DATA STRUCTURE AND ALGORITHM

Lecturer: Er. Saroj ghimire

Qualification: Msc.CSIT, BE(COMPUTER)

Lincoln University College

OVERVIEW

- Basics concept of stack
- Stack ADT
- Stack Operations
- Stack Applications
- Conversion From Infix To Postfix/Prefix
- Evaluations Of Postfix/Prefix

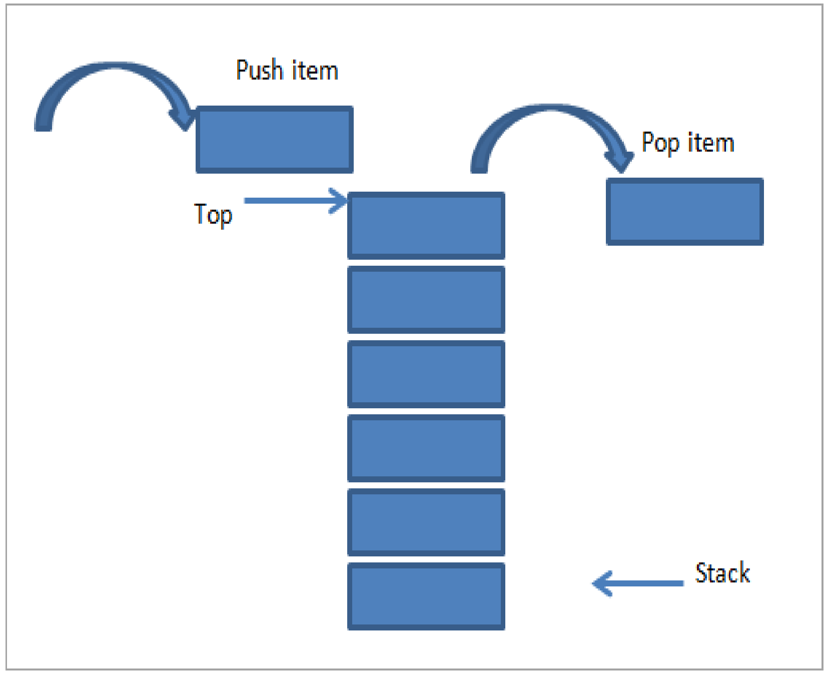
OBJECTIVES

- After studying this unit, you will be able to:
  - Describe the stack model
  - Explain the implementation and applications of stacks

