CS2x1:Data Structures and Algorithms

Koteswararao Kondepu

k.kondepu@iitdh.ac.in

Outline

- Classifications of Data Structures
- Data Structures Overview
- Abstract Data Structures
    - Stack
    - Queue
    - Linked List

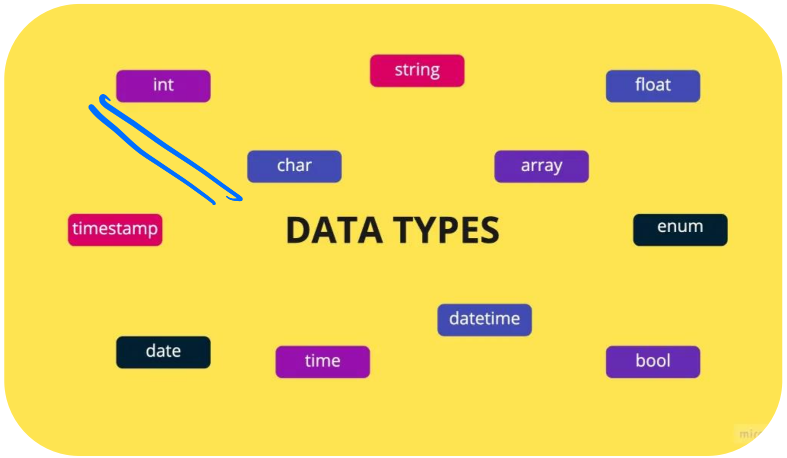
Data Type

- **Why do we need data types?**

x16
x32
x64

a = 6;

