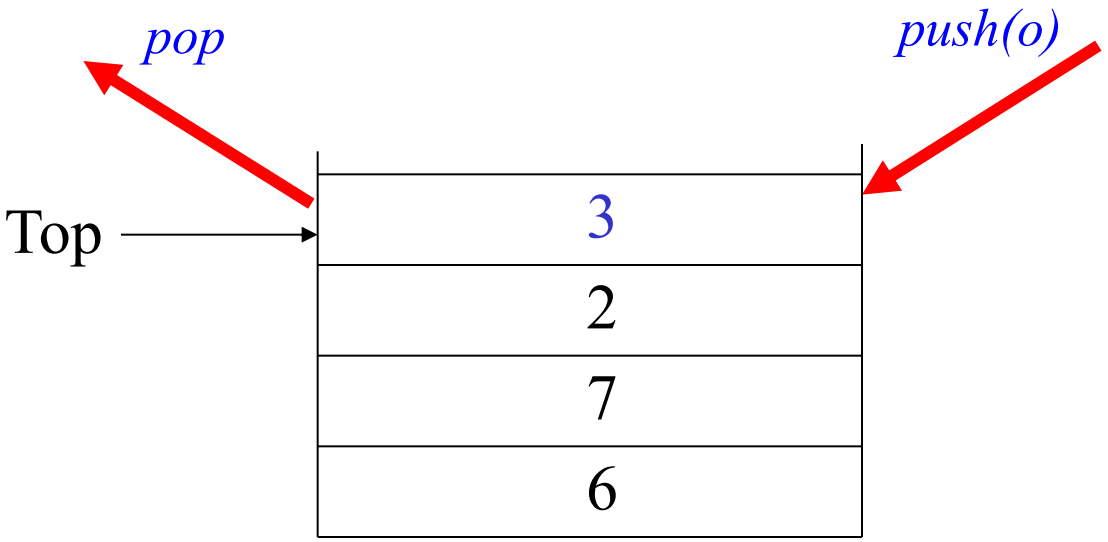
Stacks

Taken from: CS201: Data Structures and Discrete Mathematics I
Linked Lists, Stacks and Queues

What is a Stack?

- A *stack* is a list with the restriction that insertions and deletions can be performed in only one position, namely, the end of the list, called the *top*.
- The operations: push (insert) and pop (delete)



Designing and Building a Stack class

- The basic functions are:
  - Constructor: construct an empty stack
  - Empty(): Examines whether the stack is empty or not
  - Push(): Add a value at the top of the stack
  - Top(): Read the value at the top of the stack
  - Pop(): Remove the value at the top of the stack
  - Display(): Displays all the elements in the stack
  - Size(): Number of elements in stack

Implementation of the Operations

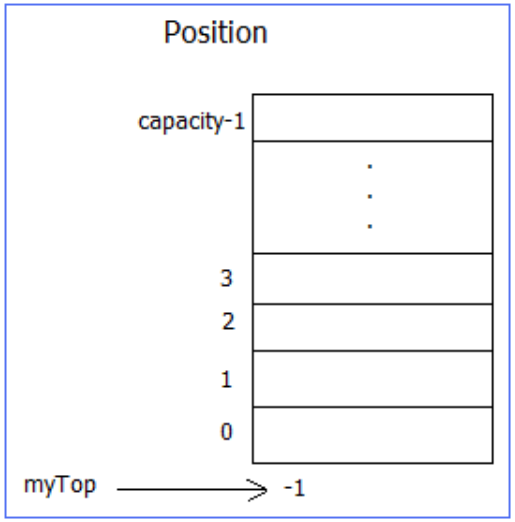
- Constructor:

  Create an array: (int) array[capacity]

  Set myTop = -1

- Empty():

  check if myTop == -1



Implementation of the Operations

- Push(int x):

  if array is not FULL (myTop < capacity-1)

  myTop++

  store the value x in array[myTop]

  else

  output “out of space”

Implementation of the Operations

- Top():

  If the stack is not empty

  return the value in array[myTop]

  else:

  output “no elements in the stack”

Implementation of the Operations

- Pop():

  If the stack is not empty

  myTop -= 1

  else:

  output “no elements in the stack”

Further Considerations

- What if static array initially allocated for stack is too small?
  - Terminate execution?
  - Replace with larger array!

- Creating a larger array
  - Allocate larger array
  - Use loop to copy elements into new array
  - Delete old array

Growing the stack

- Double the stack each time
- Increase stack size by a constant k

Which one is preferred ?

Double the Stack size

Initially size 1(phase1), then size 2 (in phase 2), size4(in phase 3) and so on

In phase i, the size is 2^{i-1}

Cost of each phase: $2^{i-1} + 2^{i-2} + 2^{i-2} = 2^i$

- Construct array of size 2^{i-1}
- Copy elements from previous array 2^{i-2}
- Fill remaining 2^{i-2} elements that can be accommodated.

For n elements we require log n phases.

Total cost: $2 + 4 + 8 + \ldots.. + 2^{\log n} = O(n)$

Increase Stack size by c

Initially size c(phase1), then size 2c (in phase 2), size 3c (in phase 3) and so on
In phase i, the size is i.c

Cost of each phase: i.c + (i-1).c+ c =2.i.c
- Construct array of size i.c
- Copy elements from previous array (i-1).c
- Fill remaining elements that can be accommodated $\therefore$ c

For n elements we require n/c phases.

