CSE 1201
Data Structure

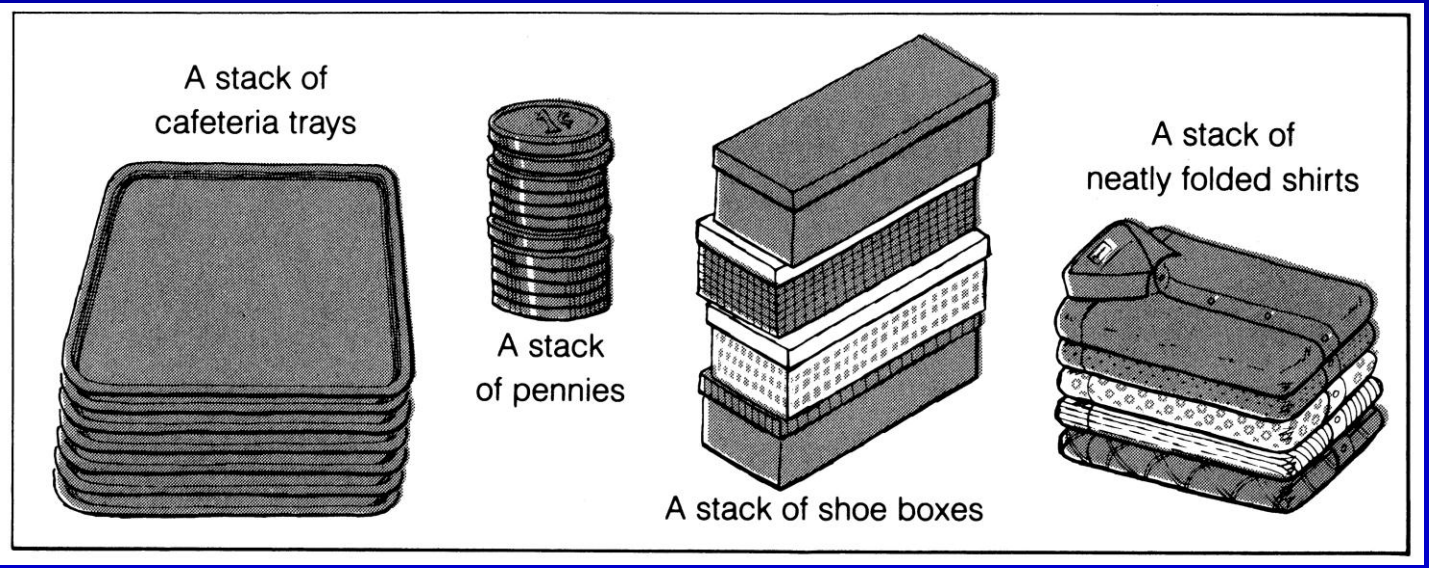
Chapter 3

Stacks

Instructor: Md. Shahid Uz Zaman
Dept. of CSE, RUET, Bangladesh

What is a stack?

- It is an ordered group of homogeneous items of elements.
- Elements are added to and removed from the top of the stack (the most recently added items are at the top of the stack).
- The last element to be added is the first to be removed (**LIFO**: Last In, First Out).



Stack Specification

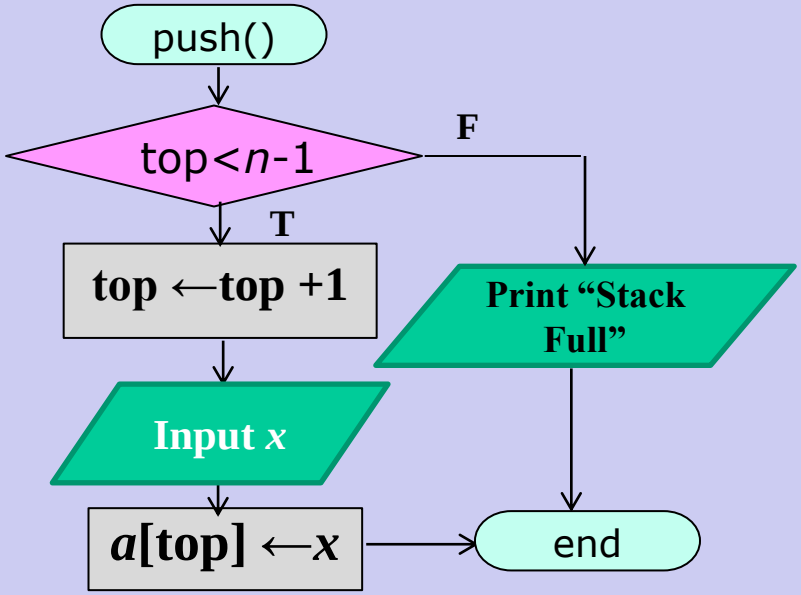
- Definitions: (provided by the user)
  - *MAX_ITEMS*: Max number of items that might be on the stack
  - *ItemType*: Data type of the items on the stack
- Operations
  - Push (ItemType newItem)
  - Pop ()

Push (ItemType newItem)

