

Data Structure & Algorithm

STACK

Linear Data Structures

- There are certain frequent situations in computer science when one wants to restrict insertion and deletions so that they can take place only at the beginning or at the end not in the middle.

– Stack

Stack



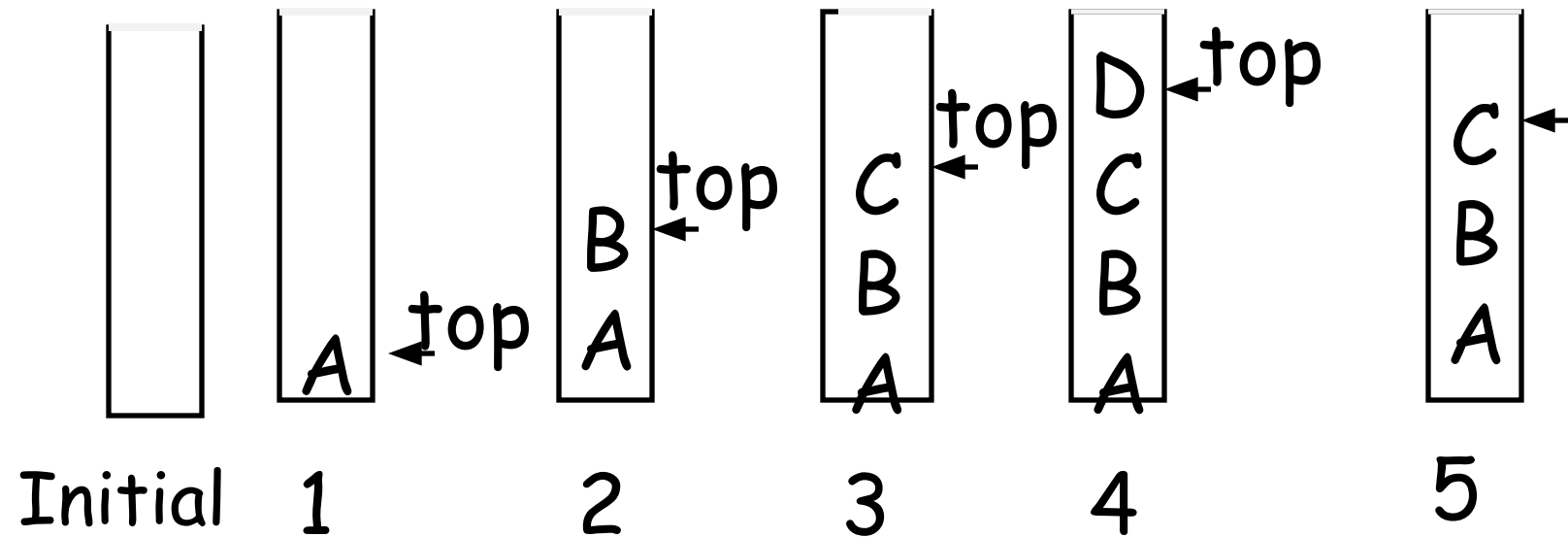
Stack

- A Stack is a list of elements in which an element may be inserted or deleted only at one end, call top of the Stack
- Two basic operations are associated with Stack
 - Push : Insert an element into a stack
 - Pop : Delete an element from a stack