Total cost: 2c+ 2.2c + 2.3c + 2.4c+ …..+ n/c. 2c= $=2c(n/c)^2=2n^2/c$

Stack ADT Interface

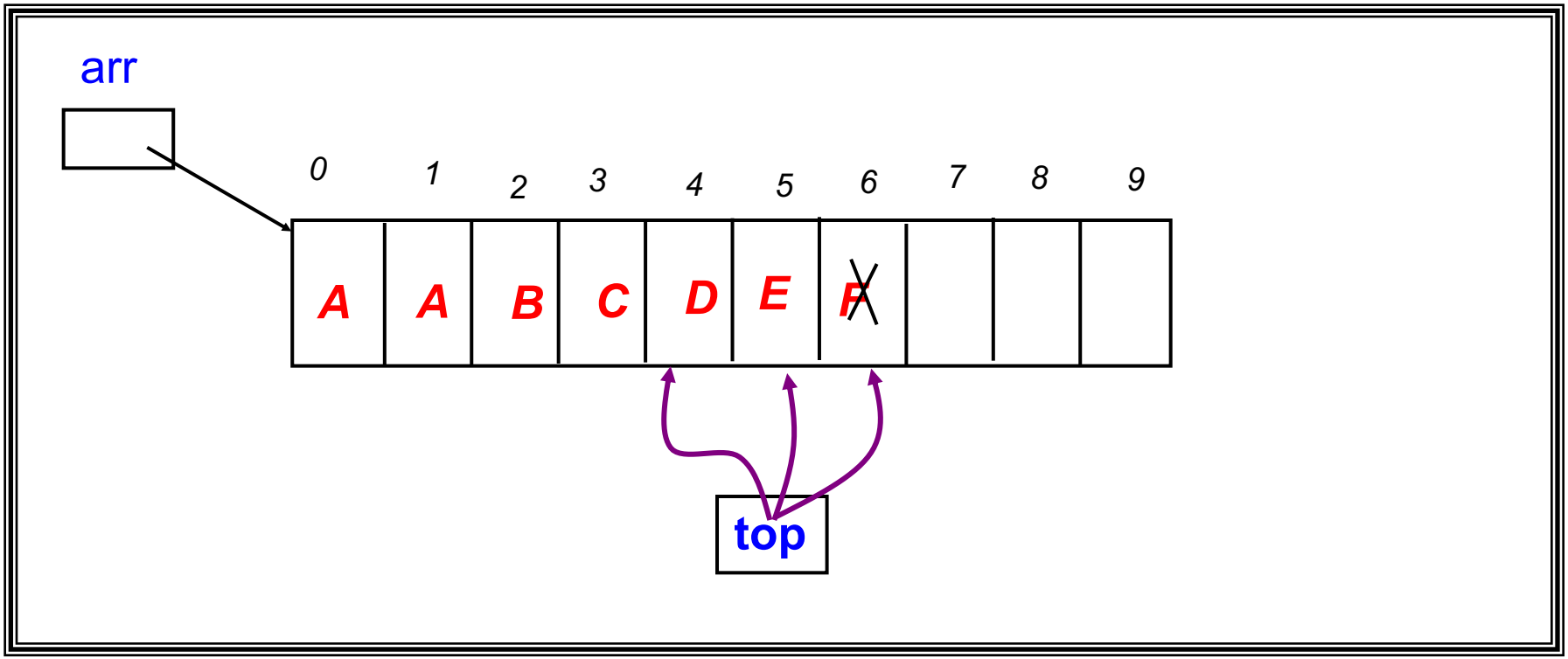
- The main functions in the Stack ADT are (S is the stack)

```
boolean isEmpty();              // return true if empty
boolean isFull(S);              // return true if full
void push(S, item);             // insert item into stack
void pop(S);                    // remove most recent item
void clear(S);                  // remove all items from stack
Item top(S);                    // retrieve most recent item
Item topAndPop(S);              // return & remove most recent item
```

Implementation by Array

- use Array with a top index pointer as an implementation of stack

StackAr



- int top=-1,stack[MAX];

```
void pop()
{
    if(top==-1)
    {
        printf("\nStack is empty!!");
    }
    else
    {
        printf("\nDeleted element is
%d",stack[top]);
        top=top-1;
    }
}
```

```
void push()
{
    int val;

    if(top==MAX-1)
    {
        printf("\nStack is full!!");
    }
    else
    {
        printf("\nEnter element to push:");
        scanf("%d",&val);
        top=top+1;
        stack[top]=val;
    }
}
```

Applications

- Balanced Paranthesis
- Convert infix to postfix and vice versa
- Evaluating PostFix notation

- Infix to prefix and vice versa
- Evaluating Prefix notation

Infix to postfix

- If (push
- If) pop and print everything on stack until (
- If operand print
- If operator,

  (a)-pop and print elements on stack of same or higher priority. If no such elements then do nothing.

