CHAPTER 6

Stacks, Queues and Recursion

Introduction

- Stacks: A stack is a linear list of elements in which an element may be inserted or deleted only at one end called the top of the stack.

- This means, the elements are removed from a stack in the reverse order of that in which they were inserted into the stack.

  Example: a stack of dishes, a stack of pennies etc.

- Stacks are also called Last in First out(LIFO) list.

- Push: is the term used to insert an element into a stack.
- Pop; is the term used to delete an element from a stack.

Stacks Continue…

- Suppose we have five data to be inserted to the stack:

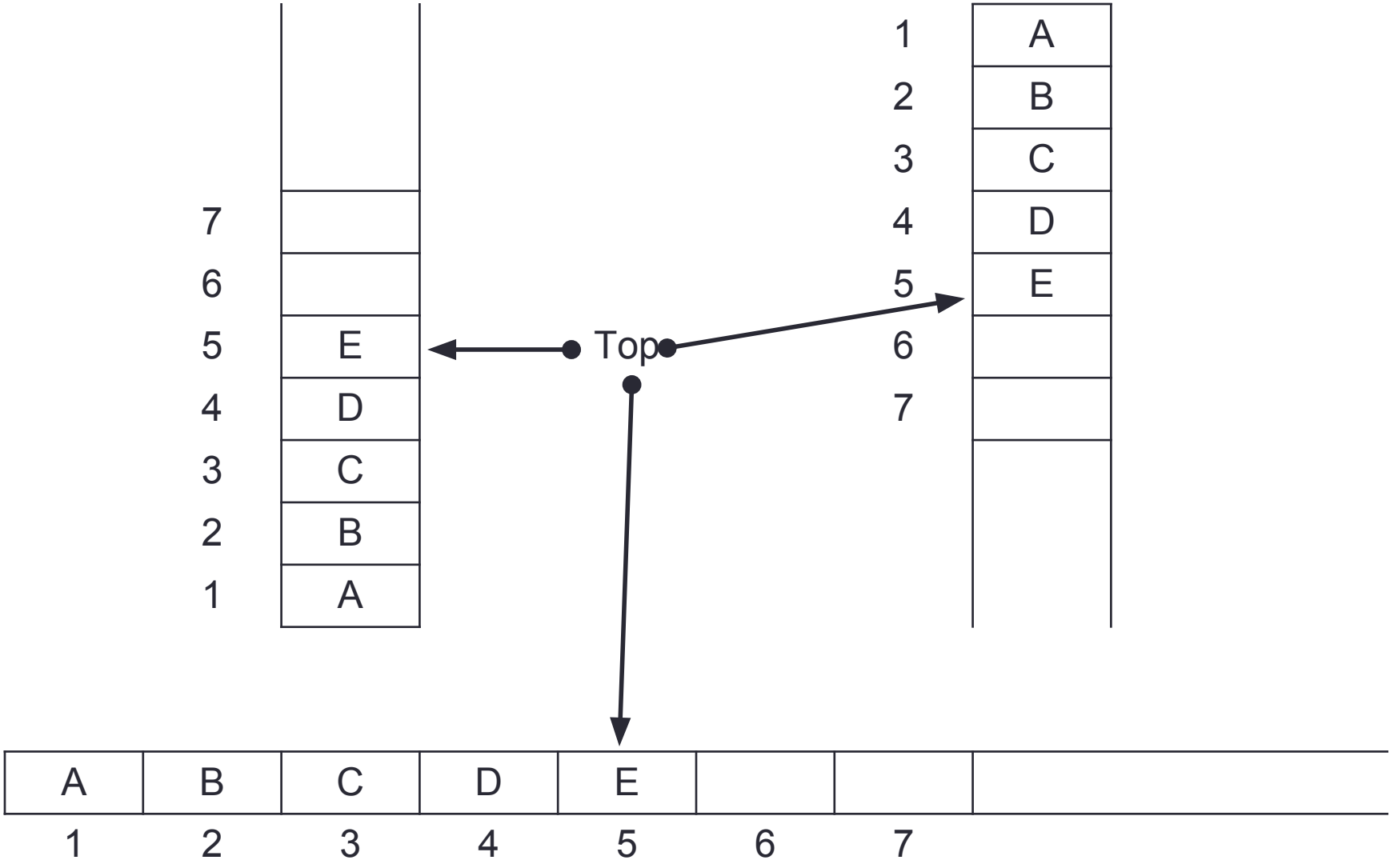
  A, B, C, D, E
- The diagram of the stack is shown on the next slide. There we show the three ways to picture a stack.