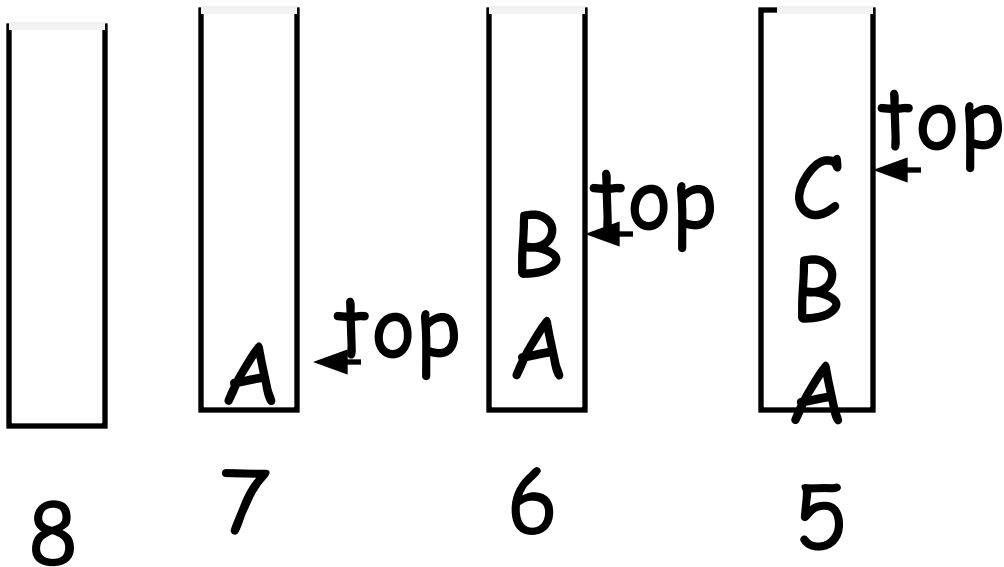
Stack

- Stores a set of elements in a particular order
- Stack principle: **LAST IN FIRST OUT = LIFO**
- It means: the last element inserted is the first one to be removed
- Which is the first element to pick up?

Last In First Out



Last In First Out



Representation of Stack

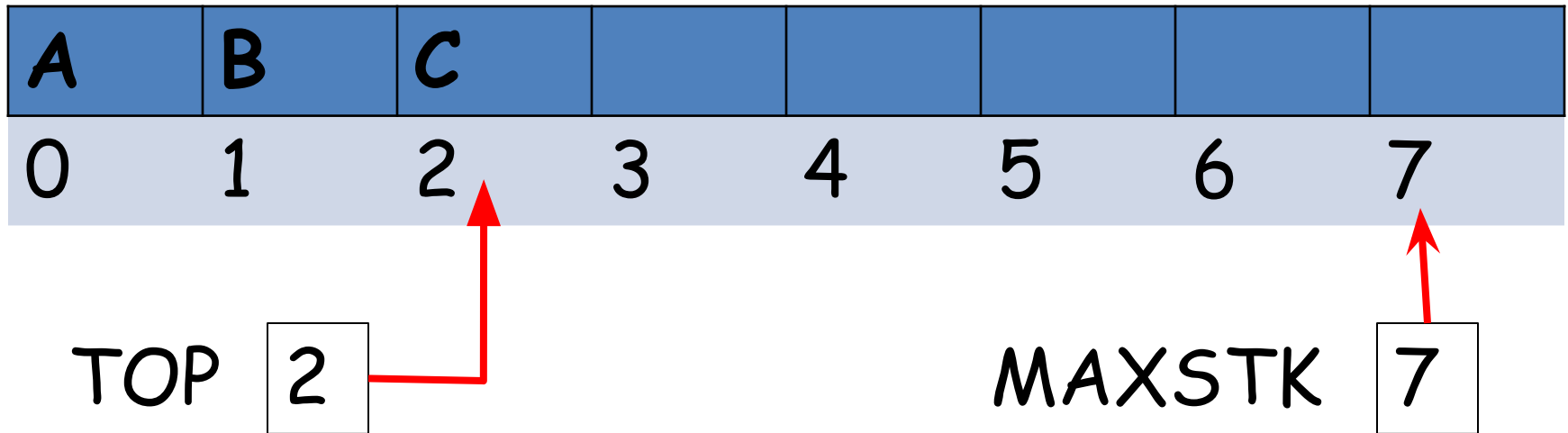
Stack can be represented in two different ways :

[1] Linear ARRAY

[2] One-way Linked list

Array Representation of Stack

STACK



TOP = -1 or TOP = NULL will indicate that the stack is empty

PUSH Operation

Perform the following steps to PUSH an ITEM onto a Stack

- [1] If $TOP = MAXSTK$, Then print:
Overflow, Exit [Stack already filled]
- [2] Set $TOP = TOP + 1$
- [3] Set $STACK[TOP] = ITEM$ [Insert
Item into new TOP Position]
- [4] Exit