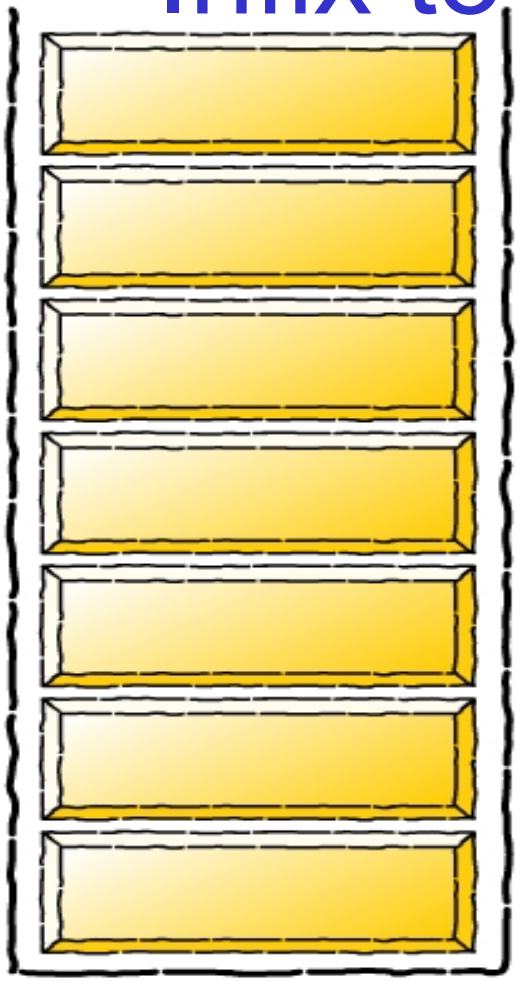
  (b)-push operator

Priority order (, +-, */, ^

Try

- (a+b-c)*d-(e+f)
- 4^2*3-3+8/4/(1+1)
- a+b/c*(d+e)-f

Infix to postfix conversion



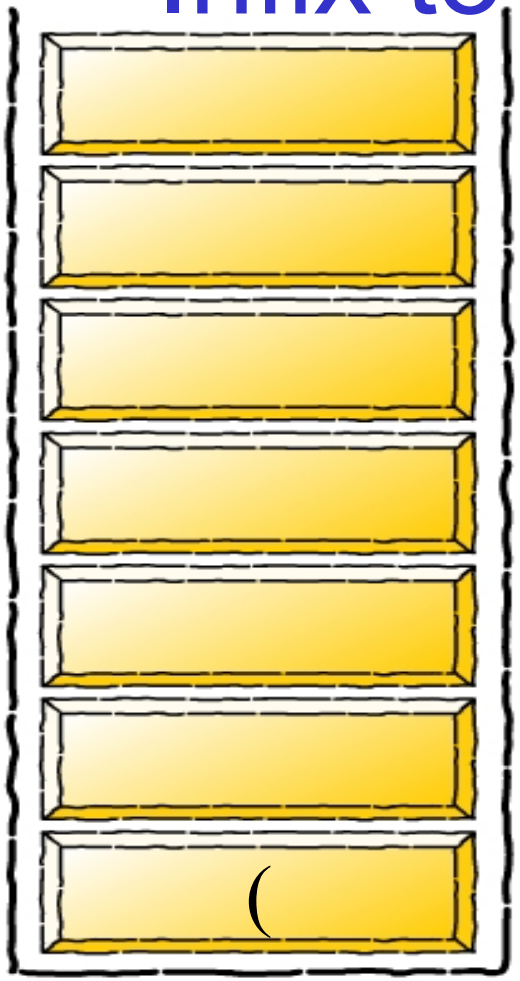
infixVect

(a + b - c) * d – (e + f)

postfixVect

Infix to postfix conversion

stackVect



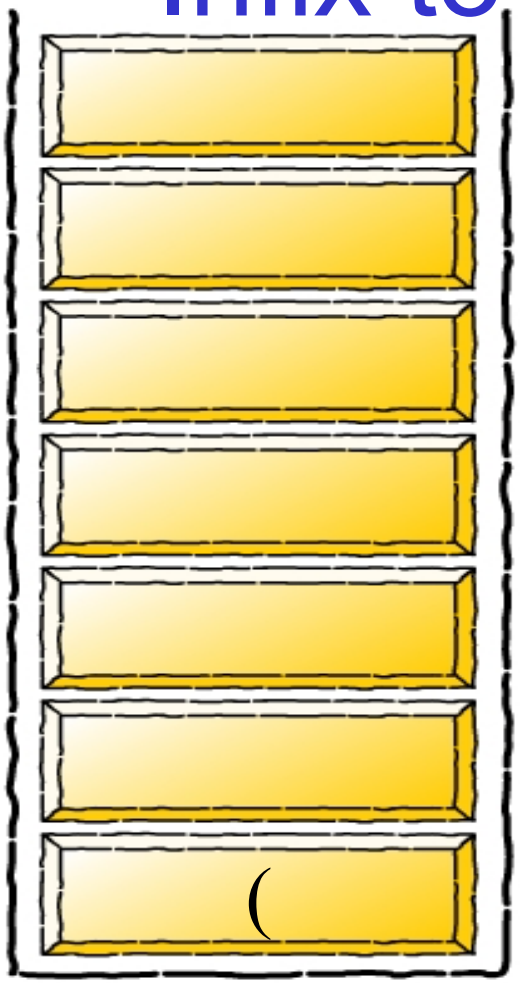
infixVect

a + b - c) * d – (e + f)

