COMPUTER SCIENCE DEPARTMENT FACULTY OF ENGINEERING AND TECHNOLOGY

COMP2321

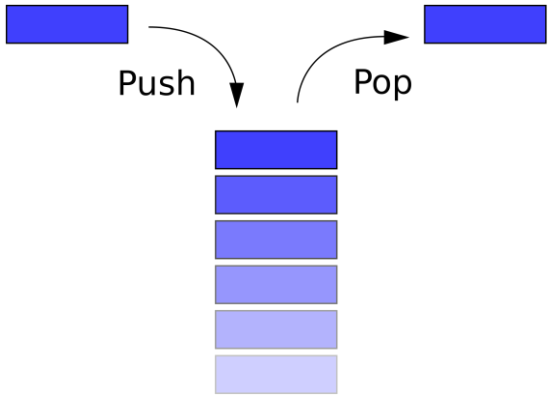
Data Structures And Algorithms in C

Chapter 3
Stack

Instructor: Murad Njoum

stack

- Collection of,
  - Homogeneous info elements where,
  - Insertion and deletion take place, At one end only.
- Also called,
  - LIFO structure.
    - Last In First Out.
  - FILO structure.
    - First In Last Out.

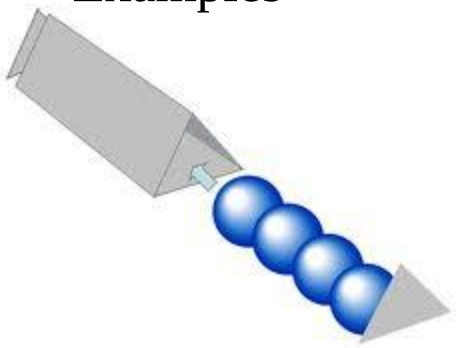


- Stack can be implemented,
  - Either as an Array,
    - Stores the elements in continuous memory locations and,
    - Has all advantages/disadvantages of array.
  - Or as a linked List.
    - Stores the elements in discontinuous memory locations and,
    - Has all advantages/disadvantages of linked list.

Stack

Stack of Books

Examples



Box of Tennis Balls
(Closed at 1 end)

Stack of Plates

Stack of Rings / Discs

Stack of Clothes.

Trains in a railway yard.

Goods in a cargo.

Type:

LIFO / FILO

Stack of Chairs

Instructor: Murad Njoum

Stack

- **Stack:**
  - Insertion operation in a Stack is called:
    - PUSH
      - When PUSH is not possible, situation is called:
        - » Stack Overflow
  - Deletion operation in a Stack is called:
    - POP
      - When POP is not possible, situation is called:
        - » Stack Underflow

Stack using Array

PUSH an element

Item → 25 15 35 65 55 75

Array A

Top	4	55
Top	3	65
Top	2	35
Top	1	15
Top	0	25

Top = -1

Steps:

Upper bound of array

If(Top > 5)

print “Stack overflow.”

Else

Top = Top + 1

A[Top] = Item

EndIf

Stack using Array

POP	POP	POP	POP	POP	POP
55	65	35	15	25	NULL

Array A

Top	4	55
Top	3	65
Top	2	35
Top	1	15
Top	0	25

Top = -1

Steps:

Item = Null

If(Top < 0)

print “Stack underflow.”

Else

Item = A[Top] // 55 65 35 15 25

A[Top] = Null

Top = Top - 1

EndIf

Return(Item)

Stack using Array

PUSH an element

Steps:

```
If(Top > U)
        print “Stack overflow.”
Else
        Top = Top + 1
        A[Top] = Item
EndIf
```

POP an element

Steps:

```
Item = Null
If(Top < L)
        print “Stack underflow.”
Else
        Item = A[Top]
        A[Top] = Null
        Top = Top - 1
EndIf
Return(Item)
```

