**On Memory:**

**Static:**

* Memory allocated on compile time.
* Memory allocated on prediction.
* Not memory efficient
* Fixed: once allocated cannot be changed.
* Easy to allocate: int a = 10; (4 byte in java)

**Dynamic:**

* Memory allocated on run time
* Memory allocated on demand only.
* Memory efficient (as per need)
* On demand: can increment and decrement as per need.
* Many languages need special command to give

E.g. C++: new/delete.

**On Access:**

**Linear:**

* Accessed in a sequence one after another
* Simple to code
* Slow in access

E.g. Stack, queue, linked list

Array is Linear structure which can be accessed directly also.

**Non-linear:**

**List of Data Structures:**

* Stack
* Queue
* Linked list
* Tree
* Graph

**ADT (Abstract Data Types):**

* Set of methods one must implement to realize /implement data structure.
* Concentrates on encapsulation and abstraction.
* Allow

**ADT:**

**create(size):** creates array/stack of n size, init tos =-1

**push(e):** enter element on stack by tos +1

**pop():** e removed element from stack[tos], tos-1

**isFull():** T/F returns True if full else false.

**isEmpty():** T/F returns True if empty else false.

**peek():** returns element at tos

**printStack():** prints in LIFO manner from tos to 0. (LIFO)

Applications:

1. **Recursion:**

int fact (int no)

{

if (no==1)

return 1;

else

return no\*fact(no-1);

}

1. **Wellness/balance of: { [ ( ) ] }**

Output of code: Balance /expected /unexpected

Stack Push/Pop

{ { { } } }

* .length() is property in array.
* .length() is method in string.

1. **Decimal to Binary Conversion**

Take remainder and put on stack.

At end remove the data from stack and accept it as answer

1. **Reversing a String**

Char by Char Push(e) and then pop()

1. **Expression evaluation and conversion**

Prefix: +ab for software

Infix: a+b for humans

Postfix: ab+ for hardware

Rules of Precedence:

Always go left to right

First solve higher then lower (**^**: highest **\***,**/**,**%**: higher **+**,**-**: lower)

In case of brackets start from the innermost.

Infix To Postfix:

1. Go left to right
2. If operand copy to postfix
3. If { [ ( push on stack
4. If } ] ) pop all till ( [ { and copy poped to postfix
5. If operator push IFF prec(new)>prec(tos\_operator) else pop till condition satisfies or stack becomes empty.

Infix To Prefix:

1. Go right to left (end to start)
2. If operand copy to prefix
3. If } ] ) push on stack
4. If { [ ( pop all till } ] ) and copy poped to prefix
5. If operator push IFF prec(new)>=prec(tos\_operator) else pop till condition satisfies or stack becomes empty.
6. Reverse and print prefix.

=

1. **To check if the string is palindrome or not**

STACK:

* Linear
* Nature wise LIFO.
* Uses TOS(top of stack) peak.