Towers of Hanoi

Task

Solve the Towers of Hanoi (https://en.wikipedia.org/wiki/Towers_of_Hanoi) problem with recursion.

(/wiki/Category:Solutions_by

Towers of Hanoi
You are encouraged to solve this task (/wiki/Rosetta_Code:Solve_a according to the task description, using any language you may know.

Contents

11l (/wiki/Category:11l)

Translation of: Python

```
F hanoi(ndisks, startPeg = 1, endPeg = 3) -> N
   I ndisks
      hanoi(ndisks - 1, startPeg, 6 - startPeg - endPeg)
      print('Move disk #. from peg #. to peg #.'.format(ndisks, startPeg, endPeg))
      hanoi(ndisks - 1, 6 - startPeg - endPeg, endPeg)

hanoi(ndisks' 3)
```

Output:

```
Move disk 1 from peg 1 to peg 3
Move disk 2 from peg 1 to peg 2
Move disk 1 from peg 3 to peg 2
Move disk 3 from peg 1 to peg 3
Move disk 1 from peg 2 to peg 1
Move disk 2 from peg 2 to peg 3
Move disk 1 from peg 1 to peg 3
```

360 Assembly (/wiki/Category:360_Assembly)

Translation of: PL/I

```
*        Towers of Hanoi           08/09/2015
HANOITOW CSECT
         USING  HANOITOW,R12       r12 : base register
         LR     R12,R15            establish base register
         ST     R14,SAVE14         save r14
BEGIN    LH     R2,=H'4'           n <===
         L      R3,=C'123 '        stating position
         BAL    R14,MOVE           r1=move(m,n)
RETURN   L      R14,SAVE14         restore r14
         BR     R14                return to caller
SAVE14   DS     F                  static save r14
PG       DC     CL44'xxxxxxxxxxxx Move disc from pole X to pole Y'
NN       DC     F'0'
POLEX    DS     F                  current poles
POLEN    DS     F                  new poles
*        ....   recursive          subroutine move(n, poles)  [r2,r3]
MOVE     LR     R10,R11            save stackptr (r11) in r10 temp
         LA     R1,STACKLEN        amount of storage required
         GETMAIN RU,LV=(R1)        allocate storage for stack
         USING  STACKDS,R11        make storage addressable
         LR     R11,R1             establish stack addressability
         ST     R14,SAVE14M        save previous r14
         ST     R10,SAVE11M        save previous r11
         LR     R1,R5              restore saved argument r5
BEGINM   STM    R2,R3,STACK        push arguments to stack
         ST     R3,POLEX
         CH     R2,=H'1'           if n<>1
         BNE    RECURSE            then goto recurse
         L      R1,NN
         LA     R1,1(R1)           nn=nn+1
         ST     R1,NN
         XDECO  R1,PG              nn
         MVC    PG+33(1),POLEX+0   from
         MVC    PG+43(1),POLEX+1   to
         XPRNT  PG,44              print "move disk from to"
         B      RETURNM
RECURSE  L      R2,N               n
         BCTR   R2,0               n=n-1
         MVC    POLEN+0(1),POLES+0 from
         MVC    POLEN+1(1),POLES+2 via
         MVC    POLEN+2(1),POLES+1 to
         L      R3,POLEN           new poles
         BAL    R14,MOVE           call move(n-1,from,via,to)
         LA     R2,1               n=1
         MVC    POLEN,POLES
         L      R3,POLEN           new poles
         BAL    R14,MOVE           call move(1,from,to,via)
         L      R2,N               n
         BCTR   R2,0               n=n-1
         MVC    POLEN+0(1),POLES+2 via
         MVC    POLEN+1(1),POLES+1 to
         MVC    POLEN+2(1),POLES+0 from
         L      R3,POLEN           new poles
         BAL    R14,MOVE           call move(n-1,via,to,from)
RETURNM  LM     R2,R3,STACK        pull arguments from stack
         LR     R1,R11             current stack
         L      R14,SAVE14M        restore r14
         L      R11,SAVE11M        restore r11
         LA     R0,STACKLEN        amount of storage to free
         FREEMAIN A=(R1),LV=(R0)   free allocated storage
         BR     R14                return to caller
         LTORG
         DROP   R12                base no longer needed
STACKDS  DSECT                     dynamic area
SAVE14M  DS     F                  saved r14
SAVE11M  DS     F                  saved r11
STACK    DS     0F                 stack
N        DS     F                  r2 n
POLES    DS     F                  r3 poles
STACKLEN EQU    *-STACKDS
         YREGS
         END    HANOITOW
```

Output:

```
          1 Move disc from pole 1 to pole 3
          2 Move disc from pole 1 to pole 2
          3 Move disc from pole 3 to pole 2
          4 Move disc from pole 1 to pole 3
          5 Move disc from pole 2 to pole 1
          6 Move disc from pole 2 to pole 3
```

8080 Assembly (/wiki/Category:8080_Assembly)

```
        org     100h
        lhld    6       ; Top of CP/M usable memory
        sphl            ; Put the stack there
        lxi     b,0401h ; Set up first arguments to move()
        lxi     d,0203h
        call    move    ; move(4, 1, 2, 3)
        rst     0       ; quit program
        ;;;     Move B disks from C via D to E.
move:   dcr     b       ; One fewer disk in next iteration
        jz      mvout   ; If this was the last disk, print move and stop
        push    b       ; Otherwise, save registers,
        push    d
        mov     a,d     ; First recursive call
        mov     d,e
        mov     e,a
        call    move    ; move(B-1, C, E, D)
        pop     d       ; Restore registers
        pop     b
        call    mvout   ; Print current move
        mov     a,c     ; Second recursive call
        mov     c,d
        mov     d,a
        jmp     move    ; move(B-1, D, C, E) - tail call optimization
        ;;;     Print move, saving registers.
mvout:  push    b       ; Save registers on stack
        push    d
        mov     a,c     ; Store 'from' as ASCII digit in 'from' space
        adi     '0'
        sta     out1
        mov     a,e     ; Store 'to' as ASCII digit in 'to' space
        adi     '0'
        sta     out2
        lxi     d,outstr
        mvi     c,9     ; CP/M call to print the string
        call    5
        pop     d       ; Restore register contents
        pop     b
        ret
        ;;;     Move output with placeholder for pole numbers
outstr: db      'Move disk from pole '
out1:   db      '* to pole '
out2:   db      '*',13,10,'$'
```

Output:

```
Move disk from pole 1 to pole 2
Move disk from pole 1 to pole 3
Move disk from pole 2 to pole 3
Move disk from pole 1 to pole 2
Move disk from pole 3 to pole 1
Move disk from pole 3 to pole 2
Move disk from pole 1 to pole 2
Move disk from pole 1 to pole 3
Move disk from pole 2 to pole 3
Move disk from pole 2 to pole 1
Move disk from pole 3 to pole 1
Move disk from pole 2 to pole 3
Move disk from pole 1 to pole 2
Move disk from pole 1 to pole 3
Move disk from pole 2 to pole 3
```

8086 Assembly (/wiki/Category:8086_Assembly)

```
        cpu     8086
        bits    16
        org     100h
section .text
        mov     bx,0402h        ; Set up first arguments to move()
        mov     cx,0103h        ; Registers chosen s.t. CX contains output
        ;;;     Move BH disks from CH via BL to CL
move:   dec     bh              ; One fewer disk in next iteration
        jz      .out            ; If this was last disk, just print move
        push    bx              ; Save the registers for a recursive call
        push    cx
        xchg    bl,cl           ; Swap the 'to' and 'via' registers
        call    move            ; move(BH, CH, CL, BL)
        pop     cx              ; Restore the registers from the stack
        pop     bx
        call    .out            ; Print the move
        xchg    ch,bl           ; Swap the 'from' and 'via' registers
        jmp     move            ; move(BH, BL, CH, CL)
        ;;;     Print the move
.out:   mov     ax,'00'         ; Add ASCII 0 to both 'from' and 'to'
        add     ax,cx           ; in one 16-bit operation
        mov     [out1],ah       ; Store 'from' field in output
        mov     [out2],al       ; Store 'to' field in output
        mov     dx,outstr       ; MS-DOS system call to print string
        mov     ah,9
        int     21h
        ret
section .data
outstr: db      'Move disk from pole '
out1:   db      '* to pole '
out2:   db      '*',13,10,'$'
```

Output:

```
Move disk from pole 1 to pole 2
Move disk from pole 1 to pole 3
Move disk from pole 2 to pole 3
Move disk from pole 1 to pole 2
Move disk from pole 3 to pole 1
Move disk from pole 3 to pole 2
Move disk from pole 1 to pole 2
Move disk from pole 1 to pole 3
Move disk from pole 2 to pole 3
Move disk from pole 2 to pole 1
Move disk from pole 3 to pole 1
Move disk from pole 2 to pole 3
Move disk from pole 1 to pole 2
Move disk from pole 1 to pole 3
Move disk from pole 2 to pole 3
```

8th (/wiki/Category:8th)

```
5 var, disks
var sa
var sb
var sc

: save sc ! sb ! sa ! disks ! ;
: get sa @ sb @ sc @ ;
: get2 get swap ;
: hanoi
        save disks @ not if ;; then
        disks @ get
        disks @ n:1- get2 hanoi save
        cr
        " move a ring from " .  sa @ . " to " . sb @ .
        disks @ n:1- get2 rot hanoi
;

" Tower of Hanoi, with " . disks @ . " rings: " .
disks @ 1 2 3 hanoi cr bye
```

ActionScript (/wiki/Category:ActionScript)

```
public function move(n:int, from:int, to:int, via:int):void
{
    if (n > 0)
    {
        move(n - 1, from, via, to);
        trace("Move disk from pole " + from + " to pole " + to);
        move(n - 1, via, to, from);
    }
}
```

Ada (/wiki/Category:Ada)

```
with Ada.Text_Io; use Ada.Text_Io;

procedure Towers is
   type Pegs is (Left, Center, Right);
   procedure Hanoi (Ndisks : Natural; Start_Peg : Pegs := Left; End_Peg : Pegs := Right; Via_Peg : Pegs := Center) is
   begin
      if Ndisks > 0 then
         Hanoi(Ndisks - 1, Start_Peg, Via_Peg, End_Peg);
         Put_Line("Move disk" & Natural'Image(Ndisks) & " from " & Pegs'Image(Start_Peg) & " to " & Pegs'Image(End_Peg));
         Hanoi(Ndisks - 1, Via_Peg, End_Peg, Start_Peg);
      end if;
   end Hanoi;
begin
   Hanoi(4);
end Towers;
```

Agena (/wiki/Category:Agena)

```
move := proc(n::number, src::number, dst::number, via::number) is
   if n > 0 then
      move(n - 1, src, via, dst)
      print(src & ' to ' & dst)
      move(n - 1, via, dst, src)
   fi
end

move(4, 1, 2, 3)
```

ALGOL 68 (/wiki/Category:ALGOL_68)

```
PROC move = (INT n, from, to, via) VOID:
  IF n > 0 THEN
    move(n - 1, from, via, to);
    printf(($"Move disk from pole "g" to pole "gl$, from, to));
    move(n - 1, via, to, from)
  FI
;

main: (
  move(4, 1,2,3)
)
```

COMMENT Disk number is also printed in this code (works with a68g): COMMENT

```

PROC move = (INT n, from, to, via) VOID:
  IF n > 0 THEN
    move(n - 1, from, via, to);
    printf(($"Move disk "g"     from pole "g"     to pole "gl$, n,  from, to));
    move(n - 1, via, to, from)
  FI ;
main: (
  move(4, 1,2,3)
)

```

ALGOL-M (/wiki/Category:ALGOL-M)

```
begin
procedure move(n, src, via, dest);
integer n;
string(1) src, via, dest;
begin
    if n > 0 then
    begin
        move(n-1, src, dest, via);
        write("Move disk from pole ");
        writeon(src);
        writeon(" to pole ");
        writeon(dest);
        move(n-1, via, src, dest);
    end;
end;

move(4, "1", "2", "3");
end
```

Output:

```
Move disk from pole 1 to pole 2
Move disk from pole 1 to pole 3
Move disk from pole 2 to pole 3
Move disk from pole 1 to pole 2
Move disk from pole 3 to pole 1
Move disk from pole 3 to pole 2
Move disk from pole 1 to pole 2
Move disk from pole 1 to pole 3
Move disk from pole 2 to pole 3
Move disk from pole 2 to pole 1
Move disk from pole 3 to pole 1
Move disk from pole 2 to pole 3
Move disk from pole 1 to pole 2
Move disk from pole 1 to pole 3
Move disk from pole 2 to pole 3
```

ALGOL W (/wiki/Category:ALGOL_W)

Following Agena, Algol 68, AmigaE...

```
begin
    procedure move ( integer value n, from, to, via ) ;
        if n > 0 then begin
            move( n - 1, from, via, to );
            write( i_w := 1, s_w := 0, "Move disk from peg: ", from, " to peg: ", to );
            move( n - 1, via, to, from )
        end move ;

    move( 4, 1, 2, 3 )
end.
```

AmigaE (/wiki/Category:AmigaE)

```
PROC move(n, from, to, via)
  IF n > 0
    move(n-1, from, via, to)
    WriteF('Move disk from pole \d to pole \d\n', from, to)
    move(n-1, via, to, from)
  ENDIF
ENDPROC

PROC main()
  move(4, 1,2,3)
ENDPROC
```

APL (/wiki/Category:APL)

Works with: Dyalog APL (/wiki/Dyalog_APL)

```
hanoi←{
    move←{
        n from to via←⍵
        n≤0:⍬
        l←∇(n-1) from via to
        r←∇(n-1) via to from
        l,(⊂from to),r
    }
    'Move disk from pole ⍵,⍨1↓,⍵,⍨' to pole ⍵,⍨1↓'⎕FMT↑move ⍵
}
```

Output:

```
      hanoi 4 1 2 3
Move disk from pole 1 to pole 3
Move disk from pole 1 to pole 2
Move disk from pole 3 to pole 2
Move disk from pole 1 to pole 3
Move disk from pole 2 to pole 1
Move disk from pole 2 to pole 3
Move disk from pole 1 to pole 3
Move disk from pole 1 to pole 2
Move disk from pole 3 to pole 2
Move disk from pole 3 to pole 1
Move disk from pole 2 to pole 1
Move disk from pole 3 to pole 2
Move disk from pole 1 to pole 3
Move disk from pole 1 to pole 2
Move disk from pole 3 to pole 2
```

AppleScript (/wiki/Category:AppleScript)

```
---------------------- TOWERS OF HANOI ----------------------

-- hanoi :: Int -> (String, String, String) -> [(String, String)]
on hanoi(n, abc)
    script go
        on |λ|(n, {x, y, z})
            if n > 0 then
                |λ|(n - 1, {x, z, y}) & ¬
                    {{x, y}} & |λ|(n - 1, {z, y, x})
            else
                {}
            end if
        end |λ|
    end script
    go's |λ|(n, abc)
end hanoi


---------------------------- TEST ---------------------------
on run
    script arrow
        on |λ|(abc)
            item 1 of abc & " -> " & item 2 of abc
        end |λ|
    end script

    unlines(map(arrow, ¬
        hanoi(3, {"left", "right", "mid"})))
end run


--------------------- GENERIC FUNCTIONS ---------------------

-- Lift 2nd class handler function into 1st class script wrapper
-- mReturn :: First-class m => (a -> b) -> m (a -> b)
on mReturn(f)
    if class of f is script then
        f
    else
        script
            property |λ| : f
        end script
    end if
end mReturn

-- map :: (a -> b) -> [a] -> [b]
on map(f, xs)
    tell mReturn(f)
        set lng to length of xs
        set lst to {}
        repeat with i from 1 to lng
            set end of lst to |λ|(item i of xs, i, xs)
        end repeat
        return lst
    end tell
end map

-- unlines :: [String] -> String
on unlines(xs)
    set {dlm, my text item delimiters} to ¬
        {my text item delimiters, linefeed}
    set str to xs as text
    set my text item delimiters to dlm
    str
end unlines
```

Output:

```
left -> right
left -> mid
right -> mid
left -> right
mid -> left
mid -> right
left -> right
```

More illustratively:

(I've now eliminated the recursive |move|() handler's tail calls. So it's now only called 2 ^ (n - 1) times as opposed to 2 ^ (n + 1) - 1 with full recursion. The maximum call depth of n is only reached once, whereas with full recursion, the maximum depth was n + 1 and this was reached 2 ^ n times.)

```
on hanoi(n, source, target)
    set t1 to tab & "tower 1: " & tab
    set t2 to tab & "tower 2: " & tab
    set t3 to tab & "tower 3: " & tab

    script o
        property m : 0
        property tower1 : {}
        property tower2 : {}
        property tower3 : {}
        property towerRefs : {a reference to tower1, a reference to tower2, a reference to tower3}
        property process : missing value

        on |move|(n, source, target)
            set aux to 6 - source - target
            repeat with n from n to 2 by -1 -- Tail call elimination repeat.
                |move|(n - 1, source, aux)
                set end of item target of my towerRefs to n
                tell item source of my towerRefs to set its contents to reverse of rest of its reverse
                set m to m + 1
                set end of my process to ¬
                    {(m as text) & ". move disc " & n & (" from tower " & source) & (" to tower " & target & ":"), ¬
                        t1 & tower1, ¬
                        t2 & tower2, ¬
                        t3 & tower3}
                tell source
                    set source to aux
                    set aux to it
                end tell
            end repeat
            -- Specific code for n = 1:
            set end of item target of my towerRefs to 1
            tell item source of my towerRefs to set its contents to reverse of rest of its reverse
            set m to m + 1
            set end of my process to ¬
                {(m as text) & ". move disc 1 from tower " & source & (" to tower " & target & ":"), ¬
                    t1 & tower1, ¬
                    t2 & tower2, ¬
                    t3 & tower3}
        end |move|
    end script

    repeat with i from n to 1 by -1
        set end of item source of o's towerRefs to i
    end repeat

    set astid to AppleScript's text item delimiters
    set AppleScript's text item delimiters to ", "
    set o's process to {"Starting with " & n & (" discs on tower " & (source & ":")), ¬
        t1 & o's tower1, t2 & o's tower2, t3 & o's tower3}
    if (n > 0) then tell o to |move|(n, source, target)
    set end of o's process to "That's it!"
    set AppleScript's text item delimiters to linefeed
    set process to o's process as text
    set AppleScript's text item delimiters to astid

    return process
end hanoi

-- Test:
set numberOfDiscs to 3
set sourceTower to 1
set destinationTower to 2
hanoi(numberOfDiscs, sourceTower, destinationTower)
```

Output:

```
"Starting with 3 discs on tower 1:
    tower 1:     3, 2, 1
    tower 2:
    tower 3:
1. move disc 1 from tower 1 to tower 2:
    tower 1:     3, 2
    tower 2:     1
    tower 3:
2. move disc 2 from tower 1 to tower 3:
    tower 1:     3
    tower 2:     1
    tower 3:     2
3. move disc 1 from tower 2 to tower 3:
    tower 1:     3
    tower 2:
    tower 3:     2, 1
4. move disc 3 from tower 1 to tower 2:
    tower 1:
    tower 2:     3
    tower 3:     2, 1
5. move disc 1 from tower 3 to tower 1:
    tower 1:     1
    tower 2:     3
    tower 3:     2
6. move disc 2 from tower 3 to tower 2:
    tower 1:     1
    tower 2:     3, 2
    tower 3:
7. move disc 1 from tower 1 to tower 2:
    tower 1:
    tower 2:     3, 2, 1
    tower 3:
That's it!"
```

ARM Assembly (/wiki/Category:ARM_Assembly)

```
.text
.global _start
_start: mov     r0,#4           @ 4 disks,
        mov     r1,#1           @ from pole 1,
        mov     r2,#2           @ via pole 2,
        mov     r3,#3           @ to pole 3.
        bl      move
        mov     r0,#0           @ Exit to Linux afterwards
        mov     r7,#1
        swi     #0
        @@@     Move r0 disks from r1 via r2 to r3
move:   subs    r0,r0,#1        @ One fewer disk in next iteration
        beq     show            @ If last disk, just print move
        push    {r0-r3,lr}      @ Save all the registers incl. link register
        eor     r2,r2,r3        @ Swap the 'to' and 'via' registers
        eor     r3,r2,r3
        eor     r2,r2,r3
        bl      move            @ Recursive call
        pop     {r0-r3}         @ Restore all the registers except LR
        bl      show            @ Show current move
        eor     r1,r1,r3        @ Swap the 'to' and 'via' registers
        eor     r3,r1,r3
        eor     r1,r1,r3
        pop     {lr}            @ Restore link register
        b       move            @ Tail call
        @@@     Show move
show:   push    {r0-r3,lr}      @ Save all the registers
        add     r1,r1,#'0       @ Write the source pole
        ldr     lr,=spole
        strb    r1,[lr]
        add     r3,r3,#'0       @ Write the destination pole
        ldr     lr,=dpole
        strb    r3,[lr]
        mov     r0,#1           @ 1 = stdout
        ldr     r1,=moves       @ Pointer to string
        ldr     r2,=mlen        @ Length of string
        mov     r7,#4           @ 4 = Linux write syscall
        swi     #0              @ Print the move
        pop     {r0-r3,pc}      @ Restore all the registers and return
.data
moves:  .ascii  "Move disk from pole "
spole:  .ascii  "* to pole "
dpole:  .ascii  "*\n"
mlen    =       . - moves
```

Output:

```
Move disk from pole 1 to pole 2
Move disk from pole 1 to pole 3
Move disk from pole 3 to pole 1
Move disk from pole 1 to pole 2
Move disk from pole 2 to pole 3
Move disk from pole 2 to pole 1
Move disk from pole 1 to pole 2
Move disk from pole 1 to pole 3
Move disk from pole 3 to pole 1
Move disk from pole 3 to pole 2
Move disk from pole 2 to pole 3
Move disk from pole 3 to pole 1
Move disk from pole 1 to pole 2
Move disk from pole 1 to pole 3
Move disk from pole 3 to pole 1
```

Arturo (/wiki/Category:Arturo)

Translation of: D

```
hanoi: function [n f dir via][
        if n>0 [
                hanoi n-1 f via dir
                print ["Move disk" n "from" f "to" dir]
                hanoi n-1 via dir f
        ]
]

hanoi 3 'L 'M 'R
```

Output:

```
Move disk 1 from L to M
Move disk 2 from L to R
Move disk 1 from M to R
Move disk 3 from L to M
Move disk 1 from R to L
Move disk 2 from R to M
Move disk 1 from L to M
```

AutoHotkey (/wiki/Category:AutoHotkey)

```
move(n, from, to, via)  ;n = # of disks, from = start pole, to = end pole, via = remaining pole
{
  if (n = 1)
  {
    msgbox (http://www.autohotkey.com/docs/commands/MsgBox.htm) , Move disk from pole %from% to pole %to%
  }
  else
  {
    move(n-1, from, via, to)
    move(1, from, to, via)
    move(n-1, via, to, from)
  }
}
move(64, 1, 3, 2)
```

AutoIt (/wiki/Category:AutoIt)

```
Func (https://www.autoitscript.com/autoit3/docs/keywords.htm) move($n, $from, $to, $via)
        If (https://www.autoitscript.com/autoit3/docs/keywords.htm) ($n = 1) Then (https://www.autoitscript.com/autoit3/docs/keywords.
htm)
                ConsoleWrite (https://www.autoitscript.com/autoit3/docs/functions/ConsoleWrite.htm)(StringFormat (https://www.autoitsc
ript.com/autoit3/docs/functions/StringFormat.htm)("Move disk from pole "&$from&" To pole "&$to&"\n"))
        Else (https://www.autoitscript.com/autoit3/docs/keywords.htm)
                move($n - 1, $from, $via, $to)
                move(1, $from, $to, $via)
                move($n - 1, $via, $to, $from)
        EndIf (https://www.autoitscript.com/autoit3/docs/keywords.htm)
EndFunc (https://www.autoitscript.com/autoit3/docs/keywords.htm)

move(4, 1,2,3)
```

AWK (/wiki/Category:AWK)

Translation of: Logo

```
$ awk 'func hanoi(n,f,t,v){if(n>0){hanoi(n-1,f,v,t);print(f,"->",t);hanoi(n-1,v,t,f)}}
BEGIN{hanoi(4,"left","middle","right")}'
```

Output:

```
left -> right
left -> middle
right -> middle
left -> right
middle -> left
middle -> right
left -> right
left -> middle
right -> middle
right -> left
middle -> left
right -> middle
left -> right
left -> middle
right -> middle
```

