Ex1: write program to create stack with size=10 to store the student's degree, compute number of succeed students and print the original stack.

Ex2: write function to insert three elements to stack.

Stack applications:

1. **Treat the programs which have sub programs.**
2. Mathematic expiration
   As now the Mathematic expiration divides into three expirations:

- Infix notation:
  Where the sine in middle of operands → x/2, a-b, 3+4
- Prefix Notation: → /x 20, -a b, + 3 4
- Postfix Notation: → x 20 /, a b -, 3 4 +

Note: In high level languages, infix notation cannot be used to evaluate expressions.

Type of operation	the Priority
^ (power), Unary (-), Unary (+), Not	4
*, /, AND, DIV, MOD	3
+ , - , OR	2
= , < , > , != , <= , >=	1

Note:

- In high level languages, infix notation cannot be used to evaluate expressions.
- A common technique is to convert a infix notation into postfix notation, then evaluate it.

Concert Infix notation to Postfix Notation algorithm using two stacks:

1. Initialize two empty stacks, (st1) to store operands, (st2) to store the operators of the mathematic operations.

- Read the tokens from the infix string one at a time from left to right
- If the token is operands then push it in **st1**.
- If the token is lift parentheses then push it in **st2.**
- If the token is right parentheses then pop all the operators from **st2** and push it in **st1 .**

- Repeat this step until lift parentheses is encountered, then remove and ignore the lift parentheses and right parentheses.
- If the token is operator then pop all operators (if you found) which have priority higher than or equal the priority of the current operator from **st2** and push it in **st1,** then push the current operators in **st2.**

  Else

  Push the current operator in **st2**

2. After all the characters are read and the **st2** is not empty then Pop the stack and add the tokens to the **st1**
3. Return the Postfix **st1.**

Ex1: convert the following expressions from infix to postfix using two stacks.

a-b(c+d)/(e-f)^g*h*

st2	st1	symbol	steps
.....	a	a	1
-	a	-	2
-	ab	b	3
- *	ab	*	4
- * (	ab	(	5
- * (	abc	c	6
-* (+	abc	+	7
- * (+	abcd	d	8
- *	abcd+	)	9

نلاحظ هنا عند ورود القوس الأيمن يتم إخراج (نقل) جميع العمليات الحسابية لغاية القوس الأيسر من المكدس(ST2) إلى(ST1) مع إخراج القوس الأيسر ليهمل هو والقوس الأيمن.

- /	abcd+*	/	10
- / (	abcd+*	(	11
- / (	abcd+*e	e	12
- / (-	abcd+*e	-	13
- / (-	abcd+*ef	f	14
- /	abcd+*ef-	)	15
- / ^	abcd+*ef-	^	16
- / ^	abcd+*ef-g	g	17
- *	abcd+*ef-g^/	*	18

لان أسبقية الضرب (*) =< اسبقية الرفع (^) والقسمة(/)

- *	abcd+*ef-g^/h	h	19

هنا انتهت جميع المدخلات لذا ينقل المتبقي في المكدس(ST2)الى المكدس(ST1) بالتتابع ليصبح

abcd+*ef-g^/h*-	20

Ex2: convert from infix to postfix using two stacks. *M:=X/6+(a-2*(b/3)^5+f)^2*

st2 for operators	st1 for operands	input char	Step no.
…..	M	M	1
:=	M	:=	2
:=	MX	X	3
:=/	MX	/	4
:=/	MX6	6	5
:=+	MX6/	+	6
:=+(	MX6/	(	7
:=+(	MX6/a	A	8
:=+(-	MX6/a	-	9
:=+(-	MX6/a2	2	10
:=+(-*	MX6/a2	*	11
:=+(-*(	MX6/a2	(	12
:=+(-*(	MX6/a2b	B	13
:=+(-*(/	MX6/a2b	/	14
:=+(-*(/	MX6/a2b3	3	15
:=+(-*	MX6/a2b3/	)	16
:=+(-*^	MX6/a2b3/	^	17
:=+(-*^	MX6/a2b3/	5	18
:=+(+	MX6/a2b3/5^*-	+	19
:=+(+	MX6/a2b3/5^*-F	F	20
:=+	MX6/a2b3/5^*-F+	)	21
:=+^	MX6/a2b3/5^*-F+	^	22
:=+^	MX6/a2b3/5^*-F+2	2	23
……	MX6/a2b3/5^*-F+2^+:	…..	24

Ex3: convert from infix to postfix using two stacks. (A>B)AND ((E-C>A)OR(G<F)).

Evaluate postfix expressions algorithm:

1. Initialize one stack (**st**) .

2. Read the tokens from the postfix string one at a time from left to right •

3. If token is operand, push it in **st.**

4. If the token is operator, implement this operation on the two operand in **st** and then put the result in **st**.

5. After all the characters are read, the result in st is the final result.

Ex: convert the following infix expression to postfix and compute its value.

7+8-6*3/2.

After conversion we have the postfix 78+63*2/-

To get the value :

St	Input char	Step no.
7	7	1
7 8	8	2
15	+	3
15 6	6	4
15 6 3	3	5
15 18	*	6
15 18 2	2	7
15 9	/	8
6	-	9