Data Structures
Module 2.2
Stack and Queues

Ms. Sarika Dharangaonkar,
Assistant Professor,
Information Technology Department, KJSCE

Outline

- Introduction to Stacks
- Stack ADT
- Operations on stack
- Representation of stack in memory
- Algorithms for stack operations
- Array implementation of stack

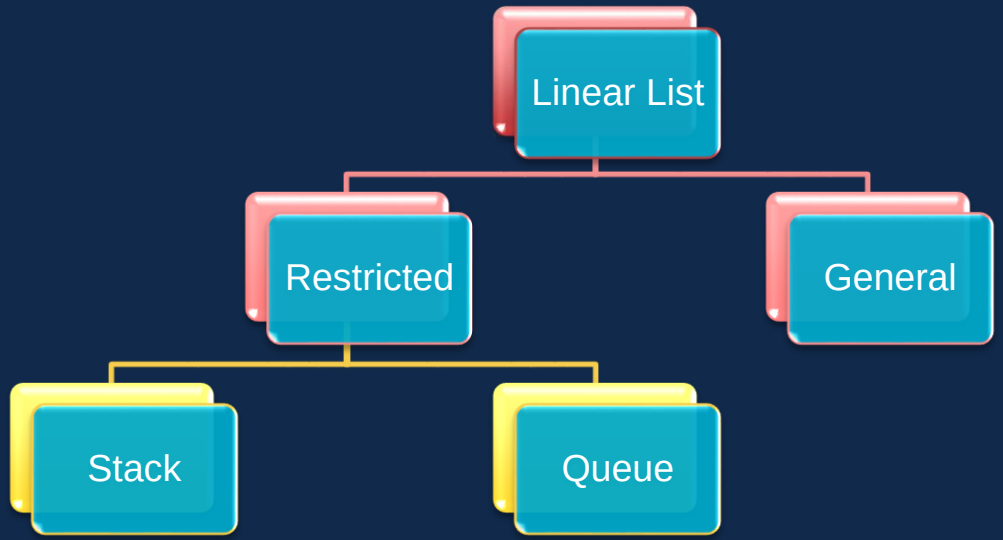
Learning Objectives

At the end of the lecture, students will be able to

- Give examples of real world stack.
- Explain stack ADT.
- Demonstrate different operations performed on a stack.
- Represent stack in memory.
- Write algorithms for stack operations.
- Develop program to implement stack ADT using array.

Linear Lists

A linear list is a list in which each element has a unique successor.



Stack

- A stack is an Abstract Data Type (ADT), commonly used in most programming languages.
- It is named stack as it behaves like a real-world stack, for example-