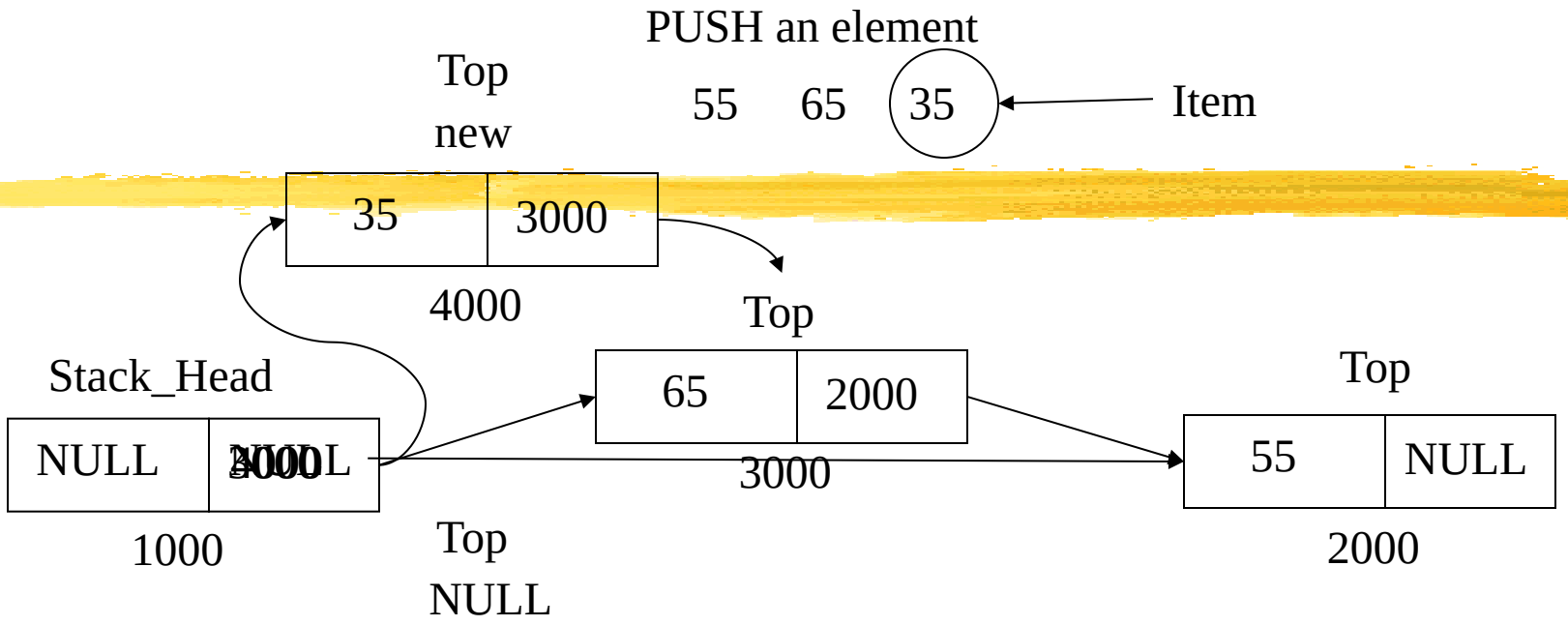
```c
#include <stdio.h>
#define MAXSIZE 5

struct stack
{
    int stk[MAXSIZE];
    int top;
};
typedef struct stack STACK;
STACK s;

void push(void);
int  pop(void);
void display(void);
```



```c
void push ()
{
    int num;
    if (s.top == (MAXSIZE - 1))
    {
        printf ("Stack is Full\n");
        return;
    }
    else
    {
        printf ("Enter the element to be
pushed\n");
        scanf ("%d", &num);
        s.top = s.top + 1;
        s.stk[s.top] = num;
    }
    return;
}
```

```
int pop ()
{
    int num;
    if (s.top == - 1)
    {
        printf ("Stack is Empty\n");
        return (s.top);
    }
    else
    {
        num = s.stk[s.top];
        printf ("poped element is = %d\n",
s.stk[s.top]);
        s.top = s.top - 1;
    }
    return(num);
}
```

Stack using linked List



Steps:

new = GetNode(NODE)

If(new == NULL)

print "Memory Insufficient. PUSH not possible."

Else

Steps: (cntd)

new->info = 35

new->next = Top

Stack_Head->next = new

Top = new

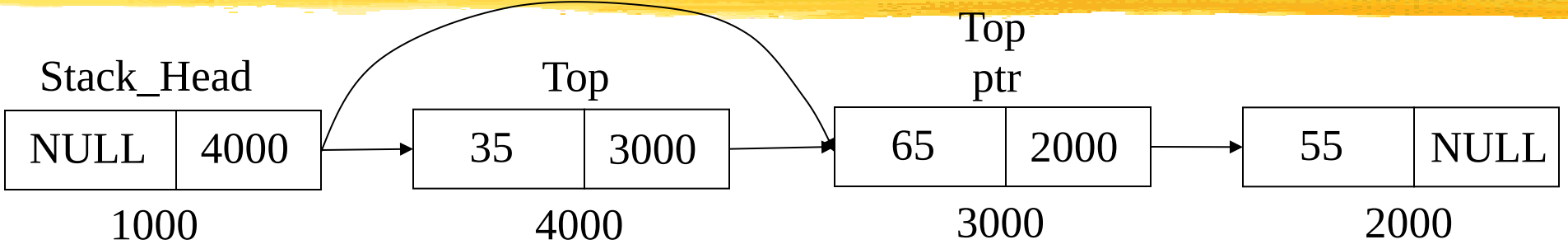
EndIf

Stop

Stack using linked List

POP an element

35



Steps:

Item = Top->info

ptr = Top->next

Stack_Head->next = ptr

ReturnNode(Top)

Top = ptr

Return(Item)

Stop

Stack using linked List

POP POP

55 NULL

Steps:

Item = NULL

If(Top == NULL)

print “Stack Underflow.”