int a=4

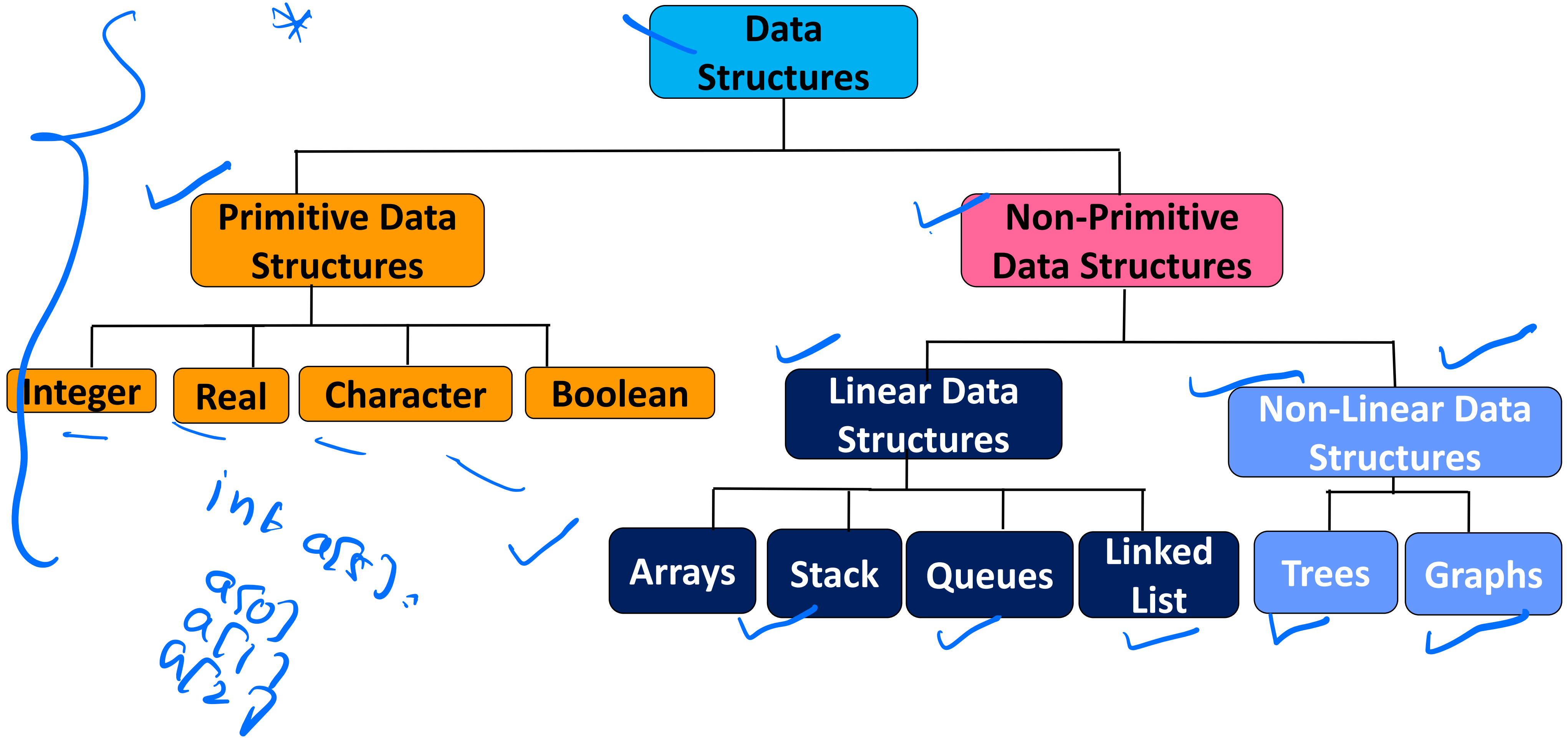


- *Data type* is an attribute of data, which tells the compiler (or interpreter) how the programmer intended to use the data.

- Is it really helps to reduce the coding effort?

- Is the data type determines how the computation is performed in underlying hardware?

Classification of Data Structures



Define: Data Structures

- *Data structure* is a special format for organizing and strong data
- *Data structure* is used to denote a particular way of organizing data for particular type of operations
- *Data structure* is a data organization, management, and storage format that enables *efficient* access and modification.
- *Data structure* is collection of data values, the relationships among them, and the functions or operations that can be applied to the data.

Abstraction

- *A real life example*: Lets consider a program that manages the student records, which allows to check if the student are opting a particular course or not.

- What student information is need for the record?
  - *Name, DoB, ID, email, mailing address, transcripts, hobbies, etc …*

- IS all the above properties are necessary to solve the given problem?

- Student Properties: **TYPE OF DATA**
  - Name
  - ID
- Operations performed by this program: **OPERATIONS ON THE DATA**
  - ✓ ADD (to add a student to the class)
  - ✓ SEARCH (if the particular student enrolled to the class or not)
  - ✓ DELETE (if the student dropped the course)

➢ No information about, how to store this data in computer memory and how to implement them.

Define: Abstract Data Type (ADT)

Abstract + Datatype

- *Abstract data type* defines the logical form of the data type → data and operations

- *Abstract data type* contains functions/operations that operate on its data (e.g., add, search and delete)

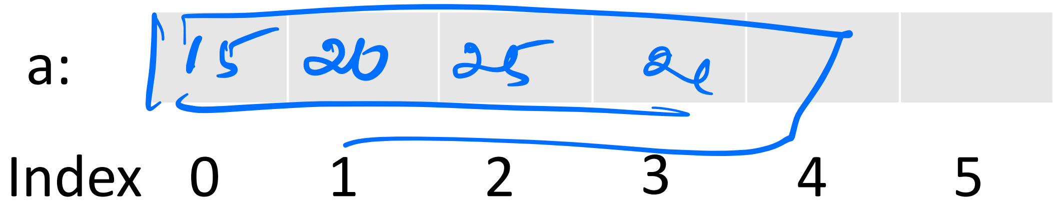
- *Abstract data type* describes the expected *behaviour* associated with a concrete data structure.