- However, we normally use the following notation to represent a stack.

  Stack: A, B, C, D, E (The right most is the top).

- ‘C’, can not be deleted from the list before ‘E’ and ‘D’.

Diagrams of Stacks



Array Representation of Stacks

- Stacks may be represented in the computer in various ways usually by means of linear array.
- For this reason, we require a linear array Stack; a pointer variable Top which contains the location of the top element and a variable MAXSTK which indicates the maximum no. of elements that can be held by the stack.
- Top=0 or NULL indicates that the stack is empty and no elements can be deleted from the stack.
- Top=MAXSTK indicates that the stack is holding maximum elements and no further elements can be inserted.

PUSH and POP procedure

- The elements can be added(pushed) into the stack by the following PUSH procedure.

PUSH(STACK, TOP, MAXSTK, ITEM)

This procedure pushes an ITEM onto a STACK

1. If TOP=MAXSTK then print: Overflow and Return.
2. Set TOP:=TOP+1
3. Set STACK[TOP]:=ITEM
4. Return.

- The elements can be removed(popped) from the stack by the following POP procedure.

POP(STACK, TOP, ITEM)

This procedure deletes the top element and assigns it to the variable ITEM

1. If TOP=0 then print: Underflow and Return.
2. Set ITEM:=STACK[TOP]
3. Set TOP:=TOP-1
4. Return.

Arithmetic Expression: Polish Notation

- Polish notation, named after the Polish Mathematician Jan Lukasiewicz, refers to the notation in which the operator symbol is placed before its two operand.
- Example: A+B can be written as +AB.
- This notation also known as prefix notation.

- Another notation which is reverse of Polish notation is postfix notation.
- Example: A*B can be written as AB*.
- The common mathematical expression such as (A+B)*C is known as infix notation.

Polish Notation Continue…

- Advantage: To evaluate infix expression, we need to follow the precedence of the operators; otherwise correct result can not be obtained.

- For example: The following two expressions (A+B)*C and A+(B*C) almost same but result is different, because of the operator precedence.

- In case of prefix or postfix notation, parentheses never used, to determine the order of the operations.

Infix to postfix and prefix

- (A + B ↑ D) / (E – F) + G to postfix

=(A + [B D ↑]) / [E F -] + G

=[A B D ↑ +] / [E F -] + G

=[A B D ↑ + E F - /] + G

=A B D ↑ + E F - / G +

- (A + B ↑ D) / (E - F) + G to prefix

=(A + [↑ B D]) / [- E F] + G

=[+ A ↑ B D] / [- E F] + G

=[/ + A ↑ B D - E F] + G

=+ / +A ↑ B D - E F G

Infix to postfix and prefix

- A * (B + D) / E – F * (G + H / K) to postfix

=A * [B D +] / E – F * (G + [H K /])

=[A B D + *] / E – F * [G H K / +]

=[A B D + * E /] - [F G H K / + *]

=A B D + * E / F G H K / + * -

- A * (B + D) / E – F * (G + H / K) to prefix

=A * [+ B D] / E – F * (G + [/ H K])

=[* A + B D] / E – F * [+ G / H K]

=[/ * A + B D E] - [* F + G / H K]

=- / * A + B D E * F + G / H K

Postfix Expression Evaluation

- Evaluate postfix expression: 5, 6, 2, +, *, 12, 4, /, -

=5, [6+2], *, 12, 4, /, -

=[5*8],12, 4, /, -

=40, 12, 4, /, -

=40, [12/4], -

=40-3

=37

Note: Postfix expression can be easily evaluated using computer. How ever infix expression can also be evaluated with some extra cost (Infix to postfix then postfix evaluation).