postfixVect

Infix to postfix conversion

stackVect



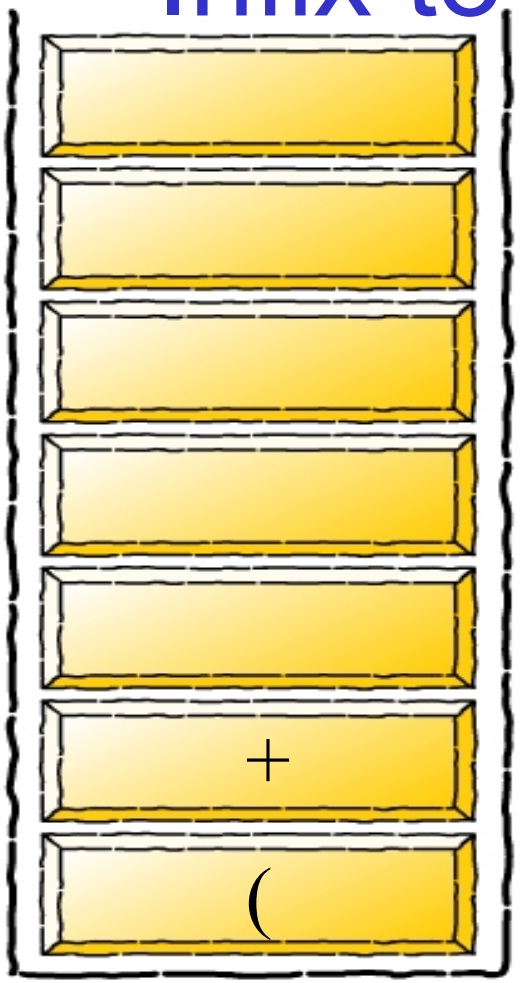
infixVect

+ b - c) * d – (e + f)

postfixVect

a

Infix to postfix conversion



infixVect

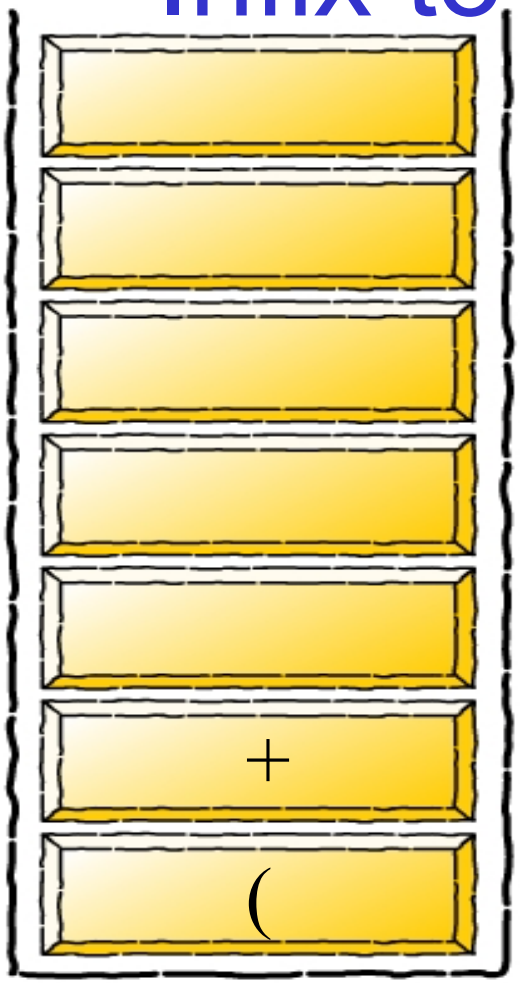
b - c) * d – (e + f)

postfixVect

a

Infix to postfix conversion

stackVect



infixVect

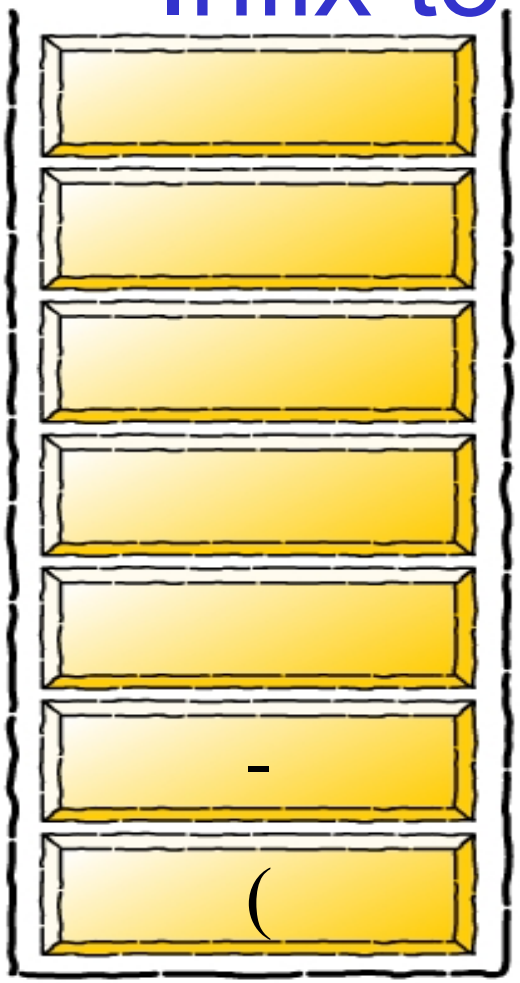
- c) * d – (e + f)

postfixVect

a b

Infix to postfix conversion

stackVect



infixVect

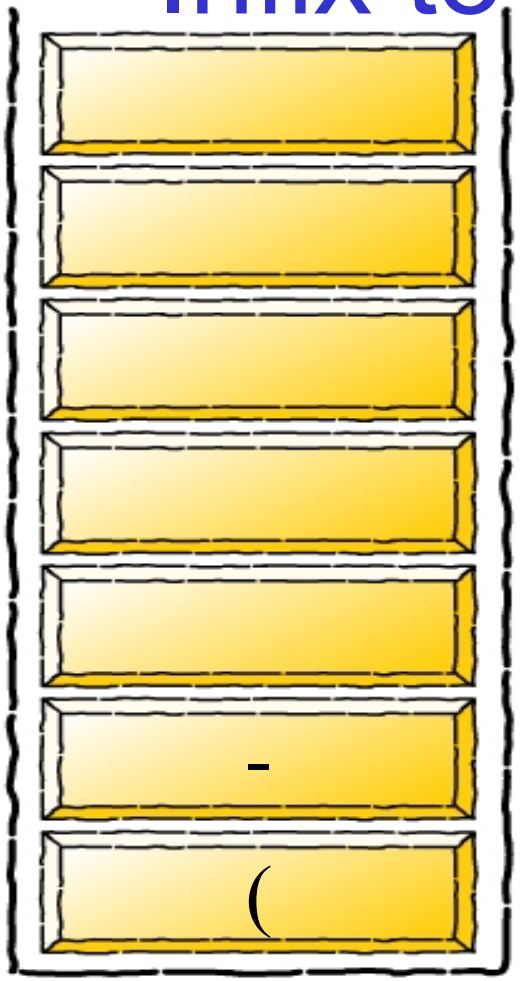
c) * d – (e + f)