BASIC (/wiki/Category:BASIC)

Using a Subroutine

Works with: FreeBASIC (/wiki/FreeBASIC)
Works with: RapidQ (/wiki/RapidQ)

```
SUB move (n AS Integer, fromPeg AS Integer, toPeg AS Integer, viaPeg AS Integer)
    IF n>0 THEN
        move n-1, fromPeg, viaPeg, toPeg
        PRINT "Move disk from "; fromPeg; " to "; toPeg
        move n-1, viaPeg, toPeg, fromPeg
    END IF
END SUB

move 4,1,2,3
```

Using `GOSUB`s

Here's an example of implementing recursion in an old BASIC that only has global variables:

Works with: Applesoft BASIC (/wiki/Applesoft_BASIC)
Works with: Commodore BASIC (/wiki/Commodore_BASIC)

```
10 DIM N(1024), F(1024), T(1024), V(1024): REM STACK PER PARAMETER
20 SP = 0:    REM STACK POINTER
30 N(SP) = 4: REM START WITH 4 DISCS
40 F(SP) = 1: REM ON PEG 1
50 T(SP) = 2: REM MOVE TO PEG 2
60 V(SP) = 3: REM VIA PEG 3
70 GOSUB 100
80 END
90 REM MOVE SUBROUTINE
100 IF N(SP) = 0 THEN RETURN
110 OS = SP:           REMEMBER STACK POINTER
120 SP = SP + 1:       REM INCREMENT STACK POINTER
130 N(SP) = N(OS) - 1: REM MOVE N-1 DISCS
140 F(SP) = F(OS)    : REM FROM START PEG
150 T(SP) = V(OS)    : REM TO VIA PEG
160 V(SP) = T(OS)    : REM VIA TO PEG
170 GOSUB 100
180 OS = SP - 1: REM OS WILL HAVE CHANGED
190 PRINT "MOVE DISC FROM"; F(OS); "TO"; T(OS)
200 N(SP) = N(OS) - 1: REM MOVE N-1 DISCS
210 F(SP) = V(OS)    : REM FROM VIA PEG
220 T(SP) = T(OS)    : REM TO DEST PEG
230 V(SP) = F(OS)    : REM VIA FROM PEG
240 GOSUB 100
250 SP = SP - 1      : REM RESTORE STACK POINTER FOR CALLER
260 RETURN
```

Using binary method

Works with: Commodore BASIC (/wiki/Commodore_BASIC)
Very fast version in BASIC V2 on Commodore C-64

```
  10 def fnm(x)=x-int(x/3)*3:rem modulo
  20 n=4:gosub 100
  30 end
 100 rem hanoi
 110 :for m=1 to 2^n-1
 120 ::print m;":",fnm(m and m-1)+1;" to ";fnm((m or m-1)+1)+1
 130 :next
 140 return
```

Output:

```
1 :     1  to 3
2 :     1  to 2
3 :     3  to 2
4 :     1  to 3
5 :     2  to 1
6 :     2  to 3
7 :     1  to 3
8 :     1  to 2
9 :     3  to 2
10 :    3  to 1
11 :    2  to 1
12 :    3  to 2
13 :    1  to 3
14 :    1  to 2
15 :    3  to 2
```

BASIC256 (/wiki/Category:BASIC256)

```
call move(4,1,2,3)
print "Towers of Hanoi puzzle completed!"
end

subroutine move (n, fromPeg, toPeg, viaPeg)
    if n>0 then
        call move(n-1, fromPeg, viaPeg, toPeg)
        print "Move disk from "+fromPeg+" to "+toPeg
        call move(n-1, viaPeg, toPeg, fromPeg)
    end if
end subroutine
```

Output:

```
Move disk from 1 to 3
Move disk from 1 to 2
Move disk from 3 to 2
Move disk from 1 to 3
Move disk from 2 to 1
Move disk from 2 to 3
Move disk from 1 to 3
Move disk from 1 to 2
Move disk from 3 to 2
Move disk from 3 to 1
Move disk from 2 to 1
Move disk from 3 to 2
Move disk from 1 to 3
Move disk from 1 to 2
Move disk from 3 to 2
Towers of Hanoi puzzle completed!
```

Batch File (/wiki/Category:Batch_File)

```
@echo (https://www.ss64.com/nt/echo.html) off
setlocal (https://www.ss64.com/nt/setlocal.html) enabledelayedexpansion

        %==The main thing==%
        %==First param - Number of disks==%
        %==Second param - Start pole==%
        %==Third param - End pole==%
        %==Fourth param - Helper pole==%
call (https://www.ss64.com/nt/call.html) :move (https://www.ss64.com/nt/move.html) 4 START END HELPER
echo (https://www.ss64.com/nt/echo.html).
pause (https://www.ss64.com/nt/pause.html)
exit (https://www.ss64.com/nt/exit.html) /b 0

        %==The "function"==%
:move (https://www.ss64.com/nt/move.html)
        setlocal (https://www.ss64.com/nt/setlocal.html)
        set (https://www.ss64.com/nt/set.html) n=%1
        set (https://www.ss64.com/nt/set.html) from=%2
        set (https://www.ss64.com/nt/set.html) to=%3
        set (https://www.ss64.com/nt/set.html) via=%4

        if (https://www.ss64.com/nt/if.html) %n% gtr (https://www.ss64.com/nt/gtr.html) 0 (
                set (https://www.ss64.com/nt/set.html) /a x=!n!-1
                call (https://www.ss64.com/nt/call.html) :move (https://www.ss64.com/nt/move.html) !x! %from% %via% %to%
                echo (https://www.ss64.com/nt/echo.html) Move (https://www.ss64.com/nt/move.html) top disk from pole %from% to pole %t
o%.
                call (https://www.ss64.com/nt/call.html) :move (https://www.ss64.com/nt/move.html) !x! %via% %to% %from%
        )
        exit (https://www.ss64.com/nt/exit.html) /b 0
```

Output:

```
Move top disk from pole START to pole HELPER.
Move top disk from pole START to pole END.
Move top disk from pole HELPER to pole END.
Move top disk from pole START to pole HELPER.
Move top disk from pole END to pole START.
Move top disk from pole END to pole HELPER.
Move top disk from pole START to pole HELPER.
Move top disk from pole START to pole END.
Move top disk from pole HELPER to pole END.
Move top disk from pole HELPER to pole START.
Move top disk from pole END to pole START.
Move top disk from pole HELPER to pole END.
Move top disk from pole START to pole HELPER.
Move top disk from pole START to pole END.
Move top disk from pole HELPER to pole END.

Press any key to continue . . .
```

BBC BASIC (/wiki/Category:BBC_BASIC)

Works with: BBC BASIC for Windows (/wiki/BBC_BASIC_for_Windows)

```
      DIM Disc$(13),Size%(3)
      FOR disc% = 1 TO 13
        Disc$(disc%) = STRING$(disc%," ")+STR$disc%+STRING$(disc%," ")
        IF disc%>=10 Disc$(disc%) = MID$(Disc$(disc%),2)
        Disc$(disc%) = CHR$17+CHR$(128+disc%-(disc%>7))+Disc$(disc%)+CHR$17+CHR$128
      NEXT disc%

      MODE 3
      OFF
      ndiscs% = 13
      FOR n% = ndiscs% TO 1 STEP -1
        PROCput(n%,1)
      NEXT
      INPUT TAB(0,0) "Press Enter to start" dummy$
      PRINT TAB(0,0) SPC(20);
      PROChanoi(ndiscs%,1,2,3)
      VDU 30
      END

      DEF PROChanoi(a%,b%,c%,d%)
      IF a%=0 ENDPROC
      PROChanoi(a%-1,b%,d%,c%)
      PROCtake(a%,b%)
      PROCput(a%,c%)
      PROChanoi(a%-1,d%,c%,b%)
      ENDPROC

      DEF PROCput(disc%,peg%)
      PRINTTAB(13+26*(peg%-1)-disc%,20-Size%(peg%))Disc$(disc%);
      Size%(peg%) = Size%(peg%)+1
      ENDPROC

      DEF PROCtake(disc%,peg%)
      Size%(peg%) = Size%(peg%)-1
      PRINTTAB(13+26*(peg%-1)-disc%,20-Size%(peg%))STRING$(2*disc%+1," ");
      ENDPROC
```

BCPL (/wiki/Category:BCPL)

```
get "libhdr"

let start() be move(4, 1, 2, 3)
and move(n, src, via, dest) be if n > 0 do
$(  move(n-1, src, dest, via)
    writef("Move disk from pole %N to pole %N*N", src, dest);
    move(n-1, via, src, dest)
$)
```

Output:

```
Move disk from pole 1 to pole 2
Move disk from pole 1 to pole 3
Move disk from pole 2 to pole 3
Move disk from pole 1 to pole 2
Move disk from pole 3 to pole 1
Move disk from pole 3 to pole 2
Move disk from pole 1 to pole 2
Move disk from pole 1 to pole 3
Move disk from pole 2 to pole 3
Move disk from pole 2 to pole 1
Move disk from pole 3 to pole 1
Move disk from pole 2 to pole 3
Move disk from pole 1 to pole 2
Move disk from pole 1 to pole 3
Move disk from pole 2 to pole 3
```

Befunge (/wiki/Category:Befunge)

This is loosely based on the Python (/wiki/Towers_of_Hanoi#Python) sample. The number of disks is specified by the first integer on the stack (the initial character 4 in the example below). If you want the program to prompt the user for that value, you can replace the 4 with a & (the read integer command).

```
48*2+1>#v_:!#@_0" ksid evoM">:#,_$:8/:.v
>8v8:<$#<+9-+*2%3\*3/3:,+55.+1%3:$_,#!>#:<
: >/!#^_:0\:8/1-8vv,_$8%:3/1+.>0" gep ot"^
^++3-%3\*2/3:%8\*<>:^:"from peg "0\*8-1<
```

Output:

```
Move disk 1 from peg 1 to peg 2
Move disk 2 from peg 1 to peg 3
Move disk 1 from peg 2 to peg 3
Move disk 3 from peg 1 to peg 2
Move disk 1 from peg 3 to peg 1
Move disk 2 from peg 3 to peg 2
Move disk 1 from peg 1 to peg 2
Move disk 4 from peg 1 to peg 3
Move disk 1 from peg 2 to peg 3
Move disk 2 from peg 2 to peg 1
Move disk 1 from peg 3 to peg 1
Move disk 3 from peg 2 to peg 3
Move disk 1 from peg 1 to peg 2
Move disk 2 from peg 1 to peg 3
Move disk 1 from peg 2 to peg 3
```

Bracmat (/wiki/Category:Bracmat)

```
( ( move
  =   n from to via
    .   !arg:(?n,?from,?to,?via)
      & (   !n:>0
          & move$(!n+-1,!from,!via,!to)
          & out$("Move disk from pole " !from " to pole " !to)
          & move$(!n+-1,!via,!to,!from)
        |
        )
  )
& move$(4,1,2,3)
);
```

Output:

```
Move disk from pole  1  to pole  3
Move disk from pole  1  to pole  2
Move disk from pole  3  to pole  2
Move disk from pole  1  to pole  3
Move disk from pole  2  to pole  1
Move disk from pole  2  to pole  3
Move disk from pole  1  to pole  3
Move disk from pole  1  to pole  2
Move disk from pole  3  to pole  2
Move disk from pole  3  to pole  1
Move disk from pole  2  to pole  1
Move disk from pole  3  to pole  2
Move disk from pole  1  to pole  3
Move disk from pole  1  to pole  2
Move disk from pole  3  to pole  2
```

Brainf*** (/wiki/Category:Brainf***)

```
[
This implementation is recursive and uses
a stack, consisting of frames that are 8
bytes long. The layout is as follows:

Byte   Description
   0   recursion flag
       (the program stops if the flag is
        zero)
   1   the step which is currently
       executed
       4 means a call to
               move(a, c, b, n - 1)
       3 means a call to
               move(a, b, c, 1)
       2 means a call to
               move(b, a, c, n - 1)
       1 prints the source and dest pile
   2   flag to check whether the current
       step has already been done or if
       it still must be executed
   3   the step which will be executed
       in the next loop
   4   the source pile
   5   the helper pile
   6   the destination pile
   7   the number of disks to move

The first stack frame (0 0 0 0 0 0 0 0)
is used to abort the recursion.
]

>>>>>>>>

These are the parameters for the program
(1 4 1 0 'a 'b 'c 5)
+>++++>+>>
>>>>++++++++[<++++++++++++>-]<
[<<<+>+>+>-]<<<+>++>+++>+++++>
<<<<<<<<

[> while (recurse)
  [- if (step gt 0)
    >[-]+< todo = 1
    [- if (step gt 1)
      [- if (step gt 2)
        [- if (step gt 3)
          >>++++< next = 3
          >-< todo = 0
          >>>>>>[>+>+<<-]>[<+>-]> n dup
          -
          [[-] if (sub(n 1) gt 0)
            <+>>>++++> push (1 0 0 4)

            copy and push a
            <<<<<<<<[>>>>>>>>>+>+
            <<<<<<<<<-]>>>>>>>>
            >[<<<<<<<<<+>>>>>>>>>-]< >

            copy and push c
```

```
            <<<<<<<[>>>>>>>+>+
            <<<<<<<-]>>>>>>>
            >[<<<<<<<<+>>>>>>>>-]< >

            copy and push b
            <<<<<<<<<[>>>>>>>>>+>+
            <<<<<<<<<-]>>>>>>>>>
            >[<<<<<<<<<<+>>>>>>>>>>-]< >

            copy n and push sub(n 1)
            <<<<<<<<[>>>>>>>>+>+
            <<<<<<<<<-]>>>>>>>>
            >[<<<<<<<<<+>>>>>>>>>-]< -
            >>
          ]
          <<<<<<<<
        ]
        >[-< if ((step gt 2) and todo)
          >>++<< next = 2
          >>>>>>>
          +>>>+> push 1 0 0 1 a b c 1
          <<<<<<<<[>>>>>>>>+>+
          <<<<<<<<<-]>>>>>>>>
          >[<<<<<<<<<+>>>>>>>>>-]< > a
          <<<<<<<<[>>>>>>>>+>+
          <<<<<<<<<-]>>>>>>>>
          >[<<<<<<<<<+>>>>>>>>>-]< > b
          <<<<<<<<[>>>>>>>>+>+
          <<<<<<<<<-]>>>>>>>>
          >[<<<<<<<<<+>>>>>>>>>-]< > c
          + >>
        >]<
      ]
      >[-< if ((step gt 1) and todo)
        >>>>>>[>+>+<<-]>[<+>-]> n dup
        -
        [[-] if (n sub 1 gt 0)
          <+>>>++++> push (1 0 0 4)

          copy and push b
          <<<<<<<[>>>>>>>+
          <<<<<<<-]>>>>>>>
          >[<<<<<<<<+>>>>>>>>-]< >

          copy and push a
          <<<<<<<<<[>>>>>>>>>+
          <<<<<<<<<-]>>>>>>>>>
          >[<<<<<<<<<<+>>>>>>>>>>-]< >

          copy and push c
          <<<<<<<<[>>>>>>>>+
          <<<<<<<<-]>>>>>>>>
          >[<<<<<<<<<+>>>>>>>>>-]< >

          copy n and push sub(n 1)
          <<<<<<<<[>>>>>>>>+>+
          <<<<<<<<<-]>>>>>>>>
          >[<<<<<<<<<+>>>>>>>>>-]< -
          >>
        ]
        <<<<<<<<
      >]<
    ]
    >[-< if ((step gt 0) and todo)
      >>>>>>
      >++++[<++++++++>-]<
      >>+++++++++[<++++++++++>-]<++++
      >>+++++++++[<++++++++++++>-]<+++++
      >>+++++++++[<++++++++++++>-]<+++
      <<<
      >.+++++++>.++.--.<<.
      >>-.+++++.----.<<.
      >>>.<---.+++.>+++.+.+.<.<<.
      >.>--.+++++.---.++++.
        -------.+++.<<.
      >>>++.-------.-.<<<.
      >+.>>+++++++.---.-----.<<<.
      <<<<.>>>>.
      >>----.>++++++++.<+++++.<<.
      >.>>.---.-----.<<<.
      <<.>>+++++++++++++++.
```

```
      >>>[-]<[-]<[-]<[-]
      ++++++++++++.---.[-]
      <<<<<<<
    >]<
    >>[<<+>>-]<< step = next
  ]
  return with clear stack frame
  <[-]>[-]>[-]>[-]>[-]>[-]>[-]>[-]<<<<<<
  <<<<<<<<
  >>[<<+>>-]<< step = next
  <
]
```

C (/wiki/Category:C)

```
#include <stdio.h>

void move(int n, int from, int via, int to)
{
  if (n > 1) {
    move(n - 1, from, to, via);
    printf (https://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("Move disk from pole %d to pole %d\n", from, to);
    move(n - 1, via, from, to);
  } else {
    printf (https://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("Move disk from pole %d to pole %d\n", from, to);
  }
}
int main()
{
  move(4, 1,2,3);
  return 0;
}
```

Animate it for fun:

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

typedef struct { int *x, n; } tower;
tower *new_tower(int cap)
{
        tower *t = calloc (https://www.opengroup.org/onlinepubs/009695399/functions/calloc.html)(1, sizeof(tower) + sizeof(int) * cap)
;
        t->x = (int*)(t + 1);
        return t;
}

tower *t[3];
int height;

void text(int y, int i, int d, const char *s)
{
        printf (https://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("\033[%d;%dH", height - y + 1, (height + 1) * (2
 * i + 1) - d);
        while (d--) printf (https://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("%s", s);
}

void add_disk(int i, int d)
{
        t[i]->x[t[i]->n++] = d;
        text(t[i]->n, i, d, "==");

        usleep(100000);
        fflush (https://www.opengroup.org/onlinepubs/009695399/functions/fflush.html)(stdout);
}

int remove_disk(int i)
{
        int d = t[i]->x[--t[i]->n];
        text(t[i]->n + 1, i, d, "  ");
        return d;
}

void move(int n, int from, int to, int via)
{
        if (!n) return;

        move(n - 1, from, via, to);
        add_disk(to, remove_disk(from));
        move(n - 1, via, to, from);
}

int main(int c, char *v[])
{
        puts (https://www.opengroup.org/onlinepubs/009695399/functions/puts.html)("\033[H\033[J");

        if (c <= 1 || (height = atoi (https://www.opengroup.org/onlinepubs/009695399/functions/atoi.html)(v[1])) <= 0)
                height = 8;
        for (c = 0; c < 3; c++)  t[c] = new_tower(height);
        for (c = height; c; c--) add_disk(0, c);

        move(height, 0, 2, 1);

        text(1, 0, 1, "\n");
        return 0;
}
```

C# (/wiki/Category:C_sharp)

```c
public  void move(int n, int from, int to, int via) {
   if (n == 1) {
     System.Console.WriteLine("Move disk from pole " + from + " to pole " + to);
   } else {
     move(n - 1, from, via, to);
     move(1, from, to, via);
     move(n - 1, via, to, from);
   }
 }
```

C++ (/wiki/Category:C%2B%2B)

Works with: g++ (/wiki/G%2B%2B)

```
void move(int n, int from, int to, int via) {
  if (n == 1) {
    std::cout << "Move disk from pole " << from << " to pole " << to << std::endl;
  } else {
    move(n - 1, from, via, to);
    move(1, from, to, via);
    move(n - 1, via, to, from);
  }
}
```

Clojure (/wiki/Category:Clojure)

Side-Effecting Solution

```
(defn towers-of-hanoi [n from to via]
  (when (pos? n)
    (towers-of-hanoi (dec n) from via to)
    (printf "Move from %s to %s\n" from to)
    (recur (dec n) via to from)))
```

Lazy Solution

```
(defn towers-of-hanoi [n from to via]
  (when (pos? n)
    (lazy-cat (towers-of-hanoi (dec n) from via to)
              (cons [from '-> to]
                    (towers-of-hanoi (dec n) via to from)))))
```

COBOL (/wiki/Category:COBOL)

Translation of: C
Works with: OpenCOBOL (/wiki/OpenCOBOL) version 2.0

```
       >>SOURCE FREE
IDENTIFICATION DIVISION.
PROGRAM-ID. towers-of-hanoi.

PROCEDURE DIVISION.
    CALL "move-disk" USING 4, 1, 2, 3
    .
END PROGRAM towers-of-hanoi.

IDENTIFICATION DIVISION.
PROGRAM-ID. move-disk RECURSIVE.

DATA DIVISION.
LINKAGE SECTION.
01  n                      PIC 9 USAGE COMP.
01  from-pole              PIC 9 USAGE COMP.
01  to-pole                PIC 9 USAGE COMP.
01  via-pole               PIC 9 USAGE COMP.

PROCEDURE DIVISION USING n, from-pole, to-pole, via-pole.
    IF n > 0
       SUBTRACT 1 FROM n
       CALL "move-disk" USING CONTENT n, from-pole, via-pole, to-pole
       DISPLAY "Move disk from pole " from-pole " to pole " to-pole
       CALL "move-disk" USING CONTENT n, via-pole, to-pole, from-pole
    END-IF
    .
END PROGRAM move-disk.
```

Template:Number of disks also (/mw/index.php?title=Template:Number_of_disks_also&action=edit&redlink=1)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. towers-of-hanoi.

PROCEDURE DIVISION.
    CALL "move-disk" USING 4, 1, 2, 3
    .
END PROGRAM towers-of-hanoi.

IDENTIFICATION DIVISION.
PROGRAM-ID. move-disk RECURSIVE.

DATA DIVISION.
LINKAGE SECTION.
01  n                      PIC 9 USAGE COMP.
01  from-pole              PIC 9 USAGE COMP.
01  to-pole                PIC 9 USAGE COMP.
01  via-pole               PIC 9 USAGE COMP.

PROCEDURE DIVISION USING n, from-pole, to-pole, via-pole.
    IF n > 0
       SUBTRACT 1 FROM n
       CALL "move-disk" USING CONTENT n, from-pole, via-pole, to-pole
       ADD 1 TO n
       DISPLAY "Move disk number "n " from pole " from-pole " to pole " to-pole
       SUBTRACT 1 FROM n
       CALL "move-disk" USING CONTENT n, via-pole, to-pole, from-pole
    END-IF
    .
END PROGRAM move-disk.
```

ANSI-74 solution

Early versions of COBOL did not have recursion. There are no locally-scoped variables and the call of a procedure does not have to use a stack to save return state. Recursion would cause undefined results. It is therefore necessary to use an iterative algorithm. This solution is an adaptation of Kolar's Hanoi Tower algorithm no. 1 (http://hanoitower.mkolar.org/algo.html).

Works with: CIS COBOL (/wiki/CIS_COBOL) version 4.2
Works with: GnuCOBOL (/wiki/GnuCOBOL) version 3.0-rc1.0

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. ITERATIVE-TOWERS-OF-HANOI.
       AUTHOR. SOREN ROUG.
       DATE-WRITTEN. 2019-06-28.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. LINUX.
       OBJECT-COMPUTER. KAYPRO4.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77  NUM-DISKS                   PIC 9 VALUE 4.
       77  N1                          PIC 9 COMP.
       77  N2                          PIC 9 COMP.
       77  FROM-POLE                   PIC 9 COMP.
       77  TO-POLE                     PIC 9 COMP.
       77  VIA-POLE                    PIC 9 COMP.
       77  FP-TMP                      PIC 9 COMP.
       77  TO-TMP                      PIC 9 COMP.
       77  P-TMP                       PIC 9 COMP.
       77  TMP-P                       PIC 9 COMP.
       77  I                           PIC 9 COMP.
       77  DIV                         PIC 9 COMP.
       01  STACKNUMS.
           05  NUMSET OCCURS 3 TIMES.
               10  DNUM                PIC 9 COMP.
       01  GAMESET.
           05  POLES OCCURS 3 TIMES.
               10  STACK OCCURS 10 TIMES.
                   15  POLE            PIC 9 USAGE COMP.