Else

Item = Top->info

ptr = Top->next

Stack_Head->next = ptr

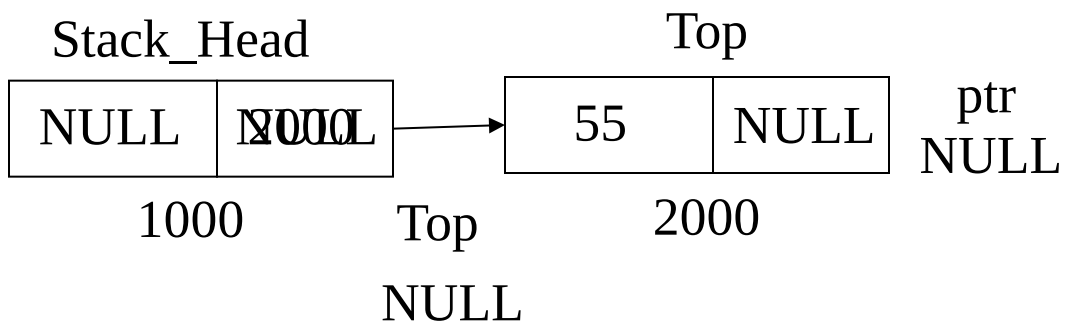
ReturnNode(Top)

Top = ptr

EndIf

Return(Item)

Stop



```
#include <stdio.h>
#include <stdlib.h>

struct node
{
    int info;
    struct node *ptr;
}*top,*temp_cell,*temp;

int topelement();
void push(int data);
void pop();
void empty();
void display();
void destroy();
void stack_count();
void create();
......
......
...
```

```
struct stack
{
   int info;
   struct node *next;
}
typedef struct stack Stack
```

```
Stack  push( Stack top, int data)
{if (top == NULL)
    {
       top =(struct node *)malloc(sizeof(struct node));
       top->next = NULL;
       top->info = data;
    }
    else
    {
       temp =(struct node *)malloc(sizeof(struct node));
       temp->next = top;
       temp->info = data;
       top = temp;
    }
 return top;
}
```

```
Stack  pop(Stack top)
 {
   Stack   temp_cell = top;

   if (temp_cell == NULL)
   {
      printf("\n Error : Trying to pop from empty stack");
      return;
   }
   else
      temp_cell = temp_cell->next;
      printf("\n Popped value : %d", top->info);

  free(top);
 top = temp_cell;

return top; }
```

