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

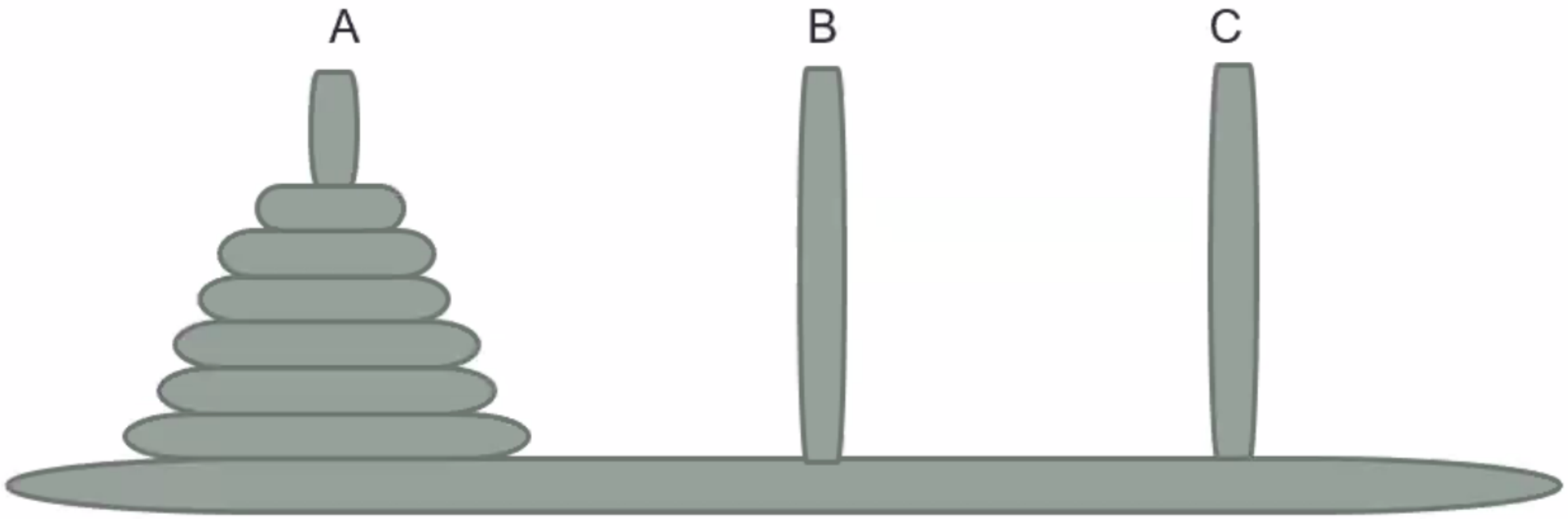
- In Fibonacci series $F_0=0$ and $F_1=1$ and each succeeding term is the sum of two preceding terms.
- The series therefore: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55…..

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 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.

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	A	B	C		
Front=2, Rear=3		B	C		
Front=2, Rear=5		B	C	D	E
Front=2, Rear=1	K	B	C	D	E
Front=5, Rear=1	K				E
Front=1, Rear=1	K				

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.