       PROCEDURE DIVISION.
       HANOI.
           DISPLAY "TOWERS OF HANOI PUZZLE WITH ", NUM-DISKS, " DISKS.".
           ADD NUM-DISKS, 1 GIVING N1.
           ADD NUM-DISKS, 2 GIVING N2.
           MOVE 1 TO DNUM (1).
```

```
           MOVE N1 TO DNUM (2), DNUM (3).
           MOVE N1 TO POLE (1, N1), POLE (2, N1), POLE (3, N1).
           MOVE 1 TO POLE (1, N2).
           MOVE 2 TO POLE (2, N2).
           MOVE 3 TO POLE (3, N2).
           MOVE 1 TO I.
           PERFORM INIT-PUZZLE UNTIL I = N1.
           MOVE 1 TO FROM-POLE.
           DIVIDE 2 INTO NUM-DISKS GIVING DIV.
           MULTIPLY 2 BY DIV.
           IF DIV NOT = NUM-DISKS PERFORM INITODD ELSE PERFORM INITEVEN.
           PERFORM MOVE-DISK UNTIL DNUM (3) NOT > 1.
           DISPLAY "TOWERS OF HANOI PUZZLE COMPLETED!".
           STOP RUN.
       INIT-PUZZLE.
           MOVE I TO POLE (1, I).
           MOVE 0 TO POLE (2, I), POLE (3, I).
           ADD 1 TO I.
       INITEVEN.
           MOVE 2 TO TO-POLE.
           MOVE 3 TO VIA-POLE.
       INITODD.
           MOVE 3 TO TO-POLE.
           MOVE 2 TO VIA-POLE.
       MOVE-DISK.
           MOVE DNUM (FROM-POLE) TO FP-TMP.
           MOVE POLE (FROM-POLE, FP-TMP) TO I.
           DISPLAY "MOVE DISK FROM ", POLE (FROM-POLE, N2),
               " TO ", POLE (TO-POLE, N2).
           ADD 1 TO DNUM (FROM-POLE).
           MOVE VIA-POLE TO TMP-P.
           SUBTRACT 1 FROM DNUM (TO-POLE).
           MOVE DNUM (TO-POLE) TO TO-TMP.
           MOVE I TO POLE (TO-POLE, TO-TMP).
           DIVIDE 2 INTO I GIVING DIV.
           MULTIPLY 2 BY DIV.
           IF I NOT = DIV PERFORM MOVE-TO-VIA ELSE
               PERFORM MOVE-FROM-VIA.
       MOVE-TO-VIA.
           MOVE TO-POLE TO VIA-POLE.
           MOVE DNUM (FROM-POLE) TO FP-TMP.
           MOVE DNUM (TMP-P) TO P-TMP.
           IF POLE (FROM-POLE, FP-TMP) > POLE (TMP-P, P-TMP)
               PERFORM MOVE-FROM-TO
           ELSE MOVE TMP-P TO TO-POLE.
       MOVE-FROM-TO.
           MOVE FROM-POLE TO TO-POLE.
           MOVE TMP-P TO FROM-POLE.
           MOVE DNUM (FROM-POLE) TO FP-TMP.
           MOVE DNUM (TMP-P) TO P-TMP.
       MOVE-FROM-VIA.
           MOVE FROM-POLE TO VIA-POLE.
           MOVE TMP-P TO FROM-POLE.
```

CoffeeScript (/wiki/Category:CoffeeScript)

```
hanoi = (ndisks, start_peg=1, end_peg=3) ->
  if ndisks
    staging_peg = 1 + 2 + 3 - start_peg - end_peg
    hanoi(ndisks-1, start_peg, staging_peg)
    console.log "Move disk #{ndisks} from peg #{start_peg} to #{end_peg}"
    hanoi(ndisks-1, staging_peg, end_peg)

hanoi(4)
```

Common Lisp (/wiki/Category:Common_Lisp)

```
(defun move (n from to via)
  (cond ((= n 1)
         (format t "Move from ~A to ~A.~%" from to))
        (t
         (move (- n 1) from via to)
         (format t "Move from ~A to ~A.~%" from to)
         (move (- n 1) via to from))))
```

D (/wiki/Category:D)

Recursive Version

```
import std.stdio;

void hanoi(in int n, in char from, in char to, in char via) {
    if (n > 0) {
        hanoi(n - 1, from, via, to);
        writefln("Move disk %d from %s to %s", n, from, to);
        hanoi(n - 1, via, to, from);
    }
}

void main() {
    hanoi(3, 'L', 'M', 'R');
}
```

Output:

```
Move disk 1 from L to M
Move disk 2 from L to R
Move disk 1 from M to R
Move disk 3 from L to M
Move disk 1 from R to L
Move disk 2 from R to M
Move disk 1 from L to M
```

Fast Iterative Version

See: The shortest and "mysterious" TH algorithm (http://hanoitower.mkolar.org/shortestTHalgo.html)

```
// Code found and then improved by Glenn C. Rhoads,
// then some more by M. Kolar (2000).
void main(in string[] args) {
    import core.stdc.stdio, std.conv, std.typetuple;

    immutable size_t n = (args.length > 1) ? args[1].to!size_t : 3;
    size_t[3] p = [(1 << n) - 1, 0, 0];

    // Show the start configuration of the pegs.
    '|'.putchar;
    foreach_reverse (immutable i; 1 .. n + 1)
        printf(" %d", i);
    "\n|\n|".puts;

    foreach (immutable size_t x; 1 .. (1 << n)) {
        {
            immutable size_t i1 = x & (x - 1);
            immutable size_t fr = (i1 + i1 / 3) & 3;
            immutable size_t i2 = (x | (x - 1)) + 1;
            immutable size_t to = (i2 + i2 / 3) & 3;

            size_t j = 1;
            for (size_t w = x; !(w & 1); w >>= 1, j <<= 1) {}

            // Now j is not the number of the disk to move,
            // it contains the single bit to be moved:
            p[fr] &= ~j;
            p[to] |= j;
        }

        // Show the current configuration of pegs.
        foreach (immutable size_t k; TypeTuple!(0, 1, 2)) {
            "\n|".printf;
            size_t j = 1 << n;
            foreach_reverse (immutable size_t w; 1 .. n + 1) {
                j >>= 1;
                if (j & p[k])
                    printf(" %zd", w);
            }
        }
        '\n'.putchar;
    }
}
```

Output:

```
| 3 2 1
|
|

| 3 2
|
| 1

| 3
| 2
| 1

| 3
| 2 1
|

|
| 2 1
| 3

| 1
| 2
| 3

| 1
|
| 3 2

|
|
| 3 2 1
```

Dart (/wiki/Category:Dart)

```
main() {
  moveit(from,to) {
    print("move ${from} ---> ${to}");
  }

  hanoi(height,toPole,fromPole,usePole) {
    if (height>0) {
      hanoi(height-1,usePole,fromPole,toPole);
      moveit(fromPole,toPole);
      hanoi(height-1,toPole,usePole,fromPole);
    }
  }

  hanoi(3,3,1,2);
}
```

The same as above, with optional static type annotations and styled according to http://www.dartlang.org/articles/style-guide/ (https://www.dartlang.org/articles/style-guide

```
main() {
  String say(String from, String to) => "$from ---> $to";

  hanoi(int height, int toPole, int fromPole, int usePole) {
    if (height > 0) {
      hanoi(height - 1, usePole, fromPole, toPole);
      print(say(fromPole.toString(), toPole.toString()));
      hanoi(height - 1, toPole, usePole, fromPole);
    }
  }

  hanoi(3, 3, 1, 2);
}
```

Output:

```
move 1 ---> 3
move 1 ---> 2
move 3 ---> 2
move 1 ---> 3
move 2 ---> 1
move 2 ---> 3
move 1 ---> 3
```

Dc (/wiki/Category:Dc)

From Here (http://se.aminet.net/pub/OpenBSD/src/regress/usr.bin/dc/t20.in)

```
[ # move(from, to)
   n           # print from
   [ --> ]n    # print " --> "
   p           # print to\n
   sw          # p doesn't pop, so get rid of the value
]sm

[ # init(n)
   sw          # tuck n away temporarily
   9           # sentinel as bottom of stack
   lw          # bring n back
   1           # "from" tower's label
   3           # "to" tower's label
   0           # processed marker
]si

[ # Move()
   lt          # push to
   lf          # push from
   lmx         # call move(from, to)
]sM

[ # code block <d>
   ln          # push n
   lf          # push from
   lt          # push to
   1           # push processed marker 1
   ln          # push n
   1           # push 1
   -           # n - 1
   lf          # push from
   ll          # push left
   0           # push processed marker 0
]sd

[ # code block <e>
   ln          # push n
   1           # push 1
   -           # n - 1
   ll          # push left
   lt          # push to
   0           # push processed marker 0
]se

[ # code block <x>
   ln 1 =M
   ln 1 !=d
]sx

[ # code block <y>
   lMx
   lex
]sy

[ # quit()
   q           # exit the program
]sq

[ # run()
   d 9 =q      # if stack empty, quit()
   sp          # processed
   st          # to
   sf          # from
   sn          # n
   6           #
   lf          #
   -           #
   lt          #
   -           # 6 - from - to
   sl          #
   lp 0 =x     #
   lp 0 !=y    #
   lrx         # loop
]sr

5lix # init(n)
lrx # run()
```

Delphi (/wiki/Category:Delphi)

See Pascal (https://rosettacode.org/wiki/Towers_of_Hanoi#Pascal).

Dyalect (/wiki/Category:Dyalect)

Translation of: Swift

```
func hanoi(n, a, b, c) {
    if n > 0 {
        hanoi(n - 1, a, c, b)
        print("Move disk from \(a) to \(c)")
        hanoi(n - 1, b, a, c)
    }
}

hanoi(4, "A", "B", "C")
```

Output:

```
Move disk from A to B
Move disk from A to C
Move disk from B to C
Move disk from A to B
Move disk from C to A
Move disk from C to B
Move disk from A to B
Move disk from A to C
Move disk from B to C
Move disk from B to A
Move disk from C to A
Move disk from B to C
Move disk from A to B
Move disk from A to C
Move disk from B to C
```

E (/wiki/Category:E)

```
def (http://wiki.erights.org/wiki/def) move(out, n, fromPeg, toPeg, viaPeg) {
    if (http://wiki.erights.org/wiki/if) (n.aboveZero()) {
        move(out, n.previous(), fromPeg, viaPeg, toPeg)
        out.println (http://wiki.erights.org/wiki/println)(`Move disk $n from $fromPeg to $toPeg.`)
        move(out, n.previous(), viaPeg, toPeg, fromPeg)
    }
}

move(stdout (http://wiki.erights.org/wiki/stdout), 4, def (http://wiki.erights.org/wiki/def) left {}, def (http://wiki.erights.org/wik
i/def) right {}, def (http://wiki.erights.org/wiki/def) middle {})
```

EasyLang (/wiki/Category:EasyLang)

```
func hanoi n src dst aux . .
  if n >= 1
    call hanoi n - 1 src aux dst
    print "Move " & src & " to " & dst
    call hanoi n - 1 aux dst src
  .
.
call hanoi 5 1 2 3
```

Eiffel (/wiki/Category:Eiffel)

```
class
	APPLICATION

create
	make

feature {NONE (https://www.google.com/search?q=site%3Ahttp%3A%2F%2Fdocs.eiffel.com%2Feiffelstudio%2Flibraries+none&btnI=I%27m+Feeling+Lucky)} -- Initialization

	make
		do
			move (4, "A", "B", "C")
		end

feature -- Towers of Hanoi

	move (n: INTEGER (https://www.google.com/search?q=site%3Ahttp%3A%2F%2Fdocs.eiffel.com%2Feiffelstudio%2Flibraries+integer&btnI=I%27m+Feeling+Lucky); frm, to, via: STRING (https://www.google.com/search?q=site%3Ahttp%3A%2F%2Fdocs.eiffel.com%2Feiffelstudio%2Flibraries+string&btnI=I%27m+Feeling+Lucky))
		require
			n > 0
	    do
			if n = 1 then
			print ("Move disk from pole " + frm + " to pole " + to + "%N")
		    else
			move (n - 1, frm, via, to)
			move (1, frm, to, via)
			move (n - 1, via, to, frm)
		    end
	    end
end
```

Ela (/wiki/Category:Ela)

Translation of: Haskell

```
open monad io
:::IO

//Functional approach
hanoi 0 _ _ _ = []
hanoi n a b c = hanoi (n - 1) a c b ++ [(a,b)] ++ hanoi (n - 1) c b a

hanoiIO n = mapM_ f $ hanoi n 1 2 3 where
  f (x,y) = putStrLn $ "Move " ++ show x ++ " to " ++ show y

//Imperative approach using IO monad
hanoiM n = hanoiM' n 1 2 3 where
  hanoiM' 0 _ _ _ = return ()
  hanoiM' n a b c = do
    hanoiM' (n - 1) a c b
    putStrLn $ "Move " ++ show a ++ " to " ++ show b
    hanoiM' (n - 1) c b a
```

Elena (/wiki/Category:Elena)

ELENA 4.x :

```
move = (n,from,to,via)
{
    if (n == 1)
    {
        console.printLine("Move disk from pole ",from," to pole ",to)
    }
    else
    {
        move(n-1,from,via,to);
        move(1,from,to,via);
        move(n-1,via,to,from)
    }
};
```

Elixir (/wiki/Category:Elixir)

```
defmodule RC do
  def hanoi(n) when 0<n and n<10, do: hanoi(n, 1, 2, 3)

  defp hanoi(1, f, _, t), do: move(f, t)
  defp hanoi(n, f, u, t) do
    hanoi(n-1, f, t, u)
    move(f, t)
    hanoi(n-1, u, f, t)
  end

  defp move(f, t), do: IO.puts "Move disk from #{f} to #{t}"
end

RC.hanoi(3)
```

Output:

```
Move disk from 1 to 3
Move disk from 1 to 2
Move disk from 3 to 2
Move disk from 1 to 3
Move disk from 2 to 1
Move disk from 2 to 3
Move disk from 1 to 3
```

Emacs Lisp (/wiki/Category:Emacs_Lisp)

Translation of: Common Lisp

```
(defun move (n from to via)
  (cond ((= n 1)
         (print (format "Move from %S to %S" from to)))
        (t
         (progn
           (move (- n 1) from via to)
           (print (format "Move from %S to %S" from to))
           (move (- n 1) via to from)))))
```

Erlang (/wiki/Category:Erlang)

```
move(1, F, T, _V) ->
  io (http://erlang.org/doc/man/io.html):format("Move from ~p to ~p~n", [F, T]);
move(N, F, T, V) ->
  move(N-1, F, V, T),
  move(1  , F, T, V),
  move(N-1, V, T, F).
```

ERRE (/wiki/Category:ERRE)

```
!------------------------------------------------------------
! HANOI.R : solve tower of Hanoi puzzle using a recursive
! modified algorithm.
!------------------------------------------------------------

PROGRAM HANOI

!$INTEGER

!VAR I,J,MOSSE,NUMBER

PROCEDURE PRINTMOVE
  LOCAL SOURCE$,DEST$
  MOSSE=MOSSE+1
  CASE I OF
     1-> SOURCE$="Left" END ->
     2-> SOURCE$="Center" END ->
     3-> SOURCE$="Right" END ->
  END CASE
  CASE J OF
     1-> DEST$="Left" END ->
     2-> DEST$="Center" END ->
     3-> DEST$="Right" END ->
  END CASE
  PRINT("I move a disk from ";SOURCE$;" to ";DEST$)
END PROCEDURE

PROCEDURE MOVE
  IF NUMBER<>0 THEN
     NUMBER=NUMBER-1
     J=6-I-J
     MOVE
     J=6-I-J
     PRINTMOVE
     I=6-I-J
     MOVE
     I=6-I-J
     NUMBER=NUMBER+1
  END IF
END PROCEDURE

BEGIN
  MAXNUM=12
  MOSSE=0
  PRINT(CHR$(12);TAB(25);"--- TOWERS OF HANOI ---")
  REPEAT
     PRINT("Number of disks ";)
     INPUT(NUMBER)
  UNTIL NUMBER>1 AND NUMBER<=MAXNUM
  PRINT
  PRINT("For ";NUMBER;"disks the total number of moves is";2^NUMBER-1)
  I=1  ! number of source pole
  J=3  ! number of destination pole
  MOVE
END PROGRAM
```

Output:

```
                       --- TOWER OF HANOI ---
Number of disks ? 3

For  3 disks the total number of moves is 7
I move a disk from Left to Right
I move a disk from Left to Center
I move a disk from Right to Center
I move a disk from Left to Right
I move a disk from Center to Left
I move a disk from Center to Right
I move a disk from Left to Right
```

Excel (/wiki/Category:Excel)

LAMBDA

With the names HANOI and SHOWHANOI bound to the following lambdas in the Excel worksheet Name Manager:

(See LAMBDA: The ultimate Excel worksheet function (https://www.microsoft.com/en-us/research/blog/lambda-the-ultimatae-excel-worksheet-function/))

Works with: Office 365 Betas 2021 (/mw/index.php?title=Office_365_Betas_2021&action=edit&redlink=1)

```
SHOWHANOI
=LAMBDA(n,
    FILTERP(
        LAMBDA(x, "" <> x)
    )(
        HANOI(n)("left")("right")("mid")
    )
)


HANOI
=LAMBDA(n,
    LAMBDA(l,
        LAMBDA(r,
            LAMBDA(m,
                IF(0 = n,
                    "",
                    LET(
                        next, n - 1,
                        APPEND(
                            APPEND(
                                HANOI(next)(l)(m)(r)
                            )(
                                CONCAT(l, " -> ", r)
                            )
                        )(
                            HANOI(next)(m)(r)(l)
                        )
                    )
                )
            )
        )
    )
)
```

And assuming that these generic lambdas are also bound to the following names in Name Manager:

```
APPEND
=LAMBDA(xs,
    LAMBDA(ys,
        LET(
            nx, ROWS(xs),
            rowIndexes, SEQUENCE(nx + ROWS(ys)),
            colIndexes, SEQUENCE(
                1,
                MAX(COLUMNS(xs), COLUMNS(ys))
            ),
            IF(
                rowIndexes <= nx,
                INDEX(xs, rowIndexes, colIndexes),
                INDEX(ys, rowIndexes - nx, colIndexes)
            )
        )
    )
)


FILTERP
=LAMBDA(p,
    LAMBDA(xs,
        FILTER(xs, p(xs))
    )
)
```

In the output below, the expression in B2 defines an array of strings which additionally populate the following cells.

Output:

	fx	**=SHOWHANOI(A2)**
	A	B
1	**Disks**	**Steps**
2	**3**	left -> right
3		left -> mid
4		right -> mid

5		left -> right
6		mid -> left
7		mid -> right
8		left -> right

Ezhil (/wiki/Category:Ezhil)

```
# (C) 2013 Ezhil Language Project
# Tower of Hanoi – recursive solution

நிரல்பாகம் ஹோனாய்(வட்டுகள், முதல்அச்சு, இறுதிஅச்சு,வட்டு)

  @(வட்டுகள் == 1 ) ஆனால்
     பதிப்பி  "வட்டு " + str(வட்டு) + "ஐ \t  (" + str(முதல்அச்சு) + "  -> " +  str(இறுதிஅச்சு)+ ") அச்சிற்கு நகர்த்துக."
  இல்லை

  @( ["இ", "அ",  "ஆ"]  இல் அச்சு ) ஒவ்வொன்றாக
          @( (முதல்அச்சு != அச்சு)  && (இறுதிஅச்சு  != அச்சு) ) ஆனால்
              நடு = அச்சு
          முடி
  முடி

    # solve problem for n-1 again between src and temp pegs
    ஹோனாய்(வட்டுகள்-1,   முதல்அச்சு,நடு,வட்டுகள்-1)

    # move largest disk from src to destination
    ஹோனாய்(1, முதல்அச்சு, இறுதிஅச்சு,வட்டுகள்)

    # solve problem for n-1 again between different pegs
    ஹோனாய்(வட்டுகள்-1, நடு, இறுதிஅச்சு,வட்டுகள்-1)
  முடி
முடி

ஹோனாய்(4,"அ","ஆ",0)
```

F# (/wiki/Category:F_Sharp)

```
#light
let rec hanoi num start finish =
  match num with
  | 0 -> [ ]
  | _ -> let temp = (6 - start - finish)
         (hanoi (num-1) start temp) @ [ start, finish ] @ (hanoi (num-1) temp finish)

[<EntryPoint>]
let main args =
  (hanoi 4 1 2) |> List (http://research.microsoft.com/en-us/um/cambridge/projects/fsharp/manual/namespaces.html).iter (fun pair -> ma
tch pair with
                                          | a, b -> printf "Move disc from %A to %A\n" a b)
  0
```

Factor (/wiki/Category:Factor)

```
USING: formatting kernel locals math ;
IN: rosettacode.hanoi

: move ( from to -- )
    "%d->%d\n" printf ;
:: hanoi ( n from to other -- )
    n 0 > [
        n 1 - from other to hanoi
        from to move
        n 1 - other to from hanoi
    ] when ;
```

In the REPL:

```
( scratchpad ) 3 1 3 2 hanoi
1->3
1->2
3->2
1->3
2->1
2->3
1->3
```

FALSE (/wiki/Category:FALSE)

```
["Move disk from "$!\" to "$!\"
"]p:  { to from }
[n;0>[n;1-n: @\ h;! @\ p;! \@ h;! \@ n;1+n:]?]h:  { via to from }
4n:["right"]["middle"]["left"]h;!%%%
```

Fermat (/wiki/Category:Fermat)

```
Func Hanoi( n, f, t, v ) =
if n = 0 then
    !'';
else
    Hanoi(n - 1, f, v, t);
    !f;!' -> ';!t;!',   ';
    Hanoi(n - 1, v, t, f)
fi.
```

Output:

```
1 -> 3,   1 -> 2,   3 -> 2,   1 -> 3,   2 -> 1,   2 -> 3,   1 -> 3,   1 -> 2,   3 -> 2,   3 -> 1,   2 -> 1,   3 -> 2,   1 -> 3,   1 ->
 2,   3 -> 2,
```

FOCAL (/wiki/Category:FOCAL)

```
01.10 S N=4;S S=1;S V=2;S T=3
01.20 D 2
01.30 Q

02.02 S N(D)=N(D)-1;I (N(D)),2.2,2.04
02.04 S D=D+1
02.06 S N(D)=N(D-1);S S(D)=S(D-1)
02.08 S T(D)=V(D-1);S V(D)=T(D-1)
02.10 D 2
02.12 S D=D-1
02.14 D 3
02.16 S A=S(D);S S(D)=V(D);S V(D)=A
02.18 G 2.02
02.20 D 3

03.10 T %1,"MOVE DISK FROM POLE",S(D)
03.20 T " TO POLE",T(D),!
```

Output:

```
MOVE DISK FROM POLE= 1 TO POLE= 2
MOVE DISK FROM POLE= 1 TO POLE= 3
MOVE DISK FROM POLE= 2 TO POLE= 3
MOVE DISK FROM POLE= 1 TO POLE= 2
MOVE DISK FROM POLE= 3 TO POLE= 1
MOVE DISK FROM POLE= 3 TO POLE= 2
MOVE DISK FROM POLE= 1 TO POLE= 2
MOVE DISK FROM POLE= 1 TO POLE= 3
MOVE DISK FROM POLE= 2 TO POLE= 3
MOVE DISK FROM POLE= 2 TO POLE= 1
MOVE DISK FROM POLE= 3 TO POLE= 1
MOVE DISK FROM POLE= 2 TO POLE= 3
MOVE DISK FROM POLE= 1 TO POLE= 2
MOVE DISK FROM POLE= 1 TO POLE= 3
MOVE DISK FROM POLE= 2 TO POLE= 3
```

Forth (/wiki/Category:Forth)

With locals:

```
CREATE peg1 ," left "
CREATE peg2 ," middle "
CREATE peg3 ," right "

: .$   COUNT TYPE ;
: MOVE-DISK
  LOCALS| via to from n |
  n 1 =
  IF   CR ." Move disk from " from .$ ." to " to .$
  ELSE n 1- from via to RECURSE
       1    from to via RECURSE
       n 1- via to from RECURSE
  THEN ;
```

Without locals, executable pegs:

```
: left   ." left" ;
: right  ." right" ;
: middle ." middle" ;

: move-disk ( v t f n -- v t f )
  dup 0= if drop exit then
  1-       >R
  rot swap R@ ( t v f n-1 ) recurse
  rot swap
  2dup cr ." Move disk from " execute ."  to " execute
  swap rot R> ( f t v n-1 ) recurse
  swap rot ;
: hanoi ( n -- )
  1 max >R ['] right ['] middle ['] left R> move-disk drop drop drop ;
```

Fortran (/wiki/Category:Fortran)

Works with: Fortran (/wiki/Fortran) version 90 and later

```
PROGRAM TOWER

  CALL Move(4, 1, 2, 3)

