24. Recursion (Cont'd), The Calculator
Chapter 10 November 26, 2018

- Recursion
  - Factorial (recurrence vs. recursion)
  - Towers of Hanoi

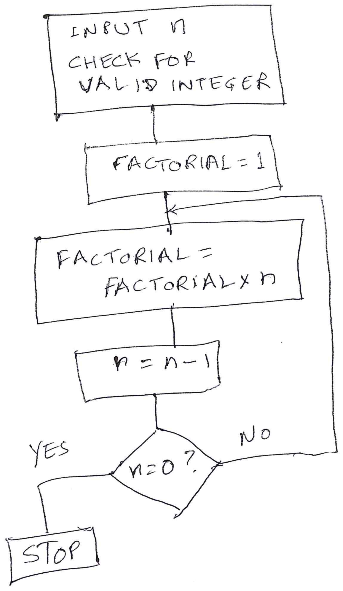
- Calculator
  - High-Level View
  - Subroutine details

Recurrence vs. Recursion

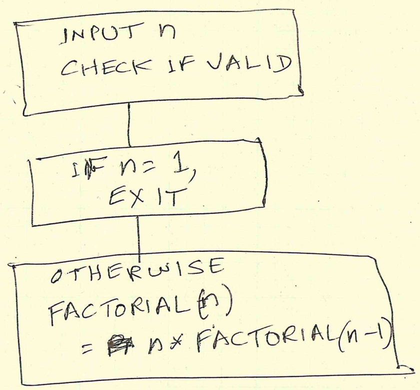
EXAMPLE: FACTORIAL

$n! = n \times (n-1) \times (n-2) \times \cdots 1$

RECURRENCE:

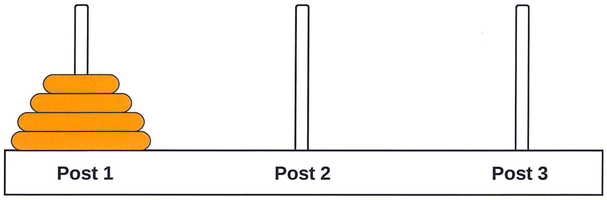


RECURSIVE IMPLEMENTATION OF FACTORIAL



High-Level Example: Towers of Hanoi

Task: Move all disks from current post to another post.



Rules:

(1) Can only move one disk at a time.

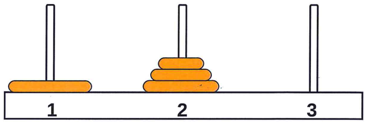
(2) A larger disk can never be placed on top of a smaller disk.

(3) May use third post for temporary storage.

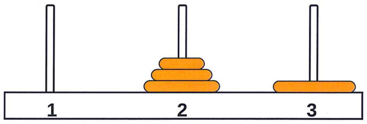
Task Decomposition

Suppose disks start on Post 1, and target is Post 3.

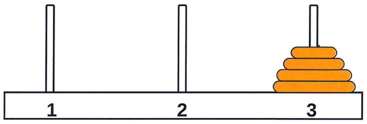
1. Move top n-1 disks to Post 2.



2. Move largest disk to Post 3.



3. Move n-1 disks from Post 2 to Post 3.



High-Level Description of Towers of Hanoi

```
Call towers(num, Frompeg, Topeg, Auxpeg)
return;

towers(num, Frompeg, Topeg, Auxpeg)
  {if num == 1
     {Move disk 1 from peg Frompeg to Topeg;
       return;}
   Call towers(num-1, Frompeg, Auxpeg, Topeg);
   Move disk num, from peg Frompeg, to Topeg;
   Call towers(num-1, Auxpeg, Topeg, Frompeg);
  }
```

Implementation of Towers of Hanoi in LC-3

A (From Peg), B (To Peg), C (Int. Peg)

Calling Hanoi(N, Parm1, Parm2, Parm3) [Hanoi(N, From, To, Int.)]

R2: Parm1, R3: Parm2, R4: Parm3

R6 points to the top element of the stack (x4000 is the empty stack)

Get parameters N, Parm1, Parm2, Parm3 from stack
STACK frame:

<return address>	R6 - 1	
N	R6 - 2	
Parm3	R6 - 3	From
Parm2	R6 - 4	To
Parm1	R6 - 5	Intermediate

<Compute> (including recursive call)
Restore Return address
Return

Task Decomposition (cont.)

Task 1 is really the same problem, with fewer disks and a different target post.

- **"Move n-1 disks from Post 1 to Post 2."**

And Task 3 is also the same problem, with fewer disks and different starting and target posts.