postfixVect

a b +

Infix to postfix conversion

stackVect



infixVect

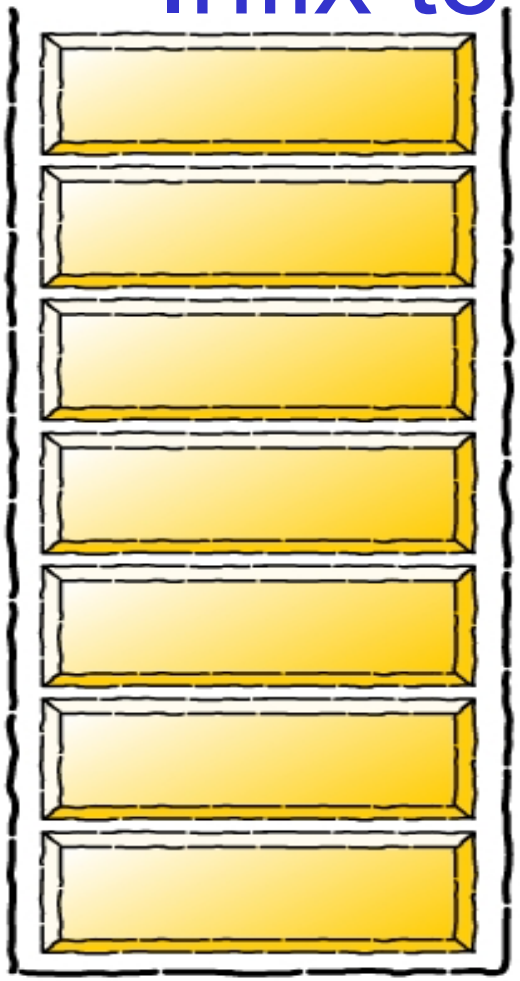
) * d – (e + f)

postfixVect

a b + c

Infix to postfix conversion

stackVect



infixVect

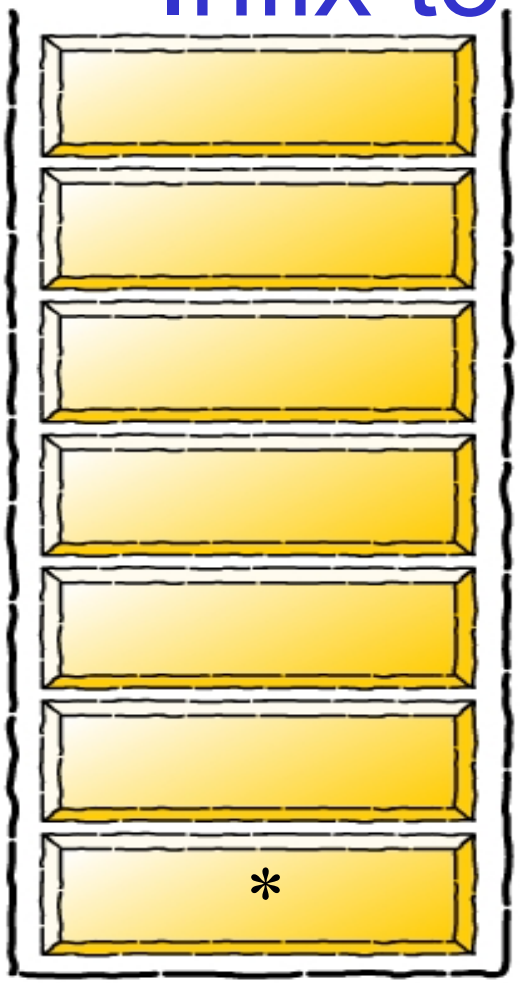
* d – (e + f)

postfixVect

a b + c -

Infix to postfix conversion

stackVect



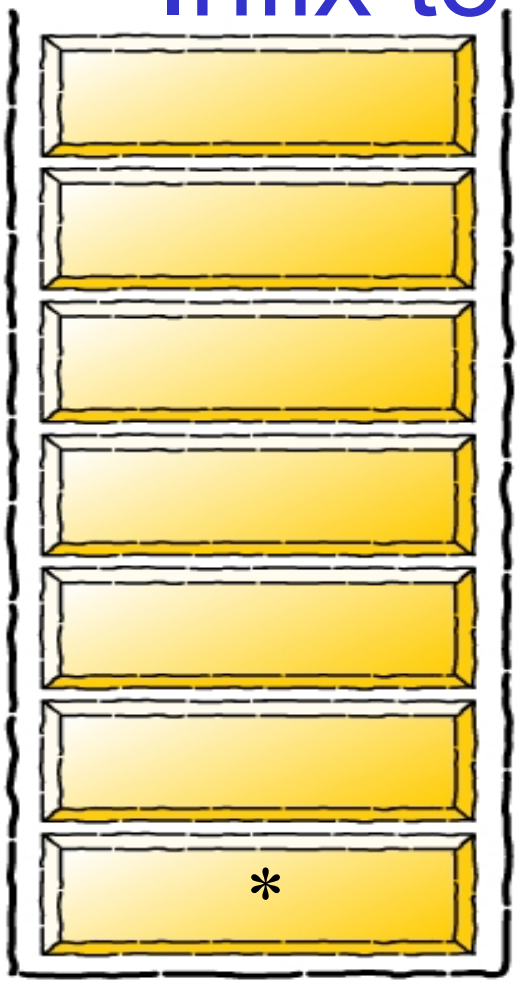
infixVect

d – (e + f)