CONTAINS

  RECURSIVE SUBROUTINE Move(ndisks, from, to, via)
    INTEGER, INTENT (IN) :: ndisks, from, to, via

    IF (ndisks == 1) THEN
       WRITE(*, "(A,I1,A,I1)") "Move disk from pole ", from, " to pole ", to
    ELSE
       CALL Move(ndisks-1, from, via, to)
       CALL Move(1, from, to, via)
       CALL Move(ndisks-1, via, to, from)
    END IF
  END SUBROUTINE Move

END PROGRAM TOWER
```

Template:More informative version (/mw/index.php?title=Template:More_informative_version&action=edit&redlink=1)

```
PROGRAM TOWER2

  CALL Move(4, 1, 2, 3)

CONTAINS

  RECURSIVE SUBROUTINE Move(ndisks, from, via, to)
    INTEGER, INTENT (IN) :: ndisks, from, via, to

    IF (ndisks > 1) THEN
       CALL Move(ndisks-1, from, to, via)
       WRITE(*, "(A,I1,A,I1,A,I1)") "Move disk ", ndisks, "  from pole ", from, " to pole ", to
       Call Move(ndisks-1,via,from,to)
    ELSE
       WRITE(*, "(A,I1,A,I1,A,I1)") "Move disk ", ndisks, "  from pole ", from, " to pole ", to
    END IF
  END SUBROUTINE Move

END PROGRAM TOWER2
```

FreeBASIC (/wiki/Category:FreeBASIC)

```
' FB 1.05.0 Win64

Sub move(n As Integer, from As Integer, to_ As Integer, via As Integer)
  If n > 0 Then
    move(n - 1, from, via, to_)
    Print "Move disk"; n; " from pole"; from; " to pole"; to_
    move(n - 1, via, to_, from)
  End If
End Sub

Print "Three disks" : Print
move 3, 1, 2, 3
Print
Print "Four disks" : Print
move 4, 1, 2, 3
Print "Press any key to quit"
Sleep
```

Output:

```
Three disks

Move disk 1 from pole 1 to pole 2
Move disk 2 from pole 1 to pole 3
Move disk 1 from pole 2 to pole 3
Move disk 3 from pole 1 to pole 2
Move disk 1 from pole 3 to pole 1
Move disk 2 from pole 3 to pole 2
Move disk 1 from pole 1 to pole 2

Four disks

Move disk 1 from pole 1 to pole 3
Move disk 2 from pole 1 to pole 2
Move disk 1 from pole 3 to pole 2
Move disk 3 from pole 1 to pole 3
Move disk 1 from pole 2 to pole 1
Move disk 2 from pole 2 to pole 3
Move disk 1 from pole 1 to pole 3
Move disk 4 from pole 1 to pole 2
Move disk 1 from pole 3 to pole 2
Move disk 2 from pole 3 to pole 1
Move disk 1 from pole 2 to pole 1
Move disk 3 from pole 3 to pole 2
Move disk 1 from pole 1 to pole 3
Move disk 2 from pole 1 to pole 2
Move disk 1 from pole 3 to pole 2
```

Frink (/wiki/Category:Frink)

```
/** Set up the recursive call for n disks */
hanoi[n] := hanoi[n, 1, 3, 2]

/** The recursive call. */
hanoi[n, source, target, aux] :=
{
   if n > 0
   {
      hanoi[n-1, source, aux, target]
      println["Move from $source to $target"]
      hanoi[n-1, aux, target, source]
   }
}

hanoi[7]
```

FutureBasic (/wiki/Category:FutureBasic)

```
include "ConsoleWindow"

void local fn move( n as long, fromPeg as long, toPeg as long, viaPeg as long )
if n > 0
   fn move( n-1, fromPeg, viaPeg, toPeg )
   print "Move disk from "; fromPeg; " to "; toPeg
   fn move( n-1, viaPeg, toPeg, fromPeg )
end if
end fn

fn move( 4, 1, 2, 3 )
print
print "Towers of Hanoi puzzle solved."
end
```

Output:

```
Move disk from  1 to  3
Move disk from  1 to  2
Move disk from  3 to  2
Move disk from  1 to  3
Move disk from  2 to  1
Move disk from  2 to  3
Move disk from  1 to  3
Move disk from  1 to  2
Move disk from  3 to  2
Move disk from  3 to  1
Move disk from  2 to  1
Move disk from  3 to  2
Move disk from  1 to  3
Move disk from  1 to  2
Move disk from  3 to  2

Towers of Hanoi puzzle solved.
```

Fōrmulæ (/wiki/Category:F%C5%8Drmul%C3%A6)

Fōrmulæ programs are not textual, visualization/edition of programs is done showing/manipulating structures but not text. Moreover, there can be multiple visual representations of the same program. Even though it is possible to have textual representation —i.e. XML, JSON— they are intended for storage and transfer purposes more than visualization and edition.

Programs in Fōrmulæ are created/edited online in its website (https://formulae.org), However they run on execution servers. By default remote servers are used, but they are limited in memory and processing power, since they are intended for demonstration and casual use. A local server can be downloaded and installed, it has no limitations (it runs in your own computer). Because of that, example programs can be fully visualized and edited, but some of them will not run if they require a moderate or heavy computation/memory resources, and no local server is being used.

In **this (https://formulae.org/?example=Tower_of_Hanoi)** page you can see the program(s) related to this task and their results.

GAP (/wiki/Category:GAP)

```
Hanoi := function(n)
        local move;
        move := function(n, a, b, c) # from, through, to
                if n = 1 then
                        Print(a, " -> ", c, "\n");
                else
                        move(n - 1, a, c, b);
                        move(1, a, b, c);
                        move(n - 1, b, a, c);
                fi;
        end;
        move(n, "A", "B", "C");
end;

Hanoi(1);
# A -> C

Hanoi(2);
# A -> B
# A -> C
# B -> C

Hanoi(3);
# A -> C
# A -> B
# C -> B
# A -> C
# B -> A
# B -> C
# A -> C
```

Go (/wiki/Category:Go)

```
package main

import "fmt"

// a towers of hanoi solver just has one method, play
type solver interface {
    play(int)
}

func main() {
    var t solver    // declare variable of solver type
    t = new(towers) // type towers must satisfy solver interface
    t.play(4)
}

// towers is example of type satisfying solver interface
type towers struct {
    // an empty struct.  some other solver might fill this with some
    // data representation, maybe for algorithm validation, or maybe for
    // visualization.
}

// play is sole method required to implement solver type
func (t *towers) play(n int) {
    // drive recursive solution, per task description
    t.moveN(n, 1, 2, 3)
}

// recursive algorithm
func (t *towers) moveN(n, from, to, via int) {
    if n > 0 {
        t.moveN(n-1, from, via, to)
        t.move1(from, to)
        t.moveN(n-1, via, to, from)
    }
}

// example function prints actions to screen.
// enhance with validation or visualization as needed.
func (t *towers) move1(from, to int) {
    fmt.Println("move disk from rod", from, "to rod", to)
}
```

In other words:

```
package main

import "fmt"

func main() {
        move(3, "A", "B", "C")
}

func move(n uint64, a, b, c string) {
        if n > 0 {
                move(n-1, a, c, b)
                fmt.Println("Move disk from " + a + " to " + c)
                move(n-1, b, a, c)
        }
}
```

Groovy (/wiki/Category:Groovy)

Unlike most solutions here this solution manipulates more-or-less actual stacks of more-or-less actual rings.

```
def (https://www.google.de/search?q=site%3Agroovy.codehaus.org/%20def) tail = { list, n ->  def (https://www.google.de/search?q=site%3Agroovy.codehaus.org/%20def) m = list.size (https://www.google.de/search?q=site%3Agroovy.codehaus.org/%20size)(); list.subList([m - n, 0].max (https://www.google.de/search?q=site%3Agroovy.codehaus.org/%20max)(),m) }

final (https://www.google.de/search?q=site%3Agroovy.codehaus.org/%20final) STACK = [A:[],B:[],C:[]].asImmutable (https://www.google.de/search?q=site%3Agroovy.codehaus.org/%20asImmutable)()

def (https://www.google.de/search?q=site%3Agroovy.codehaus.org/%20def) report = { it -> }
def (https://www.google.de/search?q=site%3Agroovy.codehaus.org/%20def) check = { it -> }

def (https://www.google.de/search?q=site%3Agroovy.codehaus.org/%20def) moveRing = { from, to ->  to << from.pop (https://www.google.de/search?q=site%3Agroovy.codehaus.org/%20pop)(); report(); check(to) }

def (https://www.google.de/search?q=site%3Agroovy.codehaus.org/%20def) moveStack
moveStack = { from, to, using = STACK.values().find (https://www.google.de/search?q=site%3Agroovy.codehaus.org/%20find) { !(it.is(from) || it.is(to)) } ->
    if (https://www.google.de/search?q=site%3Agroovy.codehaus.org/%20if) (!from) return (https://www.google.de/search?q=site%3Agroovy.codehaus.org/%20return)
    def (https://www.google.de/search?q=site%3Agroovy.codehaus.org/%20def) n = from.size (https://www.google.de/search?q=site%3Agroovy.codehaus.org/%20size)()
    moveStack(tail(from, n-1), using, to)
    moveRing(from, to)
    moveStack(tail(using, n-1), to, from)
}
```

Test program:

```
enum (https://www.google.de/search?q=site%3Agroovy.codehaus.org/%20enum) Ring {
    S('°'), M('o'), L('O'), XL('( )');
    private (https://www.google.de/search?q=site%3Agroovy.codehaus.org/%20private) sym
    private (https://www.google.de/search?q=site%3Agroovy.codehaus.org/%20private) Ring(sym) { this (https://www.google.de/search?q=site%3Agroovy.codehaus.org/%20this).sym=sym }
    String (https://www.google.de/search?as_q=String&num=100&hl=en&as_occt=url&as_sitesearch=java.sun.com%2Fj2se%2F1%2E5%2E0%2Fdocs%2Fapi%2F) toString() { sym }
}

report = { STACK.each (https://www.google.de/search?q=site%3Agroovy.codehaus.org/%20each) { k, v ->  println (https://www.google.de/search?q=site%3Agroovy.codehaus.org/%20println) "${k}: ${v}" }; println (https://www.google.de/search?q=site%3Agroovy.codehaus.org/%20println)() }
check = { to -> assert (https://www.google.de/search?q=site%3Agroovy.codehaus.org/%20assert) to == ([] + to).sort (https://www.google.de/search?q=site%3Agroovy.codehaus.org/%20sort)().reverse (https://www.google.de/search?q=site%3Agroovy.codehaus.org/%20reverse)() }

Ring.values().reverseEach (https://www.google.de/search?q=site%3Agroovy.codehaus.org/%20reverseEach) { STACK.A << it }
report()
check(STACK.A)
moveStack(STACK.A, STACK.C)
```

Output:

```
A: [( ), O, o, °]
B: []
C: []

A: [( ), O, o]
B: [°]
C: []

A: [( ), O]
B: [°]
C: [o]
```

Haskell (/wiki/Category:Haskell)

Most of the programs on this page use an imperative approach (i.e., print out movements as side effects during program execution). Haskell favors a purely functional approach, where you would for example return a (lazy) list of movements from a to b via c:

```
hanoi :: Integer (https://haskell.org/ghc/docs/latest/html/libraries/base/Prelude.html#t:Integer) -> a -> a -> a -> [(a, a)]
hanoi 0 _ _ _ = []
hanoi n a b c = hanoi (n-1) a c b ++ [(a,b)] ++ hanoi (n-1) c b a
```

You can also do the above with one tail-recursion call:

```
hanoi :: Integer (https://haskell.org/ghc/docs/latest/html/libraries/base/Prelude.html#t:Integer) -> a -> a -> a -> [(a, a)]

hanoi n a b c = hanoiToList n a b c []
  where
    hanoiToList 0 _ _ _ l = l
    hanoiToList n a b c l = hanoiToList (n-1) a c b ((a, b) : hanoiToList (n-1) c b a l)
```

One can use this function to produce output, just like the other programs:

```
hanoiIO n = mapM_ (https://haskell.org/ghc/docs/latest/html/libraries/base/Prelude.html#v:mapM_) f $ hanoi n 1 2 3 where
  f (x,y) = putStrLn (https://haskell.org/ghc/docs/latest/html/libraries/base/Prelude.html#v:putStrLn) $ "Move " ++ show (https://hask
ell.org/ghc/docs/latest/html/libraries/base/Prelude.html#v:show) x ++ " to " ++ show (https://haskell.org/ghc/docs/latest/html/librari
es/base/Prelude.html#v:show) y
```

or, instead, one can of course also program imperatively, using the IO monad directly:

```
hanoiM :: Integer (https://haskell.org/ghc/docs/latest/html/libraries/base/Prelude.html#t:Integer) -> IO (https://haskell.org/ghc/docs
/latest/html/libraries/base/Prelude.html#t:IO) ()
hanoiM n = hanoiM' n 1 2 3 where
  hanoiM' 0 _ _ _ = return (https://haskell.org/ghc/docs/latest/html/libraries/base/Prelude.html#v:return) ()
  hanoiM' n a b c = do
    hanoiM' (n-1) a c b
    putStrLn (https://haskell.org/ghc/docs/latest/html/libraries/base/Prelude.html#v:putStrLn) $ "Move " ++ show (https://haskell.org/
ghc/docs/latest/html/libraries/base/Prelude.html#v:show) a ++ " to " ++ show (https://haskell.org/ghc/docs/latest/html/libraries/base/
Prelude.html#v:show) b
    hanoiM' (n-1) c b a
```

or, defining it as a monoid, and adding some output:

```
-------------------------- HANOI --------------------------

hanoi ::
  Int (https://haskell.org/ghc/docs/latest/html/libraries/base/Prelude.html#t:Int) ->
  String (https://haskell.org/ghc/docs/latest/html/libraries/base/Prelude.html#t:String) ->
  String (https://haskell.org/ghc/docs/latest/html/libraries/base/Prelude.html#t:String) ->
  String (https://haskell.org/ghc/docs/latest/html/libraries/base/Prelude.html#t:String) ->
  [(String (https://haskell.org/ghc/docs/latest/html/libraries/base/Prelude.html#t:String), String (https://haskell.org/ghc/docs/lates
t/html/libraries/base/Prelude.html#t:String))]
hanoi 0 _ _ _ = mempty
hanoi n l r m =
  hanoi (n - 1) l m r
    <> [(l, r)]
    <> hanoi (n - 1) m r l

--------------------------- TEST --------------------------
main :: IO (https://haskell.org/ghc/docs/latest/html/libraries/base/Prelude.html#t:IO) ()
main = putStrLn (https://haskell.org/ghc/docs/latest/html/libraries/base/Prelude.html#v:putStrLn) $ showHanoi 5

------------------------- DISPLAY -------------------------
showHanoi :: Int (https://haskell.org/ghc/docs/latest/html/libraries/base/Prelude.html#t:Int) -> String (https://haskell.org/ghc/docs/
latest/html/libraries/base/Prelude.html#t:String)
showHanoi n =
  unlines (https://haskell.org/ghc/docs/latest/html/libraries/base/Prelude.html#v:unlines) $
    fmap (https://haskell.org/ghc/docs/latest/html/libraries/base/Prelude.html#v:fmap)
      ( \(from, to) ->
          concat (https://haskell.org/ghc/docs/latest/html/libraries/base/Prelude.html#v:concat) [justifyRight 5 ' ' from, " -> ", to]
      )
      (hanoi n "left" "right" "mid")

justifyRight :: Int (https://haskell.org/ghc/docs/latest/html/libraries/base/Prelude.html#t:Int) -> Char (https://haskell.org/ghc/docs
/latest/html/libraries/base/Prelude.html#t:Char) -> String (https://haskell.org/ghc/docs/latest/html/libraries/base/Prelude.html#t:Str
ing) -> String (https://haskell.org/ghc/docs/latest/html/libraries/base/Prelude.html#t:String)
justifyRight n c = (drop (https://haskell.org/ghc/docs/latest/html/libraries/base/Prelude.html#v:drop) . length (https://haskell.org/g
hc/docs/latest/html/libraries/base/Prelude.html#v:length)) <*> (replicate n c <>)
```

Output:

```
 left -> right
 left -> mid
right -> mid
 left -> right
  mid -> left
  mid -> right
 left -> right
 left -> mid
right -> mid
right -> left
  mid -> left
right -> mid
 left -> right
 left -> mid
right -> mid
 left -> right
  mid -> left
  mid -> right
 left -> right
  mid -> left
right -> mid
right -> left
  mid -> left
  mid -> right
 left -> right
 left -> mid
right -> mid
 left -> right
  mid -> left
  mid -> right
 left -> right
```

HolyC (/wiki/Category:HolyC)

Translation of: C

```
U0 Move(U8 n, U8 from, U8 to, U8 via) {
  if (n > 0) {
    Move(n - 1, from, via, to);
    Print("Move disk from pole %d to pole %d\n", from, to);
    Move(n - 1, via, to, from);
  }
}

Move(4, 1, 2, 3);
```

Icon (/wiki/Category:Icon) and Unicon (/wiki/Category:Unicon)

The following is based on a solution in the Unicon book.

```
procedure main(arglist)
hanoi(arglist[1]) | stop("Usage: hanoi n\n\rWhere n is the number of disks to move.")
end

#procedure hanoi(n:integer, needle1:1, needle2:2)   # unicon shorthand for icon code 1,2,3 below

procedure hanoi(n, needle1, needle2)   #: solve towers of hanoi by moving  n disks from needle 1 to needle2 via other
local other

n := integer(0 < n) | runerr(n,101)       # 1 ensure integer (this also ensures it's positive too)
/needle1 := 1                             # 2 default
/needle2 := 2                             # 3 default

if n = 1 then
   write("Move disk from ", needle1, " to ", needle2)
else {
   other := 6 - needle1 - needle2         # clever but somewhat un-iconish way to find other
   hanoi(n-1, needle1, other)
   write("Move disk from ", needle1, " to ", needle2)
   hanoi(n-1, other, needle2)
}
return
end
```

Inform 7 (/wiki/Category:Inform_7)

```
Hanoi is a room.

A post is a kind of supporter. A post is always fixed in place.

The left post, the middle post, and the right post are posts in Hanoi.

A disk is a kind of supporter.
The red disk is a disk on the left post.
The orange disk is a disk on the red disk.
The yellow disk is a disk on the orange disk.
The green disk is a disk on the yellow disk.

Definition: a disk is topmost if nothing is on it.

When play begins:
        move 4 disks from the left post to the right post via the middle post.

To move (N - number) disk/disks from (FP - post) to (TP - post) via (VP - post):
        if N > 0:
                move N - 1 disks from FP to VP via TP;
                say "Moving a disk from [FP] to [TP]...";
                let D be a random topmost disk enclosed by FP;
                if a topmost disk (called TD) is enclosed by TP, now D is on TD;
                otherwise now D is on TP;
                move N - 1 disks from VP to TP via FP.
```

Io (/wiki/Category:Io)

```
hanoi := method(n, from, to, via,
  if (n == 1) then (
    writeln("Move from ", from, " to ", to)
  ) else (
    hanoi(n - 1, from, via, to  )
    hanoi(1    , from, to , via )
    hanoi(n - 1, via , to , from)
  )
)
```

Ioke (/wiki/Category:Ioke)

```
 = method(n, f, u, t,
  if(n < 2,
    "#{f} --> #{t}" println,

    H(n - 1, f, t, u)
    "#{f} --> #{t}" println
    H(n - 1, u, f, t)
  )
)

hanoi = method(n,
  H(n, 1, 2, 3)
)
```

IS-BASIC (/wiki/Category:IS-BASIC)

```
100 PROGRAM "Hanoi.bas"
110 CALL HANOI(4,1,3,2)
120 DEF HANOI(DISK,FRO,TO,WITH)
130   IF DISK>0 THEN
140     CALL HANOI(DISK-1,FRO,WITH,TO)
150     PRINT "Move disk";DISK;"from";FRO;"to";TO
160     CALL HANOI(DISK-1,WITH,TO,FRO)
170   END IF
180 END DEF
```

J (/wiki/Category:J)

Solutions

```
H =: i.@,&2 ` (({&0 2 1,0 2,{&1 0 2)@$:@<:) @. *    NB. tacit using anonymous recursion
```

Example use:

```
   H 3
0 2
0 1
2 1
0 2
1 2
1 0
2 0
```

The result is a 2-column table; a row `i,j` is interpreted as: move a disk (the top disk) from peg `i` to peg `j`. Or, using explicit rather than implicit code:

```
H1=: monad define                                NB. explicit equivalent of H
  if. y do.
    ({&0 2 1 , 0 2 , {&1 0 2) H1 y-1
  else.
    i.0 2
  end.
)
```

The usage here is the same:

```
   H1 2
0 1
0 2
1 2
```

Alternative solution

If a textual display is desired, similar to some of the other solutions here (counting from 1 instead of 0, tracking which disk is on the top of the stack, and of course formattin the result for a human reader instead of providing a numeric result):

```
hanoi=: monad define
  moves=. H y
  disks=.  $~` ((],[,]) $:@<:) @.* y
  ('move disk ';' from peg ';' to peg ');@,."1 ":&.>disks,.1+moves
)
```

Demonstration:

```
   hanoi 3
move disk 1 from peg 1 to peg 3
move disk 2 from peg 1 to peg 2
move disk 1 from peg 3 to peg 2
move disk 3 from peg 1 to peg 3
move disk 1 from peg 2 to peg 1
move disk 2 from peg 2 to peg 3
move disk 1 from peg 1 to peg 3
```

Java (/wiki/Category:Java)

```
public void move(int n, int from, int to, int via) {
  if (n == 1) {
    System (https://www.google.com/search?hl=en&q=allinurl%3Asystem+java.sun.com&btnI=I%27m%20Feeling%20Lucky).out.println("Move disk
from pole " + from + " to pole " + to);
  } else {
    move(n - 1, from, via, to);
    move(1, from, to, via);
    move(n - 1, via, to, from);
  }
}
```

JavaScript (/wiki/Category:JavaScript)

ES5

```
function move(n, a, b, c) {
  if (n > 0) {
    move(n-1, a, c, b);
    console.log("Move disk from " + a + " to " + c);
    move(n-1, b, a, c);
  }
}
move(4, "A", "B", "C");
```

Or, as a functional expression, rather than a statement with side effects:

```
(function () {

    // hanoi :: Int -> String -> String -> String -> [[String, String]]
    function hanoi(n, a, b, c) {
        return n ? hanoi(n - 1, a, c, b)
            .concat([
                [a, b]
            ])
            .concat(hanoi(n - 1, c, b, a)) : [];
    }

    return hanoi(3, 'left', 'right', 'mid')
        .map(function (d) {
            return d[0] + ' -> ' + d[1];
        });
})();
```

Output:

```
["left -> right", "left -> mid",
 "right -> mid", "left -> right",
 "mid -> left", "mid -> right",
 "left -> right"]
```

ES6

```
(() => {
    'use strict';

    // hanoi :: Int -> String -> String -> String -> [[String, String]]
    const hanoi = (n, a, b, c) =>
        n ? hanoi(n - 1, a, c, b)
        .concat([
            [a, b]
        ])
        .concat(hanoi(n - 1, c, b, a)) : [];

    // show :: a -> String
    const show = x => JSON.stringify(x, null, 2);

    return show(
        hanoi(3, 'left', 'right', 'mid')
        .map(d => d[0] + ' -> ' + d[1])
    );
})();
```

Output:

```
[
  "left -> right",
  "left -> mid",
  "right -> mid",
  "left -> right",
  "mid -> left",
  "mid -> right",
  "left -> right"
]
```

Joy (/wiki/Category:Joy)

From here (http://www.latrobe.edu.au/phimvt/joy/jp-nestrec.html)

```
DEFINE hanoi == [[rolldown] infra] dip
                [ [ [null] [pop pop] ]
                  [ [dup2 [[rotate] infra] dip pred]
                    [ [dup rest put] dip
                      [[swap] infra] dip pred ]
                    [] ] ]
                condnestrec.
```

Using it (5 is the number of disks.)

```
[source destination temp] 5 hanoi.
```

jq (/wiki/Category:Jq)

Works with: jq (/wiki/Jq) version 1.4
The algorithm used here is used elsewhere on this page but it is worthwhile pointing out that it can also be read as a proof that:

(a) move(n;"A";"B";"C") will logically succeed for n>=0; and

(b) move(n;"A";"B";"C") will generate the sequence of moves, assuming sufficient computing resources.

The proof of (a) is by induction:

- As explained in the comments, the algorithm establishes that move(n;x;y;z) is possible for all n>=0 and distinct x,y,z if move(n-1;x;y;z)) is possible;
- Since move(0;x;y;z) evidently succeeds, (a) is established by induction.

The truth of (b) follows from the fact that the algorithm emits an instruction of what to do when moving a single disk.