Exercise

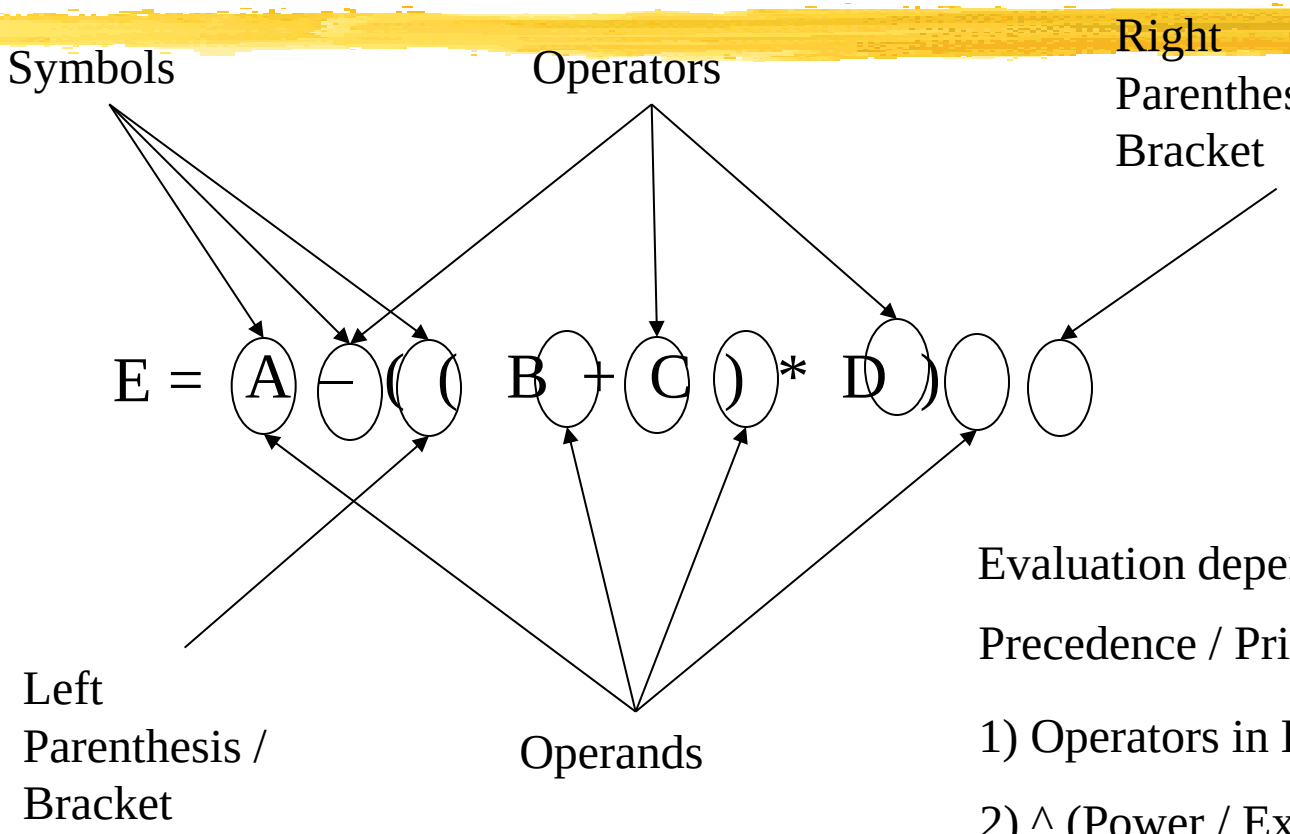
- Given a Stack (using linked list), write a function to reverse it (without extra space).
  Examples:
  Input: S = [5->24-> 9->6->8->4->1->8->3->6]
  Output: S = [6->3->8->1->4->8->6->9->24-> 5]

Applications of Stack

- **Uses of Stack:**
  - Main uses of Stack are:
    - Evaluation of Arithmetic Expression
      - Example:
        - » (A + B) * C or (1 + 2) * 3
    - Execute recursive function calls / Recursion.

Applications of Stack

Evaluation of Arithmetic Expression using a Stack



Evaluation depends on,

Precedence / Priority:

1) Operators in Brackets ()

2) ^ (Power / Exponentiation)

3) *, /

4) +, -

Applications of Stack

- **Evaluation of Arithmetic Expression using a Stack:**
  - Process consist of 2 steps:
    - 1) Convert the given arithmetic expression into a special notation/representation using a Stack.
      - Algorithm would be required for this step.
    - 2) Evaluate (Find the value of) that special notation (obtained from step-1) again using a Stack.
      - Algorithm would be required for this step.

Stack should be empty
at the end of the conversion.

Postfix expression
never contains parentheses.

Infix to Postfix

Infix Expression: A * B + C

Infix Expression enclosed in parentheses:

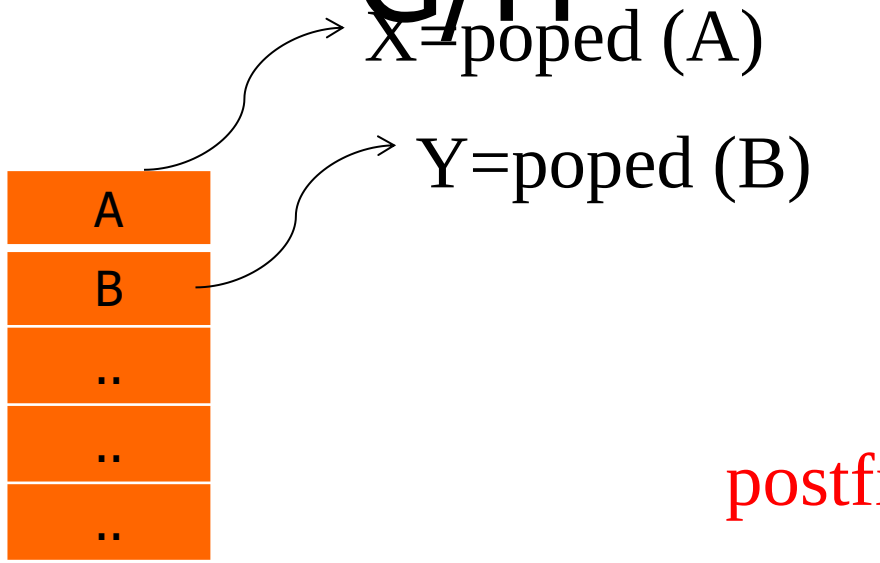
(A * B + C)

