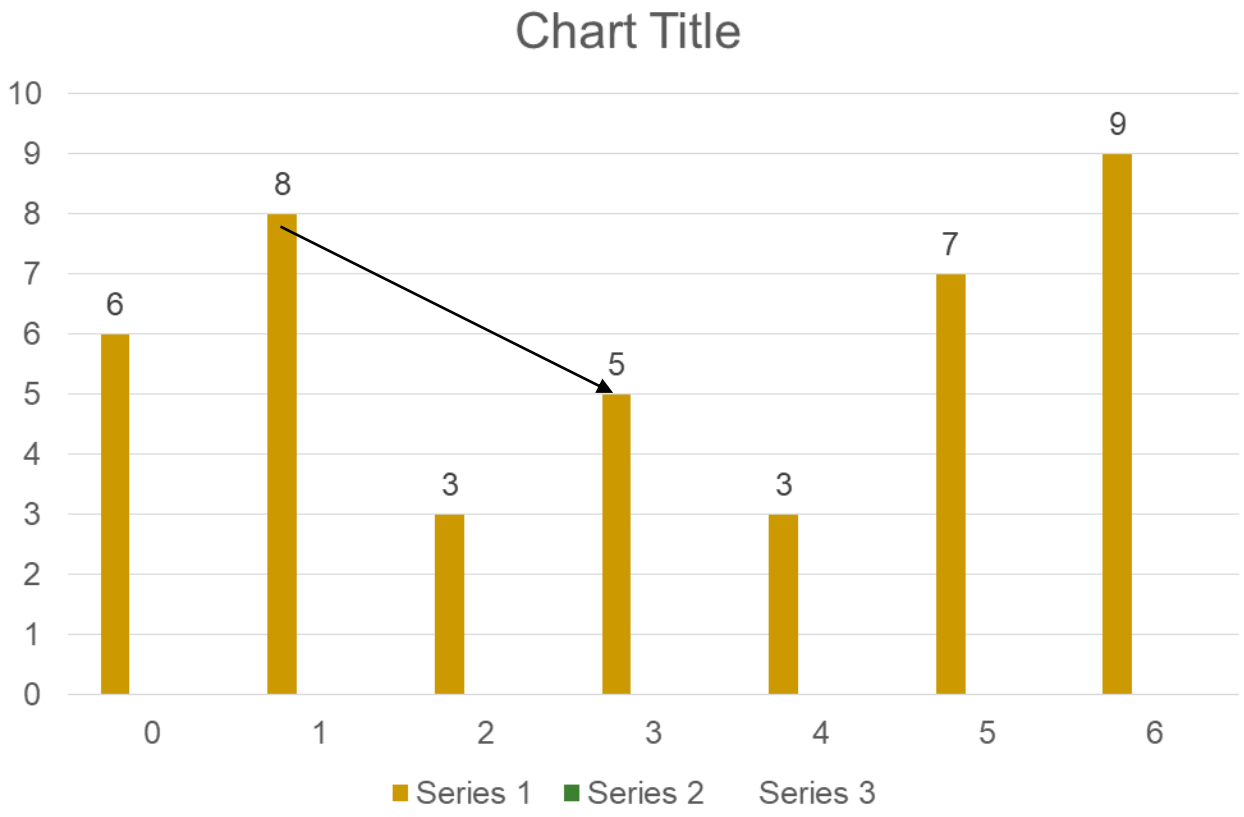
Pop



Stack 1

Span **1** **2** **1**

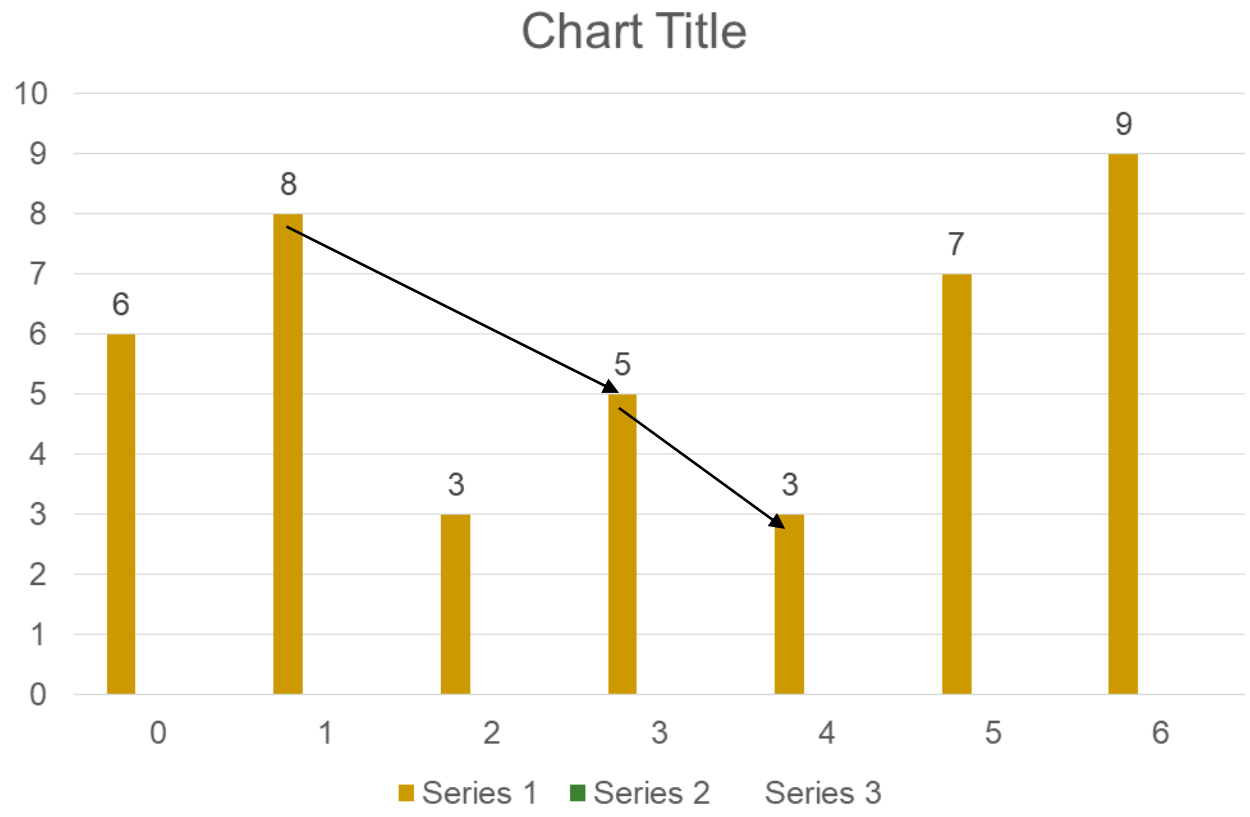