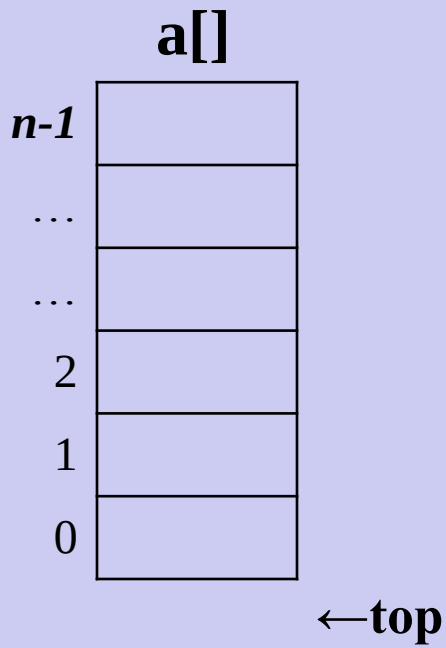
- *Function*: Adds newItem to the top of the stack.
- *Preconditions*: Stack has been initialized and is not full.
- *Postconditions*: newItem is at the top of the stack.

Stack: push() function

Topic 1: Write an Algorithm to push a new element in a stack



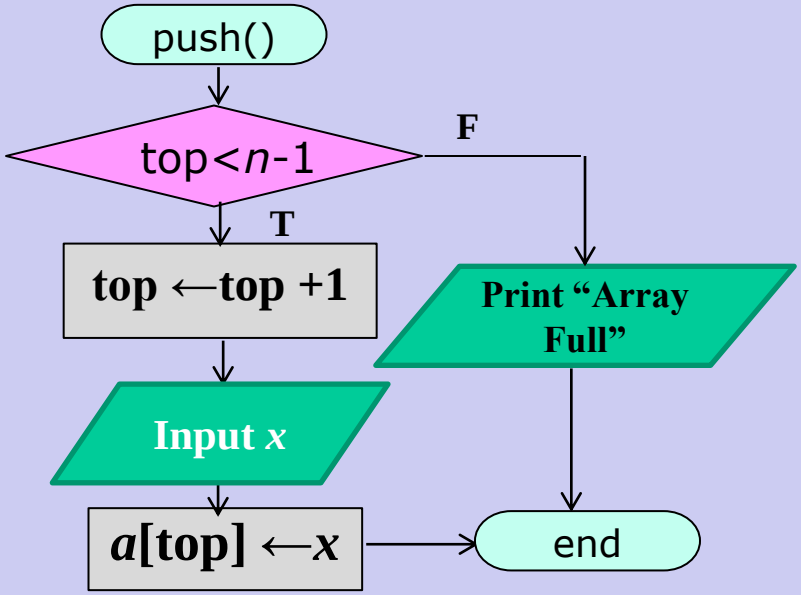
n: size of *a[]*
x: input variable
top: index of last input, declare it global.
Array Elements: *a*[0]…..*a*[*n*-1]



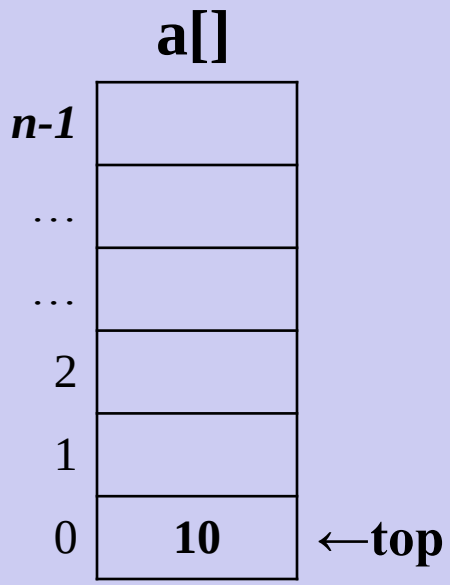
Initially top = -1

Stack: push() function

Topic 1: Write an Algorithm to push a new element in a stack



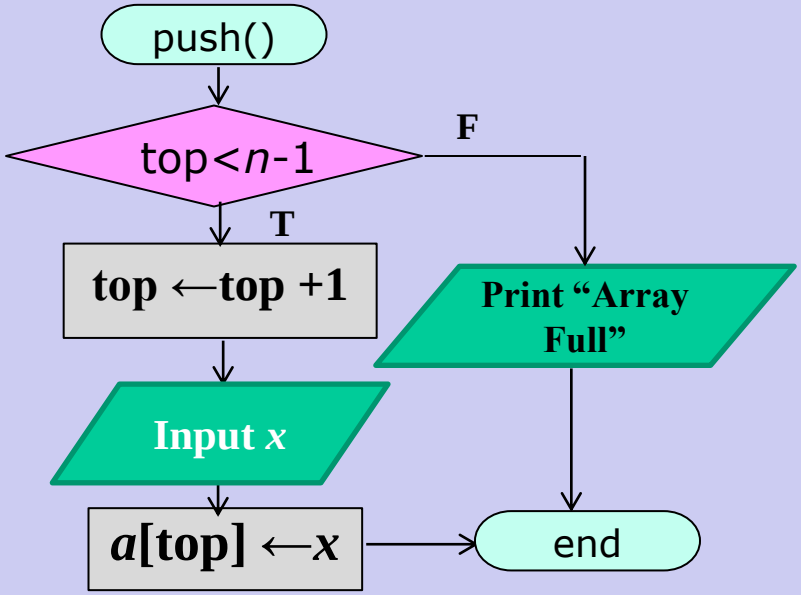
n: size of *a[]*
x: input variable
top: index of last input, declare it global.
Array Elements: *a*[0]…..*a*[*n*-1]