```
# n is the number of disks to move from From to To
def move(n; From; To; Via):
  if n > 0 then
     # move all but the largest at From to Via (according to the rules):
     move(n-1; From; Via; To),
     # ... so the largest disk at From is now free to move to its final destination:
     "Move disk from \(From) to \(To)",
     # Move the remaining disks at Via to To:
     move(n-1; Via; To; From)
  else empty
  end;
```

Example:

```
move(5; "A"; "B"; "C")
```

Jsish (/wiki/Category:Jsish)

From Javascript ES5 entry.

```
/* Towers of Hanoi, in Jsish */

function move(n, a, b, c) {
  if (n > 0) {
    move(n-1, a, c, b);
    puts("Move disk from " + a + " to " + c);
    move(n-1, b, a, c);
  }
}

if (Interp.conf('unitTest')) move(4, "A", "B", "C");

/*
=!EXPECTSTART!=
Move disk from A to B
Move disk from A to C
Move disk from B to C
Move disk from A to B
Move disk from C to A
Move disk from C to B
Move disk from A to B
Move disk from A to C
Move disk from B to C
Move disk from B to A
Move disk from C to A
Move disk from B to C
Move disk from A to B
Move disk from A to C
Move disk from B to C
=!EXPECTEND!=
*/
```

Output:

```
prompt$ jsish -u towersOfHanoi.jsi
[PASS] towersOfHanoi.jsi
```

Julia (/wiki/Category:Julia)

Translation of: R

```
function solve(n::Integer, from::Integer, to::Integer, via::Integer)
  if n == 1
    println("Move disk from $from to $to")
  else
    solve(n - 1, from, via, to)
    solve(1, from, to, via)
    solve(n - 1, via, to, from)
  end
end

solve(4, 1, 2, 3)
```

Output:

```
Move disk from 1 to 3
Move disk from 1 to 2
Move disk from 3 to 2
Move disk from 1 to 3
Move disk from 2 to 1
Move disk from 2 to 3
Move disk from 1 to 3
Move disk from 1 to 2
Move disk from 3 to 2
Move disk from 3 to 1
Move disk from 2 to 1
Move disk from 3 to 2
Move disk from 1 to 3
Move disk from 1 to 2
Move disk from 3 to 2
```

K (/wiki/Category:K)

```
   h:{[n;a;b;c]if[n>0;_f[n-1;a;c;b];`0:,//$($n,":",$a,"->",$b,"\n");_f[n-1;c;b;a]]}
   h[4;1;2;3]
1:1->3
2:1->2
1:3->2
3:1->3
1:2->1
2:2->3
1:1->3
4:1->2
1:3->2
2:3->1
1:2->1
3:3->2
1:1->3
2:1->2
1:3->2
```

The disk to move in the i'th step is the same as the position of the leftmost 1 in the binary representation of 1..2^n.

```
   s:();{[n;a;b;c]if[n>0;_f[n-1;a;c;b];s,:n;_f[n-1;c;b;a]]}[4;1;2;3];s
1 2 1 3 1 2 1 4 1 2 1 3 1 2 1

   1_{*1+&|x}'a:(2_vs!_2^4)
1 2 1 3 1 2 1 4 1 2 1 3 1 2 1
```

Klingphix (/wiki/Category:Klingphix)

Translation of: MiniScript

```
include ..\Utilitys.tlhy

:moveDisc %B !B %C !C %A !A %n !n { n A C B }
    $n [
        $n 1 - $A $B $C moveDisc
        ( "Move disc " $n " from pole " $A " to pole " $C ) lprint nl
        $n 1 - $B $C $A moveDisc
    ] if
;

{ Move disc 3 from pole 1 to pole 3, with pole 2 as spare }
3 1 3 2 moveDisc

" " input
```

Output:

```
Move disc 1 from pole 1 to pole 3
Move disc 2 from pole 1 to pole 2
Move disc 1 from pole 3 to pole 2
Move disc 3 from pole 1 to pole 3
Move disc 1 from pole 2 to pole 1
Move disc 2 from pole 2 to pole 3
Move disc 1 from pole 1 to pole 3
```

Kotlin (/wiki/Category:Kotlin)

```
// version 1.1.0

class (https://scala-lang.org) Hanoi(disks: Int) {
    private (https://scala-lang.org) var (https://scala-lang.org) moves = 0

    init {
        println("Towers of Hanoi with $disks disks:\n")
        move(disks, 'L', 'C', 'R')
        println("\nCompleted in $moves moves\n")
    }

    private (https://scala-lang.org) fun move(n: Int, from: Char, to: Char, via: Char) {
        if (https://scala-lang.org) (n > 0) {
            move(n - 1, from, via, to)
            moves++
            println("Move disk $n from $from to $to")
            move(n - 1, via, to, from)
        }
    }
}

fun main(args: Array<String>) {
    Hanoi(3)
    Hanoi(4)
}
```

Output:

```
Towers of Hanoi with 3 disks:

Move disk 1 from L to C
Move disk 2 from L to R
Move disk 1 from C to R
Move disk 3 from L to C
Move disk 1 from R to L
Move disk 2 from R to C
Move disk 1 from L to C

Completed in 7 moves

Towers of Hanoi with 4 disks:

Move disk 1 from L to R
Move disk 2 from L to C
Move disk 1 from R to C
Move disk 3 from L to R
Move disk 1 from C to L
Move disk 2 from C to R
Move disk 1 from L to R
Move disk 4 from L to C
Move disk 1 from R to C
Move disk 2 from R to L
Move disk 1 from C to L
Move disk 3 from R to C
Move disk 1 from L to R
Move disk 2 from L to C
Move disk 1 from R to C

Completed in 15 moves
```

lambdatalk (/wiki/Category:Lambdatalk)

(Following NewLisp, PicoLisp, Racket, Scheme)

```
{def move
 {lambda {:n :from :to :via}
  {if {<= :n 0}
   then >
   else {move {- :n 1} :from :via :to}
         move disk :n from :from to :to {br}
        {move {- :n 1} :via :to :from} }}}
-> move
{move 4 A B C}
> move disk 1 from A to C
> move disk 2 from A to B
> move disk 1 from C to B
> move disk 3 from A to C
> move disk 1 from B to A
> move disk 2 from B to C
> move disk 1 from A to C
> move disk 4 from A to B
> move disk 1 from C to B
> move disk 2 from C to A
> move disk 1 from B to A
> move disk 3 from C to B
> move disk 1 from A to C
> move disk 2 from A to B
> move disk 1 from C to B
```

Lasso (/wiki/Category:Lasso)

```
#!/usr/bin/lasso9

define towermove(
        disks::integer,
        a,b,c
) => {
        if(#disks > 0) => {
                towermove(#disks - 1, #a, #c, #b )
                stdoutnl("Move disk from " + #a + " to " + #c)
                towermove(#disks - 1, #b, #a, #c )
        }
}

towermove((integer($argv -> second || 3)), "A", "B", "C")
```

Called from command line:

```
./towers
```

Output:

```
Move disk from A to C
Move disk from A to B
Move disk from C to B
Move disk from A to C
Move disk from B to A
Move disk from B to C
Move disk from A to C
```

Called from command line:

```
./towers 4
```

Output:

```
Move disk from A to B
Move disk from A to C
Move disk from B to C
Move disk from A to B
Move disk from C to A
Move disk from C to B
Move disk from A to B
Move disk from A to C
Move disk from B to C
Move disk from B to A
Move disk from C to A
Move disk from B to C
Move disk from A to B
Move disk from A to C
Move disk from B to C
```

Liberty BASIC (/wiki/Category:Liberty_BASIC)

This looks much better with a GUI interface.

```
   source$ ="A"
    via$    ="B"
    target$ ="C"

    call hanoi 4, source$, target$, via$        '   ie call procedure to move legally 4 disks from peg A to peg C via peg B

    wait

    sub hanoi numDisks, source$, target$, via$
        if numDisks =0 then
            exit sub
        else
            call hanoi numDisks -1, source$, via$, target$
            print " Move disk "; numDisks; " from peg "; source$; " to peg "; target$
            call hanoi numDisks -1, via$, target$, source$
        end if
    end sub

    end
```

Lingo (/wiki/Category:Lingo)

```
on hanoi (n, a, b, c)
  if n > 0 then
    hanoi(n-1, a, c, b)
    put "Move disk from" && a && "to" && c
    hanoi(n-1, b, a, c)
  end if
end
```

```
hanoi(3, "A", "B", "C")
-- "Move disk from A to C"
-- "Move disk from A to B"
-- "Move disk from C to B"
-- "Move disk from A to C"
-- "Move disk from B to A"
-- "Move disk from B to C"
-- "Move disk from A to C"
```

Logo (/wiki/Category:Logo)

```
to move :n :from :to :via
  if :n = 0 [stop]
  move :n-1 :from :via :to
  (print [Move disk from] :from [to] :to)
  move :n-1 :via :to :from
end
move 4 "left "middle "right
```

Logtalk (/wiki/Category:Logtalk)

```
:- object(hanoi).

    :- public(run/1).
    :- mode(run(+integer), one).
    :- info(run/1, [
        comment is 'Solves the towers of Hanoi problem for the specified number of disks.',
        argnames is ['Disks']]).

    run(Disks) :-
        move(Disks, left, middle, right).

    move(1, Left, _, Right):-
        !,
        report(Left, Right).
    move(Disks, Left, Aux, Right):-
        Disks2 is Disks - 1,
        move(Disks2, Left, Right, Aux),
        report(Left, Right),
        move(Disks2, Aux, Left, Right).

    report(Pole1, Pole2):-
        write('Move a disk from '),
        writeq(Pole1),
        write(' to '),
        writeq(Pole2),
        write('.'),
        nl.

:- end_object.
```

LOLCODE (/wiki/Category:LOLCODE)

```
HAI

HOW DUZ I HANOI YR N AN YR SRC AN YR DST AN YR VIA
    BTW VISIBLE SMOOSH "HANOI N=" N " SRC=" SRC " DST=" DST " VIA=" VIA MKAY
    BOTH SAEM N AN 0, O RLY?
        YA RLY
            BTW VISIBLE "Done."
            GTFO
        NO WAI
            I HAS A LOWER ITZ DIFF OF N AN 1
            HANOI DST VIA SRC LOWER
            VISIBLE SMOOSH "Move disc " N " from " SRC ...
            " to " DST MKAY
            HANOI SRC DST VIA LOWER
    OIC
IF U SAY SO

HANOI 2 3 1 4 BTW requires reversed arguments?

KTHXBYE
```

Lua (/wiki/Category:Lua)

```
function move(n, src, dst, via)
    if n > 0 then
        move(n - 1, src, via, dst)
        print(src, 'to', dst)
        move(n - 1, via, dst, src)
    end
end

move(4, 1, 2, 3)
```

Template:More informative version (/mw/index.php?title=Template:More_informative_version&action=edit&redlink=1)

```
function move(n, src, via, dst)
    if n > 0 then
        move(n - 1, src, dst, via)
        print('Disk ',n,' from ' ,src, 'to', dst)
        move(n - 1, via, src, dst)

    end
end

move(4, 1, 2, 3)
```

Hanoi Iterative

```
#!/usr/bin/env luajit
local function printf(fmt, ...) io.write(string.format(fmt, ...)) end
local runs=0
local function move(tower, from, to)
        if #tower[from]==0
                or (#tower[to]>0
                and tower[from][#tower[from]]>tower[to][#tower[to]]) then
                        to,from=from,to
        end
        if #tower[from]>0 then
                tower[to][#tower[to]+1]=tower[from][#tower[from]]
                tower[from][#tower[from]]=nil

                io.write(tower[to][#tower[to]],":",from, "→", to, " ")
        end
end

local function hanoi(n)
        local src,dst,via={},{},{}
        local tower={src,dst,via}
        for i=1,n do src[i]=n-i+1 end
        local one,nxt,lst
        if n%2==1 then -- odd
                one,nxt,lst=1,2,3
        else
                one,nxt,lst=1,3,2
        end
        --repeat
        ::loop::
                move(tower, one, nxt)
                if #dst==n then return end
                move(tower, one, lst)
                one,nxt,lst=nxt,lst,one
        goto loop
        --until false
end

local num=arg[1] and tonumber(arg[1]) or 4

hanoi(num)
```

Output:

```
> ./hanoi_iter.lua 5
1:1→2 2:1→3 1:2→3 3:1→2 1:3→1 2:3→2 1:1→2 4:1→3 1:2→3 2:2→1 1:3→1 3:2→3 1:1→2 2:1→3 1:2→3 5:1→2 1:3→1 2:3→2 1:1→2 3:3→1 1:2→3 2:2→1 1:
3→1 4:3→2 1:1→2 2:1→3 1:2→3 3:1→2 1:3→1 2:3→2 1:1→2
```

Hanoi Bitwise Fast

```
#!/usr/bin/env luajit
-- binary solution
local bit=require"bit"
local band,bor=bit.band,bit.bor
local function hanoi(n)
        local even=(n-1)%2
        for m=1,2^n-1 do
                io.write(m,":",band(m,m-1)%3+1, "→", (bor(m,m-1)+1)%3+1, " ")
        end
end

local num=arg[1] and tonumber(arg[1]) or 4

hanoi(num)
```

Output:

```
> ./hanoi_bit.lua 4
1:1→3 2:1→2 3:3→2 4:1→3 5:2→1 6:2→3 7:1→3 8:1→2 9:3→2 10:3→1 11:2→1 12:3→2 13:1→3 14:1→2 15:3→2
> time ./hanoi_bit.lua 30 >/dev/null  ; on AMD FX-8350 @ 4 GHz
./hanoi_bit.lua 30 > /dev/null  297,40s user 1,39s system 99% cpu 4:59,01 total
```

M2000 Interpreter (/wiki/Category:M2000_Interpreter)

Translation of: FreeBasic

```
Module Hanoi {
      Rem HANOI TOWERS
      Print "Three disks" : Print
      move(3, 1, 2, 3)
      Print
      Print "Four disks" : Print
      move(4, 1, 2, 3)


      Sub move(n, from, to, via)
            If n <=0 Then Exit Sub
            move(n - 1, from, via, to)
            Print "Move disk"; n; " from pole"; from; " to pole"; to
            move(n - 1, via, to, from)
      End Sub
}
Hanoi
```

Output:

same as in FreeBasic

MAD (/wiki/Category:MAD)

```
            NORMAL MODE IS INTEGER
            DIMENSION LIST(100)
            SET LIST TO LIST

            VECTOR VALUES MOVFMT =
           0  $20HMOVE DISK FROM POLE ,I1,S1,8HTO POLE ,I1*$

            INTERNAL FUNCTION(DUMMY)
            ENTRY TO MOVE.
LOOP        NUM = NUM - 1
            WHENEVER NUM.E.0
                PRINT FORMAT MOVFMT,FROM,DEST
            OTHERWISE
                SAVE RETURN
                SAVE DATA NUM,FROM,VIA,DEST
                TEMP=DEST
                DEST=VIA
                VIA=TEMP
                MOVE.(0)
                RESTORE DATA NUM,FROM,VIA,DEST
                RESTORE RETURN
                PRINT FORMAT MOVFMT,FROM,DEST
                TEMP=FROM
                FROM=VIA
                VIA=TEMP
                TRANSFER TO LOOP
            END OF CONDITIONAL
            FUNCTION RETURN
            END OF FUNCTION

            NUM  = 4
            FROM = 1
            VIA  = 2
            DEST = 3
            MOVE.(0)

            END OF PROGRAM
```

Output:

```
MOVE DISK FROM POLE 1 TO POLE 2
MOVE DISK FROM POLE 1 TO POLE 3
MOVE DISK FROM POLE 2 TO POLE 3
MOVE DISK FROM POLE 1 TO POLE 2
MOVE DISK FROM POLE 3 TO POLE 1
MOVE DISK FROM POLE 3 TO POLE 2
MOVE DISK FROM POLE 1 TO POLE 2
MOVE DISK FROM POLE 1 TO POLE 3
MOVE DISK FROM POLE 2 TO POLE 3
MOVE DISK FROM POLE 2 TO POLE 1
MOVE DISK FROM POLE 3 TO POLE 1
MOVE DISK FROM POLE 2 TO POLE 3
MOVE DISK FROM POLE 1 TO POLE 2
MOVE DISK FROM POLE 1 TO POLE 3
MOVE DISK FROM POLE 2 TO POLE 3
```

Maple (/wiki/Category:Maple)

```
Hanoi := proc(n::posint,a,b,c)
   if n = 1 then
       printf("Move disk from tower %a to tower %a.\n",a,c);
   else
       Hanoi(n-1,a,c,b);
       Hanoi(1,a,b,c);
       Hanoi(n-1,b,a,c);
    fi;
end:

printf("Moving 2 disks from tower A to tower C using tower B.\n");
Hanoi(2,A,B,C);
```

Output:

Moving 2 disks from tower A to tower C using tower B.

Move disk from tower A to tower B.

Move disk from tower A to tower C.

Move disk from tower B to tower C.

Mathematica (/wiki/Category:Mathematica)

```
Hanoi[0, from_, to_, via_] := Null
Hanoi[n_Integer, from_, to_, via_] :=
  (Hanoi[n-1, from, via, to];
   Print["Move disk from pole ", from, " to ", to, "."];
   Hanoi[n-1, via, to, from])
```

MATLAB (/wiki/Category:MATLAB)

This is a direct translation from the Python example given in the Wikipedia entry for the Tower of Hanoi puzzle.

```
function towerOfHanoi(n,A,C,B)
    if (n~=0)
        towerOfHanoi(n-1,A,B,C);
        disp (https://www.mathworks.com/access/helpdesk/help/techdoc/ref/disp.html)(sprintf (https://www.mathworks.com/access/helpdesk
/help/techdoc/ref/sprintf.html)('Move plate %d from tower %d to tower %d',[n A C]));
        towerOfHanoi(n-1,B,C,A);
    end
end
```

Sample output:

```
towerOfHanoi(3,1,3,2)
Move plate 1 from tower 1 to tower 3
Move plate 2 from tower 1 to tower 2
Move plate 1 from tower 3 to tower 2
Move plate 3 from tower 1 to tower 3
Move plate 1 from tower 2 to tower 1
Move plate 2 from tower 2 to tower 3
Move plate 1 from tower 1 to tower 3
```

MiniScript (/wiki/Category:MiniScript)

```
moveDisc = function(n, A, C, B)
    if n == 0 then return
    moveDisc n-1, A, B, C
    print "Move disc " + n + " from pole " + A + " to pole " + C
    moveDisc n-1, B, C, A
end function

// Move disc 3 from pole 1 to pole 3, with pole 2 as spare
moveDisc 3, 1, 3, 2
```

Output:

```
Move disc 1 from pole 1 to pole 3
Move disc 2 from pole 1 to pole 2
Move disc 1 from pole 3 to pole 2
Move disc 3 from pole 1 to pole 3
Move disc 1 from pole 2 to pole 1
Move disc 2 from pole 2 to pole 3
Move disc 1 from pole 1 to pole 3
```

MIPS Assembly (/wiki/Category:MIPS_Assembly)

```

# Towers of Hanoi
# MIPS assembly implementation (tested with MARS)
# Source: https://stackoverflow.com/questions/50382420/hanoi-towers-recursive-solution-using-mips/50383530#50383530

.data
prompt: .asciiz "Enter a number: "
part1: .asciiz "\nMove disk "
part2: .asciiz " from rod "
part3: .asciiz " to rod "
```

```
.text
.globl main
main:
    li $v0,  4          # print string
    la $a0,  prompt
    syscall
    li $v0,  5          # read integer
    syscall

    # parameters for the routine
    add $a0, $v0, $zero # move to $a0
    li $a1, 'A'
    li $a2, 'B'
    li $a3, 'C'

    jal hanoi           # call hanoi routine

    li $v0, 10          # exit
    syscall

hanoi:

    #save in stack
    addi $sp, $sp, -20
    sw   $ra, 0($sp)
    sw   $s0, 4($sp)
    sw   $s1, 8($sp)
    sw   $s2, 12($sp)
    sw   $s3, 16($sp)

    add $s0, $a0, $zero
    add $s1, $a1, $zero
    add $s2, $a2, $zero
    add $s3, $a3, $zero

    addi $t1, $zero, 1
    beq $s0, $t1, output

    recur1:

        addi $a0, $s0, -1
        add $a1, $s1, $zero
        add $a2, $s3, $zero
        add $a3, $s2, $zero
        jal hanoi

        j output

    recur2:

        addi $a0, $s0, -1
        add $a1, $s3, $zero
        add $a2, $s2, $zero
        add $a3, $s1, $zero
        jal hanoi

    exithanoi:

        lw   $ra, 0($sp)        # restore registers from stack
        lw   $s0, 4($sp)
        lw   $s1, 8($sp)
        lw   $s2, 12($sp)
        lw   $s3, 16($sp)

        addi $sp, $sp, 20       # restore stack pointer

        jr $ra

    output:

        li $v0,  4              # print string
        la $a0,  part1
        syscall
        li $v0,  1              # print integer
        add $a0, $s0, $zero
        syscall
        li $v0,  4              # print string
        la $a0,  part2
        syscall
        li $v0,  11             # print character
```

```
        add $a0, $s1, $zero
        syscall
        li $v0,  4              # print string
        la $a0,  part3
        syscall
        li $v0,  11             # print character
        add $a0, $s2, $zero
        syscall

        beq $s0, $t1, exithanoi
        j recur2
```

MK-61/52 (/wiki/Category:%D0%9C%D0%9A-61/52)

```
^       2       x^y     П0      <->     2       /       {x}     x#0     16
3       П3      2       П2      БП      20      3       П2      2       П3
1       П1      ПП      25      КППВ    ПП      28      КППА    ПП      31
КППВ    ПП      34      КППА    ИП1     ИП3     КППС    ИП1     ИП2     КППС
ИП3     ИП2     КППС    ИП1     ИП3     КППС    ИП2     ИП1     КППС    ИП2
ИП3     КППС    ИП1     ИП3     КППС    В/О     ИП1     ИП2     БП      62
ИП2     ИП1     КППС    ИП1     ИП2     ИП3     П1      ->      П3      ->
П2      В/О     1       0       /       +       С/П     КИП0    ИП0     x=0
89      3       3       1       ИНВ     ^       ВП      2       С/П     В/О
```

Instruction: PA = 56; PB = 60; PC = 72; N В/О С/П, where 2 <= N <= 7.

Modula-2 (/wiki/Category:Modula-2)

```
MODULE Towers;
FROM FormatString IMPORT FormatString;
FROM Terminal IMPORT WriteString,ReadChar;

PROCEDURE Move(n,from,to,via : INTEGER);
VAR buf : ARRAY[0..63] OF CHAR;
BEGIN
    IF n>0 THEN
        Move(n-1, from, via, to);
        FormatString("Move disk %i from pole %i to pole %i\n", buf, n, from, to);
        WriteString(buf);
        Move(n-1, via, to, from)
    END
END Move;

BEGIN
    Move(3, 1, 3, 2);

    ReadChar
END Towers.
```

Modula-3 (/wiki/Category:Modula-3)

```
MODULE Hanoi EXPORTS Main;

FROM IO IMPORT Put;
FROM Fmt IMPORT Int;

PROCEDURE doHanoi(n, from, to, using: INTEGER) =
  BEGIN
    IF n > 0 THEN
      doHanoi(n - 1, from, using, to);
      Put("move " & Int(from) & " --> " & Int(to) & "\n");
      doHanoi(n - 1, using, to, from);
    END;
  END doHanoi;

BEGIN
  doHanoi(4, 1, 2, 3);
END Hanoi.
```

Monte (/wiki/Category:Monte)

```
def move(n, fromPeg, toPeg, viaPeg):
    if (n > 0):
        move(n.previous(), fromPeg, viaPeg, toPeg)
        traceln(`Move disk $n from $fromPeg to $toPeg`)
        move(n.previous(), viaPeg, toPeg, fromPeg)

move(3, "left", "right", "middle")
```

Nemerle (/wiki/Category:Nemerle)