Push(3)



Stack 1 3

Span **1** **2** **1** **2**

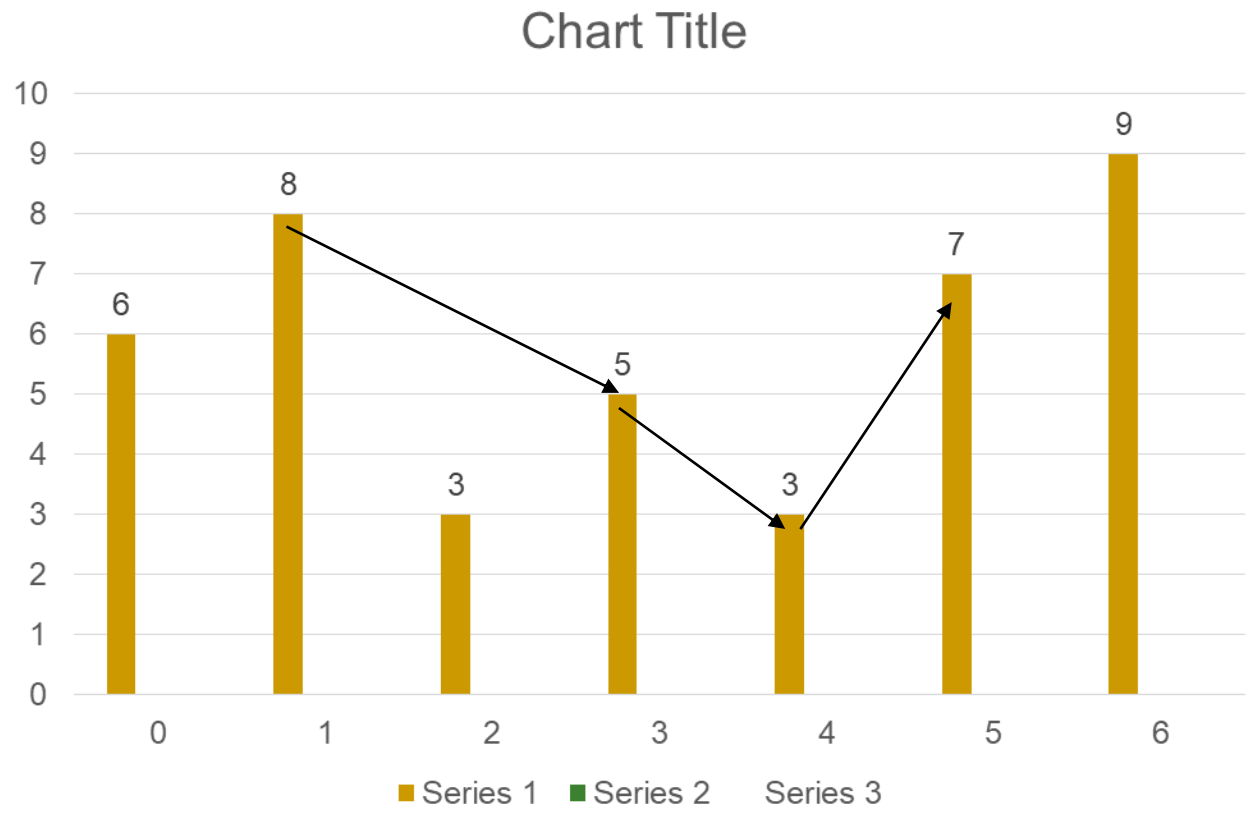
I = 4 Push(4)