postfixVect

a b + c -

Infix to postfix conversion



infixVect

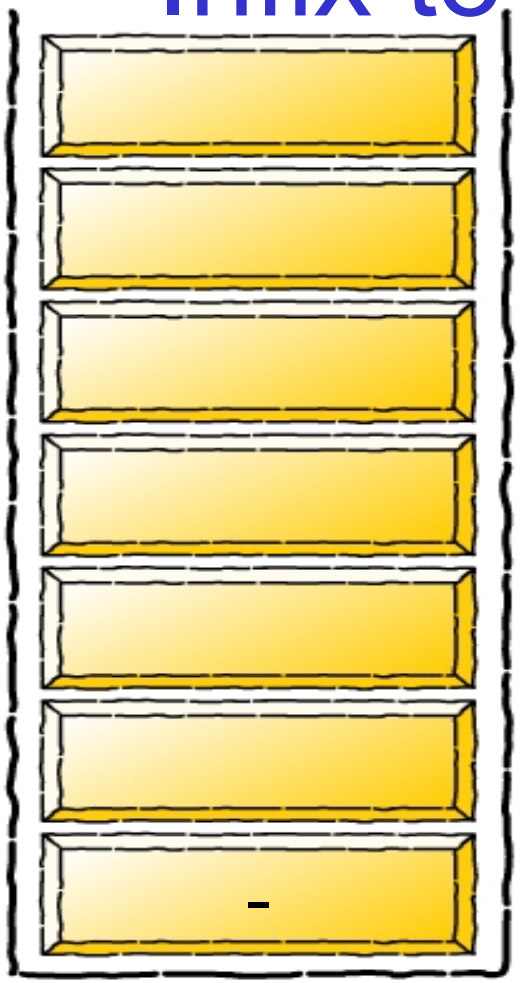
– (e + f)

postfixVect

a b + c - d

Infix to postfix conversion

stackVect



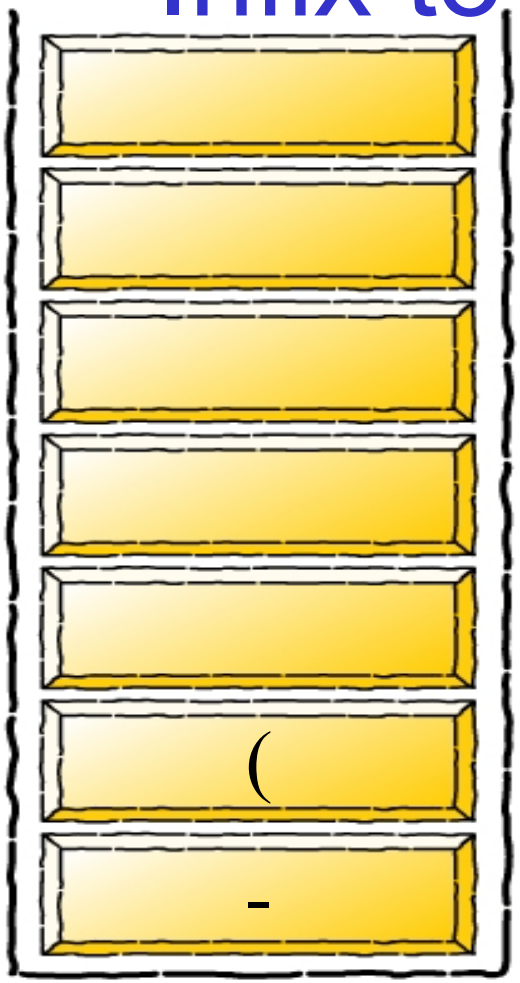
infixVect

(e + f)

postfixVect

a b + c – d *

Infix to postfix conversion



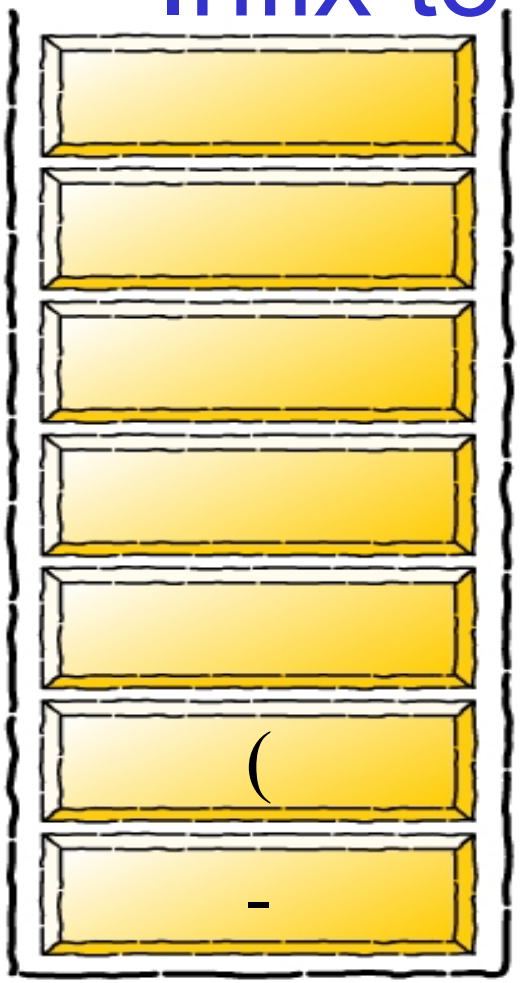
infixVect

e + f)