- *Abstract data type*: Data structures + their operations

- ✓ What are the commonly used *abstract data types*?
  - Stack, Queue and Lists

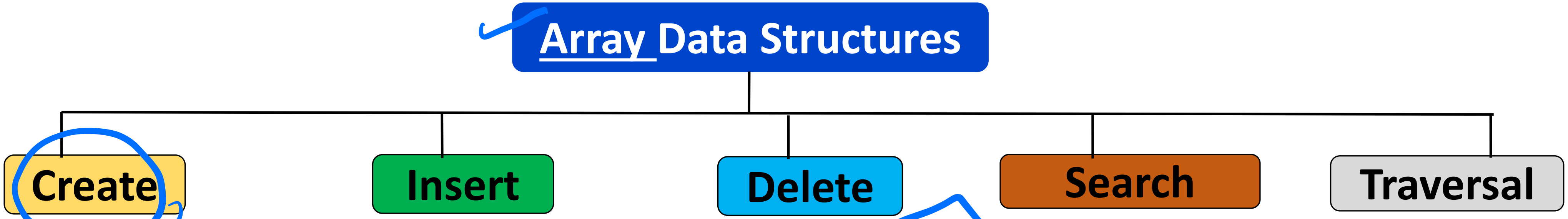
Arrays

- *Arrays* are the collection of <u>same data type</u> (homogeneous) item

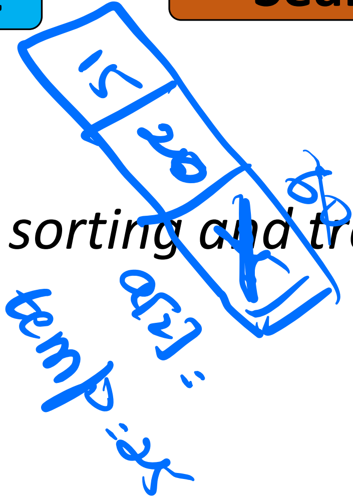


int a[6] = {15, 20, 25, 30, 5, 10}

Array: Operations



int arr[]={15, 20, 25}



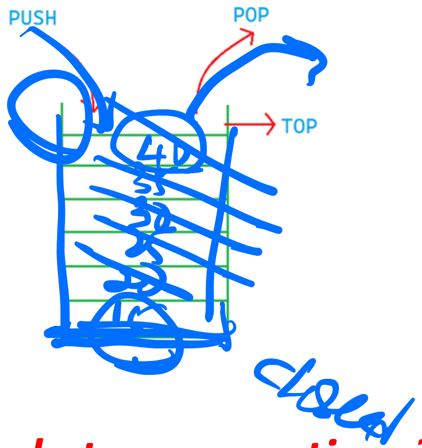
- *Operations: Insertions, deletion, searching, sorting and traversal (visiting each element)*

- ***Why do arrays fail?***
  - Cannot grow size of array dynamically
  - Allocation of larger size --- wasting space

Can we solve these problems by using any other data structures?

Define: Stacks

- *Stack is one-side open and the other side is closed* 15 20



  - *Last-In-First-Out (LIFO) or First-In-Last-Out (FILO)*
  - Insertion and deletion are done at one end, called ***TOP***

- **Major Operations**: Push / insert / add
  Pop / delete / remo
  - An element is inserted in a stack --- ***PUSH***
  - An element is removed from the stack --- ***POP***

- ***Why we have only two operations?***

- **Exceptions**:
  - ***Underflow*** – Trying to **POP** from an empty stack
  - ***Overflow*** – Trying to **PUSH** an element in a full stack

Stack of Books

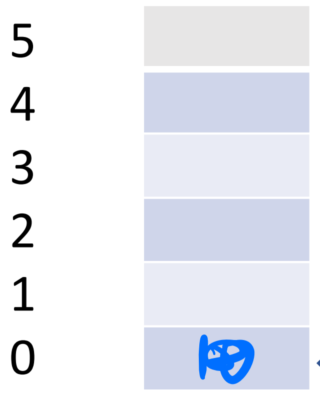
Stack of Dishes

Stack of Discs

Stack: PUSH Examples

TOP = -1

S

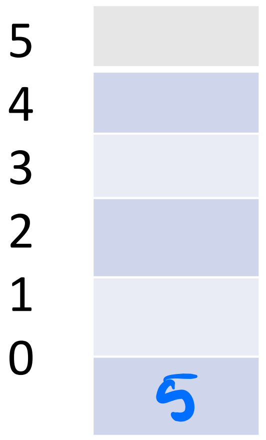


$\boxed{\text{TOP}}$ = 0

Stack is empty

S[TOP] ⇒ S[0]