Stack 1 3 4

Span **1** **2** **1** **2** **1**

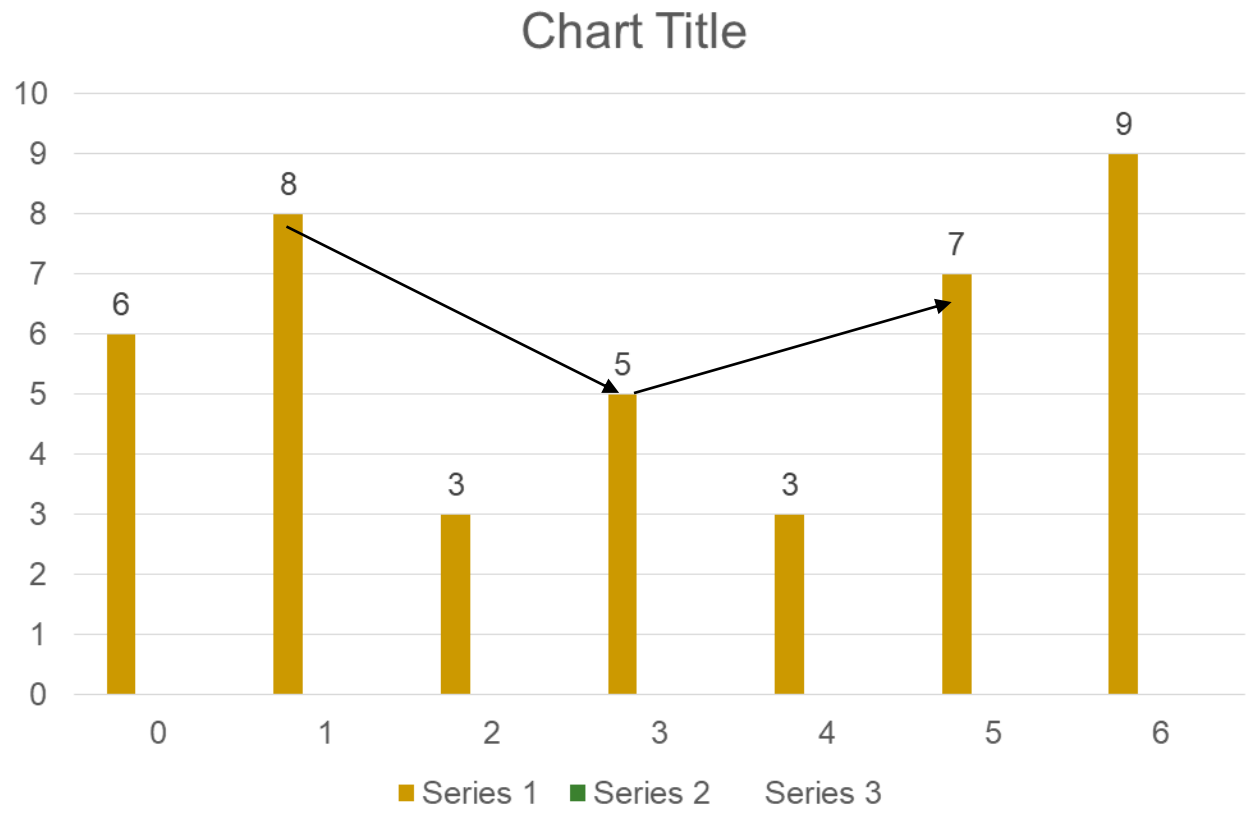
I = 5



Stack 1 3 4