Step	Symbol	Stack T	Output Expression
1	(	T (	
2	A	T (	A
3	*	T (*	A
4	B	T (*	A B
5	+	T (+	A B *
6	C	T (+	A B *C
7	)		A B *C +

Quesyion: covert The following Expression

infix to postfix

⌘A+((B-C*D)/E+F-G/H



Precdenacy // Assositivity
( ) .....L-R
^ .....R-L
/ * % .....L-R
+ - .....L-R

postfix

<operand><operand><**operator**>

<X><Y><operator>

A+((B-C*D)/E)+F-G/H

Step	Symbol	Stack	O.Expression
1			
2	A		A
3	+	+	A
4	(	+(	A
5	(	+ ((	A
6	B	+((	A B
7	-	+((-	A B
8	C	+ ((-	A B C
9	*	+((- *	A B C
10	D	+ ((- *	A B C D
11	)	+ (	A BCD*-
12	/	+ (/	A BCD*-
13	E	+(/	A BCD*- E
14	)	+	A BCD*-E/
15	+	+	ABCD*-E/+
16	F	+	ABCD*-E/+ F
17	-	-	ABCD*-E/+F+
18	G	-	ABCD*-E/+F+ G
19	/	-/	ABCD*-E/+F+ G
20	H	-/	ABCD*-E/+F+ G H

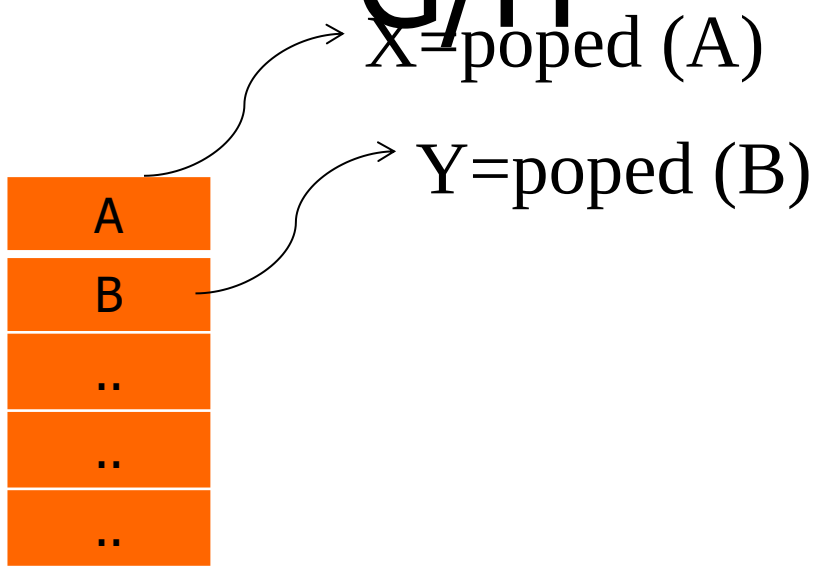
ABCD*-E/+F+ GH/

ABCD*-E/+F+GH/-

Quesyion: covert The following Expression

infix to prefix

⌘A+((B-C*D)/E)+F-G/H



prefix

<operator><operand><operand>

<operator> <X><Y>

A+((B-C*D)/E+F-G/H

Step	Symbol	Stack	O.Expression
1			
2	A		A
3	+	+	A
4	(	+(	A
5	(	+ ((	A
6	B	+((	A B
7	-	+((-	A B
8	C	+ ((-	A B C
9	*	+((- *	A B C
10	D	+ ((- *	A B C D
11	)	+ (	A -B*CD
12	/	+ (/	A -B*CD
13	E	+(/	A -B*CD E
14	)	+	A /-B*CDE
15	+	+	+A/-B*CDE
16	F	+	+A/-B*CDE F
17	-	-	++A/-B*CDEF
18	G	-	++A/-B*CDEF G
19	/	-/	++A/-B*CDEF G
20	H	-/	++A/-B*CDEF G H