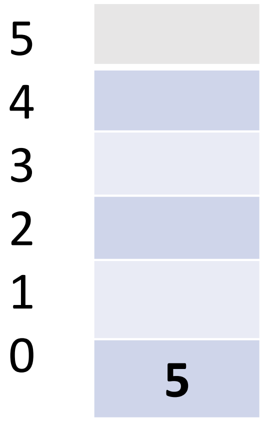
PUSH (5)



S [0] = 5

$\boxed{\text{TOP = 0 + 1}}$

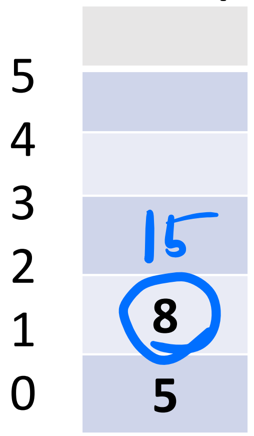
PUSH (8)



S [1] = 8

TOP = 1 + 1

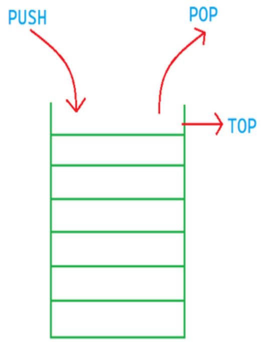
PUSH (15)



S [2] = 15

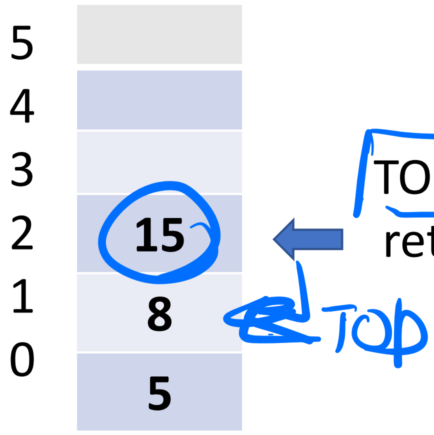
TOP = 2 + 1

```
PUSH (S, TOP, Val)
begin
    S[TOP] = Val;
    TOP = TOP + 1;
end
```



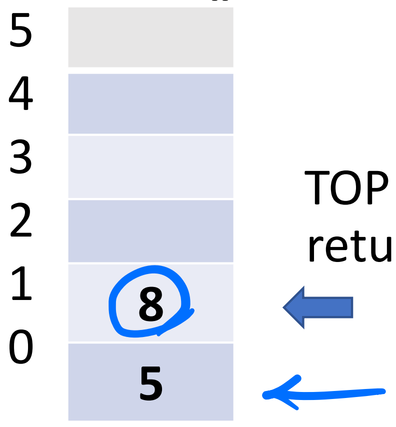
Stack: POP Examples

S = POP()



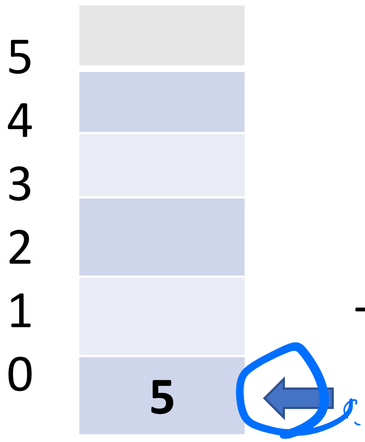
TOP = 3 - 1
return S[2]

S = POP()



TOP = 2 - 1
return S[1]