Span **1** **2** **1** **2** **1**

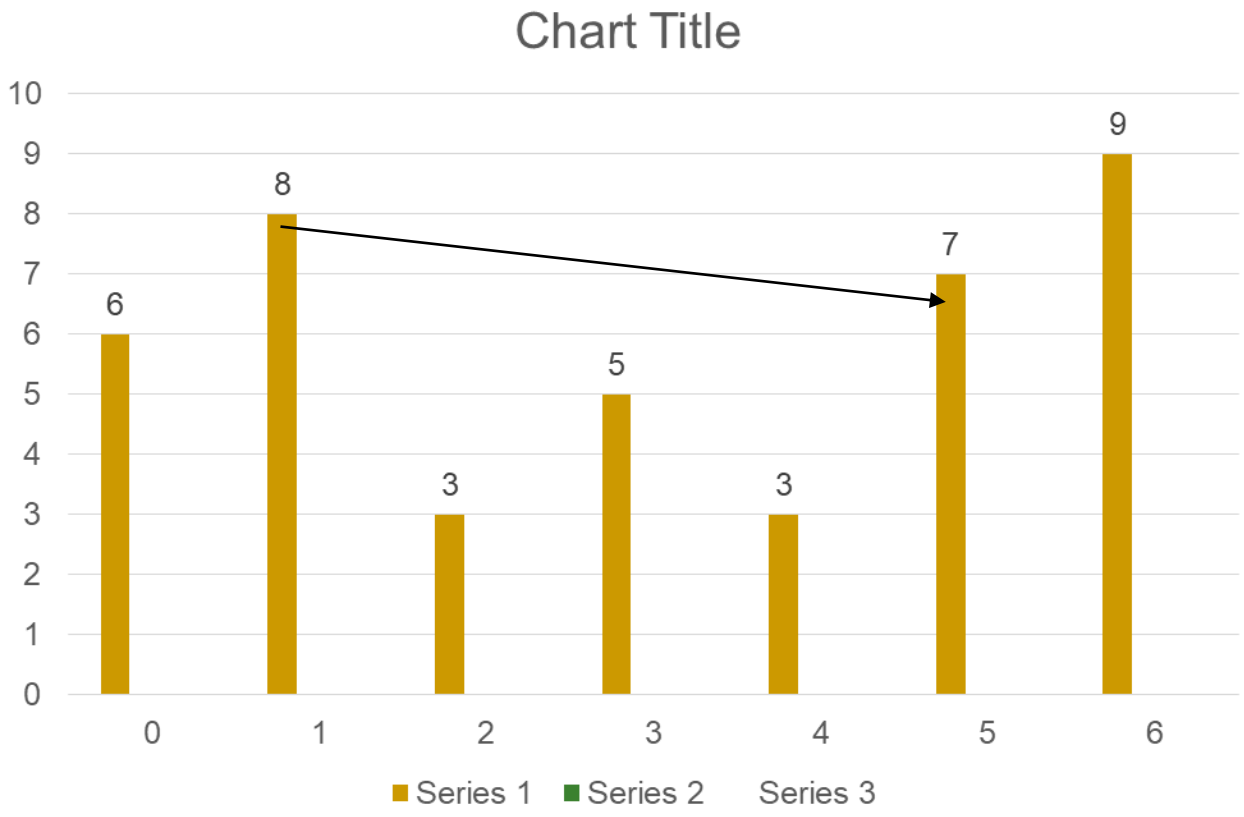
Pop



Stack 1 3

Span **1** **2** **1** **2** **1**