Deck of Cards

Stack of Books

Pile of Plates

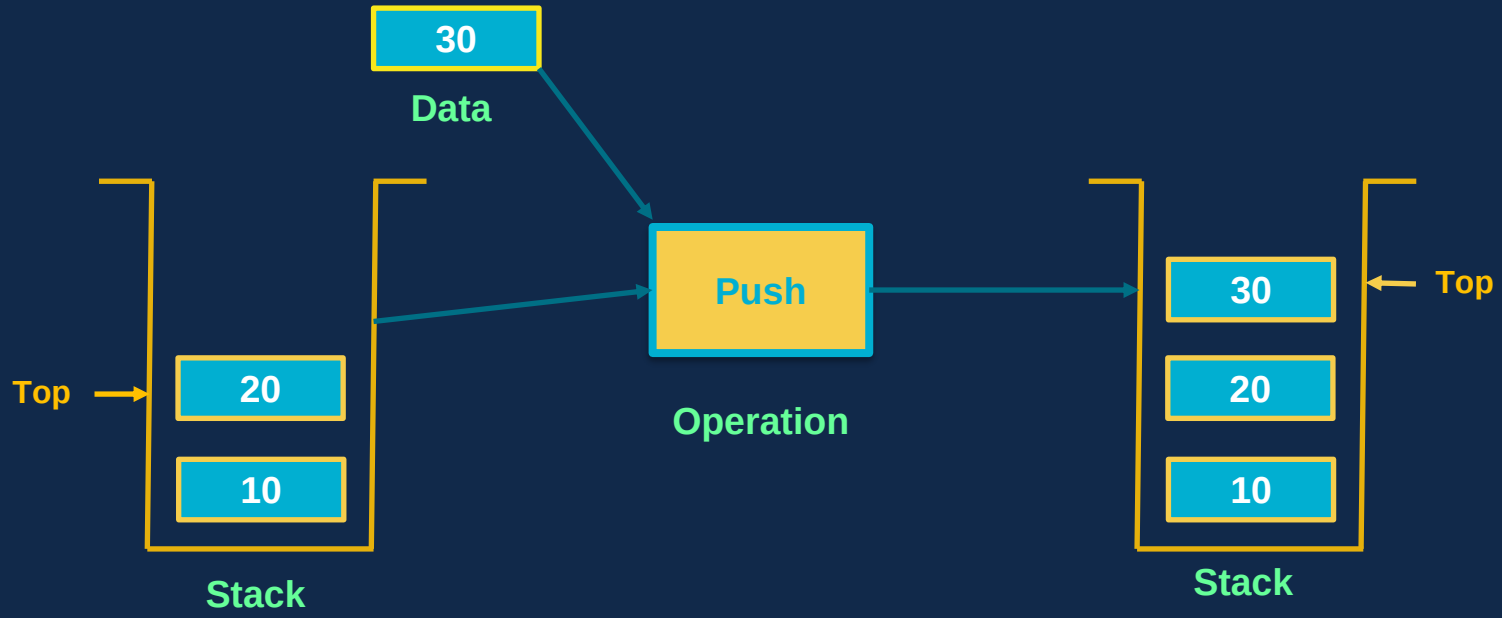
Stack

- A stack is a collection of elements, which can be stored and retrieved one at a time.
- In stack, element is inserted and deleted only from one end called the top of the stack.
- At any given time, we can only access the top element of a stack. Hence, Stack is a Last-in First-out data structure(LIFO).
- The stack is known as restrictive data structure because the operations are restricted to the one end of the structure.

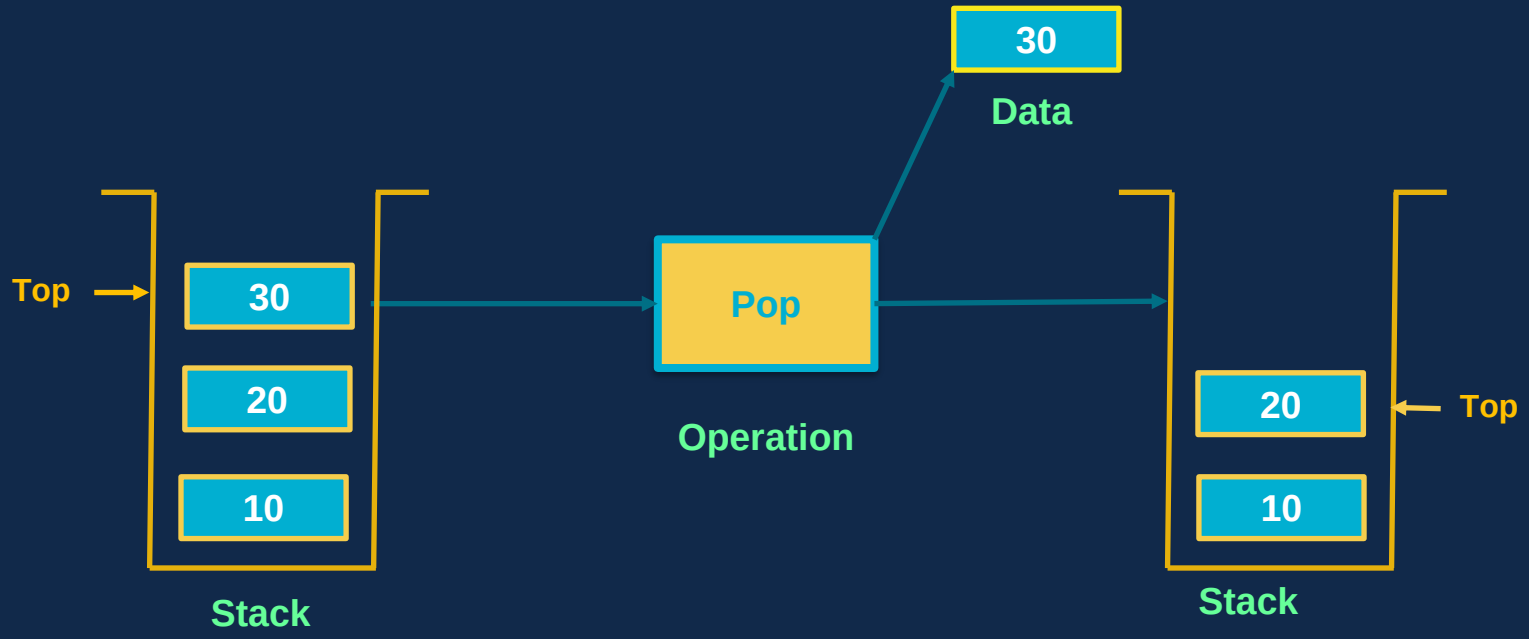
Stack ADT

Operation	Description
push()	This is used to push x into the stack
pop()	This is used to delete one element from top of the stack
peek()	This is used to get the top most element of the stack
isFull()	This is used to check whether stack is full or not
isEmpty()	This is used to check whether stack is empty or not
size()	This function is used to get number of elements present into the stack