postfixVect

a b + c – d *

Infix to postfix conversion



infixVect

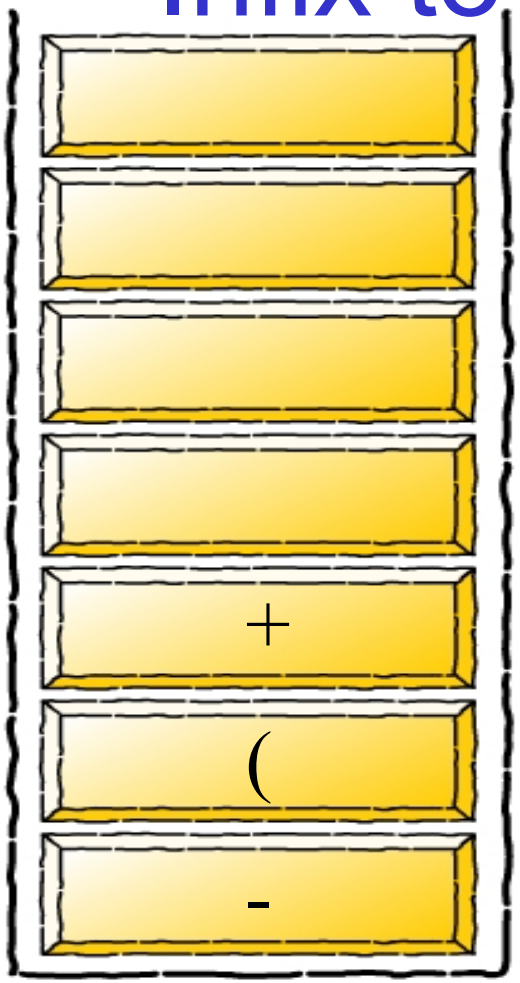
+ f)

postfixVect

a b + c – d * e

Infix to postfix conversion

stackVect



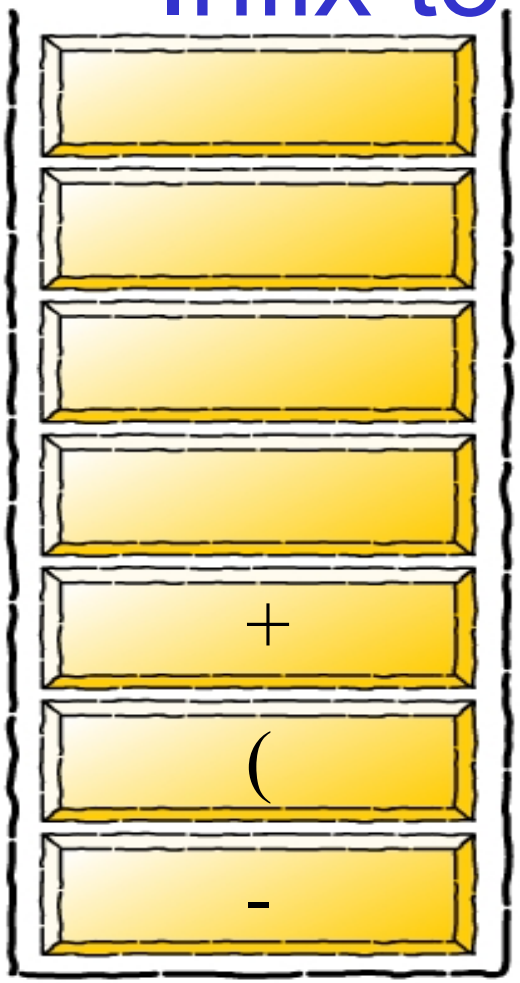
infixVect

f)

postfixVect

a b + c – d * e

Infix to postfix conversion



infixVect

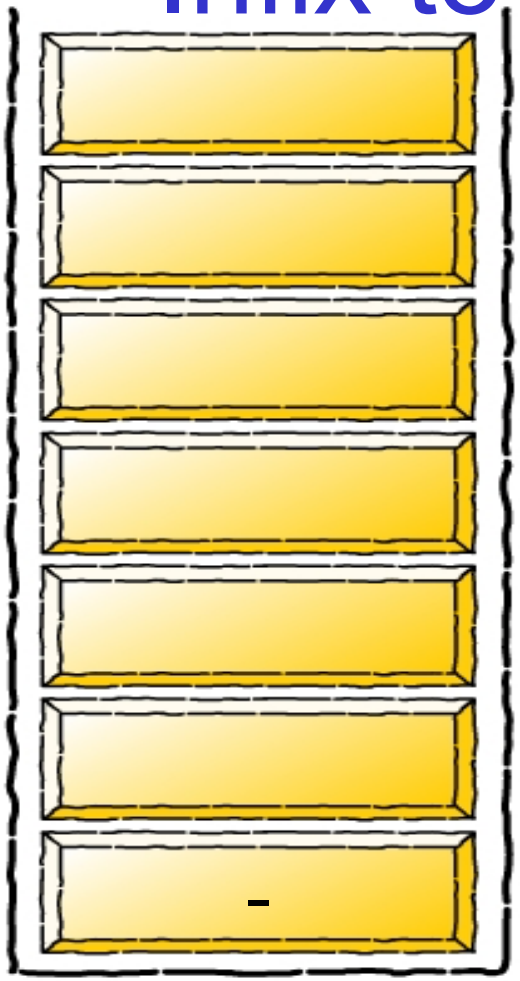
)