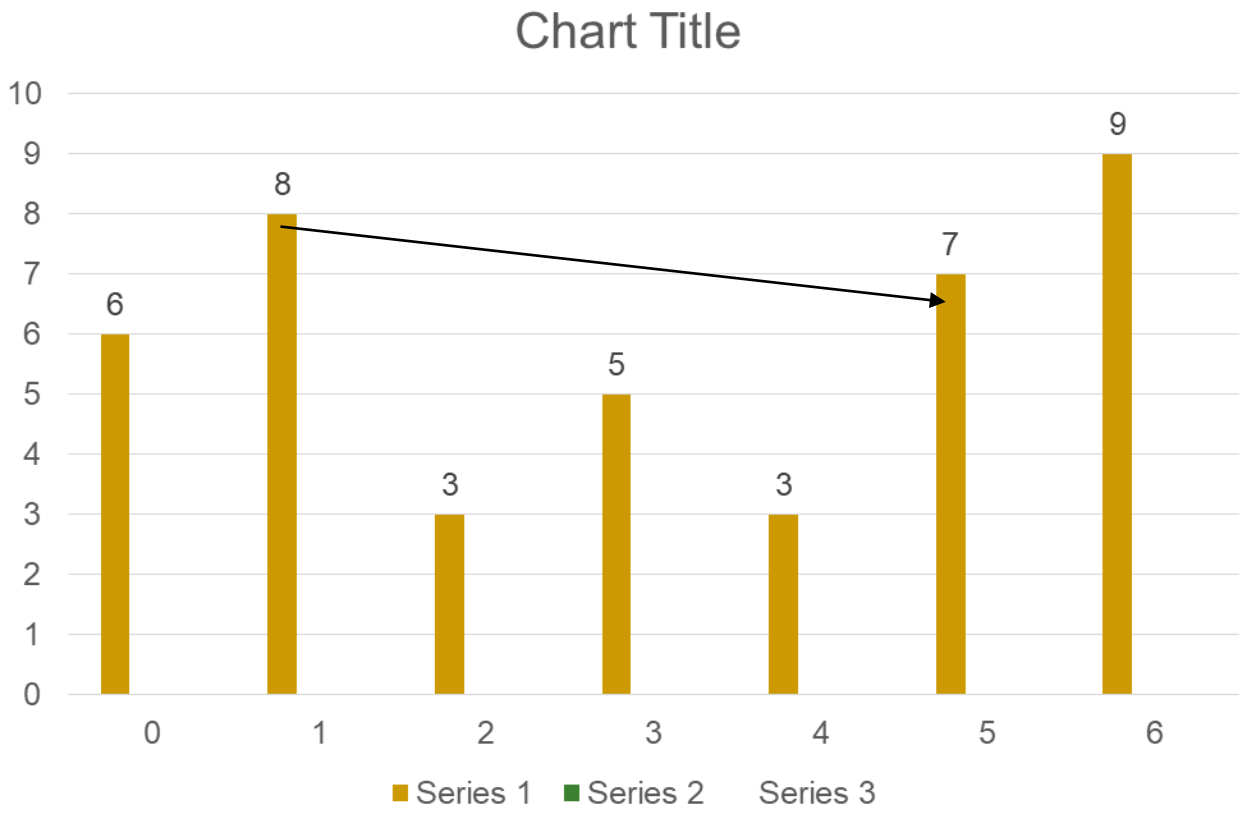
Pop



Stack 1

Span **1** **2** **1** **2** **1**

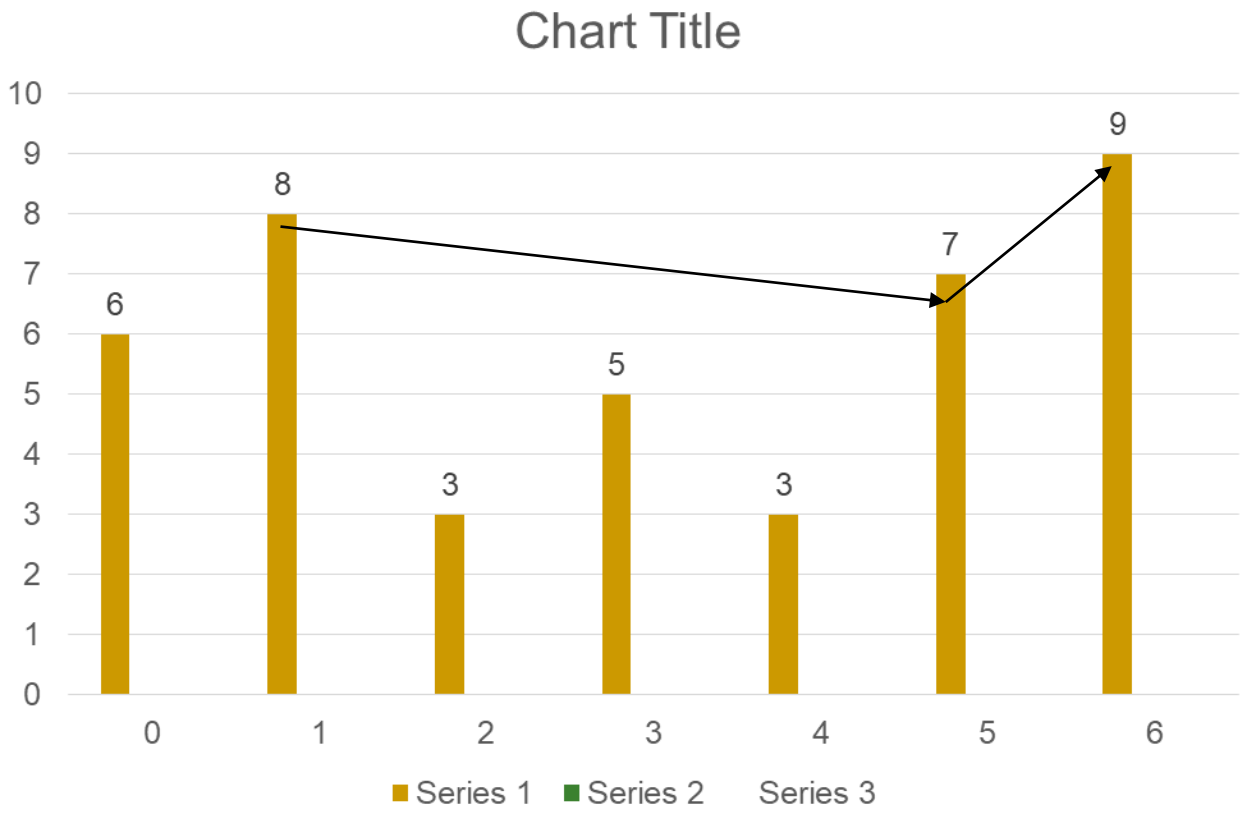
Push (5)