CONCEPT OF STACK

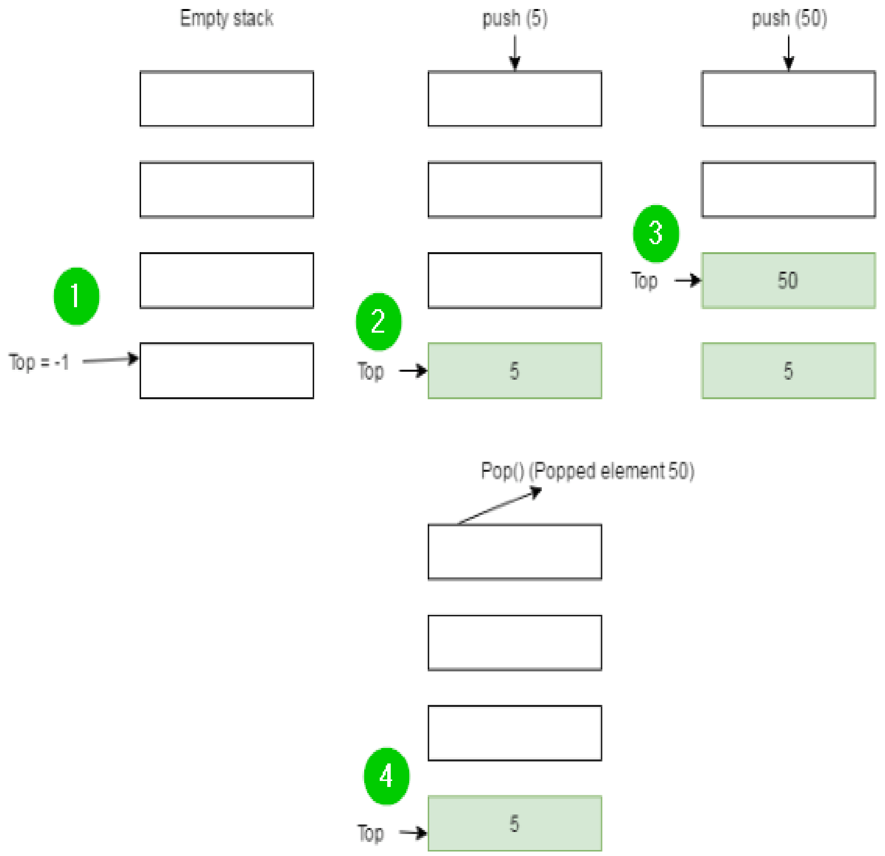
- A stack is similar to real-life stack or a pile of things that we stack one above the other.
- Stack is an **ordered list** of **similar data type**.
- Stack is a **LIFO**(Last in First out) structure or we can say **FILO**(First in Last out).
- **Push()** function is used to insert new elements into the Stack and **Pop()** function is used to remove an element from the stack. Both insertion and removal are allowed at only one end of Stack called **Top**.
- Stack is said to be in **Overflow** state when it is completely full and is said to be in **Underflow** state if it is completely empty

- Pictorial representation of stack



STACK IMPLEMENTATION

1. We initially have an empty stack. The top of an empty stack is set to -1.
2. Next, we have pushed the element 5 into the stack. The top of the stack will points to the element 5.
3. Next, we have pushed the element 50 into the stack. The top of the stack shifts and points to the element 50.
4. We have then performed a pop operation, removing the top element from the stack. The element 50 is popped from the stack. The top of the stack now points to the element 5.



STACK ADT

- **Push(item) -** Adds or pushes an element into the stack.
- **Pop() -** Removes or pops an element out of the stack.
- **peek ()-** Gets the top element of the stack but doesn't remove it.
- **Is Full ()-** Tests if the stack is full return overflow. Top= Size-1
- **Is Empty() -** Tests if the stack is empty return underflow. Top = -1

//Declaration of Stack//

```
class Stack
{
   private:
      int top;
      int ele[MAX];
   public:
      Stack();
      int     isFull();
      int     isEmpty();
      void     push(int item);
      int     pop(int *item);
}
```

STACK OPERATIONS

- Push Operation
  - The process of putting a new data element onto stack is known as a Push Operation. Push operation involves a series of steps –
    1. Checks if the stack is full. (**top == SIZE-1**)
    2. If the stack is full, produces an error and exit.
    3. If the stack is not full, increments **top** to point next empty space. (**top++**)
    4. Adds data element to the stack location, where top is pointing. (**stack[top] = value**
    5. Returns success

STACK OPERATIONS(CONTD)

- Pop operation:

  The process of removing a new data element from stack is known as a Pop Operation. Push operation involves a series of steps –

1. Checks if the stack is empty. (**top == -1**)
2. If the stack is empty, produces an error and exit.
3. If the stack is not empty, accesses the data element at which **top** is pointing. **stack[top]**
4. Decreases the value of top by 1. (**top--**)
5. Returns success

STACK APPLICATION

- Stacks can be used for expression evaluation.
- Stacks can be used to check parenthesis matching in an expression.
- Stacks can be used for Conversion from one form of expression to another.
- Stacks can be used for Memory Management.
- Stack data structures are used in backtracking problems.
- To evaluate the expressions (postfix, prefix)
- To keep the page-visited history in a Web browser
- To perform the undo sequence in a text editor
- Used in recursion
- To pass the parameters between the functions in a C program
- Can be used as an auxiliary data structure for implementing algorithms
- Can be used as a component of other data structures