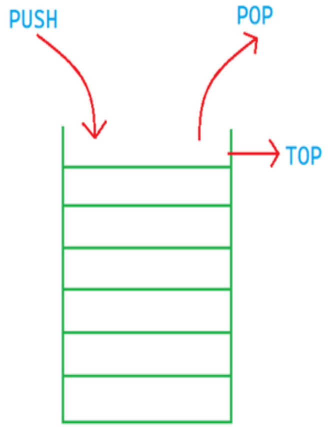
S = POP()



TOP = 1 - 1
return S[0]

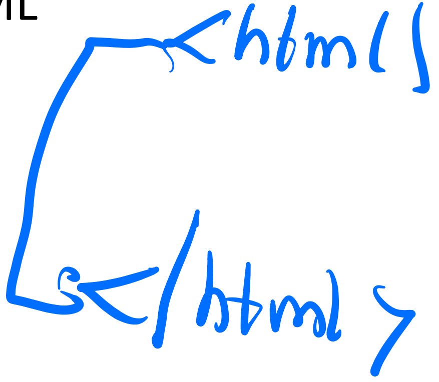
```
POP (S, TOP, Val)
begin
      TOP = TOP - 1;
      return S[TOP]
//return S[TOP--]
end
```

TO



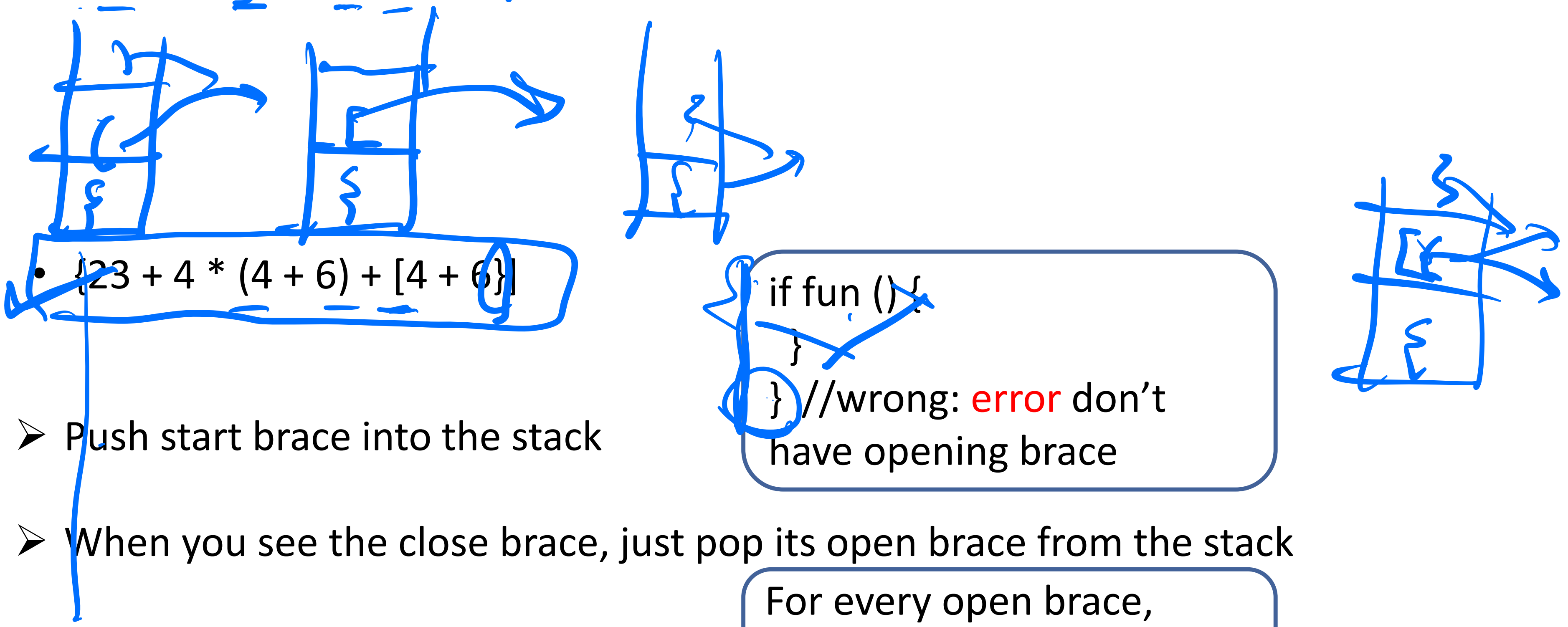
Stack: Applications

- Page-visited history in a web browser [Back Buttons]
- Balancing of symbols (e.g., (, { or [ )
- Infix to Postfix conversion
- Evaluation of postfix expression
- Implementation of function calls
- Matching Tags in HTML and XML



Stack: Applications – Parenthesis checking

- {23 + 4 * (4 + 6) + [4 + 6]}