Algorithm: Postfix Evaluation

This algorithm finds the value of an arithmetic expression P written in postfix notation.

1. Add a right parenthesis “)” at the end of P.
2. Scan P from left to right and repeat steps 3 and 4 until “)” is encountered.
3. If an operand is encountered put it onto stack.
4. If an operator θ is encounter, then
   a) Remove two top elements of Stack, where A is the top element and B is the next-to-top element.
   b) Evaluate B θ A.
   c) Place the result of (b) back on Stack.
5. Set value equal to the top element on the Stack.
6. Exit.

Postfix Evaluation Example:

Once again we evaluate postfix expression: 5, 6, 2, +, *, 12, 4, /, - by showing Stack's contents as each element is scanned.

Symbol Scanned	Stack
5	5
6	5, 6
2	5, 6, 2
+	5, 8
*	40
12	40, 12
4	40, 12, 4
/	40, 3
-	37
)	

Transforming Infix to Postfix Expression

This algorithm finds the equivalent expression P from infix Q.

1. Push “(“ onto stack and add “)” to the end of Q.
2. Scan Q from left to right and repeat steps 3 to 6 until 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 θ is encountered, then:
   a) Repeatedly pop from Stack and add to P each operator (on the top of the Stack) which has the same precedence as or higher precedence than θ.
   b) Add θ to Stack.
6. If right parenthesis is encountered, then:
   a) Repeatedly pop from Stack and add to P each operator (on the top of the stack) until a left parenthesis is encountered.
   b) Remove the left parenthesis. [Do not add it to P].
7. Exit

Transforming Infix to Postfix: Example

- Simulation of the transformation algorithm for Q: A + (B * C - (D / E ↑ F) * G) * H

Symbol Scanned	Stack	Expression
A	(	A
+	(+	A
(	(+(	A
B	(+(	A B
*	(+(*	A B
C	(+(*	A B C
-	(+(-	A B C *
(	(+(-(	A B C *
D	(+(-(	A B C * D
/	(+(-(/	A B C * D
E	(+(-(/	A B C * D E
↑	(+(-(/↑	A B C * D E
F	(+(-(/↑	A B C * D E F
)	(+(-	A B C * D E F ↑ /
*	(+(-*	A B C * D E F ↑ /
G	(+(-*	A B C * D E F ↑ / G
)	(+	A B C * D E F ↑ / G * -
*	(+*	A B C * D E F ↑ / G * -
H	(+*	A B C * D E F ↑ / G * - H
)		A B C * D E F ↑ / G * - H * +

Recursion

- Recursion is a technique that allow a procedure or a function to call itself or to call a second procedure or function that may eventually result in a call statement back to the original procedure or function.

- The procedure or function having this property is called recursive procedure or function.

- So that the program will not continue to run indefinitely, a recursive procedure must have the following two properties:

1. There must be base criteria for which the procedure does not call itself indefinitely.
2. Each time the procedure call itself, it must be closer to the base criteria.

Factorial Example

- We know n! = n.(n-1)! = n.(n-1).(n-2)….1
- If we want to calculate 4!, we first send 4, then 3, then 2, then 1 and finally 0 to a recursive function fact() as shown in the following way.

4! = 4.3! (place 4 into a stack)
3! = 3.2! (place 3 into a stack)
2! = 2.1! (place 2 into a stack)
1! = 1.0! (place 1 into a stack)
0! = 1 (0 is base value)
1! = 1.1 = 1 (remove 1 from stack)
2! = 2.1 = 2 (remove 2 from stack)
3! = 3.2 = 6 (remove 3 from stack)
4! = 4.6 = 24 (remove 4 from stack)

Factorial Calculation Algorithm

- Two way: Iterative loop process and Recursive procedure.