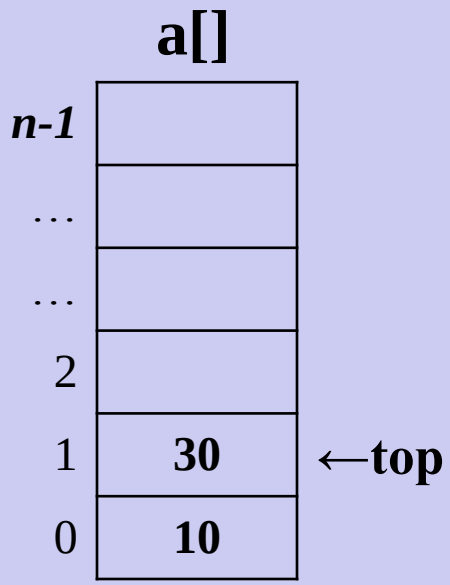
After 1st element (10) push, top = 0

Stack: push() function

Topic 1: Write an Algorithm to push a new element in a stack



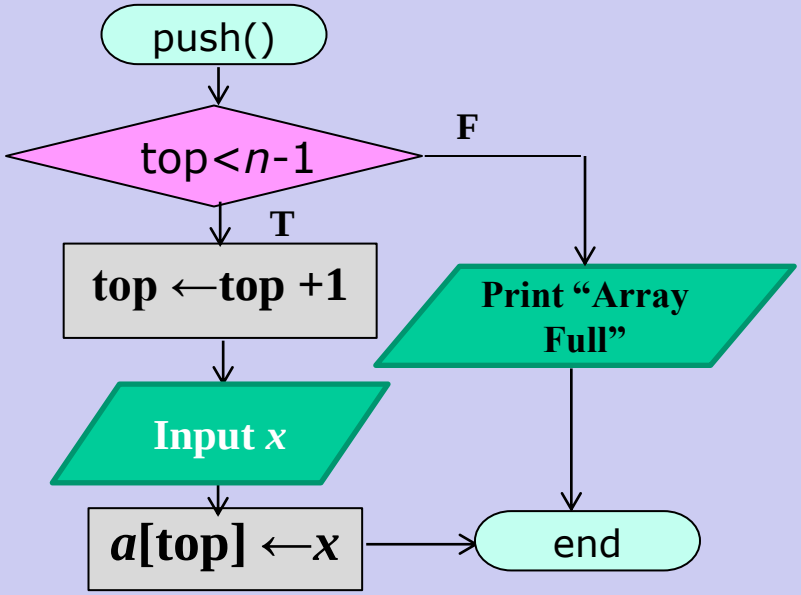
n: size of *a[]*
x: input variable
top: index of last input, declare it global.
Array Elements: *a*[0]…..*a*[*n*-1]