```
using System;
using System.Console;

module Towers
{
    Hanoi(n : int, from = 1, to = 3, via = 2) : void
    {
        when (n > 0)
        {
            Hanoi(n - 1, from, via, to);
            WriteLine("Move disk from peg {0} to peg {1}", from, to);
            Hanoi(n - 1, via, to, from);
        }
    }

    Main() : void
    {
        Hanoi(4)
    }
}
```

NetRexx (/wiki/Category:NetRexx)

```
/* NetRexx */
options replace format comments java crossref symbols binary

runSample(arg)
return

-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
method runSample(arg) private static
  parse arg discs .
  if discs = '', discs < 1 then discs = 4
  say 'Minimum moves to solution:' 2 ** discs - 1
  moves = move(discs)
  say 'Solved in' moves 'moves.'
  return

-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
method move(discs = int 4, towerFrom = int 1, towerTo = int 2, towerVia = int 3, moves = int 0) public static
  if discs == 1 then do
    moves = moves + 1
    say 'Move disc from peg' towerFrom 'to peg' towerTo '- Move No:' Rexx(moves).right(5)
    end
  else do
    moves = move(discs - 1, towerFrom, towerVia, towerTo, moves)
    moves = move(1, towerFrom, towerTo, towerVia, moves)
    moves = move(discs - 1, towerVia, towerTo, towerFrom, moves)
    end
  return moves
```

Output:

```
Minimum moves to solution: 15
Move disc from peg 1 to peg 3 – Move No:     1
Move disc from peg 1 to peg 2 – Move No:     2
Move disc from peg 3 to peg 2 – Move No:     3
Move disc from peg 1 to peg 3 – Move No:     4
Move disc from peg 2 to peg 1 – Move No:     5
Move disc from peg 2 to peg 3 – Move No:     6
Move disc from peg 1 to peg 3 – Move No:     7
Move disc from peg 1 to peg 2 – Move No:     8
Move disc from peg 3 to peg 2 – Move No:     9
Move disc from peg 3 to peg 1 – Move No:    10
Move disc from peg 2 to peg 1 – Move No:    11
Move disc from peg 3 to peg 2 – Move No:    12
Move disc from peg 1 to peg 3 – Move No:    13
Move disc from peg 1 to peg 2 – Move No:    14
Move disc from peg 3 to peg 2 – Move No:    15
Solved in 15 moves.
```

NewLISP (/wiki/Category:NewLISP)

```
(define (move n from to via)
                        (if (> n 0)
                                (move (– n 1) from via to
                                (print "move disk from pole " from " to pole " to "\n")
                                (move (– n 1) via to from))))

(move 4 1 2 3)
```

Nim (/wiki/Category:Nim)

```
proc hanoi(disks: int; fromTower, toTower, viaTower: string) =
  if disks != 0:
    hanoi(disks – 1, fromTower, viaTower, toTower)
    echo("Move disk ", disks, " from ", fromTower, " to ", toTower)
    hanoi(disks – 1, viaTower, toTower, fromTower)

hanoi(4, "1", "2", "3")
```

Output:

```
Move disk 1 from 1 to 3
Move disk 2 from 1 to 2
Move disk 1 from 3 to 2
Move disk 3 from 1 to 3
Move disk 1 from 2 to 1
Move disk 2 from 2 to 3
Move disk 1 from 1 to 3
Move disk 4 from 1 to 2
Move disk 1 from 3 to 2
Move disk 2 from 3 to 1
Move disk 1 from 2 to 1
Move disk 3 from 3 to 2
Move disk 1 from 1 to 3
Move disk 2 from 1 to 2
Move disk 1 from 3 to 2
```

Objeck (/wiki/Category:Objeck)

```
class Hanoi {
  function : Main(args : String[]) ~ Nil {
    Move(4, 1, 2, 3);
  }

  function: Move(n:Int, f:Int, t:Int, v:Int) ~ Nil {
    if(n = 1) {
      "Move disk from pole {$f} to pole {$t}"->PrintLine();
    }
    else {
      Move(n - 1, f, v, t);
      Move(1, f, t, v);
      Move(n - 1, v, t, f);
    };
  }
}
```

Objective-C (/wiki/Category:Objective-C)

From here (https://sites.google.com/site/vinodkshukla/code-snippets/objectivec-towersofhanoi)

Works with: GNUstep (/wiki/GNUstep)
It should be compatible with XCode/Cocoa on MacOS too.

The Interface - TowersOfHanoi.h:

```
#import <Foundation/NSObject.h>

@interface TowersOfHanoi: NSObject (https://developer.apple.com/documentation/Cocoa/Reference/Foundation/Classes/NSObject_Class/) {
        int pegFrom;
        int pegTo;
        int pegVia;
        int numDisks;
}

-(void) setPegFrom: (int) from andSetPegTo: (int) to andSetPegVia: (int) via andSetNumDisks: (int) disks;
-(void) movePegFrom: (int) from andMovePegTo: (int) to andMovePegVia: (int) via andWithNumDisks: (int) disks;
@end
```

The Implementation - TowersOfHanoi.m:

```
#import "TowersOfHanoi.h"
@implementation TowersOfHanoi

-(void) setPegFrom: (int) from andSetPegTo: (int) to andSetPegVia: (int) via andSetNumDisks: (int) disks {
        pegFrom = from;
        pegTo = to;
        pegVia = via;
        numDisks = disks;
}

-(void) movePegFrom: (int) from andMovePegTo: (int) to andMovePegVia: (int) via andWithNumDisks: (int) disks {
        if (disks == 1) {
            printf (https://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("Move disk from pole %i to pole %i\n", from,
 to);
        } else {
                        [self movePegFrom: from andMovePegTo: via andMovePegVia: to andWithNumDisks: disks-1];
                        [self movePegFrom: from andMovePegTo: to andMovePegVia: via andWithNumDisks: 1];
                        [self movePegFrom: via andMovePegTo: to andMovePegVia: from andWithNumDisks: disks-1];
        }
}

@end
```

Test code: TowersTest.m:

```
#import <stdio.h>
#import "TowersOfHanoi.h"

int main( int argc, const char *argv[] ) {
        @autoreleasepool {

                TowersOfHanoi *tower = [[TowersOfHanoi alloc] init];

                int from = 1;
                int to = 3;
                int via = 2;
                int disks = 3;

                [tower setPegFrom: from andSetPegTo: to andSetPegVia: via andSetNumDisks: disks];

                [tower movePegFrom: from andMovePegTo: to andMovePegVia: via andWithNumDisks: disks];

        }
        return 0;
}
```

OCaml (/wiki/Category:OCaml)

```
let rec hanoi n a b c =
  if n <> 0 then begin
    hanoi (pred (http://caml.inria.fr/pub/docs/manual-ocaml/libref/Pervasives.html#VALpred) n) a c b;
    Printf (http://caml.inria.fr/pub/docs/manual-ocaml/libref/Printf.html).printf "Move disk from pole %d to pole %d\n" a b;
    hanoi (pred (http://caml.inria.fr/pub/docs/manual-ocaml/libref/Pervasives.html#VALpred) n) c b a
  end

let () =
  hanoi 4 1 2 3
```

Octave (/wiki/Category:Octave)

```
function hanoimove(ndisks, from, to, via)
  if ( ndisks == 1 )
    printf (http://octave.sourceforge.net/octave/function/printf.html)("Move disk from pole %d to pole %d\n", from, to);
  else
    hanoimove(ndisks-1, from, via, to);
    hanoimove(1, from, to, via);
    hanoimove(ndisks-1, via, to, from);
  endif
endfunction

hanoimove(4, 1, 2, 3);
```

Oforth (/wiki/Category:Oforth)

```
: move(n, from, to, via)
   n 0 > ifTrue: [
      move(n 1-, from, via, to)
      System.Out "Move disk from " << from << " to " << to << cr
      move(n 1-, via, to, from)
      ] ;

5 $left $middle $right) move
```

Oz (/wiki/Category:Oz)

```
declare
  proc {TowersOfHanoi N From To Via}
     if N > 0 then
        {TowersOfHanoi N-1 From Via To}
        {System.showInfo "Move from "#From#" to "#To}
        {TowersOfHanoi N-1 Via To From}
     end
  end
in
  {TowersOfHanoi 4 left middle right}
```

PARI/GP (/wiki/Category:PARI/GP)

Translation of: Python

```
\\ Towers of Hanoi
\\ 8/19/2016 aev
\\ Where: n = number of disks, sp = start pole, ep = end pole.
HanoiTowers(n,sp,ep)={
  if(n!=0,
    HanoiTowers(n-1,sp,6-sp-ep);
    print("Move disk ", n, " from pole ", sp," to pole ", ep);
    HanoiTowers(n-1,6-sp-ep,ep);
    );
}
\\ Testing n=3:
HanoiTowers(3,1,3);
```

Output:

```
> HanoiTower(3,1,3);
Move disk 1 from pole 1 to pole 3
Move disk 2 from pole 1 to pole 2
Move disk 1 from pole 3 to pole 2
Move disk 3 from pole 1 to pole 3
Move disk 1 from pole 2 to pole 1
Move disk 2 from pole 2 to pole 3
Move disk 1 from pole 1 to pole 3
```

Pascal (/wiki/Category:Pascal)

Works with: Free Pascal (/wiki/Free_Pascal) version 2.0.4
I think it is standard pascal, except for the constant array "strPole". I am not sure if constant arrays are part of the standard. However, as far as I know, they are a "de facto" standard in every compiler.

```
program Hanoi;
type
  TPole = (tpLeft, tpCenter, tpRight);
const
  strPole:array[TPole] of string[6]=('left','center','right');

 procedure MoveStack (const Ndisks : integer; const Origin,Destination,Auxiliary:TPole);
 begin
  if Ndisks >0 then begin
     MoveStack(Ndisks - 1, Origin,Auxiliary, Destination );
     Writeln('Move disk ',Ndisks ,' from ',strPole[Origin],' to ',strPole[Destination]);
     MoveStack(Ndisks - 1, Auxiliary, Destination, origin);
  end;
 end;

begin
 MoveStack(4,tpLeft,tpCenter,tpRight);
end.
```

A little longer, but clearer for my taste

```
program Hanoi;
type
  TPole = (tpLeft, tpCenter, tpRight);
const
  strPole:array[TPole] of string[6]=('left','center','right');

 procedure MoveOneDisk(const DiskNum:integer; const Origin,Destination:TPole);
 begin
  Writeln('Move disk ',DiskNum,' from ',strPole[Origin],' to ',strPole[Destination]);
 end;

 procedure MoveStack (const Ndisks : integer; const Origin,Destination,Auxiliary:TPole);
 begin
  if Ndisks =1 then
       MoveOneDisk(1,origin,Destination)
  else begin
       MoveStack(Ndisks - 1, Origin,Auxiliary, Destination );
       MoveOneDisk(Ndisks,origin,Destination);
       MoveStack(Ndisks - 1, Auxiliary, Destination, origin);
  end;
 end;

begin
 MoveStack(4,tpLeft,tpCenter,tpRight);
end.
```

Perl (/wiki/Category:Perl)

```
sub hanoi {
    my ($n, $from, $to, $via) = (@_, 1, 2, 3);

    if ($n == 1) {
        print (https://perldoc.perl.org/functions/print.html) "Move disk from pole $from to pole $to.\n";
    } else {
        hanoi($n - 1, $from, $via, $to);
        hanoi(1, $from, $to, $via);
        hanoi($n - 1, $via, $to, $from);
    };
};
```

Phix (/wiki/Category:Phix)

```
constant poles = {"left","middle","right"}
enum               left,  middle,  right

sequence disks
integer moves

procedure showpegs(integer src, integer dest)
    string desc = sprintf("%s to %s:",{poles[src],poles[dest]})
    disks[dest] &= disks[src][$]
    disks[src] = disks[src][1..$-1]
    for i=1 to length(disks) do
        printf(1,"%-16s | %s\n",{desc,join(sq_add(disks[i],'0'),' ')})
        desc = ""
    end for
    printf(1,"\n")
    moves += 1
end procedure

procedure hanoir(integer n, src=left, dest=right, via=middle)
    if n>0 then
        hanoir(n-1, src, via, dest)
        showpegs(src,dest)
        hanoir(n-1, via, dest, src)
    end if
end procedure

procedure hanoi(integer n)
    disks = {reverse(tagset(n)),{},{}}
    moves = 0
    hanoir(n)
    printf(1,"completed in %d moves\n",{moves})
end procedure

hanoi(3) -- (output of 4,5,6 also shown)
```

Output:

```
left to right:   | 3 2
                 |
                 | 1

left to middle:  | 3
                 | 2
                 | 1

right to middle: | 3
                 | 2 1
                 |

left to right:   |
                 | 2 1
                 | 3

middle to left:  | 1
                 | 2
                 | 3

middle to right: | 1
                 |
                 | 3 2

left to right:   |
                 |
                 | 3 2 1

completed in 7 moves
```

```
left to middle:  | 4 3 2
                 | 1
                 |

left to right:   | 4 3
                 | 1
                 | 2

middle to right: | 4 3
                 |
                 | 2 1

   ...

left to middle:  | 2
                 | 1
                 | 4 3

left to right:   |
                 | 1
                 | 4 3 2

middle to right: |
                 |
                 | 4 3 2 1

completed in 15 moves
```

```
left to right:   | 5 4 3 2
                 |
                 | 1

left to middle:  | 5 4 3
                 | 2
                 | 1

right to middle: | 5 4 3
                 | 2 1
                 |

    ...

middle to left:  | 1
                 | 2
                 | 5 4 3

middle to right: | 1
                 |
                 | 5 4 3 2

left to right:   |
                 |
                 | 5 4 3 2 1

completed in 31 moves
```

```
left to middle:  | 6 5 4 3 2
                 | 1
                 |

left to right:   | 6 5 4 3
                 | 1
                 | 2

middle to right: | 6 5 4 3
                 |
                 | 2 1

    ...

left to middle:  | 2
                 | 1
                 | 6 5 4 3

left to right:   |
                 | 1
                 | 6 5 4 3 2

middle to right: |
                 |
                 | 6 5 4 3 2 1

completed in 63 moves
```

PHL (/wiki/Category:PHL)

Translation of: C

```
module hanoi;

extern printf;

@Void move(@Integer n, @Integer from, @Integer to, @Integer via) [
        if (n > 0) {
                move(n - 1, from, via, to);
                printf("Move disk from pole %d to pole %d\n", from, to);
                move(n - 1, via, to, from);
        }
]

@Integer main [
        move(4, 1,2,3);
        return 0;
]
```

PHP (/wiki/Category:PHP)

Translation of: Java

```
function move($n,$from,$to,$via) {
    if ($n === 1) {
        print("Move disk from pole $from to pole $to");
    } else {
        move($n-1,$from,$via,$to);
        move(1,$from,$to,$via);
        move($n-1,$via,$to,$from);
    }
}
```

PicoLisp (/wiki/Category:PicoLisp)

```
(de move (N A B C)  # Use: (move 3 'left 'center 'right)
   (unless (=0 N)
      (move (dec N) A C B)
      (println 'Move 'disk 'from A 'to B)
      (move (dec N) C B A) ) )
```

PL/I (/wiki/Category:PL/I)

Translation of: Fortran

```
tower: proc options (main);

   call Move (4,1,2,3);

Move: procedure (ndiscs, from, to, via) recursive;
   declare (ndiscs, from, to, via) fixed binary;

   if ndiscs = 1 then
      put skip edit ('Move disc from pole ', trim(from), ' to pole ',
         trim(to) ) (a);
   else
      do;
         call Move (ndiscs-1, from, via, to);
         call Move (1, from, to, via);
         call Move (ndiscs-1, via, to, from);
      end;
end Move;

end tower;
```

Plain TEX (/wiki/Category:PlainTeX)

```
\newcount\hanoidepth
\def\hanoi#1{%
  \hanoidepth = #1
  \move abc
}%
\def\move#1#2#3{%
  \advance \hanoidepth by -1
  \ifnum \hanoidepth > 0
    \move #1#3#2
  \fi
  Move the upper disk from pole #1 to pole #3.\par
  \ifnum \hanoidepth > 0
    \move#2#1#3
  \fi
  \advance \hanoidepth by 1
}

\hanoi{5}
\end
```

Pop11 (/wiki/Category:Pop11)

```
define hanoi(n, src, dst, via);
if n > 0 then
    hanoi(n - 1, src, via, dst);
    'Move disk ' >< n >< ' from ' >< src >< ' to ' >< dst >< '.' =>
    hanoi(n - 1, via, dst, src);
endif;
enddefine;

hanoi(4, "left", "middle", "right");
```

PostScript (/wiki/Category:PostScript)

A million-page document, each page showing one move.

```
%!PS-Adobe-3.0
%%BoundingBox: 0 0 300 300

/plate {
        exch 100 mul 50 add exch th mul 10 add moveto
        dup s mul neg 2 div 0 rmoveto
        dup s mul 0 rlineto
        0 th rlineto
        s neg mul 0 rlineto
        closepath gsave .5 setgray fill grestore 0 setgray stroke
} def

/drawtower {
        0 1 2 { /x exch def /y 0 def
                tower x get {
                        dup 0 gt { x y plate /y y 1 add def } {pop} ifelse
                } forall
        } for showpage
} def

/apop { [ exch aload pop /last exch def ] last } def
/apush{ [ 3 1 roll aload pop counttomark -1 roll ] } def

/hanoi {
        0 dict begin /from /mid /to /h 5 -1 2 { -1 roll def } for
        h 1 eq {
                tower from get apop tower to get apush
                tower to 3 -1 roll put
                tower from 3 -1 roll put
                drawtower
        } {
                /h h 1 sub def
                from to mid h hanoi
                from mid to 1 hanoi
                mid from to h hanoi
        } ifelse
        end
} def


/n 12 def
/s 90 n div def
/th 180 n div def
/tower [ [n 1 add -1 2 { } for ] [] [] ] def

drawtower 0 1 2 n hanoi

%%EOF
```

PowerShell (/wiki/Category:PowerShell)

Works with: PowerShell (/wiki/PowerShell) version 4.0

```
function hanoi($n, $a,  $b, $c) {
    if($n -eq 1) {
        "$a -> $c"
    } else{
         hanoi ($n - 1) $a $c $b
         hanoi 1 $a $b $c
         hanoi ($n - 1) $b $a $c
    }
}
hanoi 3 "A" "B" "C"
```

Output:

```
A -> C
A -> B
C -> B
A -> C
B -> A
B -> C
A -> C
```

Prolog (/wiki/Category:Prolog)

From Programming in Prolog by W.F. Clocksin & C.S. Mellish

```
hanoi(N) :- move(N,left,center,right).

move(0,_,_,_) :- !.
move(N,A,B,C) :-
    M is (http://pauillac.inria.fr/~deransar/prolog/bips.html) N-1,
    move(M,A,C,B),
    inform(A,B),
    move(M,C,B,A).

inform(X,Y) :- write (http://pauillac.inria.fr/~deransar/prolog/bips.html)([move,a,disk,from,the,X,pole,to,Y,pole]), nl (http://pauill
ac.inria.fr/~deransar/prolog/bips.html).
```

Using DCGs and separating core logic from IO

```
hanoi(N, Src, Aux, Dest, Moves-NMoves) :-
  NMoves is (http://pauillac.inria.fr/~deransar/prolog/bips.html) 2^N - 1,
  length(Moves, NMoves),
  phrase(move(N, Src, Aux, Dest), Moves).


move(1, Src, _, Dest) --> !,
  [Src->Dest].

move(2, Src, Aux, Dest) --> !,
  [Src->Aux,Src->Dest,Aux->Dest].

move(N, Src, Aux, Dest) -->
  { succ(N0, N) },
  move(N0, Src, Dest, Aux),
  move(1, Src, Aux, Dest),
  move(N0, Aux, Src, Dest).
```

PureBasic (/wiki/Category:PureBasic)

Algorithm according to http://en.wikipedia.org/wiki/Towers_of_Hanoi (https://en.wikipedia.org/wiki/Towers_of_Hanoi)

```
Procedure Hanoi(n, A.s, C.s, B.s)
  If n
    Hanoi(n-1, A, B, C)
    PrintN("Move the plate from "+A+" to "+C)
    Hanoi(n-1, B, C, A)
  EndIf
EndProcedure
```

Full program

```
Procedure Hanoi(n, A.s, C.s, B.s)
  If n
    Hanoi(n-1, A, B, C)
    PrintN("Move the plate from "+A+" to "+C)
    Hanoi(n-1, B, C, A)
  EndIf
EndProcedure

If OpenConsole()
  Define n=3
  PrintN("Moving "+Str(n)+" pegs."+#CRLF$)
  Hanoi(n,"Left Peg","Middle Peg","Right Peg")
  PrintN(#CRLF$+"Press ENTER to exit."): Input()
EndIf
```

Output:

```
Moving 3 pegs.

Move the plate from Left Peg to Middle Peg
Move the plate from Left Peg to Right Peg
Move the plate from Middle Peg to Right Peg
Move the plate from Left Peg to Middle Peg
Move the plate from Right Peg to Left Peg
Move the plate from Right Peg to Middle Peg
Move the plate from Left Peg to Middle Peg

Press ENTER to exit.
```

Python (/wiki/Category:Python)

Recursive

```
def hanoi(ndisks, startPeg=1, endPeg=3):
    if ndisks:
        hanoi(ndisks-1, startPeg, 6-startPeg-endPeg)
        print "Move disk %d from peg %d to peg %d" % (ndisks, startPeg, endPeg)
        hanoi(ndisks-1, 6-startPeg-endPeg, endPeg)

hanoi(ndisks=4)
```

Output:

for ndisks=2

```
Move disk 1 from peg 1 to peg 2
Move disk 2 from peg 1 to peg 3
Move disk 1 from peg 2 to peg 3
```

Or, separating the definition of the data from its display:

Works with: Python (/wiki/Python) version 3.7

```
'''Towers of Hanoi'''


# hanoi :: Int -> String -> String -> String -> [(String, String)]
def hanoi(n):
    '''A list of (from, to) label pairs,
       where a, b and c are labels for each of the
       three Hanoi tower positions.'''
    def go(n, a, b, c):
        p = n - 1
        return (
            go(p, a, c, b) + [(a, b)] + go(p, c, b, a)
        ) if 0 < n else []
    return lambda a: lambda b: lambda c: go(n, a, b, c)


# TEST ----------------------------------------------------
if __name__ == '__main__':

    # fromTo :: (String, String) -> String
    def fromTo(xy):
        '''x -> y'''
        x, y = xy
        return x.rjust(5, ' ') + ' -> ' + y

    print(__doc__ + ':\n\n' + '\n'.join(
        map(fromTo, hanoi(4)('left')('right')('mid'))
    ))
```

Output:

```
Towers of Hanoi:

 left -> mid
 left -> right
  mid -> right
 left -> mid
right -> left
right -> mid
 left -> mid
 left -> right
  mid -> right
  mid -> left
right -> left
  mid -> right
 left -> mid
 left -> right
  mid -> right
```

Graphic

Refactoring the version above to recursively generate a simple visualisation:

Works with: Python (/wiki/Python) version 3.7

```
'''Towers of Hanoi'''

from itertools import accumulate, chain, repeat
from inspect import signature
import operator


# hanoi :: Int -> [(Int, Int)]
def hanoi(n):
    '''A list of index pairs, representing disk moves
       between indexed Hanoi positions.
    '''
    def go(n, a, b, c):
        p = n - 1
        return (
            go(p, a, c, b) + [(a, b)] + go(p, c, b, a)
        ) if 0 < n else []
    return go(n, 0, 2, 1)


# hanoiState :: ([Int],[Int],[Int], String) -> (Int, Int) ->
#               ([Int],[Int],[Int], String)
def hanoiState(tpl, ab):
    '''A new Hanoi tower state'''
    a, b = ab
    xs, ys = tpl[a], tpl[b]

    w = 3 * (2 + (2 * max(map(max, filter(len, tpl[:-1])))))

    def delta(i):
        return tpl[i] if i not in ab else xs[1:] if (
            i == a
        ) else [xs[0]] + ys

    tkns = moveName(('left', 'mid', 'right'))(ab)
    caption = ' '.join(tkns)
    return tuple(map(delta, [0, 1, 2])) + (
        (caption if tkns[0] != 'mid' else caption.rjust(w, ' ')),
    )


# showHanoi :: ([Int],[Int],[Int], String) -> String
def showHanoi(tpl):
    '''Captioned string representation of an updated Hanoi tower state.'''