POP Operation

Delete top element of *STACK* and assign it to the variable *ITEM*

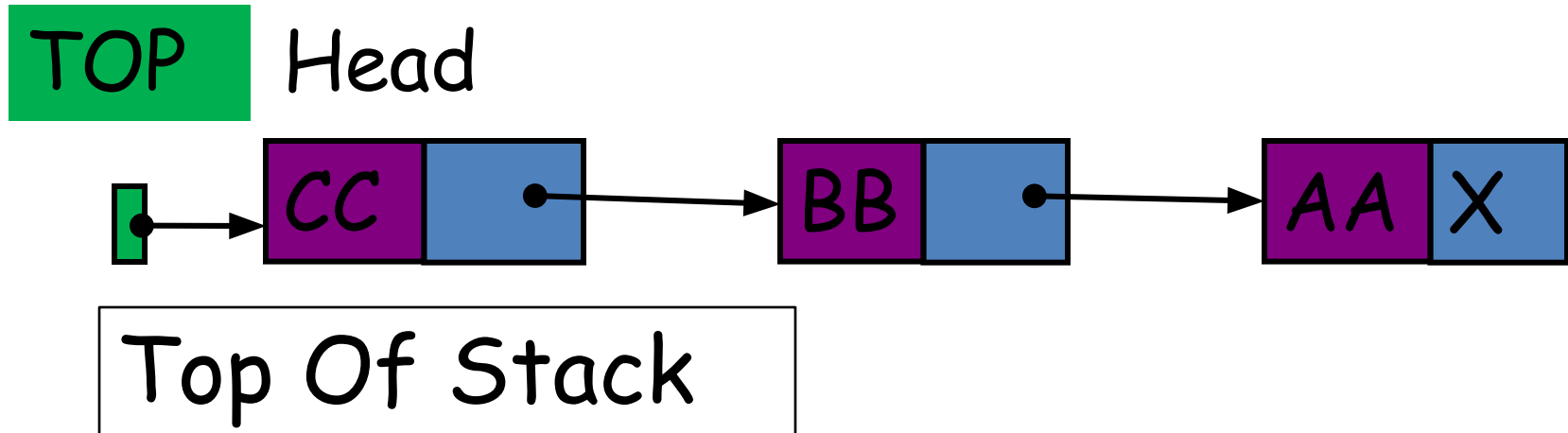
[1] If $TOP = -1$, Then print Underflow and Exit

[2] Set $ITEM = STACK[TOP]$

[3] Set $TOP = TOP - 1$

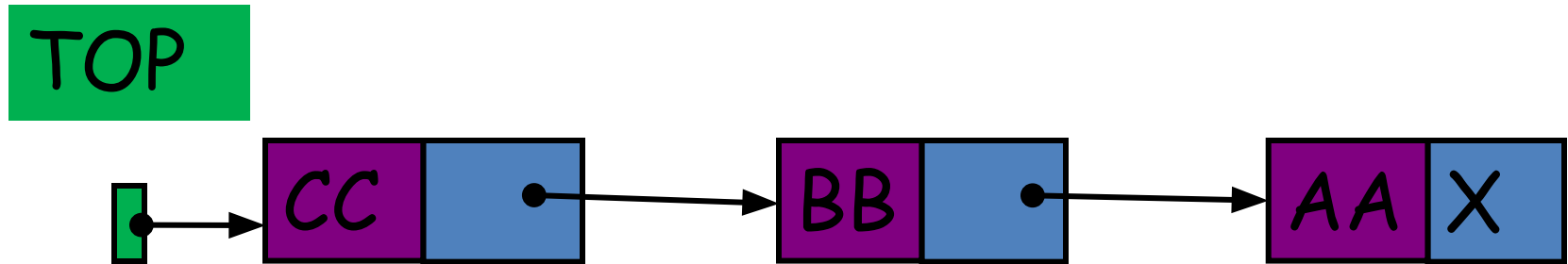
[4] Exit

Linked List Representation of Stack



PUSH Operation

- Push operation into the stack is accomplished by inserting a node into the front of the list [**Insert it as the first node in the list**]