- Iterative Loop process: FACTORIAL (FACT, N)

1. If N=0 then Set FACT:=1 and Return.
2. Set FACT:=1
3. Repeat for k=1 to N
   Set FACT:=k*FACT
4. Return.

- Recursive Procedure: FACTORIAL (FACT, N)

1. If N=0 then: Set FACT:=1, and Return.
2. Call FACTORIAL(FACT, N-1)
3. Set FACT:= N*FACT
4. Return.

Fibonacci Sequence

- In Fibonacci series F_0=0 and F_1=1 and each succeeding term is the sum of two preceding terms.
- The series therefore: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55…..

FIBONACCI(FIB, N)

This procedure calculates F_N and returns the value as FIB.

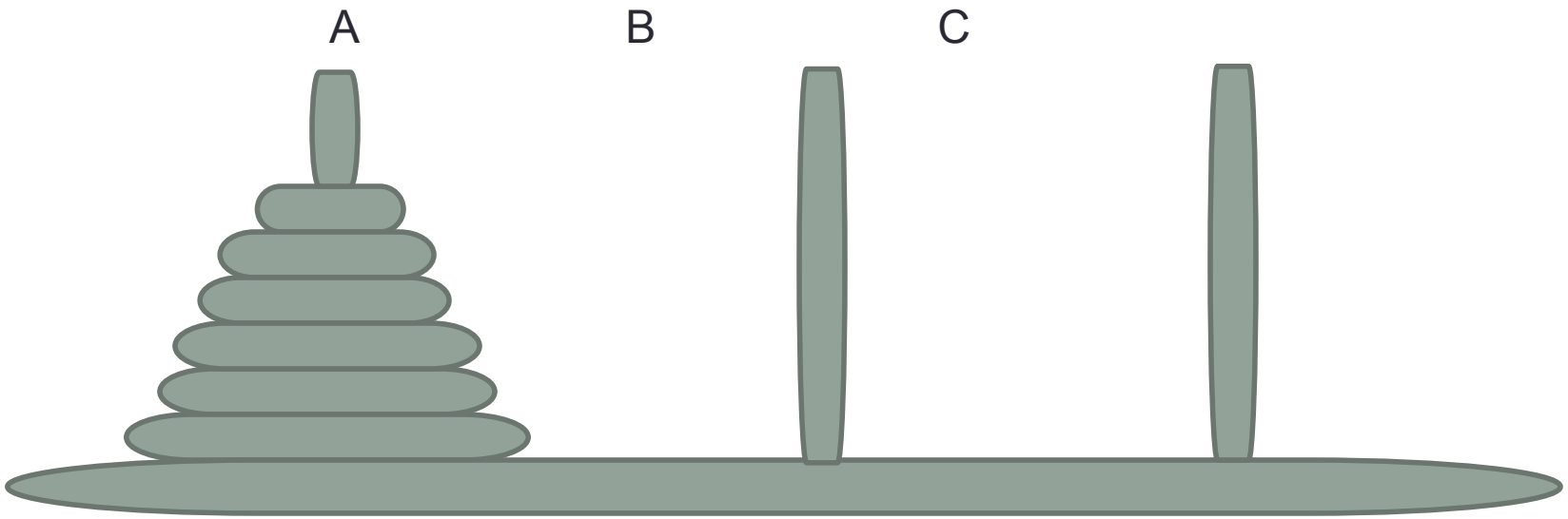
1. If N=0 or N=1 then Set FIB:=N and Return.
2. Call FIBONACCI(FIBA, N-2)
3. Call FIBONACCI(FIBB, N-1)
4. Set FIB:=FIBA+FIBB
5. Return

Towers of Hanoi

- Suppose three pegs, labeled A, B and C are given, and suppose on peg A there are placed a finite no. n of disks with decreasing size.

- The game is to move the disks from peg A to peg C using peg B as an auxiliary. The rules of the game are as follows.
  1. Only one disk(The Top one) may be moved at a time.
  2. At no time a larger disk be placed on a smaller disk.