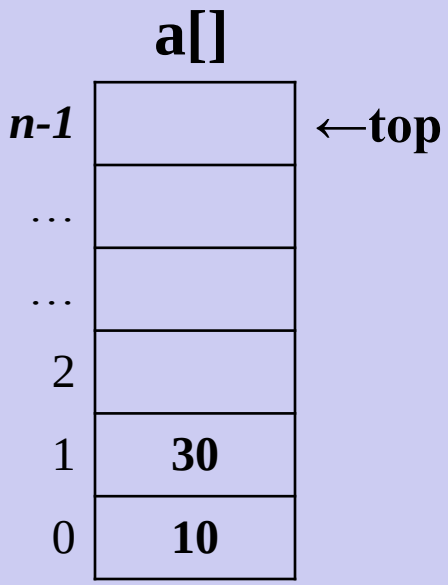
After 2nd element (30) push, top = 1

Stack: push() function

Topic 1: Write an Algorithm to push a new element in a stack



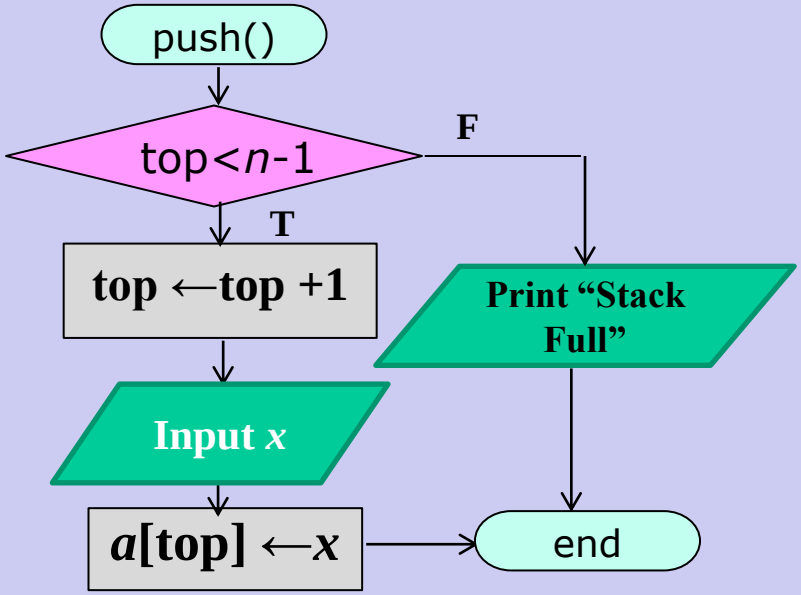
n: size of *a[]*
x: input variable
top: index of last input, declare it global.
Array Elements: *a*[0]…..*a*[*n*-1]



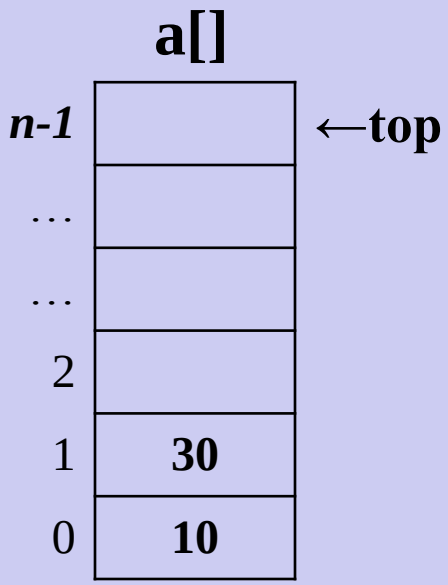
After last element (30) push, top = *n*-1

Stack: push() function

Topic 1: Write an Algorithm to push a new element in a stack



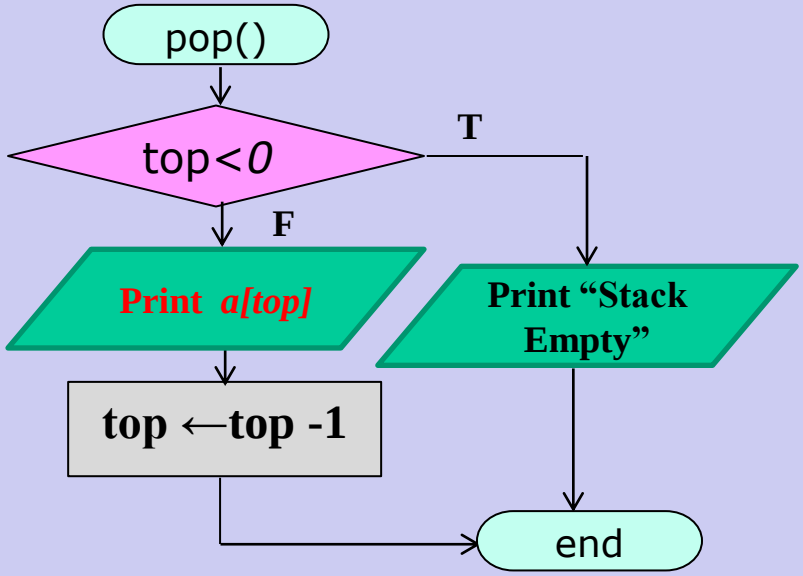
n: size of *a[]*
x: input variable
top: index of last input, declare it global.
Array Elements: *a*[0]…..*a*[*n*-1]



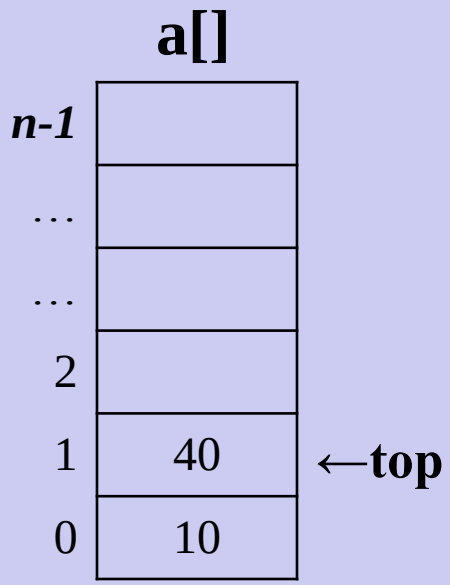
Try to push more get "Stack lFull" message