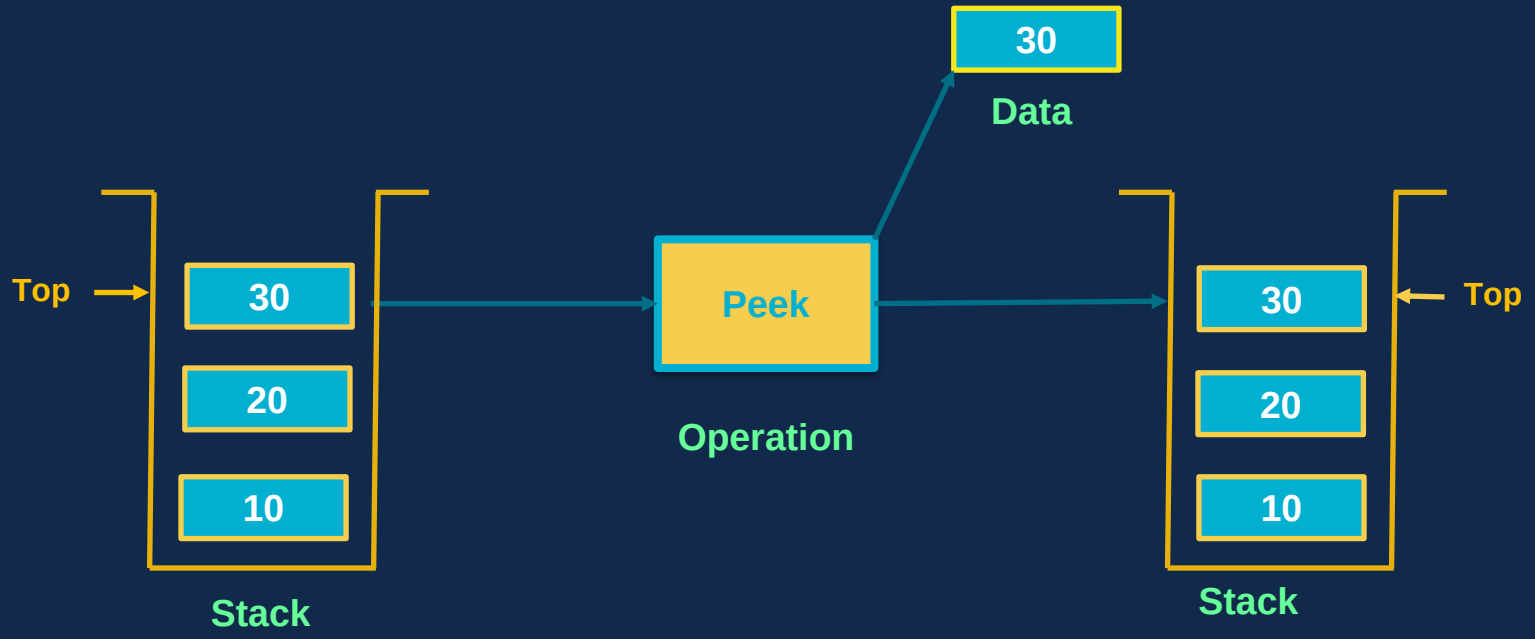
Push Operation



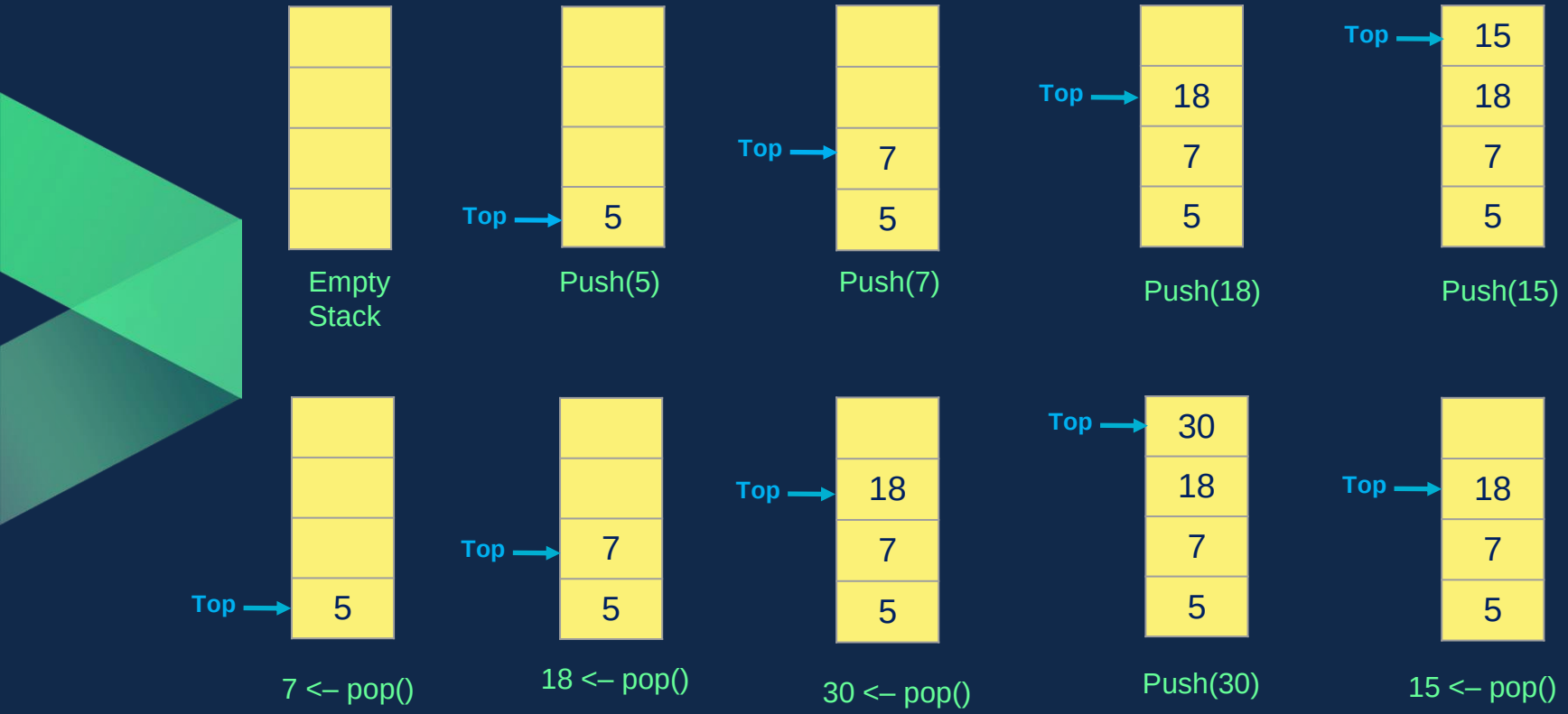
Pop Operation



Peek or Stack top Operation



Stack Example



Implementation of stacks

Stacks data structures are usually implemented using arrays or linked lists.

1. Static Implementation using Arrays :
For applications in which the maximum stack size is known ahead of time, an array is suitable.

2. Dynamic implementation using linked List:
If the maximum stack size is not known beforehand, we could use a linked list

Array Representation of Stack

- Stacks can be represented as an array in the computer memory.
- To store the address of the topmost element of the stack, every stack has a variable called TOP associated with it.
- Another variable called MAX is used which represent the maximum number of elements that the stack can hold.
- Example:

```
#define MAX 20

int stack[MAX], top=-1;
```

Operations on a Stack

Algorithm: Push operation

Step 1: IF TOP = MAX-1

PRINT OVERFLOW

Step 2: SET TOP = TOP+1

Step 3: SET STACK[TOP] = VALUE

Step 4: END

Algorithm: Pop operation

Step 1: IF TOP = -1

PRINT UNDERFLOW

Step 2: SET VAL = STACK[TOP]

Step 3: SET TOP = TOP-1

Step 4: RETURN VAL

Step 5: END

Operations on a Stack

Algorithm: Peek operation

Step 1: IF TOP = -1
PRINT STACK IS EMPTY
Goto Step 3

Step 2: RETURN STACK[TOP]

Step 3: END

Program to implement Stack using Array

References

- Data Structures using C, Reema Thareja, Oxford
- C & Data Structures, Prof. P.S. Deshpande, Prof. O.G. Kakde, DreamTech press.