Towers of Hanoi Continue…



Towers of Hanoi Continue…

- For 1 disk, total no. of disk movement is 1.
  (Just A -> C).
- For 2 disks, total no. of disk movement is 3.
  (A -> B, A -> C, B -> C)
- For 3 disks, the total no. of disk movement is 7.
  (A->C, A->B, C□B, A->C, B->A, B->C, A->C)
- As the disk no. increases, the total movement increases rapidly.
- We can use the technique of recursion to develop a general solution.
- For n>1 disks, Towers of Hanoi problems can be divided into the following sub problems.
  1. Move the top n-1 disks from peg A to peg B.
  2. Move the top disk from peg A to peg C.
  3. Move the top n-1 disks from peg B to peg C.

Towers of Hanoi: Algorithm

TOWER(N, BEG, AUX, END)

1. If N=1, then:
   a) Write: BEG->END
   b) Return.
2. Call TOWER(N-1, BEG, END, AUX)
   [Move N-1 disks from peg BEG to peg AUX]
3. Write: BEG->END
4. Call TOWER(N-1, AUX, BEG, END)
   [Move N-1 disks from peg AUX to peg END]
5. Return.

Queues

- A Queue is a linear list of elements in which deletion can take place only at one end called the front, and insertion can take place only at other end called the rear.

- It is also known as First in First out (FIFO) list, since the first element in a queue will be the first element to out of the queue.

- Examples: Peoples waiting in the ticket counter.

Implementation of Queues

- Queues can be represented in computer in many ways, usually by means of linear array QUEUE and two pointer variables; Front and Rear.
- Front: contains location of the front element of the queue.
- Rear: contains location of the rear element of the queue.
- Front=NULL, indicates that the queue is empty.

- When an element is deleted from the list front is increased.

  Front:=Front+1
- When an element is inserted into the list rear is increased.

  Rear:=Rear+1

Queues Operations

- For a queue of size n, if Rear=n, then we can not add any element since we reached at the end of the queue.
- How ever if we think of the queue as a circular we can insert more elements by changing Rear value to 1 instead of Rear:=Rear+1.
- i.e. we can reset Rear to 1 when Rear=n and we have one to insert.
- Similarly, we can reset Front to 1 when Front=n and we have to delete one.
- If Front=Rear, we have one element to delete. And after that Front and Rear both will be NULL.

Queue Examples

Initially empty (Front=0, Rear=0)

Front=1, Rear=3

Front=2, Rear=3

Front=2, Rear=5

Front=2, Rear=1

Front=5, Rear=1

Front=1, Rear=1

Queue: Algorithms

- QINSERT(queue, n, front, rear, item)
  1. If front=1 and rear=n, or if front=rear+1, then :
     write: Overflow and Return.
  2. If front=NULL, then Set front:=1 and rear:=1
  3. Else if rear=n, then Set rear:=1
  4. Else rear:=rear+1
  5. Set queue[rear]:=item.
  6. Return

- QDELETE(queue, n, front, rear, item)

1. If front=NULL then: write: Underflow and Return.
2. Set item:=queue[front]
3. If front=rear then: Set front:=NULL and rear:=NULL
4. Else if front=n then: Set front:=1
5. Else set front:=front+1
6. Return.

Deques

- A Deque is a linear list of elements in which elements can be added or removed at either end but not in the middle.
- It is a contraction of the name Double ended queue.
- Two variations of the deques are:
  1. Input Restricted Deque: allows insertion only at one end but allows deletions at both ends of the list.
  2. Output Restricted Deque: allows deletion only at one end but allows insertions at both ends of the list.

- The condition left=NULL will be used to indicate that a deque is empty.

Priority Queues

- A priority queue is a collection of elements such that each element has been assigned a priority and such that the order in which elements are deleted and processed comes from the following rules:

1. An elements of higher priority is processed before any elements of lower priority.
2. Two elements with the same priority are processed according to the order in which they were added to the queue.