- {23 + 4 * (4 + 6) + [4 + 6]}

```
if fun () {
  }
} //wrong: error don't
have opening brace
```

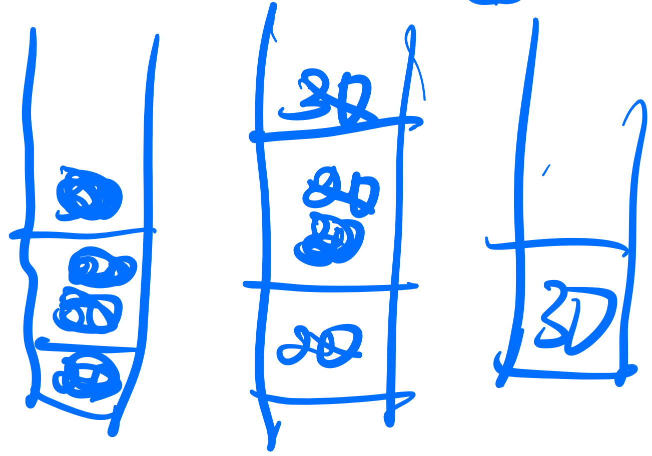
- ➢ Push start brace into the stack
- ➢ When you see the close brace, just pop its open brace from the stack
- ➢ No braces then ignore other things

For every open brace, there should be a close brace { () [] }

Stack: Applications – Examples

- Following sequence of operations is performed on the stack
  push (20), push (30), pop(), push (20), push (30), pop(), pop(), pop(),push(30), pop(). Sequence of popped elements:

a) 30, 20, 30, 20, 30 [✓ b)] 30, 30, 20, 20, 30 c) 20, 30, 30, 20, 30

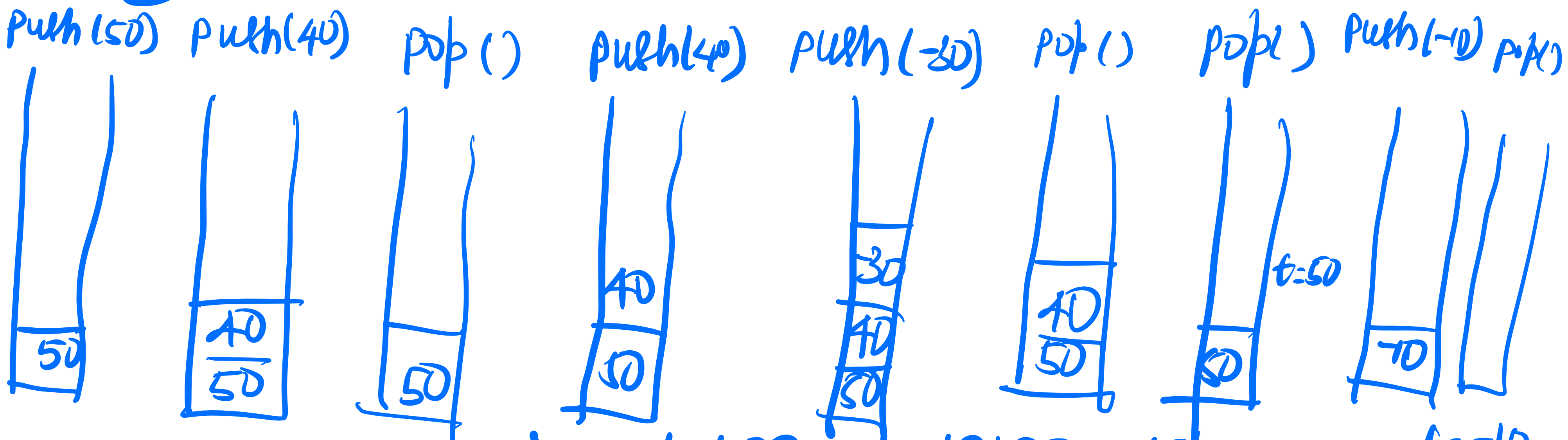


30, 30, 20, 20, 30

Stack: Applications – Examples

- Following sequence of operations is performed on the stack
  push (50), push (40), pop(), push (40), push (-30), pop(), pop(), t=pop(),push(-10), t=pop(). What is the value of t+20?

a) -10 [✓ b)] 10 c) 60 d) 70



$\Rightarrow$ $t+20 = -10+20 = 10$

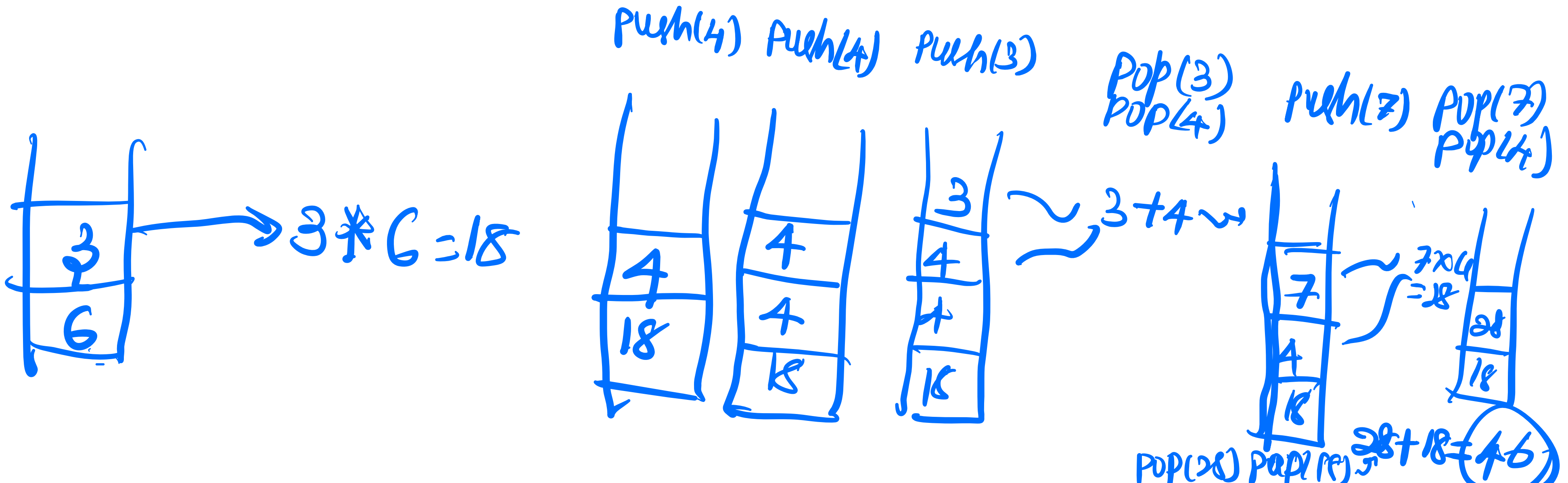
updated t value. t=-10

Stack: Applications – Examples (1)

- What is the output of the program for the following input?

  6 3 * 4 4 3 + * +

  a) 38 [✓ b)] 46 c) 40 d) 42



thank you!

email:
k.kondepu@iitdh.ac.in

NEXT Class: 20/04/2023