    def fullHeight(n):
        return lambda xs: list(repeat('', n - len(xs))) + xs

    mul = curry(operator.mul)
    lt = curry(operator.lt)
    rods = fmap(fmap(mul('__')))(
        list(tpl[0:3])
    )
    h = max(map(len, rods))
    w = 2 + max(
        map(
```

```
            compose(max)(fmap(len)),
            filter(compose(lt(0))(len), rods)
        )
    )
    xs = fmap(concat)(
        transpose(fmap(
            compose(fmap(center(w)(' ')))(
                fullHeight(h)
            )
        )(rods))
    )
    return tpl[3] + '\n\n' + unlines(xs) + '\n' + ('___' * w)


# moveName :: (String, String, String) -> (Int, Int) -> [String]
def moveName(labels):
    '''(from, to) index pair represented as an a -> b string.'''
    def go(ab):
        a, b = ab
        return [labels[a], ' to ', labels[b]] if a < b else [
            labels[b], ' from ', labels[a]
        ]
    return lambda ab: go(ab)


# TEST ----------------------------------------------------
def main():
    '''Visualisation of a Hanoi tower sequence for N discs.
    '''
    n = 3
    print('Hanoi sequence for ' + str(n) + ' disks:\n')
    print(unlines(
        fmap(showHanoi)(
            scanl(hanoiState)(
                (enumFromTo(1)(n), [], [], '')
            )(hanoi(n))
        )
    ))


# GENERIC -------------------------------------------------

# center :: Int -> Char -> String -> String
def center(n):
    '''String s padded with c to approximate centre,
       fitting in but not truncated to width n.'''
    return lambda c: lambda s: s.center(n, c)


# compose (<<<) :: (b -> c) -> (a -> b) -> a -> c
def compose(g):
    '''Right to left function composition.'''
    return lambda f: lambda x: g(f(x))


# concat :: [[a]] -> [a]
# concat :: [String] -> String
def concat(xs):
    '''The concatenation of all the elements
       in a list or iterable.'''

    def f(ys):
        zs = list(chain(*ys))
        return ''.join(zs) if isinstance(ys[0], str) else zs

    return (
        f(xs) if isinstance(xs, list) else (
            chain.from_iterable(xs)
        )
    ) if xs else []


# curry :: ((a, b) -> c) -> a -> b -> c
def curry(f):
    '''A curried function derived
       from an uncurried function.'''
    if 1 < len(signature(f).parameters):
        return lambda x: lambda y: f(x, y)
    else:
        return f
```

```
# enumFromTo :: (Int, Int) -> [Int]
def enumFromTo(m):
    '''Integer enumeration from m to n.'''
    return lambda n: list(range(m, 1 + n))


# fmap :: (a -> b) -> [a] -> [b]
def fmap(f):
    '''fmap over a list.
       f lifted to a function over a list.
    '''
    return lambda xs: list(map(f, xs))


# scanl :: (b -> a -> b) -> b -> [a] -> [b]
def scanl(f):
    '''scanl is like reduce, but returns a succession of
       intermediate values, building from the left.
    '''
    return lambda a: lambda xs: (
        accumulate(chain([a], xs), f)
    )


# showLog :: a -> IO String
def showLog(*s):
    '''Arguments printed with
       intercalated arrows.'''
    print(
        ' -> '.join(map(str, s))
    )


# transpose :: Matrix a -> Matrix a
def transpose(m):
    '''The rows and columns of the argument transposed.
       (The matrix containers and rows can be lists or tuples).
    '''
    if m:
        inner = type(m[0])
        z = zip(*m)
        return (type(m))(
            map(inner, z) if tuple != inner else z
        )
    else:
        return m


# unlines :: [String] -> String
def unlines(xs):
    '''A single string derived by the intercalation
       of a list of strings with the newline character.
    '''
    return '\n'.join(xs)


# TEST ----------------------------------------------------
if __name__ == '__main__':
    main()
```

```
Hanoi sequence for 3 disks:


   __
  ____
 ______
_____________________
left  to  right

  ____
 ______           __
_____________________
left  to  mid

 ______   ____    __
_____________________
        mid  from  right

          __
 ______   ____
_____________________
left  to  right

          __
         ____   ______
_____________________
left  from  mid

   __    ____   ______
_____________________
         mid  to  right

                ____
   __           ______
_____________________
left  to  right

                 __
                ____
               ______
_____________________
```

Library: VPython (/wiki/Category:VPython)

There is a 3D hanoi-game in the examples that come with VPython, and at github (https://github.com/vpython/visual/blob/master/examples/hanoi.py).

Quackery (/wiki/Category:Quackery)

```
  [ stack ]                    is rings    (     --> [ )

  [ rings share
    depth share -
    8 * times sp
    emit sp emit sp
    say 'move' cr ]            is echomove ( c c -->   )

  [ dup rings put
    depth put
    char a char b char c
    [ swap decurse
      rot 2dup echomove
      decurse
      swap rot ]
    3 times drop
    depth release
    rings release ]            is hanoi    (   n --> n )

  say 'How to solve a three ring Towers of Hanoi puzzle:' cr cr
  3 hanoi cr
```

Output:

```
How to solve a three ring Towers of Hanoi puzzle:

                        a c move
                a b move
                        c b move
        a c move
                        b a move
                b c move
                        a c move
a b move
                        c b move
                c a move
                        b a move
        c b move
                        a c move
                a b move
                        c b move
```

Quite BASIC (/wiki/Category:Quite_BASIC)

```
'This is implemented on the Quite BASIC website
'http://www.quitebasic.com/prj/puzzle/towers-of-hanoi/ (http://www.quitebasic.com/prj/puzzle/towers-of-hanoi/)
```

```
1000 REM Towers of Hanoi
1010 REM Quite BASIC Puzzle Project
1020 CLS
1030 PRINT "Towers of Hanoi"
1040 PRINT
1050 PRINT "This is a recursive solution for seven discs."
1060 PRINT
1070 PRINT "See the REM statements in the program if you didn't think that recursion was possible in classic BASIC!"
1080 REM Yep, recursive GOSUB calls works in Quite BASIC!
1090 REM However, to actually write useful recursive algorithms, it helps to have variable scoping and parameters to subroutines -- so
mething classic BASIC is lacking.  In this case we have only one "parameter" -- the variable N.  And subroutines are always called wit
h N-1.  This is lucky for us because we can keep track of the value by decrementing it when we enter subroutines and incrementing it b
ack when we exit.
1100 REM If we had subroutine parameters we could have written a single subroutine for moving discs from peg P to peg Q where P and Q
were subroutine parameters, but no such luck.  Instead we have to write six different subroutines for moving from peg to peg.  See Sub
routines 4000, 5000, 6000, 7000, 8000, and 9000.
1110 REM ==============================
2000 REM A, B, and C are arrays holding the discs
2010 REM We refer to the corresponding pegs as peg A, B, and C
2020 ARRAY A
2030 ARRAY B
2040 ARRAY C
2050 REM Fill peg A with seven discs
2060 FOR I = 0 TO 6
2070 LET A[I] = 7 - I
2080 NEXT I
2090 REM X, Y, Z hold the number of discs on pegs A, B, and C
2100 LET X = 7
2110 LET Y = 0
2120 LET Z = 0
2130 REM Disc colors
2140 ARRAY P
2150 LET P[1] = "cyan"
2160 LET P[2] = "blue"
2170 LET P[3] = "green"
2180 LET P[4] = "yellow"
2190 LET P[5] = "magenta"
2200 LET P[6] = "orange"
2210 LET P[7] = "red"
2220 REM Draw initial position -- all discs on the A peg
2230 FOR I = 0 TO 6
2240 FOR J = 8 - A[I] TO 8 + A[I]
2250 PLOT J, I, P[A[I]]
2260 NEXT J
2270 NEXT I
2280 REM N is the number of discs to move
2290 LET N = 7
2320 REM Move all discs from peg A to peg B
2310 GOSUB 6000
2320 END
3000 REM The subroutines 3400, 3500, 3600, 3700, 3800, 3900
3010 REM handle the drawing of the discs on the canvas as we
3020 REM move discs from one peg to another.
```

```
3030 REM These subroutines also update the variables X, Y, and Z
3040 REM which hold the number of discs on each peg.
3050 REM ============================
3400 REM Subroutine -- Remove disc from peg A
3410 LET X = X - 1
3420 FOR I = 8 - A[X] TO 8 + A[X]
3430 PLOT I, X, "gray"
3440 NEXT I
3450 RETURN
3500 REM Subroutine -- Add disc to peg A
3510 FOR I = 8 - A[X] TO 8 + A[X]
3520 PLOT I, X, P[A[X]]
3530 NEXT I
3540 LET X = X + 1
3550 PAUSE 400 * (5 - LEVEL) + 10
3560 RETURN
3600 REM Subroutine -- Remove disc from peg B
3610 LET Y = Y - 1
3620 FOR I = 24 - B[Y] TO 24 + B[Y]
3630 PLOT I, Y, "gray"
3640 NEXT I
3650 RETURN
3700 REM Subroutine -- Add disc to peg B
3710 FOR I = 24 - B[Y] TO 24 + B[Y]
3720 PLOT I, Y, P[B[Y]]
3730 NEXT I
3740 LET Y = Y + 1
3750 PAUSE 400 * (5 - LEVEL) + 10
3760 RETURN
3800 REM Subroutine -- Remove disc from peg C
3810 LET Z = Z - 1
3820 FOR I = 40 - C[Z] TO 40 + C[Z]
3830 PLOT I, Z, "gray"
3840 NEXT I
3850 RETURN
3900 REM Subroutine -- Add disc to peg C
3910 FOR I = 40 - C[Z] TO 40 + C[Z]
3920 PLOT I, Z, P[C[Z]]
3930 NEXT I
3940 LET Z = Z + 1
3950 PAUSE 400 * (5 - LEVEL) + 10
3960 RETURN
4000 REM ====================================
4010 REM Recursive Subroutine -- move N discs from peg B to peg A
4020 REM First move N-1 discs from peg B to peg C
4030 LET N = N - 1
4040 IF N <> 0 THEN GOSUB 9000
4050 REM Then move one disc from peg B to peg A
4060 GOSUB 3600
4070 LET A[X] = B[Y]
4080 GOSUB 3500
4090 REM And finally move N-1 discs from peg C to peg A
4100 IF N <> 0 THEN GOSUB 5000
4110 REM Restore N before returning
4120 LET N = N + 1
4130 RETURN
5000 REM ====================================
5010 REM Recursive Subroutine -- Move N discs from peg C to peg A
5020 REM First move N-1 discs from peg C to peg B
5030 LET N = N - 1
5040 IF N <> 0 THEN GOSUB 8000
5050 REM Then move one disc from peg C to peg A
5060 GOSUB 3800
5070 LET A[X] = C[Z]
5080 GOSUB 3500
5090 REM And finally move N-1 discs from peg B to peg A
5100 IF N <> 0 THEN GOSUB 4000
5120 REM Restore N before returning
5130 LET N = N + 1
5140 RETURN
6000 REM ====================================
6000 REM Recursive Subroutine -- Move N discs from peg A to peg B
6010 REM First move N-1 discs from peg A to peg C
6020 LET N = N - 1
6030 IF N <> 0 THEN GOSUB 7000
6040 REM Then move one disc from peg A to peg B
6050 GOSUB 3400
6060 LET B[Y] = A[X]
6070 GOSUB 3700
6090 REM And finally move N-1 discs from peg C to peg B
6100 IF N <> 0 THEN GOSUB 8000
```

```
6110 REM Restore N before returning
6120 LET N = N + 1
6130 RETURN
7000 REM ====================================
7010 REM Recursive Subroutine -- Move N discs from peg A to peg C
7020 REM First move N-1 discs from peg A to peg B
7030 LET N = N - 1
7040 IF N <> 0 THEN GOSUB 6000
7050 REM Then move one disc from peg A to peg C
7060 GOSUB 3400
7070 LET C[Z] = A[X]
7080 GOSUB 3900
7090 REM And finally move N-1 discs from peg B to peg C
7100 IF N <> 0 THEN GOSUB 9000
7110 REM Restore N before returning
7120 LET N = N + 1
7130 RETURN
8000 REM ====================================
8010 REM Recursive Subroutine -- Move N discs from peg C to peg B
8020 REM First move N-1 discs from peg C to peg A
8030 LET N = N - 1
8040 IF N <> 0 THEN GOSUB 5000
8050 REM Then move one disc from peg C to peg B
8060 GOSUB 3800
8070 LET B[Y] = C[Z]
8080 GOSUB 3700
8090 REM And finally move N-1 discs from peg A to peg B
8100 IF N <> 0 THEN GOSUB 6000
8110 REM Restore N before returning
8120 LET N = N + 1
8130 RETURN
9000 REM ====================================
9010 REM Recursive Subroutine -- Move N discs from peg B to peg C
9020 REM First move N-1 discs from peg B to peg A
9030 LET N = N - 1
9040 IF N <> 0 THEN GOSUB 4000
9050 REM Then move one disc from peg B to peg C
9060 GOSUB 3600
9070 LET C[Z] = B[Y]
9080 GOSUB 3900
9090 REM And finally move N-1 discs from peg A to peg C
9100 IF N <> 0 THEN GOSUB 7000
9110 REM Restore N before returning
9120 LET N = N + 1
9130 RETURN
```

R (/wiki/Category:R)

Translation of: Octave

```
hanoimove <- function (http://stat.ethz.ch/R-manual/R-devel/library/base/html/function.html)(ndisks, from, to, via) {
  if (http://stat.ethz.ch/R-manual/R-devel/library/base/html/if.html) (ndisks == 1) {
    cat (http://stat.ethz.ch/R-manual/R-devel/library/base/html/cat.html)("move disk from", from, "to", to, "\n")
  } else {
    hanoimove(ndisks - 1, from, via, to)
    hanoimove(1, from, to, via)
    hanoimove(ndisks - 1, via, to, from)
  }
}

hanoimove(4, 1, 2, 3)
```

Racket (/wiki/Category:Racket)

```
#lang racket
(define (hanoi n a b c)
  (when (> n 0)
    (hanoi (- n 1) a c b)
    (printf "Move ~a to ~a\n" a b)
    (hanoi (- n 1) c b a)))
(hanoi 4 'left 'middle 'right)
```

Raku (/wiki/Category:Raku)

(formerly Perl 6)

```
subset Peg of Int where 1|2|3;

multi hanoi (0,      Peg $a,     Peg $b,     Peg $c)     { }
multi hanoi (Int $n, Peg $a = 1, Peg $b = 2, Peg $c = 3) {
    hanoi $n - 1, $a, $c, $b;
    say "Move $a to $b.";
    hanoi $n - 1, $c, $b, $a;
}
```

Rascal (/wiki/Category:Rascal)

Translation of: Python

```
public void hanoi(ndisks, startPeg, endPeg){
        if(ndisks>0){
                hanoi(ndisks-1, startPeg, 6 - startPeg - endPeg);
                println("Move disk <ndisks> from peg <startPeg> to peg <endPeg>");
                hanoi(ndisks-1, 6 - startPeg - endPeg, endPeg);
        }
}
```

Output:

```
rascal>hanoi(4,1,3)
Move disk 1 from peg 1 to peg 2
Move disk 2 from peg 1 to peg 3
Move disk 1 from peg 2 to peg 3
Move disk 3 from peg 1 to peg 2
Move disk 1 from peg 3 to peg 1
Move disk 2 from peg 3 to peg 2
Move disk 1 from peg 1 to peg 2
Move disk 4 from peg 1 to peg 3
Move disk 1 from peg 2 to peg 3
Move disk 2 from peg 2 to peg 1
Move disk 1 from peg 3 to peg 1
Move disk 3 from peg 2 to peg 3
Move disk 1 from peg 1 to peg 2
Move disk 2 from peg 1 to peg 3
Move disk 1 from peg 2 to peg 3
ok
```

Raven (/wiki/Category:Raven)

Translation of: Python

```
define hanoi use ndisks, startpeg, endpeg
   ndisks 0 > if
      6 startpeg - endpeg - startpeg ndisks 1 - hanoi
      endpeg startpeg ndisks "Move disk %d from peg %d to peg %d\n" print
      endpeg 6 startpeg - endpeg - ndisks 1 - hanoi

define dohanoi use ndisks
   # startpeg=1, endpeg=3
   3 1 ndisks hanoi

# 4 disks
4 dohanoi
```

Output:

```
raven hanoi.rv
Move disk 1 from peg 1 to peg 2
Move disk 2 from peg 1 to peg 3
Move disk 1 from peg 2 to peg 3
Move disk 3 from peg 1 to peg 2
Move disk 1 from peg 3 to peg 1
Move disk 2 from peg 3 to peg 2
Move disk 1 from peg 1 to peg 2
Move disk 4 from peg 1 to peg 3
Move disk 1 from peg 2 to peg 3
Move disk 2 from peg 2 to peg 1
Move disk 1 from peg 3 to peg 1
Move disk 3 from peg 2 to peg 3
Move disk 1 from peg 1 to peg 2
Move disk 2 from peg 1 to peg 3
Move disk 1 from peg 2 to peg 3
```

REBOL (/wiki/Category:REBOL)

```
rebol [
        Title: "Towers of Hanoi"
        URL: http://rosettacode.org/wiki/Towers_of_Hanoi
]

hanoi: func [
        {Begin moving the golden disks from one pole to the next.
         Note: when last disk moved, the world will end.}
        disks [integer!] "Number of discs on starting pole."
        /poles "Name poles."
        from to via
][
    if disks = 0 [return]
        if not poles [from: 'left  to: 'middle  via: 'right]

    hanoi/poles disks - 1 from via to
        print [from "->" to]
    hanoi/poles disks - 1 via to from
]

hanoi 4
```

Output:

```
left -> right
left -> middle
right -> middle
left -> right
middle -> left
middle -> right
left -> right
left -> middle
right -> middle
right -> left
middle -> left
right -> middle
left -> right
left -> middle
right -> middle
```

Retro (/wiki/Category:Retro)

```
~~~
{ 'Num 'From 'To 'Via } [ var ] a:for-each

:set     !Via !To !From !Num ;
:display @To @From 'Move_a_ring_from_%n_to_%n\n s:format s:put ;

:hanoi (num,from,to,via-)
  set @Num n:-zero?
  [ @Num @From @To @Via
    @Num n:dec @From @Via @To hanoi set display
    @Num n:dec @Via @To @From hanoi ] if ;

#3 #1 #3 #2 hanoi nl
~~~
```

REXX (/wiki/Category:REXX)

simple text moves

```
/*REXX program  displays  the  moves  to solve  the  Tower of Hanoi  (with  N  disks).  */
parse arg N .                                    /*get optional number of disks from CL.*/
if N=='' | N=="," then N=3                       /*Not specified?  Then use the default.*/
#= 0                                             /*#:  the number of disk moves (so far)*/
z= 2**N  -  1                                    /*Z:   "     "    " minimum # of moves.*/
call mov  1, 3, N                                /*move the top disk,  then recurse ··· */
say                                              /* [↓]  Display the minimum # of moves.*/
say 'The minimum number of moves to solve a '      N"─disk  Tower of Hanoi is "     z
exit                                             /*stick a fork in it,  we're all done. */
/*──────────────────────────────────────────────────────────────────────────────────────*/
mov: procedure expose # z; parse arg  @1,@2,@3;          L= length(z)
     if @3==1  then do;  #= # + 1                /*bump the (disk) move counter by one. */
                         say 'step'   right(#, L)":  move disk on tower"   @1  '──►'   @2
                    end
               else do;  call mov @1,               6 -@1 -@2,         @3 -1
                         call mov @1,               @2,                  1
                         call mov 6 - @1 - @2,      @2,                @3 -1
                    end
     return                                      /* [↑]  this subroutine uses recursion.*/
```

output when using the default input:

```
step 1:  move disk on tower 1 ──► 3
step 2:  move disk on tower 1 ──► 2
step 3:  move disk on tower 3 ──► 2
step 4:  move disk on tower 1 ──► 3
step 5:  move disk on tower 2 ──► 1
step 6:  move disk on tower 2 ──► 3
step 7:  move disk on tower 1 ──► 3

The minimum number of moves to solve a  3─disk  Tower of Hanoi is  7
```

output when the following was entered (to solve with four disks): 4

```
step  1:  move disk on tower 1 ──► 2
step  2:  move disk on tower 1 ──► 3
step  3:  move disk on tower 2 ──► 3
step  4:  move disk on tower 1 ──► 2
step  5:  move disk on tower 3 ──► 1
step  6:  move disk on tower 3 ──► 2
step  7:  move disk on tower 1 ──► 2
step  8:  move disk on tower 1 ──► 3
step  9:  move disk on tower 2 ──► 3
step 10:  move disk on tower 2 ──► 1
step 11:  move disk on tower 3 ──► 1
step 12:  move disk on tower 2 ──► 3
step 13:  move disk on tower 1 ──► 2
step 14:  move disk on tower 1 ──► 3
step 15:  move disk on tower 2 ──► 3

The minimum number of moves to solve a  4─disk  Tower of Hanoi is  15
```

pictorial moves

This REXX version pictorially shows (via ASCII art) the moves for solving the Town of Hanoi.

Quite a bit of code has been dedicated to showing a "picture" of the towers with the disks, and the movement of the disk (for each move). "Coloring" of the disks is attempted with dithering.

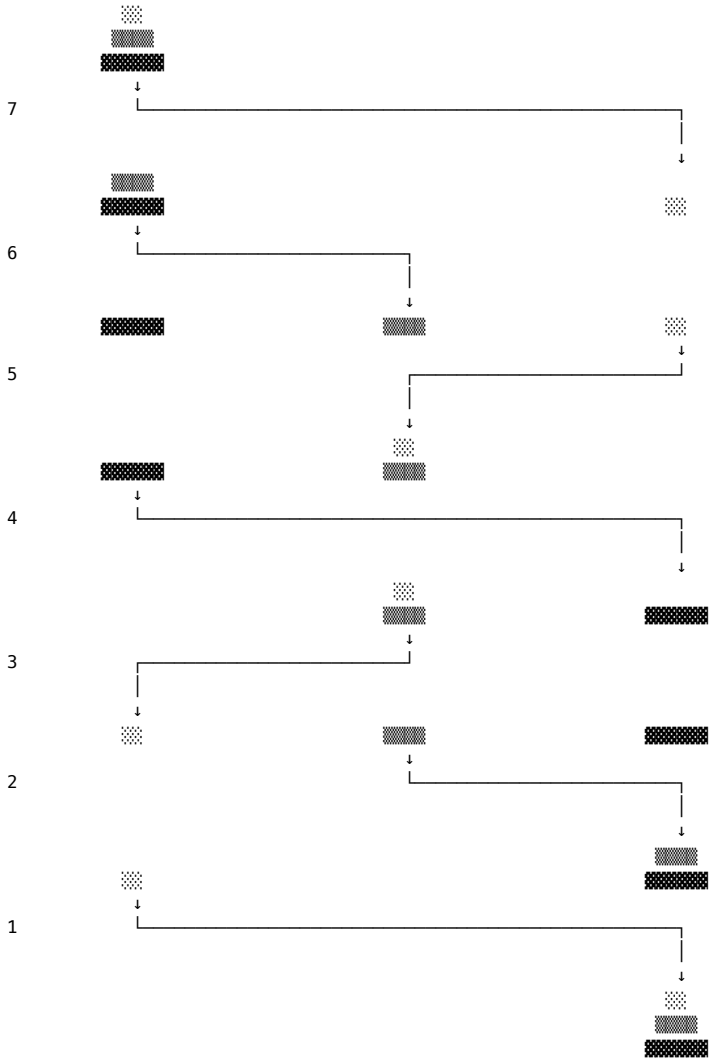
In addition, it shows each move in a countdown manner (the last move is marked as #1).

It may not be obvious from the pictorial display of the moves, but whenever a disk is moved from one tower to another, it is always the top disk that is moved (to the target tower).

Also, since the pictorial showing of the moves may be voluminous (especially for a larger number of disks), the move counter is started with the maximum and is the count shown is decremented so the viewer can see how many moves are left to display.