CONVERSION FROM INFIX TO POSTFIX/ PREFIX

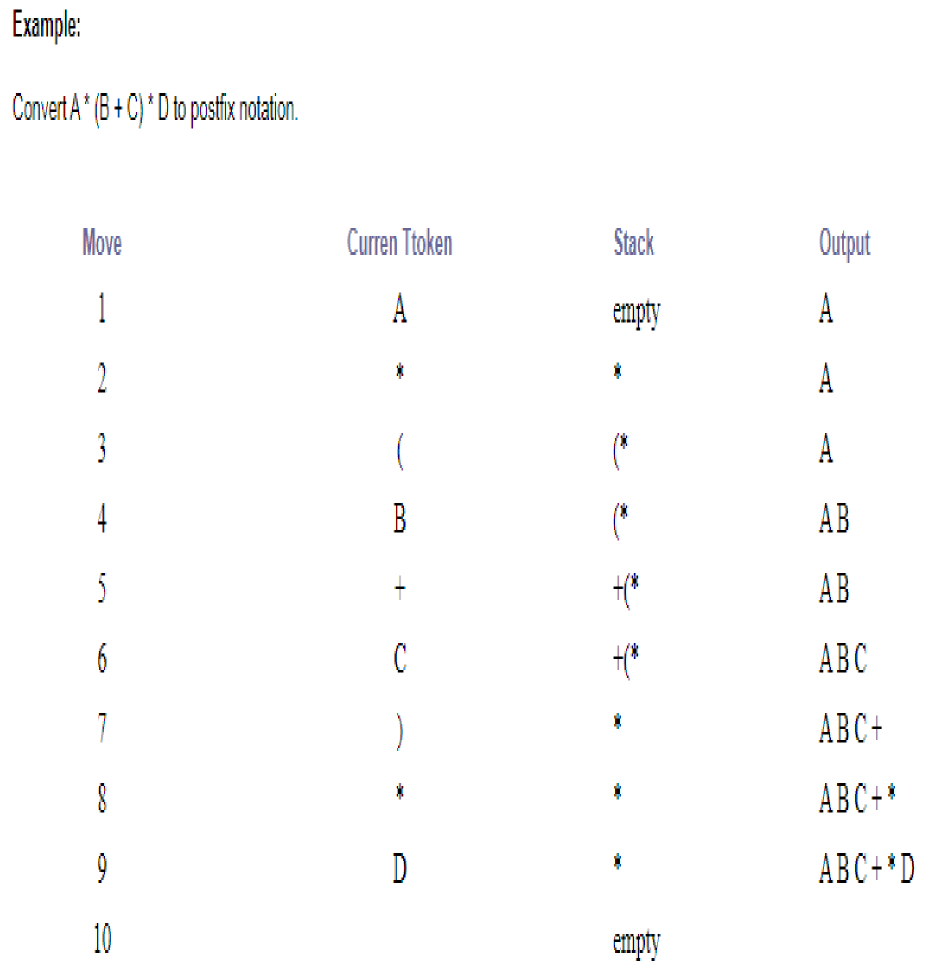
- Infix, Postfix, Prefix

  - Infix expression is the normal expression that consists of operands and operators. For example, **A+B**
  - Postfix expression consists of operands followed by operators. For example, **AB+**
  - Prefix expression consists of operators followed by operands. For example, **+AB**

INFIX TO POSTFIX EXAMPLE

In this case, We use the stacks to convert infix to postfix. We have operator's stack, output's stack and one input string. Operator's stack works as FILO(First In Last Out). Output's stack works as FIFO (First In First Out).