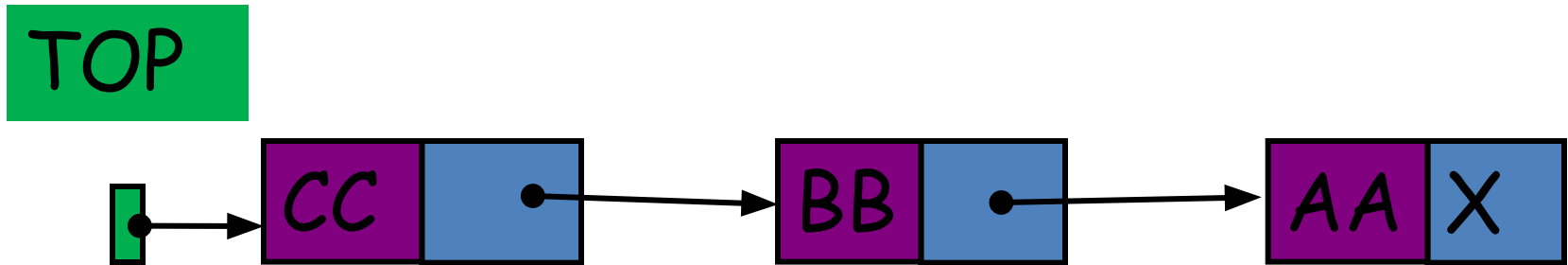


PUSH DD into STACK

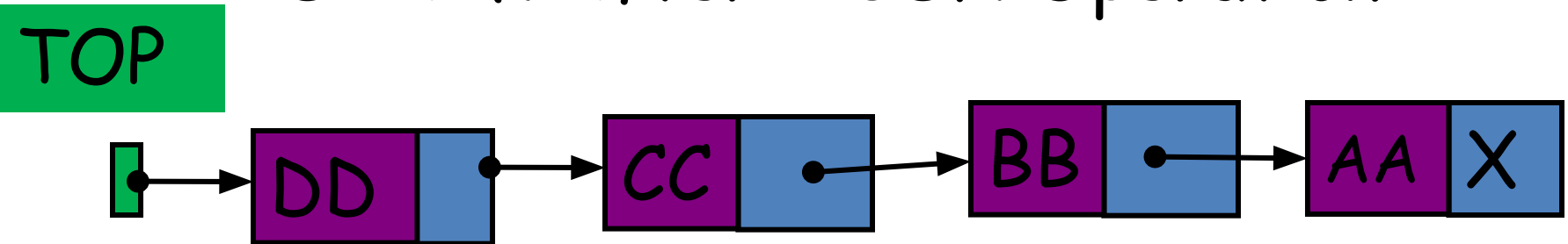


PUSH Operation

STACK before PUSH Operation



STACK After PUSH Operation



PUSH Operation

[1] NEW->INFO = ITEM

[2] NEW->LINK = TOP

[3] TOP = NEW

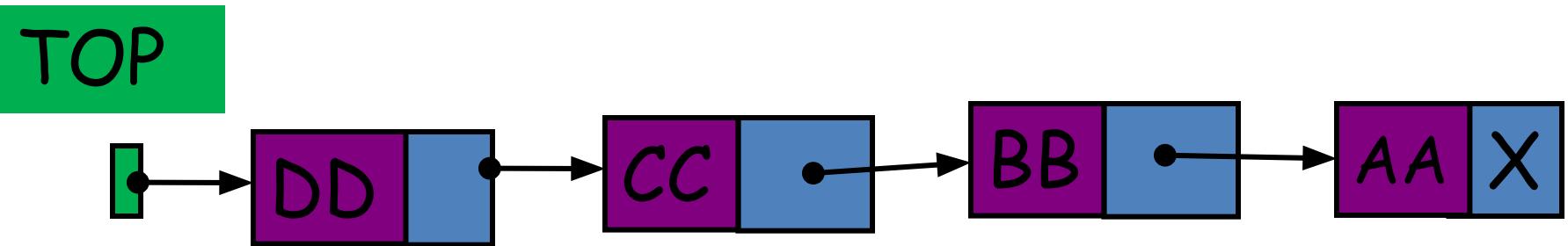
[4] Exit

POP Operation

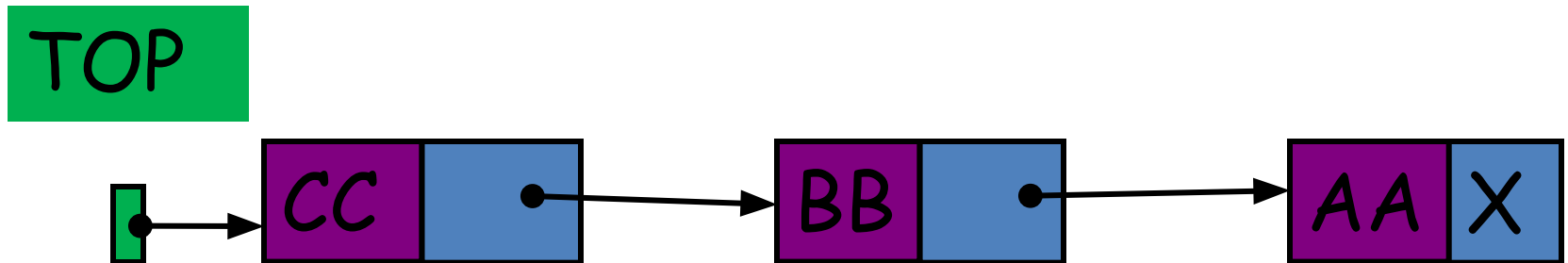
- POP operation is accomplished by deleting the node pointed to by the TOP pointer [**Delete the first node in the list**]

POP Operation

STACK before POP Operation



STACK After POP Operation



POP Operation

- [1] IF TOP == NULL Then Write Underflow and Exit
- [2] Set ITEM = TOP->INFO
- [3] Set TOP = TOP->LINK
- [4] Exit

Arithmetic Expression; Polish Notation

- Let Q be an arithmetic expression involving constant and operations
- Find the value of Q using reverse Polish (**Postfix**) Notation

- **Infix notation** [Operator symbol is placed between two Operand]

$A + B$, $C - D$, $E * F$, G / H

$(A + B) * C$ and $A + (B * C)$

- **Polish Notation** [Operator symbol is placed before its operand]

$+AB$, $-CD$, $*EF$, $/GH$

Polish Notations are frequently called
Prefix

- Infix expression to Polish Notation
[] to indicate a partial translation

$$(A+B)*C = [+AB]*C = *+ABC$$

$$A+(B*C) = A+[*BC] = +A*BC$$

$$(A+B)/(C-D) = [+AB]/[-CD] = /+AB-CD$$

Polish Notation

- The fundamental property of Polish notation is that the order in which the operations are to be performed is completely determined by the positions of the operators and operand in the expression.
- One never needs parenthesis when writing expression in Polish notations

Reverse Polish Notation

- Operator symbol is placed after its two operand

$AB+$, $CD-$, $EF*$, $GC/$

$$(A+B)/(C-D) = [AB+]/[CD-] = AB+CD-/$$

- One never needs parenthesis to determine the order of the operation in any arithmetic expression written in reverse Polish notation.
- Also known as Postfix notation

- Computer usually evaluates an arithmetic expression written in infix notation in two steps:
- First Step: Converts the **expression to Postfix notation**
- Second Step: **Evaluates the Postfix expression.**

Evaluation of Postfix Expression

- Algorithm to find the **Value** of an arithmetic expression **P** Written in **Postfix**