Stack: pop() function

Topic 1: Write an Algorithm to pop element from a stack



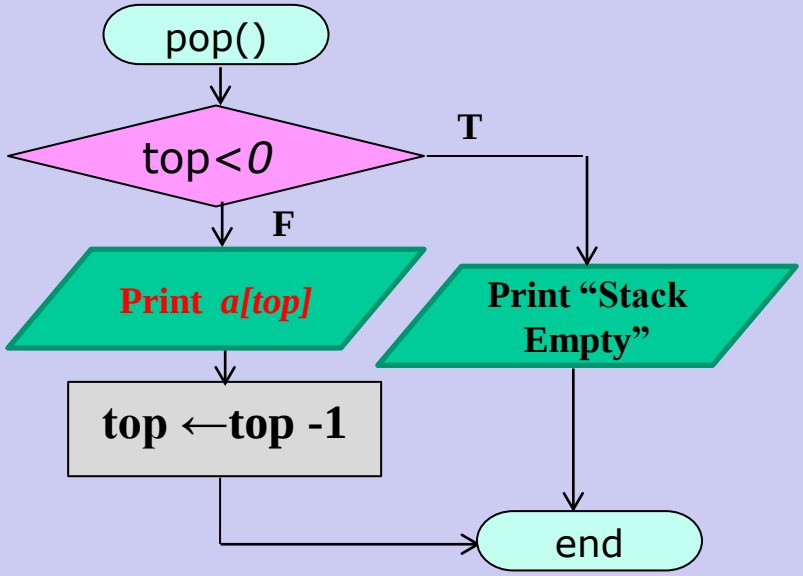
n: size of *a[]*
top: index of last input, declare it global.
Array Elements: *a*[0]…..*a*[*n*-1]



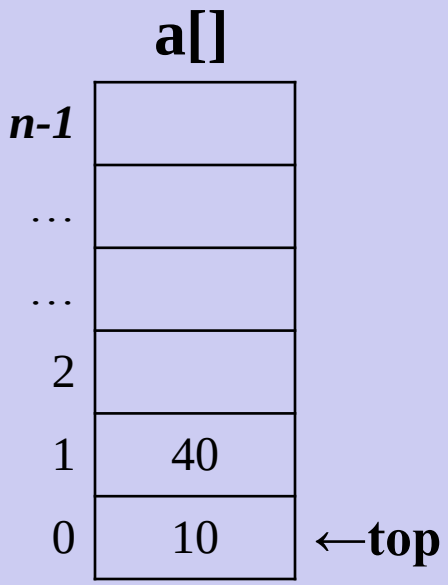
Suppose top = 1

Stack: pop() function

Topic 1: Write an Algorithm to pop element from a stack



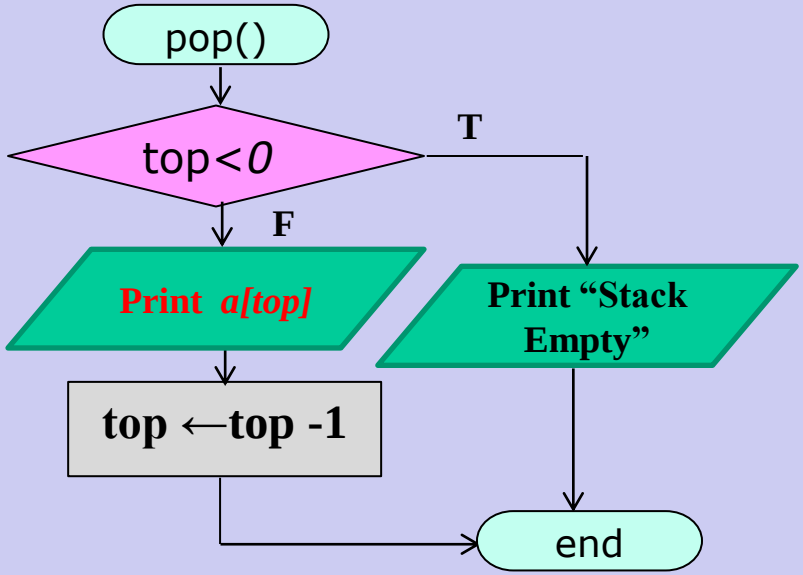
n: size of *a[]*
top: index of last input, declare it global.
Array Elements: *a*[0]…..*a*[*n*-1]