++A/-B*CDEF /GH

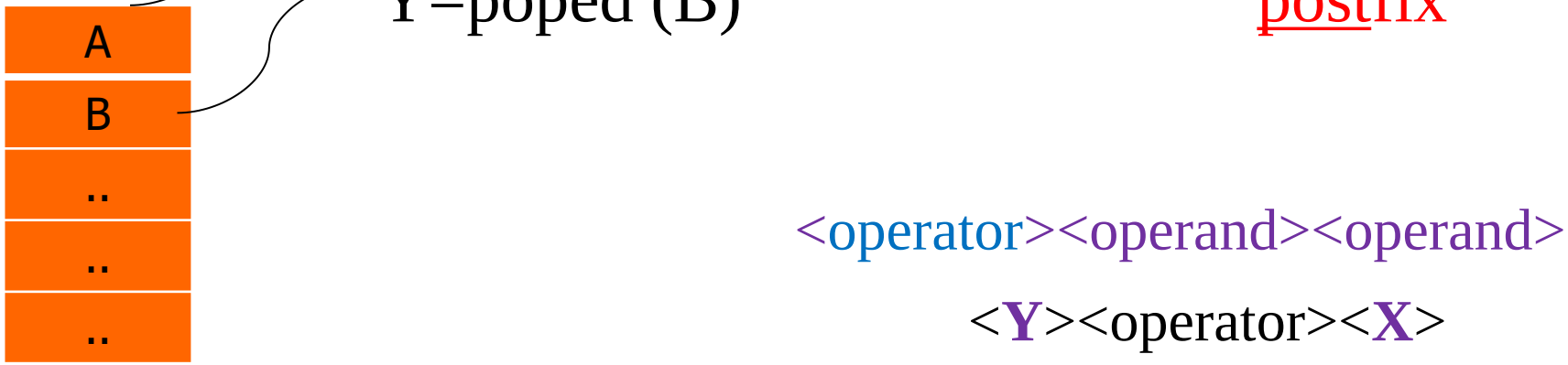
-++A/-B*CDEF/GH

Quesyion: covert The following Expression

postfix to infix

ABCD*-

E/+F+GH/-



postfix

<operator><operand><operand>

<**Y**><operator><**X**>

ABCD*-E/+F+GH/-

Step	Symbol	Reading Operator	O.Expression
1			
2	A		A
3	B		A B
4	C		A B C
5	D		A B C D
6	*	*	A B (C*D)
7	-	-	A (B-(C*D))
8	E		A (B-(C*D)) E
9	/	/	A (B-(C*D))/E
10	+	+	A +(B-(C*D))/E
11	F		A (B-(C*D))/E F
12	+	+	A+(B-((C*D)/E))+F

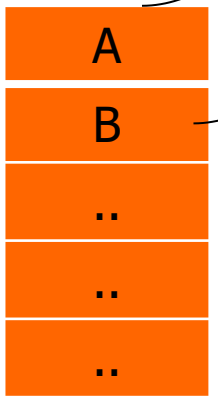
Step	Symbol	Readig	O.Expression
13	G		A+(B-((C*D)/E))+F G
14	H		A+(B-((C*D)/E))+F G H
15	/	/	A+(B-((C*D)/E))+F G/H
16	-	-	A+ (B-((C*D)/E))+F-G/H

A+(B-((C*D)/E))+F-G/H

Quesyion: covert The following Expression

pretfix to infix

-++A/-B*CDEF/GH



X=poped (A)

Y=poped (B)

prefix

<operator><operand><operand>

<**X**><operator><**Y**>

Reverse Expression : -++A/-B*CDEF/GH

-++A/-B*CDEF/GH

Step	Symbol	Reading Operator	O.Expression
1			
2	H		H
3	G		HG
4	/	/	G/H
5	F		G/H F
6	E		(G/H) F E
7	D		(G/H) F E D
8	C		(G/H) F E D C
9	*	*	(G/H) F E (C*D)
10	B		(G/H) F E (C*D) B
11	-	-	(G/H) F E (B-(C*D))
12	/	/	(G/H) F ((B-(C*D))/E)

Step	Symbol	Readig	O.Expression
13	A		(G/H) F ((B-(C*D))/E) A
14	+		(G/H) F (A+((B-(C*D))/E))
15	+	+	(G/H) (A+((B-(C*D))/E))+F)
16	-	-	(A+((B-(C*D))/E))+F)-(G/H))

(A+(B-((C*D)/E))+F-(G/H))

More Excercises

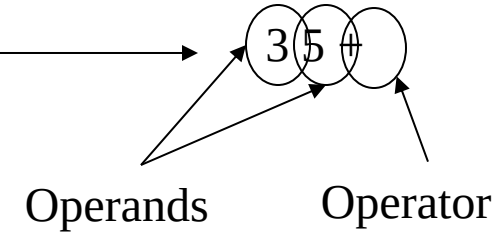
A*(B^C^D)
-E

Evaluation of Arithmetic Expression using a Stack

Step-2: Evaluate (Find the value of) special notation (Postfix) using Stack.