postfixVect

a b + c – d * e f

Infix to postfix conversion

stackVect



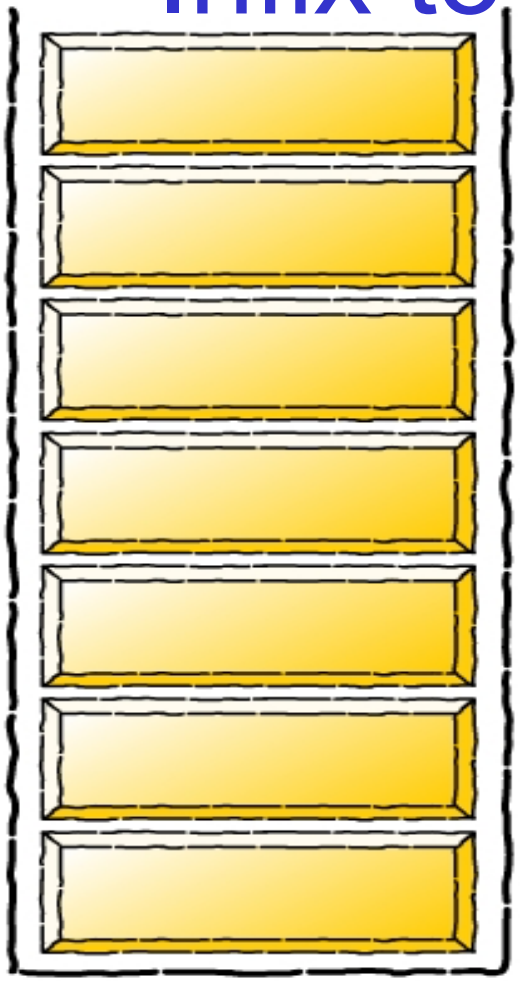
infixVect

postfixVect

a b + c – d * e f +

Infix to postfix conversion

stackVect



infixVect

postfixVect

a b + c – d * e f + -

- Scan the token list from left to right.
  - If the token is an operand, append it to the end of the output list.
  - If the token is a left parenthesis, push it on the opstack.
  - If the token is a right parenthesis, pop the opstack until the corresponding left parenthesis is removed. Append each operator to the end of the output list.
  - If the token is an operator, *, /, +, or -, push it on the opstack. However, first remove any operators already on the opstack that have higher or equal precedence and append them to the output list.
- When the input expression has been completely processed, check the opstack. Any operators still on the stack can be removed and appended to the end of the output list.

Try

- 4^2*3-3+8/4/(1+2)
- A+b/c*(d+e)-f
- A^b^c [Note exponents evaluated right to left]. H.W
- Is 2^3^2= 2^(3^2)=512 or (2^3)^2=64?

Try

- 4^2*3-3+8/4/(1+2)
  4 2 ^ 3 * 3 – 8 4 / 1 2 + / +
- A+b/c*(d+e)-f
  A b c / d e + * + f –
- A^b^c [Note exponents evaluated right to left]. H.W
- Is 2^3^2= 2^(3^2)=512 or (2^3)^2=64?

Convert Postfix to Infix

- Abc/de+*+f-
- 42^3*3-84/12+/+

Convert Postfix to Infix

- Abc/de+*+f-
  (A +(b/c) * (d+e))-f
- 42^3*3-84/12+/+

Evaluating Postfix

- Abc/de+*+f-
- 42^3*3-84/12+/+

Try

- A^b^c [Note exponents evaluated right to left]. H.W
- In case of exponents rule changed

  -From: pop same and higher priority elements

  -To: pop higher priority elements.
- However, there are no higher priority operands and hence, we only push.

- a - b ^ c ^ d + f becomes a b c d ^ ^ - f +

Notes

- Static array allocation O(1) but dynamic array allocation using malloc O(n) since n free chunks need to be checked for. These need not be contiguous

Questions

- Let S1 and S2 be two stacks. S1 has capacity of 4 elements. S2 has capacity of 2 elements. S1 already has 4 elements: 100, 200, 300, and 400(top), whereas S2 is empty.

Only the following three operations are available:

- **PushToS2:** Pop the top element from **S1** and push it on **S2**.
  PushToS1: Pop the top element from **S2** and push it on **S1**.
  GenerateOutput: Pop the top element from **S1** and output it to the user.
- Note you cant pop an empty stack and push into a full stack
- Which of the following output sequences can be generated by using the above operations?

A 100, 200, 400, 300

B 200, 300, 400, 100

C 400, 200, 100, 300

D 300, 200, 400, 100

A single array A[1..MAXSIZE] is used to implement two stacks. The two stacks grow from opposite ends of the array. Variables top1 and top2 (top1 < top2) point to the location of the topmost element in each of the stacks. If the space is to be used efficiently, the condition for "stack full" is _____.

- **(GATE CSE 2004)**
  1. (top 1 = MAXSIZE/2) AND (top 2 = MAXSIZE/2 + 1)
  2. top 1 + top 2 = MAXSIZE
  3. (top 1 = MAXSIZE/2) or (top 2 = MAXSIZE)
  4. top 1 = top 2 – 1

Homework

- Implement two stacks in an array by Dividing the space into two halves
- Sort a stack using a temporary stack Time:O(n^2), space:O(n)
- Design a stack that supports getMin() in O(1) time and O(1) extra space