Data Structures
Module – 2.2
Stack and Queues

Ms. Sarika Dharangaonkar
Assistant Professor,
Information Technology Department, KJSCE

Outline

- Linked implementation of Stack
- Applications of Stack

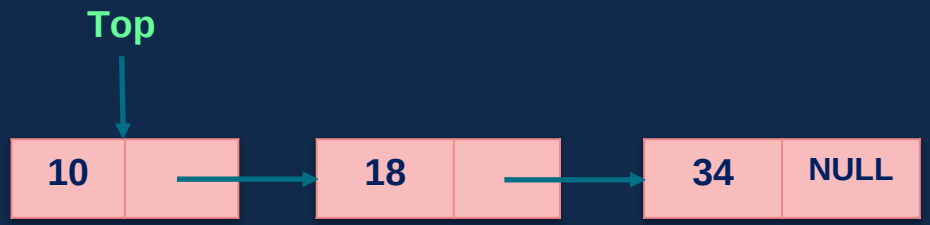
Learning Objectives

At the end of the lecture, students will be able to

- Implement stack using linked list.
- List various applications of stack.
- Identify different ways of writing Arithmetic Expressions.
- Convert given infix expression into prefix and postfix form using stack.
- Write algorithm to convert infix expression into postfix expression.

Linked Representation of Stack

- In a linked stack, every node has two parts—one that stores data and another that stores the address of the next node.
- The START pointer of the linked list is used as TOP. All insertions and deletions are done at the node pointed by TOP.
- If TOP = NULL, then it indicates that the stack is empty.

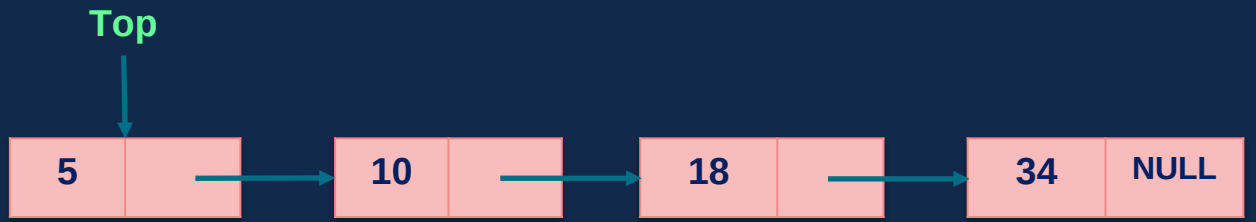


Linked Stack

Operations on a Linked Stack

Algorithm: Push operation

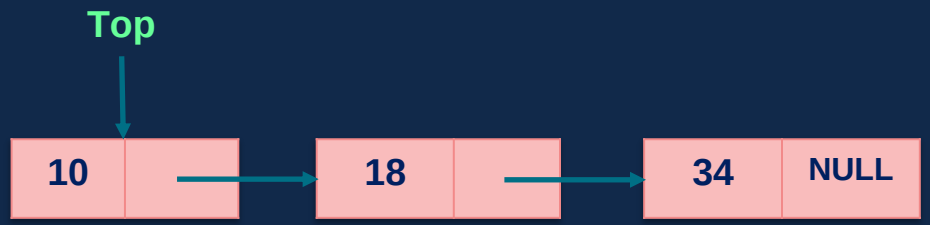
Step 1: Allocate memory for the new node and name it as NEWNODE
Step 2: Set NEWNODE->DATA = VAL
Step 3: If TOP = NULL
    Set NEWNODE -> NEXT = NULL
    Set TOP = NEWNODE
  Else
    Set NEWNODE -> NEXT = TOP
    Set TOP = NEWNODE
Step 4: END



Operations on a Linked Stack

Algorithm: Pop operation

Step 1: If TOP = NULL
Print “Stack Underflow” GOTO Step 5
Step 2: Set PTR = TOP
Step 3: Set TOP = TOP -> NEXT
Step 4: Free PTR
Step 5: END



Applications of Stack

- Expression conversion : Infix to Postfix Conversion
- Evaluation of arithmetic expression : Postfix Evaluation
- Parentheses checker/ Well form-ness of Parenthesis/ Balanced parenthesis
- Keeping track of function calls
- Recursion
- Reversal of data

Evaluation of arithmetic expression

- There are three different but equivalent notations of writing algebraic expressions:
  - Infix
  - Postfix
  - Prefix

- In Infix expression, operator is placed in between the operands.
- In Postfix expression also known as Reverse Polish Notation or RPN, operator is placed after the operands.
- In Prefix expression also known as Polish Notation, operator is placed before the operands.

Infix	Postfix	Prefix
a + b	a b +	+ a b
a + b * c	a b c * +	+ a * b c

Convert Infix Expression to Postfix and Prefix Expression

Infix	Postfix	Prefix
(a – b) * (c + d)		
(a + b) / (c + d) – (d * e)		

Convert Infix Expression into Postfix Expression

Infix : (a – b) * (c + d)

Algorithm to Convert Infix Expression into Postfix Expression

Step 1: Scan the infix expression from left to right.

Step 2: Repeat until each character in the infix notation is scanned

IF '(' is encountered, push it on the stack.

IF an operand (whether a digit or a character) is encountered, add it postfix expression.