- **"Move n-1 disks from Post 2 to Post 3."**

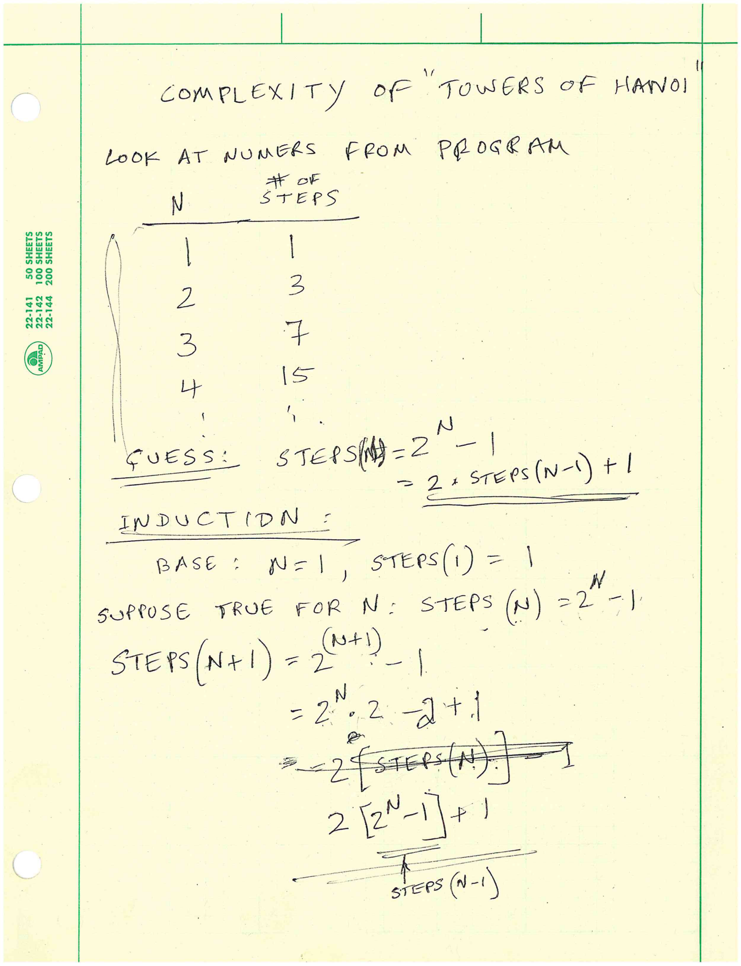
So this is a recursive algorithm.