Infix Expression: A + B —Step-1→ Postfix Expression: A B +

Values: A = 3, B = 5 Postfix Expression: A B +

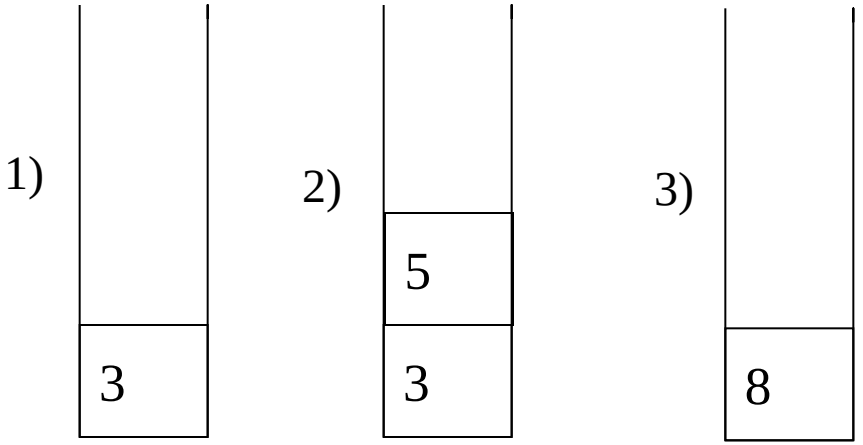


3 5 +

3 Push(3)

5 Push(5)

+



y = Pop() // y = 5

x = Pop() // x = 3

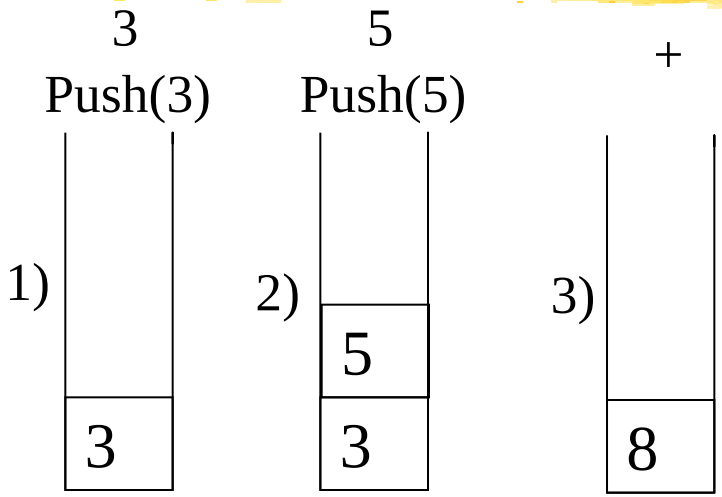
result = x operator y // result = 3 + 5 = 8

Push(result)

Evaluation of Arithmetic Expression using a Stack

Step-2: Evaluate (Find the value of) special notation (Postfix) using Stack.

3 5 +



y = Pop() // y = 5
x = Pop() // x = 3
result = x operator y // result = 3 + 5 = 8
Push(result)

Step	Symbol	Stack	Operation
1	3	3	Push(3)
2	5	3, 5	Push(5)
3	+	8	y = Pop() // 5, x = Pop() // 3, result = x operator y // 3+5, Push(result)

Evaluate Postfix

Infix: ((A+B) * (C-D)) $\xrightarrow{\text{Step-1}}$ Postfix: A B + C D -* [A=1, B=2, C=5, D=3]

1 2 + 5 3 - *

Step	Symbol	Stack	Operation
1	1	1	Push(1)
2	2	1, 2	Push(2)
3	+	3	y = Pop() // 2, x = Pop() // 1, result = x operator y // 1+2, Push(result)
4	5	3, 5	Push(5)
5	3	3, 5, 3	Push(3)
6	-	3, 2	y = Pop() // 3, x = Pop() // 5, result = x operator y // 5-3, Push(result)
7	*	6	y = Pop() // 2, x = Pop() // 3, result = x operator y // 3*2, Push(result)

Algorithm: InfixToPostfix

- **Algorithm-InfixToPostfix:**
  - Enclose the entire expression in parentheses (brackets) if not already enclosed.
    - Add '(' at the beginning and ')' at the end of the expression.
  - Scan 1 symbol at a time from left to right and do the following things depending on the type of symbol scanned:
    - If symbol is '(':
      - Push it onto the stack.
    - If symbol is '**Operand**':
      - Add/Append it to the Output Expression/Output String/Postfix Expression.
    - If symbol is '**Operator**':
      - Pop all the operators one by one from TOP of stack **which have**,
        - **The same or higher precedence** than the symbol and go on,
        - **Adding them to the Output String**.
      - **Then push the incoming operator onto the stack**.
    - **If symbol is ')':**
      - Pop all the operators one by one from TOP of stack and go on adding them to the Output String until,
        - '(' is encountered / obtained.
      - Remove that '(' from the Stack and do not do anything with ')'.

Infix to Postfix

Infix Expression

((A + B) * (C – D))

Already enclosed in parentheses.