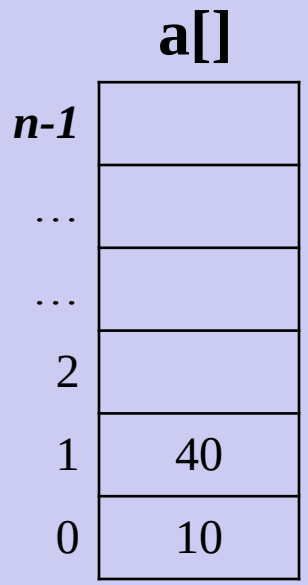
After deletion 2nd element, top=0

Stack: pop() function

Topic 1: Write an Algorithm to pop element from a stack



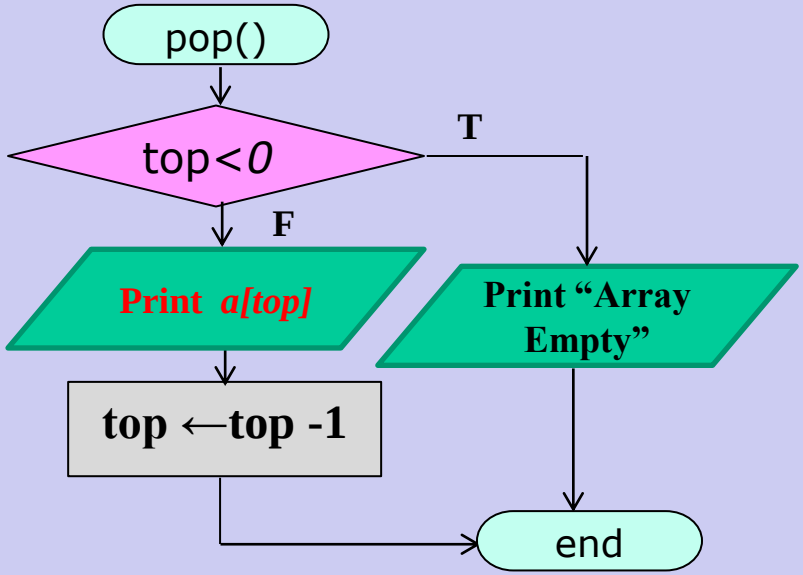
n: size of *a[]*
top: index of last input, declare it global.
Array Elements: *a*[0]…..*a*[*n*-1]



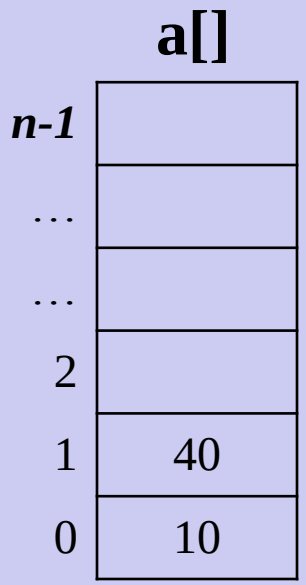
After deletion 1st element, top=-1

Stack: pop() function

Topic 1: Write an Algorithm to pop element from a stack



n: size of *a[]*
top: index of last input, declare it global.
Array Elements: *a*[0]…..*a*[*n*-1]



Try to delete more gives message "Stack Empty"

Polish Notation

What is Polish Notation?

Polish notation is a way of expressing arithmetic expression that avoids the use of brackets to define priorities for evaluation of operators.

Types of Polish Notation

1. **Infix** → Operator lies between two operands i.e. 2+3
2. **Prefix** → Operator lies before two operands i.e. +23
3. **Postfix** → Operator lies after two operands i.e. 23+

Polish Notation

Infix: 1+2x3+1/2 = ?

((1+(2x3))+(1/2)) = 7.5

Postfix (Polish notation):

Let A, B, be operands, ♦ an operator.

Instead of A♦B, write AB♦

Polish Notation

Example:

Infix:

((1+2)x(3+1))/2

Postfix:

((1+2)x(3+1))2/

(1+2)(3+1)x2/

(1+2)31+x2/

12+31+x2/

Infix:

((1+2)x(3+1))/2

Prefix:

/((1+2)x(3+1))2

/x (1+2)(3+1)2

/x+12(3+1)2

/x+12+312

Polish Notation

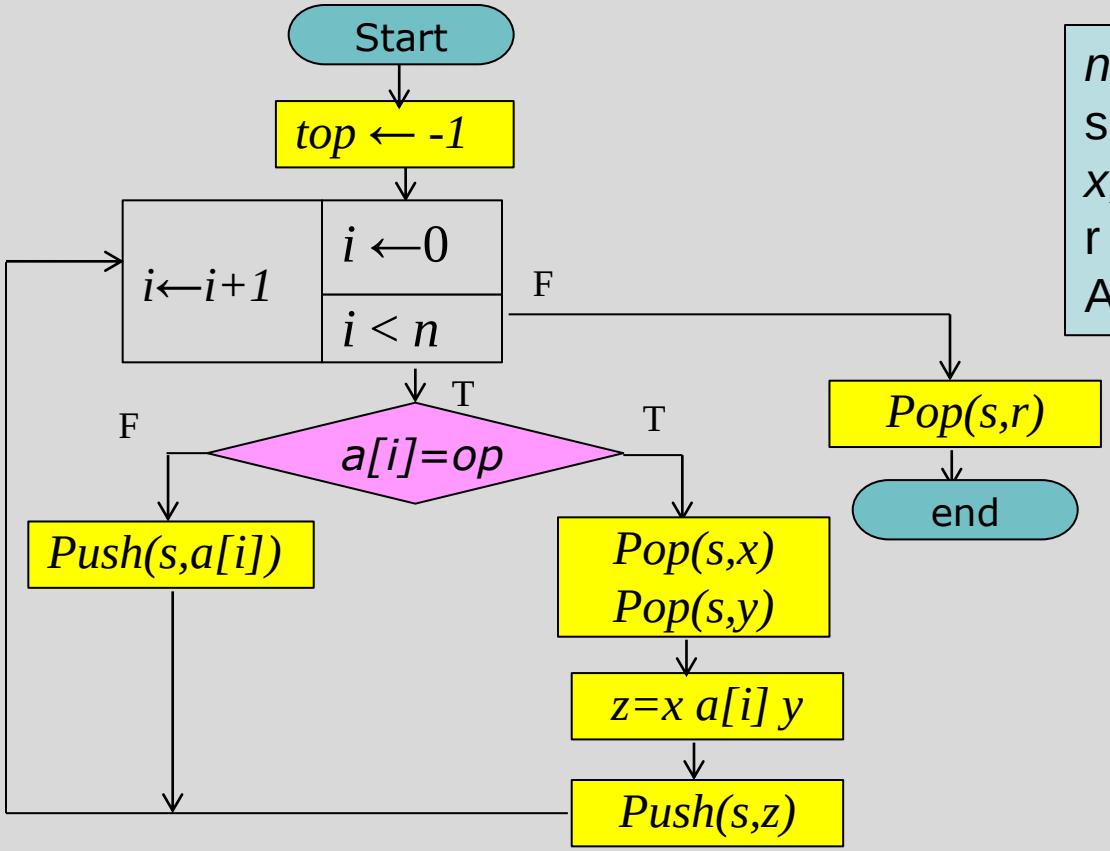
Advantage of Polish Notation:

No ambiguity!

A-B-C =? (A-B)-C A-(B-C)

AB-C- ABC--

Polish Notation



n: size of *a[]*
s: stack
x, y, z : temp variable
r : store result
Array Elements: *a*[0].....*a*[*n*-1]

a[]

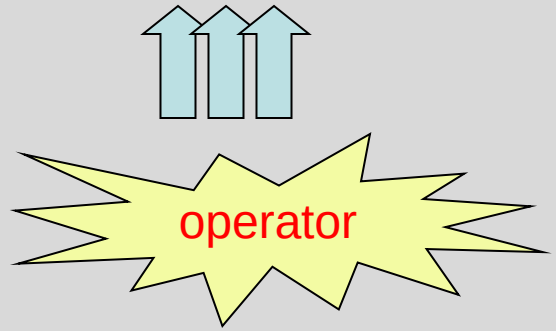
8	**/**
7	**2**
6	**x**
5	**+**
4	**1**
3	**3**
2	**+**
1	**2**
0	**1**

Example: 12+31+x2/ ((1+2)x(3+1))/2=6

Polish Notation

Example: 12+31+x2/

((1+2)x(3+1))/2=6



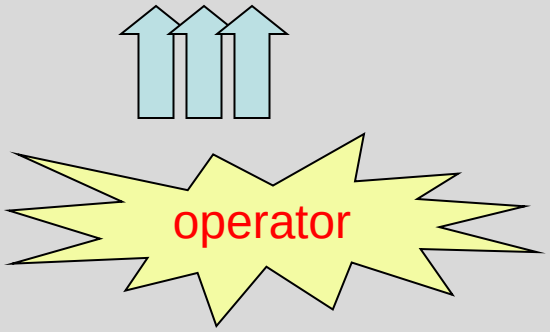
1 + 2 =3

2

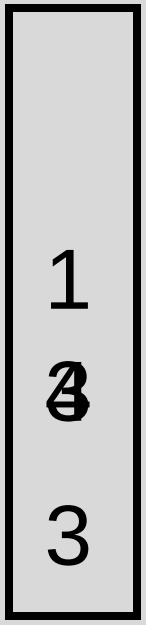
3

Polish Notation

Example: 12+31+x2/ ((1+2)x(3+1))/2=6

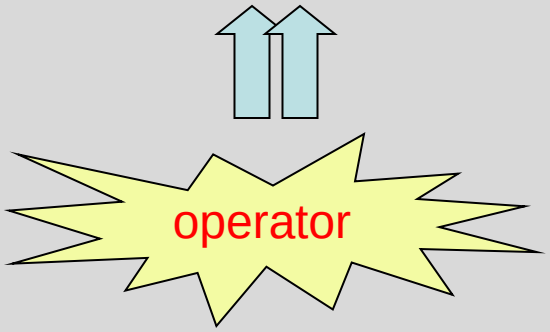


3 + 1 =4

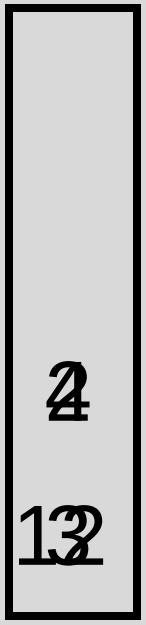


Polish Notation

Example: 12+31+x2/ ((1+2)x(3+1))/2=6



3 x 4 =12

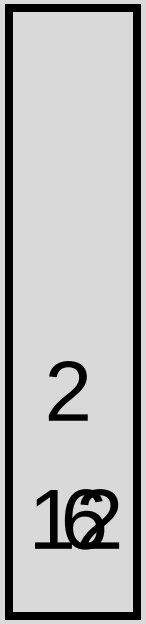


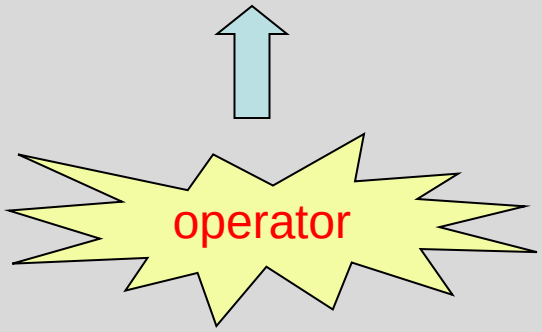
Polish Notation

Example: 12+31+x2/

((1+2)x(3+1))/2=6

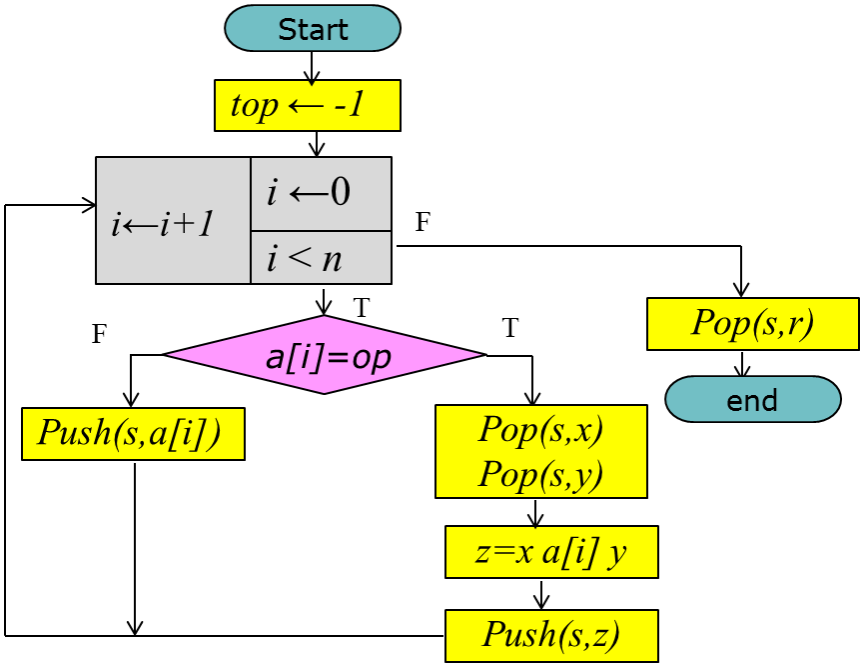
6

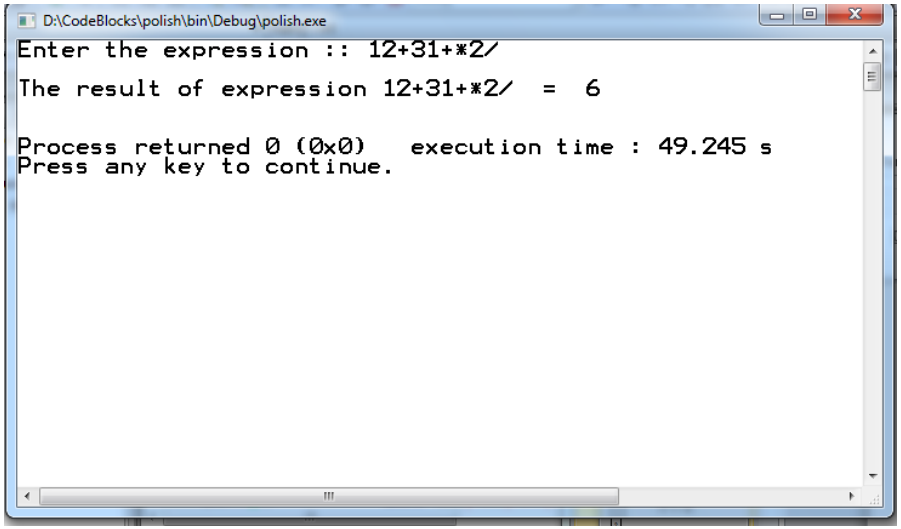




12 / 2 =6

Polish Notation






```
#include <stdio.h>
#include <stdlib.h>
int stack[20];
int top = -1;
void push(int x)
{ stack[++top] = x; }
int pop()
{  return stack[top--];}
int main()
{
        char exp[20];
        char *e;
        int n1,n2,n3,num;
        printf("Enter the expression :: ");
        scanf("%s",exp);
        e = exp;
        while(*e != '\0')
        {
                if(isdigit(*e))
                {  num = *e - 48;   push(num);    }
                else
                {
                        n1 = pop();   n2 = pop();   switch(*e)
                        {
                                case '+': { n3 = n1 + n2;   break;   }
                                case '-':  { n3 = n2 - n1;    break;   }
                                case '*':  { n3 = n1 * n2;    break;   }
                                case '/':  { n3 = n2 / n1;     break;   }
                        }
                        push(n3);
                }
                e++;
        }
        printf("\nThe result of expression %s  =  %d\n\n",exp,pop());
        return 0;
}
```

Assignments

Prob 1: An array c[] stores characters (alphabets and digits) then write an algorithm that creates two stacks to store alphabets and digits respectively from c[].

Prob 2: Write an algorithm that converts an infix expression to its prefix equivalent