IF ')' is encountered, then

a. Repeatedly pop from stack and add it to the postfix expression until a '(' is encountered.

b. Discard the '(' . That is, remove the '(' from stack and do not add it to the postfix expression.

IF an operator is encountered, then

a. Repeatedly pop from stack and add each operator (popped from the stack) to the postfix expression which has the same precedence or a higher precedence than the current scanned operator.

b. Push the operator to the stack [END OF IF]

Step 3: Repeatedly pop from the stack and add it to the postfix expression until the stack is empty

Step 4: END

Convert Infix Expression into Postfix Expression

Infix : (a + b) / (c + d) – (d * e)

Convert Infix Expression into Postfix Expression

Infix : (a – b) * (c + d)

Program to Convert Infix Expression into Postfix Expression

References

- Data Structures using C, Reema Thareja, Oxford
- C & Data Structures, Prof. P.S. Deshpande, Prof. O.G. Kakde, DreamTech press.

Data Structures
Module – 2.2,2.3
Stack and Queues

Ms. Sarika Dharangaonkar,
Assistant Professor,
Information Technology Department, KJSCE

Outline

- Postfix Evaluation

Learning Objectives

At the end of the lecture, students will be able to

- Evaluate postfix expression using stack.
- Write algorithm postfix evaluation.

Algorithm for Postfix Expression Evluation

Step 1: Scan the postfix expression from left to right.

Step 2: Repeat until each symbol in the postfix notation is scanned
IF symbol is an operand, push it on the stack.
Else IF an operator is encountered, then
opnd2 = pop
opnd1 = pop
value = result of applying symbol to opnd1 and opnd2
push value on the stack
[END OF ELSE]