- [1] Add a right parenthesis ")" at the end of P. [This act as delimiter]
- [2] Scan P from left to right and repeat Steps 3 and 4 for each element of P until the delimiter ")" is encountered

Evaluation of Postfix Expression

[3] If an operand is encountered, put it on STACK

[4] If an operator \otimes is encountered, then

(a) Remove the two top elements of STACK, where A is the top element and B is the next-to-top element

(b) Evaluate $B \otimes A$

(c) Place the result of (b) on STACK

Evaluation of Postfix Expression

[5] Set **Value** equal to the top element of
STACK

[6] Exit

Example

- $Q = 5 * (6 + 2) - 12 / 4$ [Infix]
- $P = 5, 6, 2, +, *, 12, 4, /, -,$ [Postfix]

- P :

5,	6,	2,	+,	*,	12,	4,	/,	-,)
(0)	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)

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

(1) (2) (3) (4) (5) (6) (7) (8) (9) (10)

Symbol Scanned		STACK		
(1)	5	5		
(2)	6	5,	6	
(3)	2	5,	6,	2
(4)	+	5,	8	
(5)	*	40		
(6)	12	40,	12	
(7)	4	40,	12,	4
(8)	/	40,	3	
(9)	-	37		
(10))			

Infix to Postfix

- Q is an arithmetic expression written in infix notation
- $\hat{}$, * , / , + , -
- Three level of precedence

Infix to Postfix

- Q is an arithmetic expression written in infix notation. This algorithm finds the equivalent postfix notation, P

[1] Push "(" onto STACK and ")" to the end of Q

[2] Scan Q from Left to Right and Repeat Steps 3 to 6 for each element of Q until the STACK is empty

[3] If an operand is encountered, add it to P

[4] If a left parenthesis is encountered, push it onto STACK

[5] If an operator \otimes is encountered, then:

(a) Push the operator into the stack, if the incoming operator is having higher precedence than the operator present in top of stack.

(Or) Repeatedly pop from STACK and add to P each operator (on the top of STACK) which has precedence as or higher precedence than \otimes .

(b) Add \times to STACK
○

[6] If a right parenthesis is encountered, then

(a) Repeatedly pop from the STACK and add to P each operator (on top of STACK) until a left parenthesis is encountered.

(b) Remove the left parenthesis. [Do not add it to P]

[7] Exit

Example (Infix)

- Q : $A + (B * C - (D / E \hat{=} F) * G) * H$

A	+	(B	*	C	-	(D	/	E	^	F)	*	G)	*	H)
1	2	3	4	5	6	7	8	9											20

A	+	(B	*	C	-	(D	/	E	^	F)	*	G)	*	H)
1	2	3	4	5	6	7	8	9											20

	Symbol Scanned	STACK	Expression P
		(
1	A	(A
2	+	(+	A
3	((+ (A
4	B	(+ (A B
5	*	(+ (*	A B
6	C	(+ (*	A B C
7	-	(+ (-	A B C *
8	((+ (- (A B C *

A	+	(B	*	C	-	(D	/	E	^	F)	*	G)	*	H)
1	2	3	4	5	6	7	8	9											20

	Symbol Scanned	STACK	Expression P
8	((+ (- (A B C *
9	D	(+ (- (A B C * D
10	/	(+ (- (/	A B C * D
11	E	(+ (- (/	A B C * D E
12	^	(+ (- (/ ^	A B C * D E
13	F	(+ (- (/ ^	A B C * D E F
14)	(+ (-	A B C * D E F ^ /
15	*	(+ (- *	A B C * D E F ^ /

A	+	(B	*	C	-	(D	/	E	^	F)	*	G)	*	H)
1	2	3	4	5	6	7	8	9											20

Symbol
Scanned

STACK

Expression P

15	*	(+ (- *	A B C * D E F ^ /
16	G	(+ (- *	A B C * D E F ^ / G
17)	(+	A B C * D E F ^ / G * -
18	*	(+ *	A B C * D E F ^ / G * -
19	H	(+ *	A B C * D E F ^ / G * - H
20)		A B C * D E F ^ / G * - H * +