- The following algorithm converts infix to postfix.
  - Scan input string from left to right character by character.
  - If the character is an operand, put it into output stack.
  - If the character is an operator and operator's stack is empty, push operator into operators' stack.
  - If the operator's stack is not empty, there may be following possibilities.
    - If the precedence of scanned operator is greater than the top most operator of operator's stack, push this operator into operand's stack.
    - If the precedence of scanned operator is less than or equal to the top most operator of operator's stack, pop the operators from operand's stack until we find a low precedence operator than the scanned character. Never pop out (**'('**) or (**')'**) whatever may be the precedence level of scanned character.
    - If the character is opening round bracket (**'('**), push it into operator's stack.
    - If the character is closing round bracket (**')'**), pop out operators from operator's stack until we find an opening bracket (**'('**).
    - Now pop out all the remaining operators from the operator's stack and push into output stack



Example:

Convert A * (B + C) * D to postfix notation.

Move	Curren Ttoken	Stack	Output
1	A	empty	A
2	*	*	A
3	(	(*	A
4	B	(*	AB
5	+	+(*	AB
6	C	+(*	ABC
7	)	*	ABC+
8	*	*	ABC+*
9	D	*	ABC+*D
10		empty	

INFIX TO PREFIX

In this case, we have operator's stack, output's stack and one input string. Operator's stack works as FILO(First In Last Out). Output's stack works as FIFO (First In First Out).

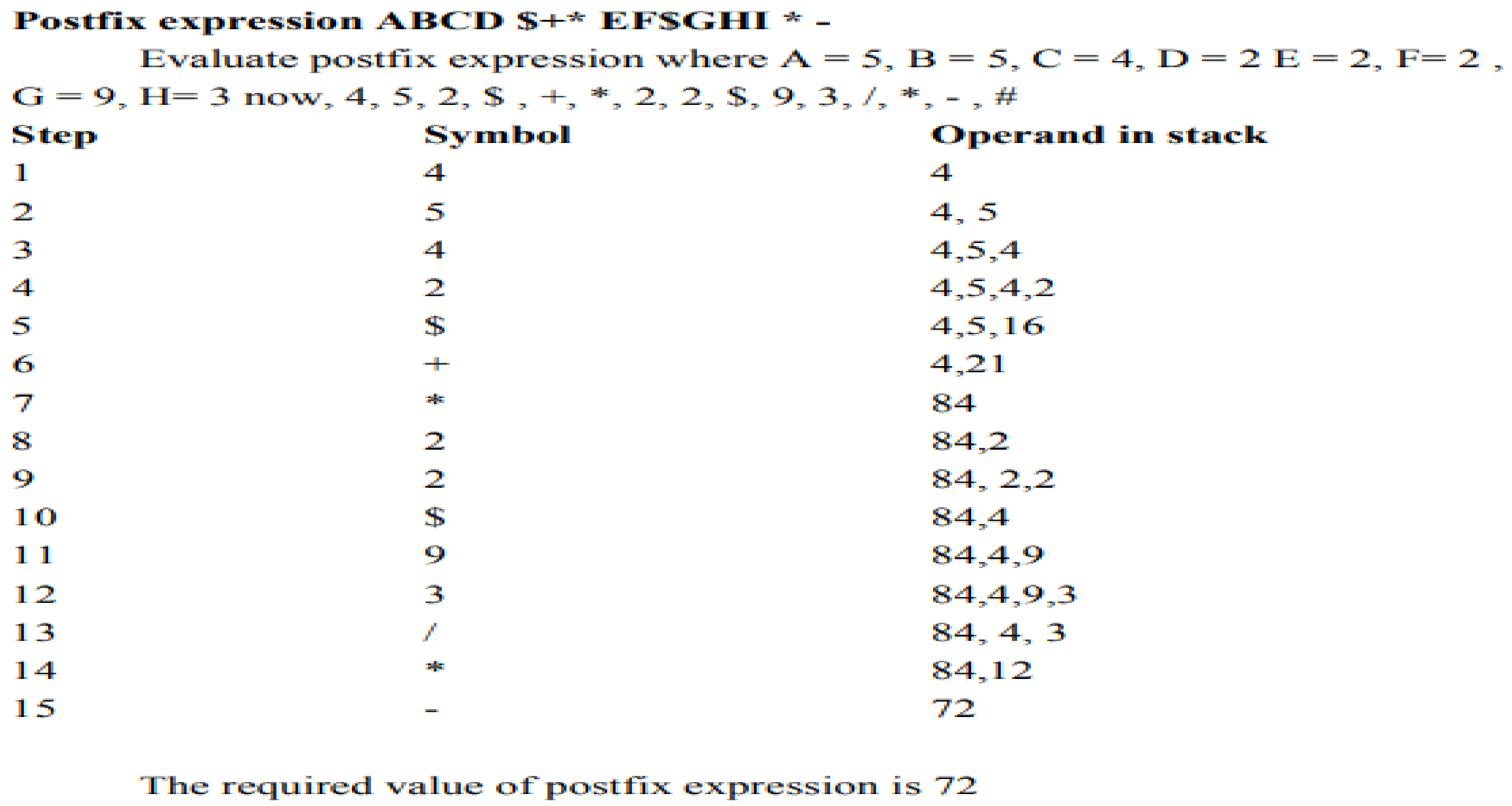
- The following algorithm must be followed for infix to prefix conversion.
  - Reverse the input string.
  - Convert the reversed string into infix expression.
  - Now reverse the resulting infix expression obtained from the previous step. The resulting expression is prefix expression