- **The terminal case is moving the smallest disk -- can move directly without using third post.**
- **Number disks from 1 (smallest) to n (largest).**

```
        ;;
        ;; Program to solve the Tower of Hanoi
        ;; N: number of disks
        ;; Pegs A, B, C kept as characters
        ;; Problem is to move disks from Peg A to Peg C (with intermediate B)
        ;; Pass parameters on stack
        ;;   Manipulate stack using R6
        .ORIG   x3000
        ;;
        LD      R6, StkBase     ; Initialize Stack Pointer
        ;; Not checking for errors here to simplify code
        ;;
        ;; Get input (number of disks)
        ;; For now, we'll assume that we get a legal integer
        ;;    between 0 and 9 (one ASCII character)
        LEA     R0, Prompt
        PUTS                    ; ask for input
        GETC
        OUT                        ; Echo character
        LD      R1, NegASCIIOffset ; Convert ASCII character into number
        ADD     R1, R1, R0         ; R1 now has the number of disks in binary
        ;;
        ;; Set up Pegs (with characters)
        ;;  A: "From" Peg
        ;;  B: "To" Peg
        ;;  C: "Intermediate" Peg
        ;;
        ;;  Push parameters, leaving a space for return address
        STR     R1, R6, #-2     ; Push N on stack
        LD      R0, A
        STR     R0, R6, #-3     ; Push "From" (A)
        LD      R0, B
        STR     R0, R6, #-4     ; Push "To" (B)
        LD      R0, C
        STR     R0, R6, #-5     ; Push "Intermediate" (C)
        ADD     R6, R6, #-5     ; Fix stack pointer
        ;;
        ;; Now call Hanoi (recursively)
        ;;  Hanoi(N, "From", "To", "Int.")
        ;;  Hanoi(N, A, B, C)
        JSR     Hanoi
        LEA     R0, Done        ; Print "Done"
        PUTS
        HALT
        ;;
ASCIIOffset    .FILL  x30       ; Add to integers to convert to ASCII
NegASCIIOffset .FILL  xFFD0     ; -x0030 to strip off ASCII template
A       .FILL   x41
B       .FILL   x42
C       .FILL   x43
        ;;
        ;; Hanoi(N, "From", "Int", "To")
Hanoi   LDR     R4, R6, #0      ; Get Parameter 3
        LDR     R3, R6, #1      ; Get Parameter 2
        LDR     R2, R6, #2      ; Get Parameter 1
        LDR     R1, R6, #3      ; Get N
        ;;  copied 4 parameters
        ADD     R6, R6, #-1     ; Stack pointer
        STR     R7, R6, #0      ; Push return address on stack
        ;; Now check number of pegs
        ADD     R0, R1, #-1     ; Check if only one peg left
        Brp     Continue        ; More pegs left
        ;; Move last disk and quit
        LEA     R0, String1


        LDR     R7, R6, #0      ; Pop return address
        ADD     R6, R6, #5      ;  Stack pointer
        RET
        ;;
Prompt  .FILL   x000A           ; force new line
        .STRINGz "Input the number of disks: "
String1 .FILL   x000A
        .STRINGz "Move Disk "
String2 .STRINGz " from Peg "
String3 .STRINGz " to Peg "
Done    .FILL   x000A
        .STRINGz "Done!"
        ;;
StkBase .FILL   x4000           ; Stack starts here
;
        .END
```

```
         PUTS
         ADD     R0, R1, #0      ; Integer 1
         LD      R5, ASCIIOffset
         ADD     R0, R0, R5       ; Make ASCII
         OUT
         LEA     R0, String2     ; " From Peg "
         PUTS
         LDR     R0, R6, #3      ; From Peg
         OUT
         LEA     R0, String3     ; " To Peg "
         PUTS
         LDR     R0, R6, #2      ; To peg
         OUT
         LDR     R7, R6, #0      ; Restore return address
         ADD     R6, R6, #5      ; Remove 4 parameters, return address
         RET
         ;;
         ;; If not done, update disks, call Hanoi again
         ;;  with Hanoi(N-1, "From", "Int.", "To")
         ;;
Continue LDR     R0, R6, #4      ; N
         ADD     R0, R0, #-1     ; N-1
         STR     R0, R6, #-1     ; Push new number N
         LDR     R0, R6, #3      ;
         STR     R0, R6, #-2     ; Push From Peg
         LDR     R0, R6, #1
         STR     R0, R6, #-3     ; Push Int. Peg
         LDR     R0, R6, #2
         STR     R0, R6, #-4     ; Push To Peg
         ADD     R6, R6, #-4     ; Fix stack pointer
         JSR     Hanoi
         ;;
         LEA     R0, String1
         PUTS
         LDR     R1, R6, #4      ; Get N from stack
         LD      R0, ASCIIOffset ; Convert Number to ASCII
         ADD     R0, R0, R1 ; Make ASCII
         OUT
         LEA     R0, String2     ; " From Peg "
         PUTS
         LDR     R0, R6, #3      ; Parameter 1
         OUT
         LEA     R0, String3     ; " To Peg "
         PUTS
         LDR     R0, R6, #2      ; Parameter 2
         OUT
         ;;
         ;; Call Hanoi(N-1, "Int.", "To", "From")
         ;;
         LDR     R1, R6, #4      ; Get N from stack
         ADD     R0, R1, #-1     ; N-1 passed to subroutine
         STR     R0, R6, #-1     ;  pushed on top of stack
         LDR     R0, R6, #1
         STR     R0, R6, #-2     ; Push Int.
         LDR     R0, R6, #2
         STR     R0, R6, #-3     ; Push To
         LDR     R0, R6, #3
         STR     R0, R6, #-4     ; Push From
         ADD     R6, R6, #-4     ; fix stack pointer
         JSR     Hanoi
         LDR     R4, R6, #1      ; Pop Parameter 3
         LDR     R3, R6, #2      ; Pop Parameter 2
         LDR     R2, R6, #3      ; Pop Parameter 1
         LDR     R1, R6, #4      ; Pop N
```



COMPLEXITY OF "TOWERS OF HANOI"

LOOK AT NUMERS FROM PROGRAM

N	# OF STEPS
1	1
2	3
3	7
4	15
⋮	⋮

GUESS: $\text{STEPS}(N) = 2^N - 1$

$= 2 * \text{STEPS}(N-1) + 1$

INDUCTION:

BASE: $N = 1$, $\text{STEPS}(1) = 1$

SUPPOSE TRUE FOR N: $\text{STEPS}(N) = 2^N - 1$

$\text{STEPS}(N+1) = 2^{(N+1)} - 1$

$= 2^N \cdot 2 - 2 + 1$

~~$= 2[\text{STEPS}(N)]$~~

$2\underbrace{[2^N - 1]}_{\text{STEPS}(N-1)} + 1$

Calculator

- Commands
  - X: Exit the simulation
  - C: Clear (all values from the stack)
  - D: Display the value at the top of the stack

Note: This is a stack-based calculator

- Operations

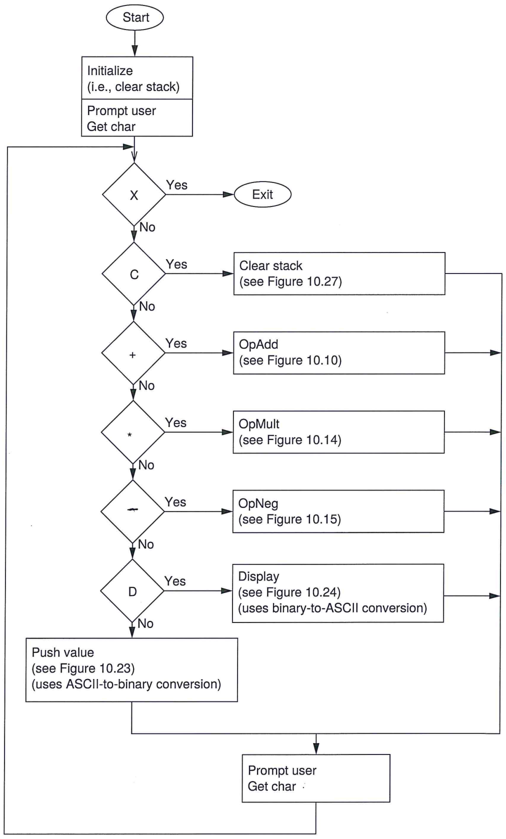
+: Replace top two elements on the stack with their sum

*: Replace top two elements on stack with their product

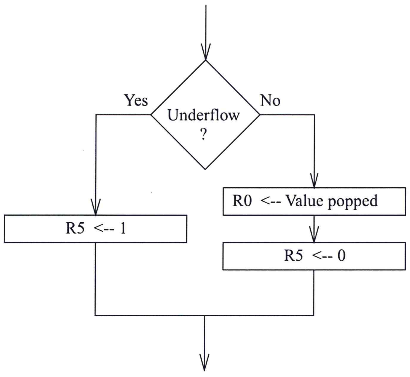
-: Negate the top element on the stack

Enter: Push value typed on keyboard onto top of the stack

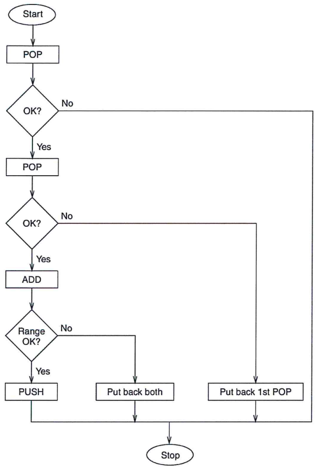
Overview of Calculator



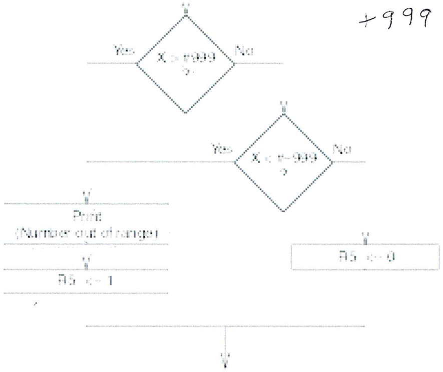
Test Underflow for POP



ADD Operands on Stack

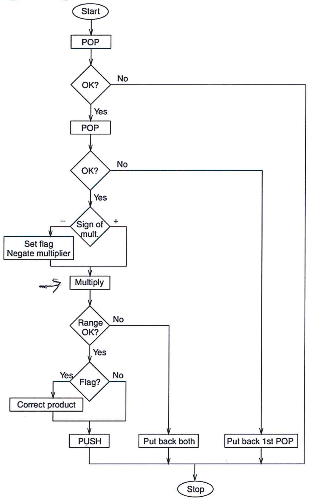


Check for Correct Range of Operands



+999 to −999

OpMult (Multiply top two stack elements)


Start
POP
OK?
No
Yes
POP
OK?
No
Yes
Sign of mult.
−
+
Set flag
Negate multiplier
Multiply
Range OK?
No
Yes
Flag?
Yes
No
Correct product
PUSH
Put back both
Put back 1st POP
Stop