Stack 1 5

Span **1** **2** **1** **2** **1** **4**

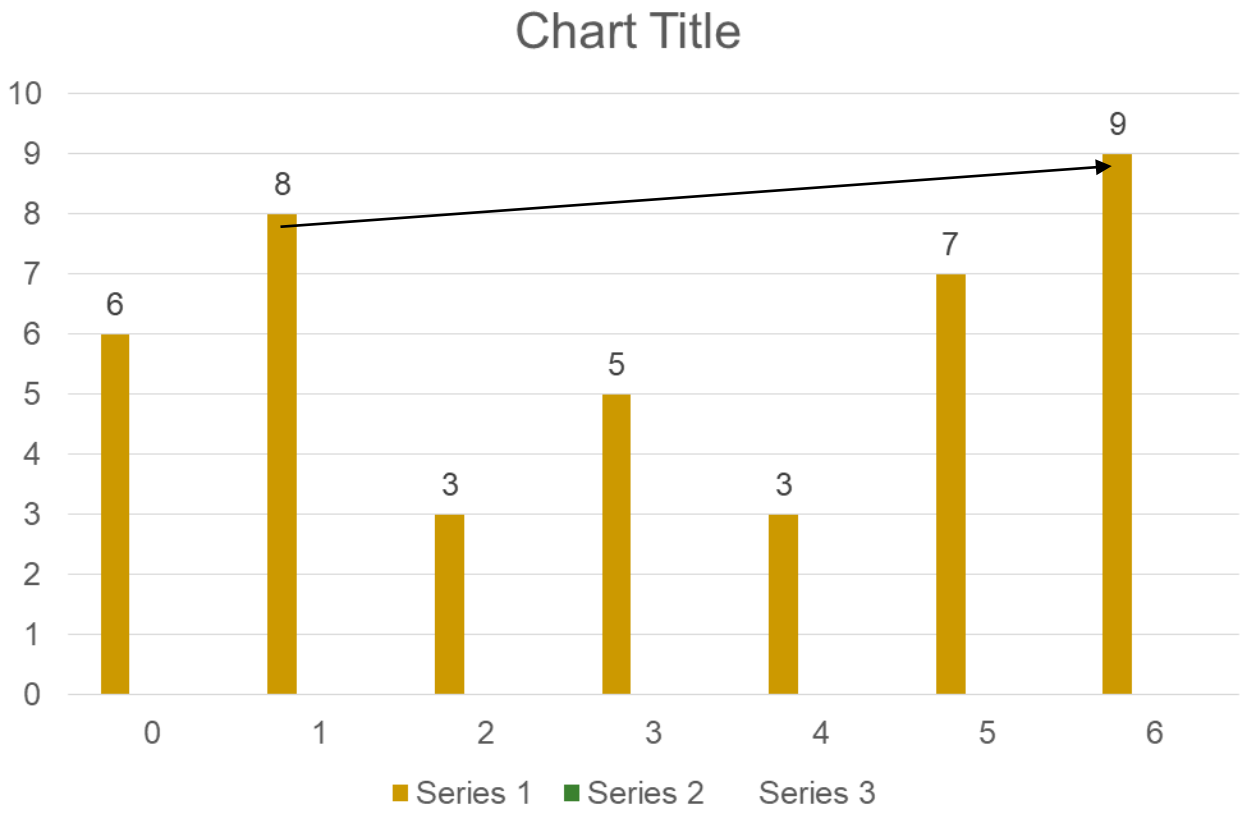
I = 6



Stack 1 5

Span **1** **2** **1** **2** **1** **4**

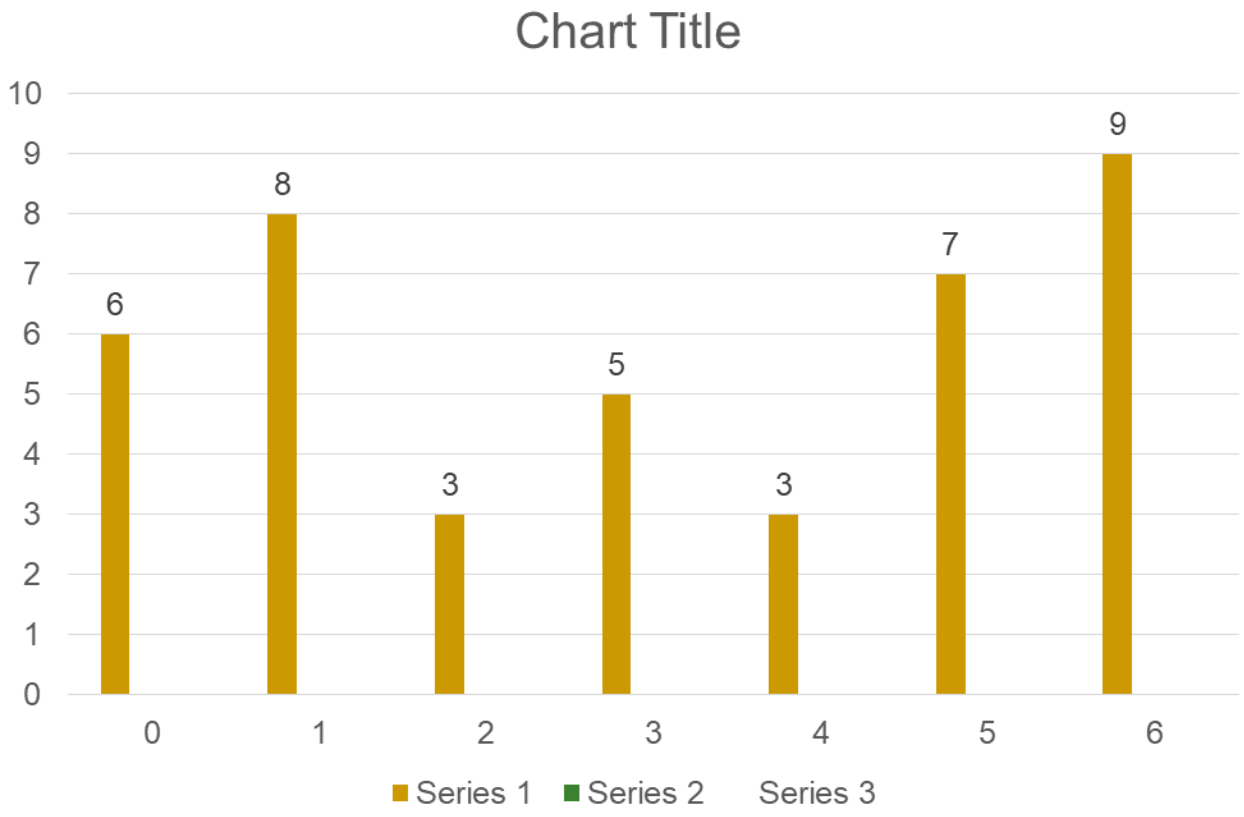
Pop



Stack 1

Span **1 2 1 2 1 4**

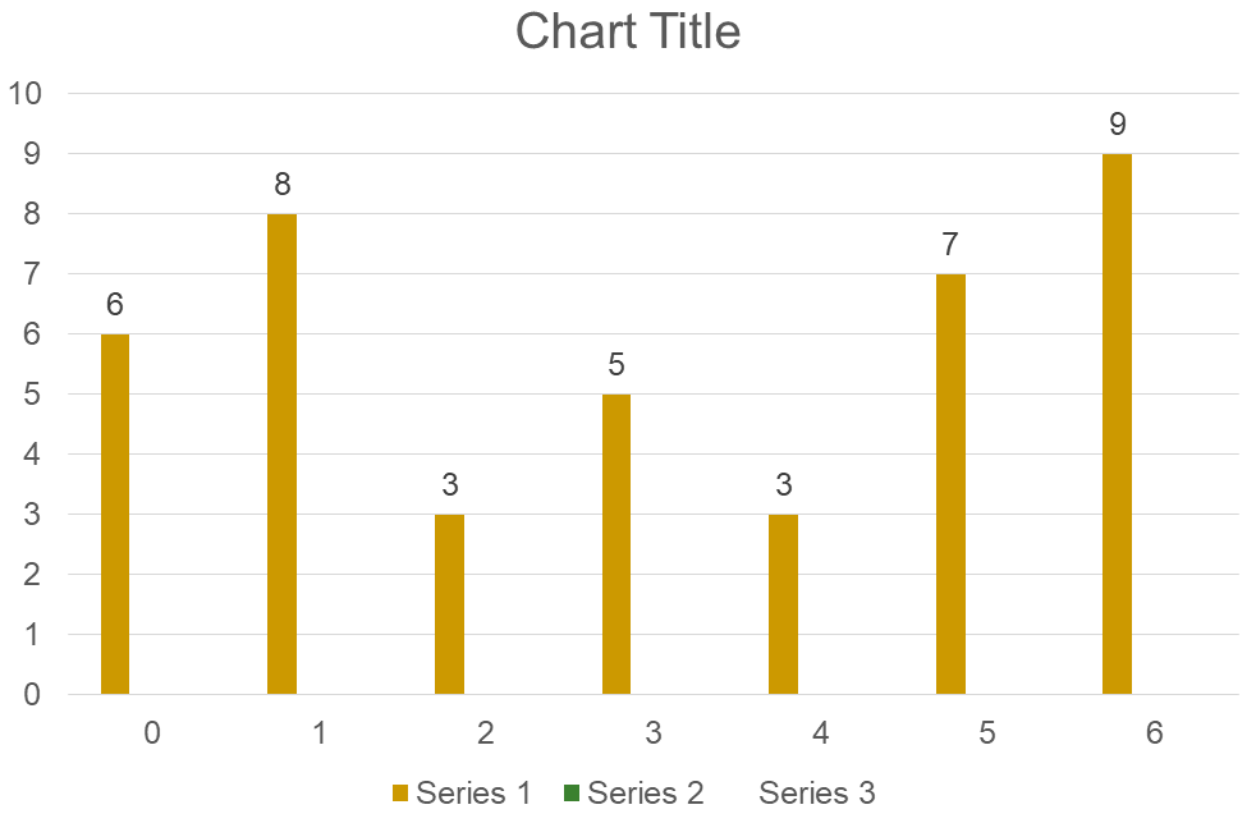
Pop



Stack

Span **1** **2** **1** **2** **1** **4**

Push (6)



Stack 6

Span **1 2 1 2 1 4 7**

Linear Time Solution

```
stack.push(0);
span[0] = 1;
for (i = 1; i < n; i++) {
    while( Stack is not empty   &&   Price[Stack[top]] <= Price[i] )
        Stack.Pop()
    If ( stack is empty )
        span[i] = i + 1
    If( Price[i] < Price[Stack[top]] )
        span[i] = i – Stack[top]
    Stack.Push(i)
}
```

Each day is pushed/popped at most once in the stack.
Time complexity = O(2n) = O(n)