EVALUATIONS OF POSTFIX/ PREFIX

- In this case stack contents the operands. In stead of operators. Whenever any operator occurs in scanning we evaluate with last two elements of stack.
  1. Add the unique symbol # at the end of array post fix.
  2. Scan the symbol of array post fix one by one from left to right.
  3. If symbol is operand, two push in to stack.
  4. If symbol is operator, then pop last two element of stack and evaluate it as [top- 1] operator [top] & push it to stack.
  5. Do the same process until ‘#’ comes in scanning.
  6. Pop the element of stack which will be value of evaluation of post fix arithmetic expression

(Note : a=4 and l= / in next page)

EVALUATIONS OF POSTFIX/ PREFIX(CONTD)



Postfix expression ABCD $+* EF$GHI * –

Evaluate postfix expression where A = 5, B = 5, C = 4, D = 2 E = 2, F= 2 , G = 9, H= 3 now, 4, 5, 2, $, +, *, 2, 2, $, 9, 3, /, *, - , #

Step	**Symbol**	**Operand in stack**
1	4	4
2	5	4, 5
3	4	4,5,4
4	2	4,5,4,2
5	$	4,5,16
6	+	4,21
7	*	84
8	2	84,2
9	2	84, 2,2
10	$	84,4
11	9	84,4,9
12	3	84,4,9,3
13	/	84, 4, 3
14	*	84,12
15	-	72

The required value of postfix expression is 72

FURTHER READINGS

- Brian W. Kernighan and Dennis M. Ritchie, The C Programming Language, Prentice Hall, 1988.
- Data Structures and Algorithms; Shi-Kuo Chang; World Scientifi c.
- Data Structures and Effi cient Algorithms, Burkhard Monien, Thomas Ottmann, Springer.
- Kruse Data Structure & Program Design, Prentice Hall of India, New Delhi
- Mark Allen Weles: Data Structure & Algorithm Analysis in C Second Adition. Addison-Wesley publishing
- RG Dromey, How to Solve it by Computer, Cambridge University Press.
- Shi-kuo Chang, Data Structures and Algorithms, World Scientifi c
- Sorenson and Tremblay: An Introduction to Data Structure with Algorithms.
- Thomas H. Cormen, Charles E, Leiserson & Ronald L. Rivest: Introduction to Algorithms.
- Prentice-Hall of India Pvt. Limited, New Delhi Timothy A. Budd, Classic Data Structures in C++, Addison Wesley