Applications of Stack

Applications of Stack

1. Expression Conversions
2. Expression Evaluations
3. Decimal to binary conversion
4. Reversing a String
5. Parsing Well formed parenthesis
6. Storing function calls

1. Expression Conversion

- Expression.

Expression is a string of operands and operators. The operands are some numerical values and operators are of two types:

Unary Operators: ‘+’ and ‘-’

Binary Operators ‘+’ , ‘-’ , ‘*’ and ‘/’ and ^ or ↑

The expressions are of three types

1. Infix Expression (A + B)
2. Postfix Expression (AB+)
3. Prefix Expression (+AB)

1. Expression Conversions.

- Infix to Postfix conversion

Priority:

Operator	Precedence in Stack	Precedence in expression
^ or ↑	3	4
*, /	2	2
+, -	1	1
(	0	4

Rules :

1. The expression is to be read from left to right.
2. Read one character at a time from infix expression
3. Make use of stack to store the operators.
4. There should not be any parenthesis in the postfix expression.

Algorithm:

1. Read the expression from left to right.
2. If the input symbol read is '(' then PUSH it on to the STACK.
3. If input symbol read is an Operand then place it in postfix expression.
4. If the input symbol read is an operator then,
   - i. Check if the precedence of the operator which is in STACK has greater or equal precedence than the precedence of the operator read, if so, then remove that symbol from STACK and place it in the postfix expression. Repeat step 4(i) till you get the operator in the STACK has greater precedence than the operator being read.
   - ii. Otherwise PUSH the operator being read onto the STACK.
5. If the input symbol read is a closing parenthesis ')' then POP all the operators from the STACK, place them in postfix expression till the opening parenthesis is not popped. The '(' should not be placed in the postfix expression.
6. Finally print the postfix expression.

Examples

1. A + B
2. A + B – C
3. (A + B) * C
4. (A + B) * (C – D)
5. A * B + (C – D / E)
6. A – B / (C * D ^ E)
7. ((A + B) * C - (D – E)) ^ (F + G)
8. A ^ B * C – D + E / F / (G + H)
9. (((A / (B ↑ C)) + (D * E)) - (F * G))

Examples

1. **A + B #**

Input Symbol Read	Stack	Output String
A		A
+	+	A
B	+	AB
#		AB+

Examples

1. A + B - C #

Input Symbol Read	Stack	Output String
A		A
+	+	A
B	+	AB
-	-	AB+
C	-	AB+C
#		AB+C-

Expression Evaluation

Algorithm:

1. Read the postfix expression from left to right.
2. If the input symbol read is an operand then PUSH it onto the STACK.
3. If the input symbol read is an operator, then POP two operands and perform arithmetic operations if operator is

+	**then, result = opearnd_2 + operand_1**
-	**then, result = operand_2 - operand_1**
*****	**then, result = operand_2 * operand_1**
/	**then, result = operand_2 / operand_1**

4. PUSH the result onto the STACK.
5. Repeat step 1- 4 till the postfix expression is not over.

e.g. A B + C D - * # if A = 4, B = 2, C = 6 and D = 3

Examples:

1. Evaluate the following postfix expressions and find the value of expression.

   i. 5, 6, 2, +, *, 12, 4, /, -

   ii. 12, 7, 3, -, /, 2, 1, 5, +, *, +

   iii. 12, 4, *, 7, 8, 9, /, -, +

2. Convert the following expressions from infix to postfix form and then evaluate them.

   i. (A + B) * C Where A = 3, B = 4, C = 5

   ii. (A + B) * (C - D) Where A = 3, B = 4, C = 5, D = 6

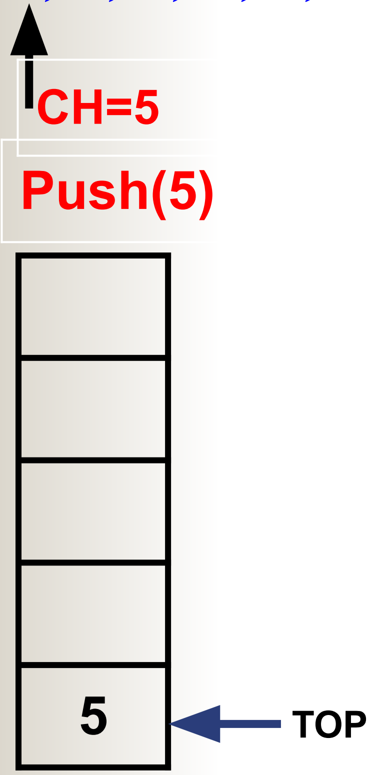
   iii. (((A / (B ↑ C)) + (D *E)) - (A * C))
   Where A = 27, B = 3, C = 2, D = 3, E = 17

Examples:

5, 6, 2, +, *, 12, 4, /, -

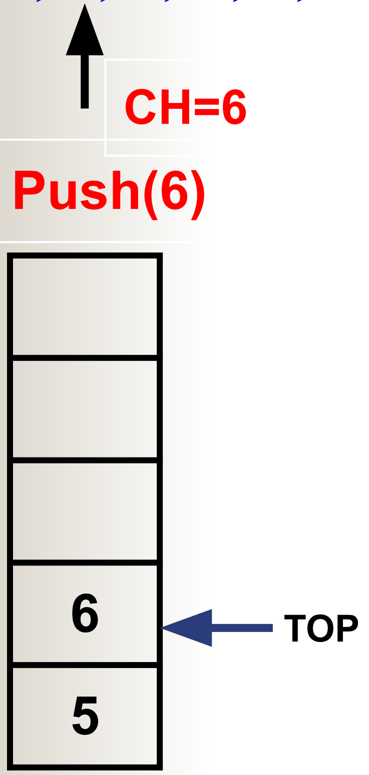
CH=5

Push(5)



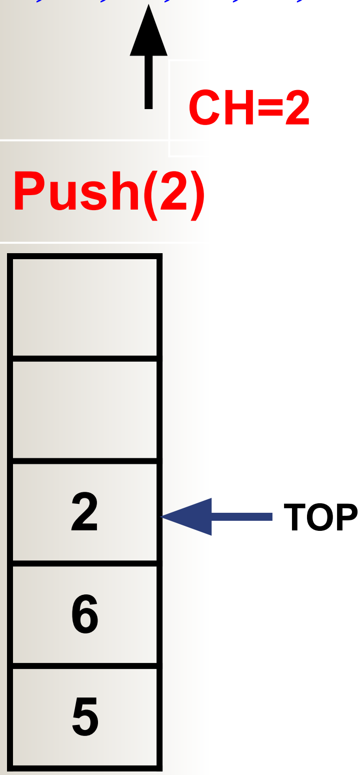
Examples:

5, 6, 2, +, *, 12, 4, /, -



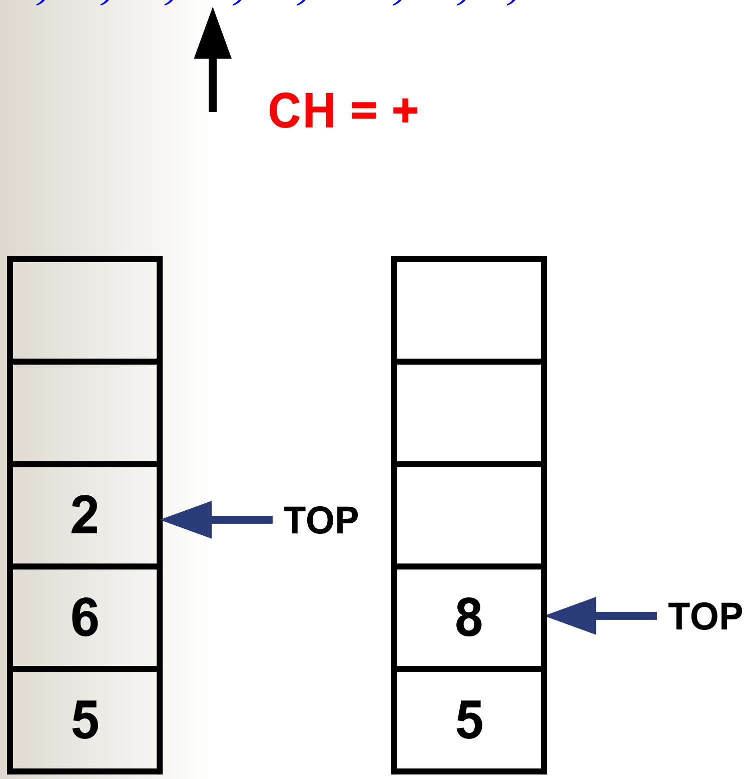
Examples:

5, 6, 2, +, *, 12, 4, /, -



Examples:

5, 6, 2, +, *, 12, 4, /, -



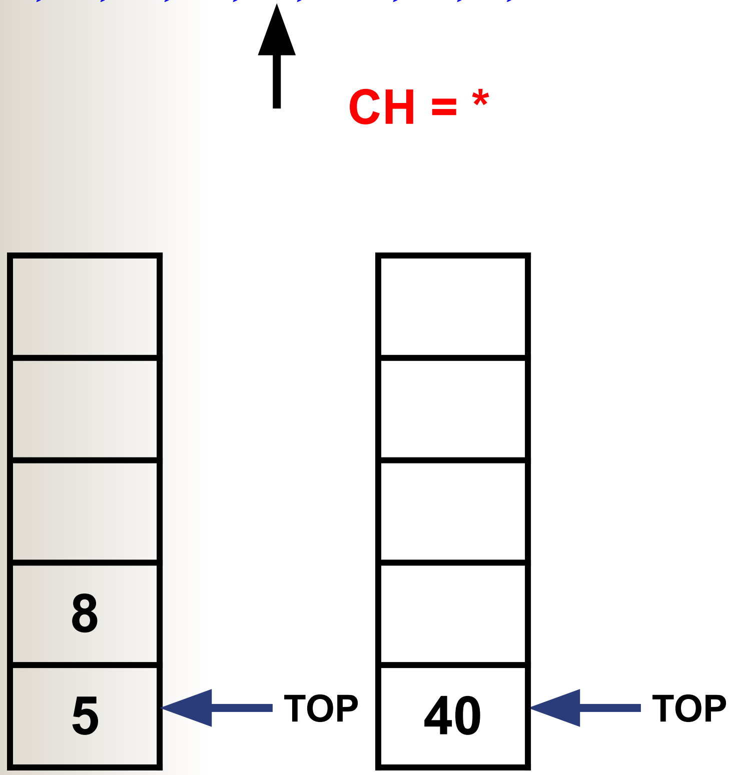
pop() top two elements i.e.
opernd_1 = 2
opernd_2 = 6

result = opernd_2 + opernd_1
result = 6 + 2 =8

push(8) back to stack

Examples:

5, 6, 2, +, *, 12, 4, /, -



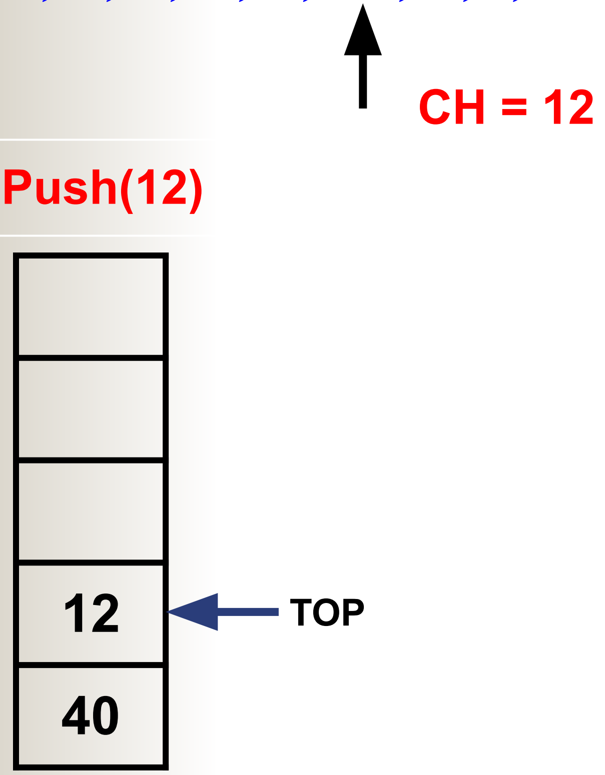
pop() top two elements i.e.
opernd_1 = 8
opernd_2 = 5

result = opernd_2 * opernd_1
result = 5 * 8 = 40

push(40) back to stack

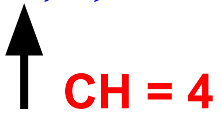
Examples:

5, 6, 2, +, *, 12, 4, /, -

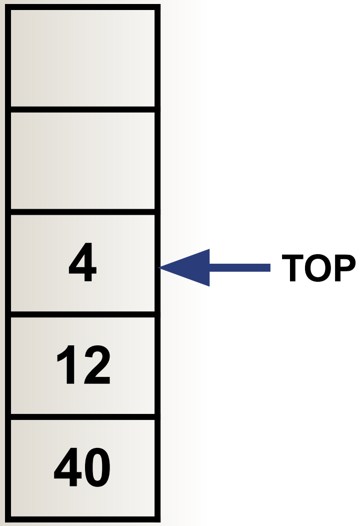


Examples:

5, 6, 2, +, *, 12, 4, /, -

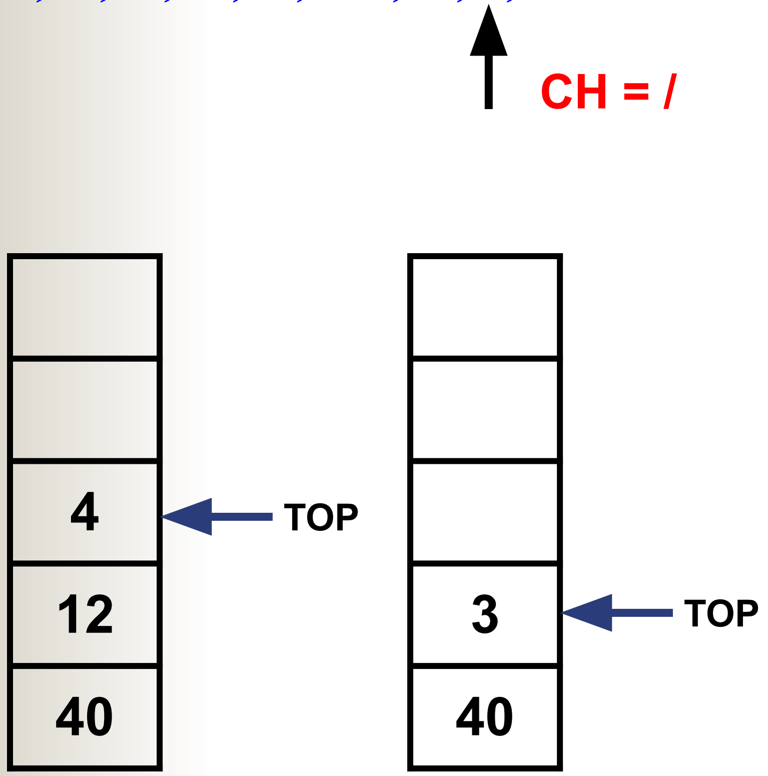


Push(4)



Examples:

5, 6, 2, +, *, 12, 4, /, -



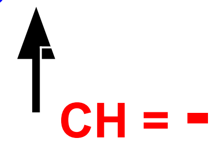
pop() top two elements i.e.
opernd_1 = 4
opernd_2 = 12

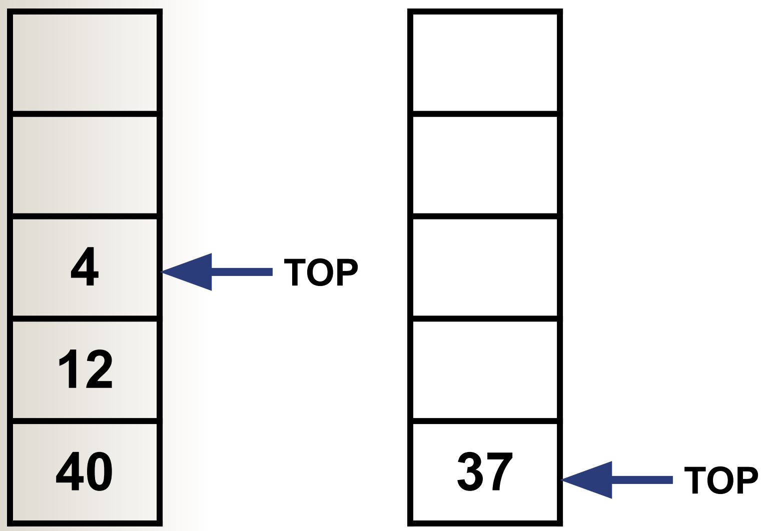
result = opernd_2 / opernd_1
result = 12 / 4 = 3

push(3) back to stack

Examples:

5, 6, 2, +, *, 12, 4, /, -





pop() top two elements i.e.
opernd_1 = 3
opernd_2 = 40

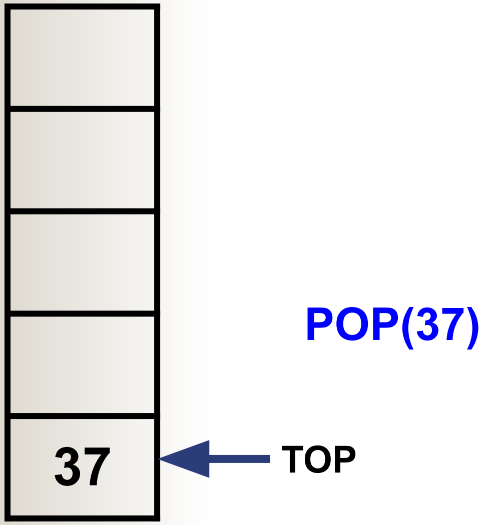
result = opernd_2 - opernd_1
result = 40 - 3 = 37

push(37) back to stack

Examples:

5, 6, 2, +, *, 12, 4, /, -

End of String(Expression)



Result = 37

Home Work

Convert the following from infix to postfix expressions

1. A * (B + C) / D - G
2. (A + B) * D + E / (F + A * D) + C
3. A + (B * C - (D / E – F) * G) * H
4. (A + B) * D + E / (F + A * D) + C
5. ((A + B) * C - (D – E)) ^ (F + G)
6. A ^ B * C – D + E / F / (G + H)
7. (((A / (B ↑ C)) + (D * E)) - (A * C))