Step 3: Pop the result from the stack

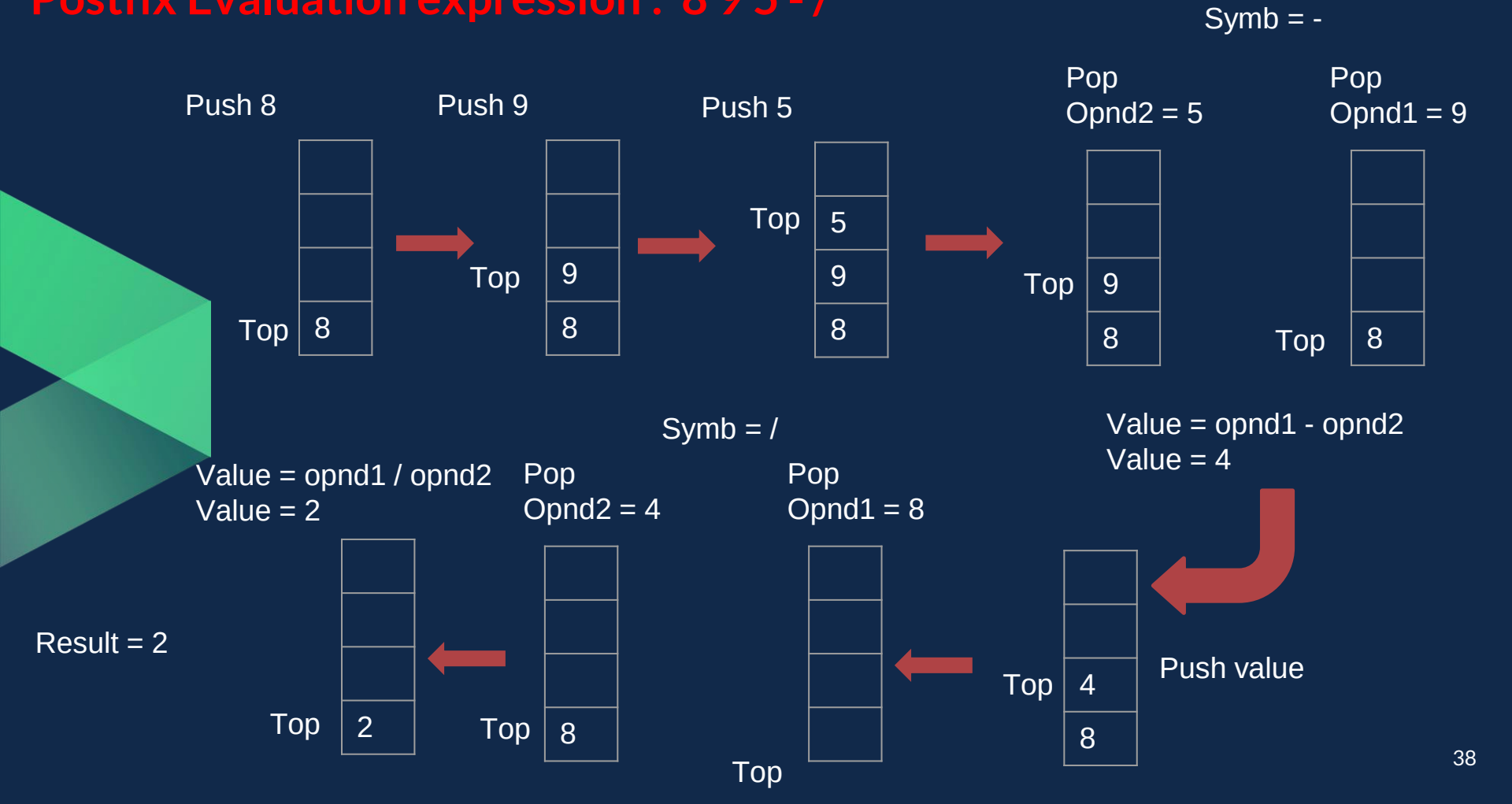
Step 4: END

Postfix Evaluation Example

6 2 3 + - 8 2 / +

Symb	opnd1	opnd2	value	opndstk
6				6
2				6, 2
3				6,2, 3
+	2	3	5	6, 5
-	6	5	1	1
8				1, 8
2				1, 8, 2
/	8	2	4	1, 4
+	1	4	5	5

Postfix Evaluation expression : 8 9 5 - /



Program to evaluate Postfix Expression

References

- Data Structures using C, Reema Thareja, Oxford
- C & Data Structures, Prof. P.S. Deshpande, Prof. O.G. Kakde, DreamTech press.

Data Structures
Module – 2.2
Stack and Queues

Ms. Sarika Dharangaonkar,
Assistant Professor,
Information Technology Department, KJSCE

Outline

- Stack Application

Learning Objectives

At the end of the lecture, students will be able to

- Write algorithm to check well-formness of parenthesis/Parenthesis checker / Balanced parenthesis.

Parentheses Matching

- Each “(”, “{”, or “[” must be paired with a matching “)”, “}”, or “]”
  - correct: ( )(( )){([( )])}
  - correct: ((( )(( )){([( )])}
  - incorrect:)(( )){([( )])}
  - incorrect: ({[ ])}
  - incorrect: (