Step	Symbol	Stack	Output Expression
1	(	(	
2	(	((	
3	A	((	A
4	+	((+	A
5	B	((+	A B
6	)	(	A B +
7	*	(*	A B +
8	(	(* (	A B +
9	C	(* (	A B + C
10	-	(* (-	A B + C
11	D	(* (-	A B + C D
12	)	(*	A B + C D -
13	)		A B + C D - *

Infix to Postfix

Infix Expression

((A + B) * (C – D))

Already enclosed in parentheses.

Step	Symbol	Stack	Output Expression
1	(	(	
2	(	((	
3	A	((	A
4	+	((+	A
5	B	((+	A B
6	)	(	A B +
7	*	(*	A B +
8	(	(* (	A B +
9	C	(* (	A B + C
10	-	(* (-	A B + C
11	D	(* (-	A B + C D
12	)	(*	A B + C D -
13	)		A B + C D - *

Infix to Postfix

Infix Expression A + B * C ^ D

Infix Expression enclosed in parentheses

(A + B * C ^ D)

Step	Symbol	Stack	Output Expression
1	(	(	
2	A	(	A
3	+	(+	A
4	B	(+	A B
5	*	(+ *	A B
6	C	(+ *	A B C
7	^	(+ * ^	A B C
8	D	(+ * ^	A B C D
9	)		A B C D ^ * +

Infix to Postfix

Infix Expression A + B / C - D

Infix Expression enclosed in parentheses

(A + B / C - D)

Step	**Symbol**	**Stack**	Output **Expression**
1	(	(	
2	A	(	A
3	+	(+	A
4	B	(+	A B
5	/	(+ /	A B
6	C	(+ /	A B C
7	-	(-	A B C / +
8	D	(-	A B C / + D
9	)		A B C / + D -

- Examples of arithmetic expressions
- Find the prefix and postfix notation for the following expression (1-6)
- 1. (A + B - C) * (E / F) – (G – H / J)
- Prefix notation - * - + A B C / E F – G / H J
- Postfix notation A B + C - E F / * G H J / - -

- 2. 4/(3+6)+5 9
- Prefix notation + / 4 + 3 6 ^ 5 9
- Postfix notation 4 3 6 + / 5 9 ^ +

- 3. (A + B) * C + D / (E + F * G) – H
- prefix: - + * + A B C / D + E * F G H
- postfix: A B + C * D E F G * + / + H -

- 4. A + ((B - C * D) / E) + F - G / H
- prefix: - + + A / - B * C D E F / G H
- postfix: A B C D * - E / + F + G H / -

5. (A * B + C) / D - E / (F + G)
prefix: - / + * A B C D / E + F G
postfix: A B * C + D / E F G + / -

6. A - B - C * (D + E / F - G) – H
prefix: - - - A B * C - + D / E F G H
postfix: A B - C D E F / + G - * - H -

7. What postfix expression is equivalent to the following infix expression?
(A + B) - C * D / (E - F / G)
Answer) A B + C D * E F G / - / -

8. What infix expression is equivalent to the following postfix expression?
A B * C / D E + F G H / * - +
Answer) A * B / C + (D + E - (F * (G / H)))

9. What is the value of the following postfix expression?
54 6 + 7 4 - * 9 / 35 15 + +
Answer) 70

Exercise very Important:

A) Convert the following expressions into its equivalent postfix or infix expressions (in other words, provide the alternatives to the one given):

prefix expression: / - + 9 9 + * 2 3 4 2

infix: 9 +9 – 2 * 3 + 4 / 2

postfix: 9 9 + 2 3 * 4 + - 2 /

B) Convert the following expressions into its equivalent postfix or infix expressions (in other words, provide the alternatives to the one given):

prefix expression: + - * ^ 4 2 3 3 / / 8 4 +1 1

infix: 4 2 ^ 3 * 3 – 8 4 / 1 1 + / +

postfix: 4 ^ 2 * 3 – 3 + 8 / 4 / 1 + 1

B) Write function to convert **prefix** to **postfix**. (**Hint, come back to Stack lecture**)

Give the prefix, find infix and postfix expressions?

prefix: - + * 2 3 / 4 * 2 2 + 3 5

infix: (((2 * 3) + (4 / (2 * 2))) - (3 + 5))
postfix: 2 3 * 4 2 2 * / + 3 5 + 2

Infix to Postfix

Convert the following arithmetic expressions from Infix to Postfix.

(A+B)^C–(D*E)/F	AB+C^DE*F/-
(A+((B^C)–D))*(E–(A/C))	ABC^D-+EAC/-*
A+(B*C)/D	ABC*D/+
(A+B)*C	AB+C*
A*B^C+D	ABC^*D+

THANK YOU

Instructor: Murad Njoum