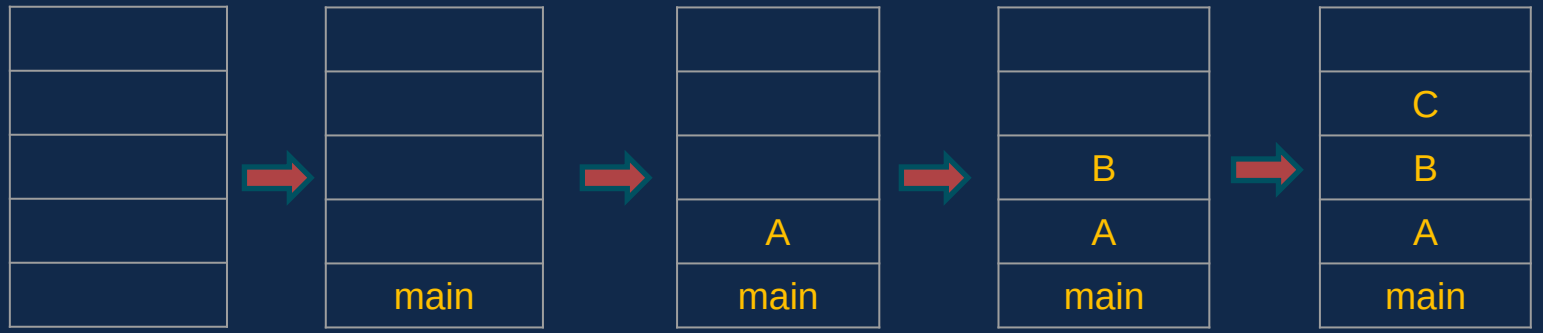
Parenthesis Checker

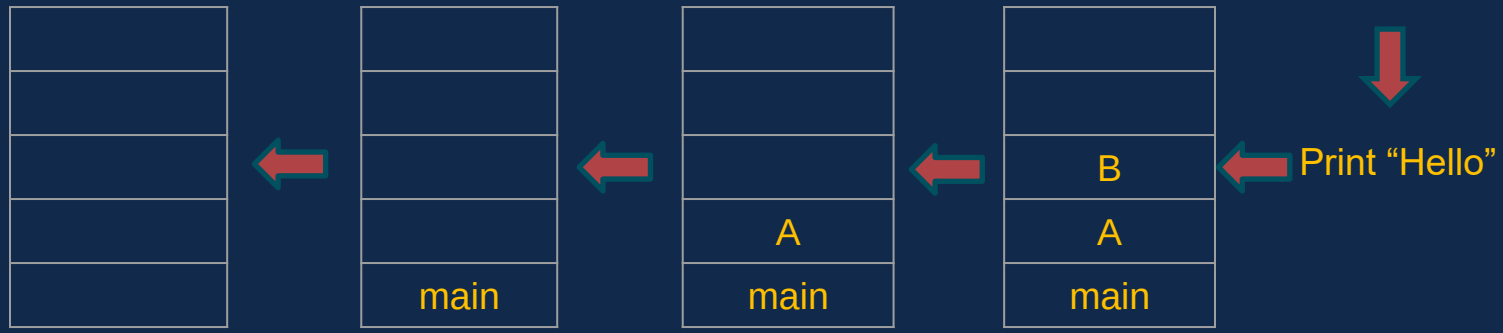
Step 1: Scan the expression from left to right.
Step 2: Set flag = 1
Step 3: Repeat until each symbol in the expression is scanned
IF symbol is '(' or '{' or '[', push it on the stack.
IF symbol is ‘)' or ‘}' or ‘]', then
If stack is empty, then set flag = 0
Else
pop top of the stack and place it in temp.
If symbol is ')' and temp is either '{' or '[‘, then set flag=0 and GOTO step 5
If symbol is '}' and temp is either '(' or '[‘, then set flag=0 and GOTO step 5
If symbol is ']' and temp is either '(' or '{‘, then set flag=0 and GOTO step 5
Step 4: If stack is not empty, then set flag=0 and GOTO step 5
Step 5: If flag =1, then Print “ Valid expression”
Else Print “Invalid expression"
Step 6: END

System Stack in the case of Function calls

```
main()
{
        A();
        printf("Bye");
}

A()
{
        B();
        printf("Inside A");
}

B()
{
        C();
        printf("Inside B");
}

C()
{
        D();
        printf("Inside C");

}

D()
{
        printf("Hello");
}
```

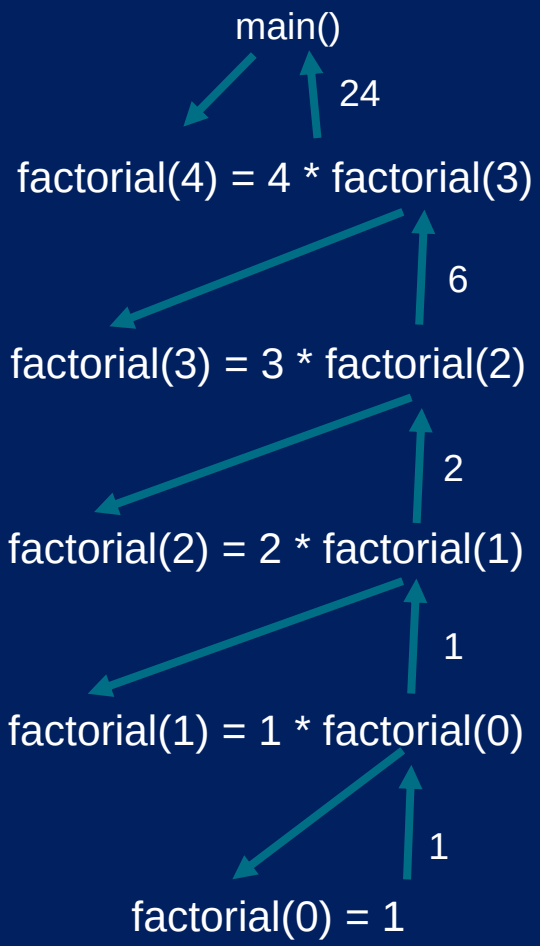
System Stack in the case of Function calls





Recursion

```
void main()
{
      int fact, n;
      printf("Enter number");
      scanf("%d", &n);
      fact=factorial(n);
      printf("Factorial of %d = %d", fact);
}

int factorial(int n)
{
      int f;
      if ( n == 0 )
            return 1;
      else
            f = n*factorial(n-1);
      return f;
}
```



Convert the Following Infix Expression to equivalent postfix expression:

- (a) A – B + C
- (b) A * B + C / D
- (c) (A – B) + C * D / E – C
- (d) (A * B) + (C / D) – (D + E)
- (e) ((A – B) + D / ((E + F) * G))
- (f) (A – 2 * (B + C) / D * E) + F
- (g) 14 / 7 * 3 – 4 + 9 / 2

References

- Data Structures using C, Reema Thareja, Oxford
- C & Data Structures, Prof. P.S. Deshpande, Prof. O.G. Kakde, DreamTech press.