```
/*REXX program  displays  the  moves  to solve  the  Tower of Hanoi  (with  N  disks).  */
parse arg N .                                  /*get optional number of disks from CL.*/
if N=='' | N=="," then N=3                     /*Not specified?  Then use the default.*/
sw= 80;    wp= sw%3 - 1;   blanks= left('', wp)  /*define some default REXX variables.  */
c.1= sw % 3 % 2                                /* [↑]  SW: assume default Screen Width*/
c.2= sw % 2 - 1                                /* ◄═══  C.1 C.2 C.2  are the positions*/
c.3= sw - 2 - c.1                              /*                   of the 3 columns.*/
#= 0;        z= 2**N - 1;           moveK= z   /*#moves; min# of moves; where to move.*/
@abc= 'abcdefghijklmnopqrstuvwxyN'              /*dithering chars when many disks used.*/
ebcdic= ('f2'x==2)                             /*determine if EBCDIC or ASCII machine.*/

if ebcdic then do;   bar= 'bf'x;    ar= "df"x;    dither= 'db9f9caf'x;         down= "9a"x
                      tr= 'bc'x;    bl= "ab"x;    br= 'bb'x;   vert= "fa"x;      tl= 'ac'x
              end
          else do;   bar= 'c4'x;    ar= "10"x;    dither= 'b0b1b2db'x;         down= "19"x
                      tr= 'bf'x;    bl= "c0"x;    br= 'd9'x;   vert= "b3"x;      tl= 'da'x
              end

verts= vert || vert;           Tcorners= tl || tr;              box     = left(dither, 1)
downs= down || down;           Bcorners= bl || br;              boxChars= dither || @abc
$.= 0;         $.1= N;         k= N;                            kk= k + k

  do j=1  for N;   @.3.j= blanks;    @.2.j= blanks;    @.1.j= center( copies(box, kk), wp)
  if N<=length(boxChars)  then @.1.j= translate( @.1.j, , substr( boxChars, kk%2, 1), box)
  kk= kk - 2
  end   /*j*/                                  /*populate the tower of Hanoi spindles.*/

call showTowers;   call mov 1,3,N;   say
say 'The minimum number of moves to solve a '        N"-disk  Tower of Hanoi is "      z
exit                                           /*stick a fork in it,  we're all done. */
/*──────────────────────────────────────────────────────────────────────────────────────*/
dsk: parse arg from dest;   #= # + 1;       pp=
     if from==1  then do;  pp= overlay(bl,  pp, c.1)
                           pp= overlay(bar, pp, c.1+1, c.dest-c.1-1, bar) || tr
                      end
     if from==2  then do
                      if dest==1  then do;  pp= overlay(tl,  pp, c.1)
                                            pp= overlay(bar, pp, c.1+1, c.2-c.1-1,bar)||br
                                       end
                      if dest==3  then do;  pp= overlay(bl,  pp, c.2)
                                            pp= overlay(bar, pp, c.2+1, c.3-c.2-1,bar)||tr
                                       end
                      end
     if from==3  then do;  pp= overlay(br,  pp, c.3)
                           pp= overlay(bar, pp, c.dest+1, c.3-c.dest-1, bar)
                           pp= overlay(tl,  pp, c.dest)
                      end
     say translate(pp, downs, Bcorners || Tcorners || bar);     say overlay(moveK, pp, 1)
     say translate(pp, verts, Tcorners || Bcorners || bar)
     say translate(pp, downs, Tcorners || Bcorners || bar);     moveK= moveK - 1
     $.from= $.from - 1;      $.dest= $.dest + 1;     _f= $.from + 1;          _t= $.dest
     @.dest._t= @.from._f;    @.from._f= blanks;      call showTowers
     return
/*──────────────────────────────────────────────────────────────────────────────────────*/
mov: if arg(3)==1  then      call dsk arg(1) arg(2)
                   else do;  call mov arg(1),             6 -arg(1) -arg(2),    arg(3) -1
                             call mov arg(1),             arg(2),               1
                             call mov 6 -arg(1) -arg(2),  arg(2),               arg(3) -1
                        end                 /* [↑]  The  MOV  subroutine is recursive,  */
     return                                 /*it uses no variables, is uses BIFs instead*/
/*──────────────────────────────────────────────────────────────────────────────────────*/
showTowers: do j=N  by -1  for N; _=@.1.j @.2.j @.3.j;  if _\=''  then say _; end;  return
```

output when using the default input:




```
The minimum number of moves to solve a  3-disk  Tower of Hanoi is  7
```

Ring (/wiki/Category:Ring)

```
move(4, 1, 2, 3)

func move n, src, dst, via
     if n > 0 move(n - 1, src, via, dst)
        see "" + src + " to " + dst + nl
        move(n - 1, via, dst, src) ok
```

Ruby (/wiki/Category:Ruby)

version 1

```
def move(num_disks, start=0, target=1, using=2)
  if num_disks == 1
   @towers[target] << @towers[start].pop
    puts "Move disk from #{start} to #{target} : #{@towers}"
  else
    move(num_disks-1, start, using, target)
    move(1,           start, target, using)
    move(num_disks-1, using, target, start)
  end
end

n = 5
@towers = [[*1..n].reverse, [], []]
move(n)
```

Output:

```
Move disk from 0 to 1 : [[5, 4, 3, 2], [1], []]
Move disk from 0 to 2 : [[5, 4, 3], [1], [2]]
Move disk from 1 to 2 : [[5, 4, 3], [], [2, 1]]
Move disk from 0 to 1 : [[5, 4], [3], [2, 1]]
Move disk from 2 to 0 : [[5, 4, 1], [3], [2]]
Move disk from 2 to 1 : [[5, 4, 1], [3, 2], []]
Move disk from 0 to 1 : [[5, 4], [3, 2, 1], []]
Move disk from 0 to 2 : [[5], [3, 2, 1], [4]]
Move disk from 1 to 2 : [[5], [3, 2], [4, 1]]
Move disk from 1 to 0 : [[5, 2], [3], [4, 1]]
Move disk from 2 to 0 : [[5, 2, 1], [3], [4]]
Move disk from 1 to 2 : [[5, 2, 1], [], [4, 3]]
Move disk from 0 to 1 : [[5, 2], [1], [4, 3]]
Move disk from 0 to 2 : [[5], [1], [4, 3, 2]]
Move disk from 1 to 2 : [[5], [], [4, 3, 2, 1]]
Move disk from 0 to 1 : [[], [5], [4, 3, 2, 1]]
Move disk from 2 to 0 : [[1], [5], [4, 3, 2]]
Move disk from 2 to 1 : [[1], [5, 2], [4, 3]]
Move disk from 0 to 1 : [[], [5, 2, 1], [4, 3]]
Move disk from 2 to 0 : [[3], [5, 2, 1], [4]]
Move disk from 1 to 2 : [[3], [5, 2], [4, 1]]
Move disk from 1 to 0 : [[3, 2], [5], [4, 1]]
Move disk from 2 to 0 : [[3, 2, 1], [5], [4]]
Move disk from 2 to 1 : [[3, 2, 1], [5, 4], []]
Move disk from 0 to 1 : [[3, 2], [5, 4, 1], []]
Move disk from 0 to 2 : [[3], [5, 4, 1], [2]]
Move disk from 1 to 2 : [[3], [5, 4], [2, 1]]
Move disk from 0 to 1 : [[], [5, 4, 3], [2, 1]]
Move disk from 2 to 0 : [[1], [5, 4, 3], [2]]
Move disk from 2 to 1 : [[1], [5, 4, 3, 2], []]
Move disk from 0 to 1 : [[], [5, 4, 3, 2, 1], []]
```

version 2

```
# solve(source, via, target)
# Example:
# solve([5, 4, 3, 2, 1], [], [])
# Note this will also solve randomly placed disks,
# "place all disk in target with legal moves only".
def solve(*towers)
  # total number of disks
  disks = towers.inject(0){|sum, tower| sum+tower.length}
  x=0 # sequence number
  p towers # initial trace
  # have we solved the puzzle yet?
  while towers.last.length < disks do
    x+=1 # assume the next step
    from = (x&x-1)%3
    to = ((x|(x-1))+1)%3
    # can we actually take from tower?
    if top = towers[from].last
      bottom = towers[to].last
      # is the move legal?
      if !bottom || bottom > top
        # ok, do it!
        towers[to].push(towers[from].pop)
        p towers # trace
      end
    end
  end
end

solve([5, 4, 3, 2, 1], [], [])
```

Output:

```
[[5, 4, 3, 2, 1], [], []]
[[5, 4, 3, 2], [], [1]]
[[5, 4, 3], [2], [1]]
[[5, 4, 3], [2, 1], []]
[[5, 4], [2, 1], [3]]
[[5, 4, 1], [2], [3]]
[[5, 4, 1], [], [3, 2]]
[[5, 4], [], [3, 2, 1]]
[[5], [4], [3, 2, 1]]
[[5], [4, 1], [3, 2]]
[[5, 2], [4, 1], [3]]
[[5, 2, 1], [4], [3]]
[[5, 2, 1], [4, 3], []]
[[5, 2], [4, 3], [1]]
[[5], [4, 3, 2], [1]]
[[5], [4, 3, 2, 1], []]
[[], [4, 3, 2, 1], [5]]
[[1], [4, 3, 2], [5]]
[[1], [4, 3], [5, 2]]
[[], [4, 3], [5, 2, 1]]
[[3], [4], [5, 2, 1]]
[[3], [4, 1], [5, 2]]
[[3, 2], [4, 1], [5]]
[[3, 2, 1], [4], [5]]
[[3, 2, 1], [], [5, 4]]
[[3, 2], [], [5, 4, 1]]
[[3], [2], [5, 4, 1]]
[[3], [2, 1], [5, 4]]
[[], [2, 1], [5, 4, 3]]
[[1], [2], [5, 4, 3]]
[[1], [], [5, 4, 3, 2]]
[[], [], [5, 4, 3, 2, 1]]
```

Run BASIC (/wiki/Category:Run_BASIC)

```
a = move(4, "1", "2", "3")
function move(n, a$, b$, c$)
if n > 0 then
        a = move(n-1, a$, c$, b$)
        print "Move disk from " ; a$ ; " to " ; c$
        a = move(n-1, b$, a$, c$)
end if
end function
```

```
Move disk from 1 to 3
Move disk from 1 to 2
Move disk from 3 to 2
Move disk from 1 to 3
Move disk from 2 to 1
Move disk from 2 to 3
Move disk from 1 to 3
Move disk from 1 to 2
Move disk from 3 to 2
Move disk from 3 to 1
Move disk from 2 to 1
Move disk from 3 to 2
Move disk from 1 to 3
Move disk from 1 to 2
Move disk from 3 to 2
```

Rust (/wiki/Category:Rust)

Translation of: C

```
fn move_(n: i32, from: i32, to: i32, via: i32) {
    if n > 0 {
        move_(n - 1, from, via, to);
        println!("Move disk from pole {} to pole {}", from, to);
        move_(n - 1, via, to, from);
    }
}

fn main() {
    move_(4, 1,2,3);
}
```

SASL (/wiki/Category:SASL)

Copied from SAL manual, Appendix II, answer (3)

```
hanoi 8 'abc"
WHERE
hanoi 0 (a,b,c,) = ()
hanoi n ( a,b,c) = hanoi (n-1) (a,c,b) ,
                   'move a disc from " , a , ' to " , b , NL ,
                   hanoi (n-1) (c,b,a)
?
```

Sather (/wiki/Category:Sather)

Translation of: Fortran

```
class MAIN is

  move(ndisks, from, to, via:INT) is
    if ndisks = 1 then
      #OUT + "Move disk from pole " + from + " to pole " + to + "\n";
    else
      move(ndisks-1, from, via, to);
      move(1, from, to, via);
      move(ndisks-1, via, to, from);
    end;
  end;

  main is
    move(4, 1, 2, 3);
  end;
end;
```

Scala (/wiki/Category:Scala)

```
def (https://scala-lang.org) move(n: Int, from: Int, to: Int, via: Int) : Unit = {
    if (https://scala-lang.org) (n == 1) {
      Console.println("Move disk from pole " + from + " to pole " + to)
    } else (https://scala-lang.org) {
      move(n - 1, from, via, to)
      move(1, from, to, via)
      move(n - 1, via, to, from)
    }
  }
```

This next example is from http://gist.github.com/66925 (https://gist.github.com/66925) it is a translation to Scala of a Prolog solution and solves the problem at compile time

```
object (https://scala-lang.org) TowersOfHanoi {
  import (https://scala-lang.org) scala.reflect.Manifest

  def (https://scala-lang.org) simpleName(m:Manifest[_]):String = {
    val (https://scala-lang.org) name = m.toString
    name.substring(name.lastIndexOf('$')+1)
  }

  trait (https://scala-lang.org) Nat
  final (https://scala-lang.org) class (https://scala-lang.org) _0 extends (https://scala-lang.org) Nat
  final (https://scala-lang.org) class (https://scala-lang.org) Succ[Pre<:Nat] extends (https://scala-lang.org) Nat

  type (https://scala-lang.org) _1 = Succ[_0]
  type (https://scala-lang.org) _2 = Succ[_1]
  type (https://scala-lang.org) _3 = Succ[_2]
  type (https://scala-lang.org) _4 = Succ[_3]

  case (https://scala-lang.org) class (https://scala-lang.org) Move[N<:Nat,A,B,C]()

  implicit (https://scala-lang.org) def (https://scala-lang.org) move0[A,B,C](implicit (https://scala-lang.org) a:Manifest[A],b:Manife
st[B]):Move[_0,A,B,C] = {
       System.out.println("Move from "+simpleName(a)+" to "+simpleName(b));null (https://scala-lang.org)
  }

  implicit (https://scala-lang.org) def (https://scala-lang.org) moveN[P<:Nat,A,B,C](implicit (https://scala-lang.org) m1:Move[P,A,C,B
],m2:Move[_0,A,B,C],m3:Move[P,C,B,A])
   :Move[Succ[P],A,B,C] = null (https://scala-lang.org)

  def (https://scala-lang.org) run[N<:Nat,A,B,C](implicit (https://scala-lang.org) m:Move[N,A,B,C]) = null (https://scala-lang.org)

  case (https://scala-lang.org) class (https://scala-lang.org) Left()
  case (https://scala-lang.org) class (https://scala-lang.org) Center()
  case (https://scala-lang.org) class (https://scala-lang.org) Right()

  def (https://scala-lang.org) main(args:Array[String]){
    run[_2,Left,Right,Center]
  }
}
```

Scheme (/wiki/Category:Scheme)

Recursive Process

```
(define (towers-of-hanoi n from to spare)
  (define (print-move from to)
    (display "Move[")
    (display from)
    (display ", ")
    (display to)
    (display "]")
    (newline))
  (cond ((= n 0) "done")
        (else
         (towers-of-hanoi (- n 1) from spare to)
         (print-move from to)
         (towers-of-hanoi (- n 1) spare to from))))

(towers-of-hanoi 3 "A" "B" "C")
```

Output:

```
Move[A, B]
Move[A, C]
Move[B, C]
Move[A, B]
Move[C, A]
Move[C, B]
Move[A, B]
"done"
```

Seed7 (/wiki/Category:Seed7)

```
const proc: hanoi (in integer: disk, in string: source, in string: dest, in string: via) is func
  begin
    if disk > 0 then
      hanoi(pred(disk), source, via, dest);
      writeln("Move disk " <& disk <& " from " <& source <& " to " <& dest);
      hanoi(pred(disk), via, dest, source);
    end if;
  end func;
```

Sidef (/wiki/Category:Sidef)

Translation of: Perl

```
func hanoi(n, from=1, to=2, via=3) {
    if (n == 1) {
        say "Move disk from pole #{from} to pole #{to}.";
    } else {
        hanoi(n-1, from, via,   to);
        hanoi(  1, from,  to,  via);
        hanoi(n-1,  via,  to, from);
    }
}

hanoi(4);
```

SNOBOL4 (/wiki/Category:SNOBOL4)

```
*       # Note: count is global

        define('hanoi(n,src,trg,tmp)') :(hanoi_end)
hanoi   hanoi = eq(n,0) 1 :s(return)
        hanoi(n - 1, src, tmp, trg)
        count  = count + 1
        output = count ': Move disc from ' src ' to ' trg
        hanoi(n - 1, tmp, trg, src) :(return)
hanoi_end

*       # Test with 4 discs
        hanoi(4,'A','C','B')
end
```

Output:

```
1: Move disc from A to B
2: Move disc from A to C
3: Move disc from B to C
4: Move disc from A to B
5: Move disc from C to A
6: Move disc from C to B
7: Move disc from A to B
8: Move disc from A to C
9: Move disc from B to C
10: Move disc from B to A
11: Move disc from C to A
12: Move disc from B to C
13: Move disc from A to B
14: Move disc from A to C
15: Move disc from B to C
```

Standard ML (/wiki/Category:Standard_ML)

```
   fun hanoi(0, a, b, c) = [] |
       hanoi(n, a, b, c) = hanoi(n-1, a, c, b) @ [(a,b)] @ hanoi(n-1, c, b, a);
```

Stata (/wiki/Category:Stata)

```
function hanoi(n, a, b, c) {
        if (n>0) {
                hanoi(n-1, a, c, b)
                printf("Move from %f to %f\n", a, b)
                hanoi(n-1, c, b, a)
        }
}

hanoi(3, 1, 2, 3)

Move from 1 to 2
Move from 1 to 3
Move from 2 to 3
Move from 1 to 2
Move from 3 to 1
Move from 3 to 2
Move from 1 to 2
```

Swift (/wiki/Category:Swift)

Translation of: JavaScript

```
func hanoi(n:Int, a:String, b:String, c:String) {
    if (n > 0) {
        hanoi(n - 1, a, c, b)
        println("Move disk from \(a) to \(c)")
        hanoi(n - 1, b, a, c)
    }
}

hanoi(4, "A", "B", "C")
```

Swift 2.1

```
func hanoi(n:Int, a:String, b:String, c:String) {
  if (n > 0) {
    hanoi(n - 1, a: a, b: c, c: b)
    print("Move disk from \(a) to \(c)")
    hanoi(n - 1, a: b, b: a, c: c)
  }
}

hanoi(4, a:"A", b:"B", c:"C")
```

Tcl (/wiki/Category:Tcl)

The use of `interp alias` shown is a sort of closure: keep track of the number of moves required

```
interp alias {} hanoi {} do_hanoi 0

proc do_hanoi {count n {from A} {to C} {via B}} {
    if {$n == 1} {
        interp alias {} hanoi {} do_hanoi [incr count]
        puts "$count: move from $from to $to"
    } else {
        incr n -1
        hanoi $n $from $via $to
        hanoi 1  $from $to $via
        hanoi $n $via $to $from
    }
}

hanoi 4
```

Output:

```
1: move from A to B
2: move from A to C
3: move from B to C
4: move from A to B
5: move from C to A
6: move from C to B
7: move from A to B
8: move from A to C
9: move from B to C
10: move from B to A
11: move from C to A
12: move from B to C
13: move from A to B
14: move from A to C
15: move from B to C
```

TI-83 BASIC (/wiki/Category:TI-83_BASIC)

TI-83 BASIC lacks recursion, so technically this task is impossible, however here is a version that uses an iterative method.

```
PROGRAM:TOHSOLVE
0→A
1→B
0→C
0→D
0→M
1→R
While A<1 or A>7
Input "No. of rings=?",A
End
randM(A+1,3)→[C]
[[1,2][1,3][2,3]]→[E]

Fill(0,[C])
For(I,1,A,1)
I?[C](I,1)
End
ClrHome
While [C](1,3)≠1 and [C](1,2)≠1

For(J,1,3)
For(I,1,A)
If [C](I,J)≠0:Then
Output(I+1,3J,[C](I,J))
End
End
End
While C=0
Output(1,3B," ")
1→I
[E](R,2)→J
While [C](I,J)=0 and I≤A
I+1→I
End
[C](I,J)→D
1→I
[E](R,1)→J
While [C](I,J)=0 and I≤A
I+1→I
End
If (D<[C](I,J) and D≠0) or [C](I,J)=0:Then
[E](R,2)→B
Else
[E](R,1)→B
End

1→I
While [C](I,B)=0 and I≤A
I+1→I
End
If I≤A:Then
[C](I,B)→C
0→[C](I,B)
Output(I+1,3B," ")
End
Output(1,3B,"V")
End

While C≠0
```

```
Output(1,3B," ")
If B=[E](R,2):Then
[E](R,1)→B
Else
[E](R,2)→B
End

1→I
While [C](I,B)=0 and I≤A
I+1→I
End
If [C](I,B)=0 or [C](I,B)>C:Then
C→[C](I−1,B)
0→C
M+1→M
End
End
Output(1,3B,"V")
R+1→R
If R=4:Then:1→R:End

End
```

Tiny BASIC (/wiki/Category:Tiny_BASIC)

Tiny BASIC does not have recursion, so only an iterative solution is possible... and it has no arrays, so actually keeping track of individual discs is not feasible.

But as if by magic, it turns out that the source and destination pegs on iteration number n are given by (n&n-1) mod 3 and ((n|n-1) + 1) mod 3 respectively, where & and | are the bitwise and and or operators. Line 40 onward is dedicated to implementing those bitwise operations, since Tiny BASIC hasn't got them natively.

```
 5  PRINT "How many disks?"
    INPUT D
    IF D < 1 THEN GOTO 5
    IF D > 10 THEN GOTO 5
    LET N = 1
10  IF D = 0 THEN GOTO 20
    LET D = D - 1
    LET N = 2*N
    GOTO 10
20  LET X = 0
30  LET X = X + 1
    IF X = N THEN END
    GOSUB 40
    LET S = S - 3*(S/3)
    GOSUB 50
    LET T = T + 1
    LET T = T - 3*(T/3)
    PRINT "Move disc on peg ",S+1," to peg ",T+1
    GOTO 30
40  LET B = X - 1
    LET A = X
    LET S = 0
    LET Z = 2048
45  LET C = 0
    IF B >= Z THEN LET C = 1
    IF A >= Z THEN LET C = C + 1
    IF C = 2 THEN LET S = S + Z
    IF A >= Z THEN LET A = A - Z
    IF B >= Z THEN LET B = B - Z
    LET Z = Z / 2
    IF Z = 0 THEN RETURN
    GOTO 45
50  LET B = X - 1
    LET A = X
    LET T = 0
    LET Z = 2048
55  LET C = 0
    IF B >= Z THEN LET C = 1
    IF A >= Z THEN LET C = C + 1
    IF C > 0 THEN LET T = T + Z
    IF A >= Z THEN LET A = A - Z
    IF B >= Z THEN LET B = B - Z
    LET Z = Z / 2
    IF Z = 0 THEN RETURN
    GOTO 55
```

Output:

```
How many discs?
4
Move disc on peg 1 to peg 3
Move disc on peg 1 to peg 2
Move disc on peg 3 to peg 2
Move disc on peg 1 to peg 3
Move disc on peg 2 to peg 1
Move disc on peg 2 to peg 3
Move disc on peg 1 to peg 3
Move disc on peg 1 to peg 2
Move disc on peg 3 to peg 2
Move disc on peg 3 to peg 1
Move disc on peg 2 to peg 1
Move disc on peg 3 to peg 2
Move disc on peg 1 to peg 3
Move disc on peg 1 to peg 2
Move disc on peg 3 to peg 2
```

Toka (/wiki/Category:Toka)

```
value| sa sb sc n |
[ to sc to sb to sa to n ] is vars!
[ ( num from to via -- )
  vars!
  n 0 <>
  [
    n sa sb sc
    n 1- sa sc sb recurse
    vars!
    ." Move a ring from " sa . ." to " sb . cr
    n 1- sc sb sa recurse
  ] ifTrue
] is hanoi
```

True BASIC (/wiki/Category:True_BASIC)

Translation of: FreeBASIC

```
DECLARE SUB hanoi

SUB hanoi(n, desde , hasta, via)
    IF n > 0 THEN
       CALL hanoi(n - 1, desde, via, hasta)
       PRINT "Mover disco"; n; "desde posición"; desde; "hasta posición"; hasta
       CALL hanoi(n - 1, via, hasta, desde)
    END IF
END SUB

PRINT "Tres discos"
PRINT
CALL hanoi(3, 1, 2, 3)
PRINT
PRINT "Cuatro discos"
PRINT
CALL hanoi(4, 1, 2, 3)
PRINT
PRINT "Pulsa un tecla para salir"
END
```

TSE SAL (/wiki/Category:TSE_SAL)

```
// library: program: run: towersofhanoi: recursive: sub <description></description> <version>1.0.0.0.0</version> <version control></ve
rsion control> (filenamemacro=runprrsu.s) [kn, ri, tu, 07-02-2012 19:54:23]
PROC PROCProgramRunTowersofhanoiRecursiveSub( INTEGER totalDiskI, STRING fromS, STRING toS, STRING viaS, INTEGER bufferI )
 IF ( totalDiskI == 0 )
  RETURN()
 ENDIF
 PROCProgramRunTowersofhanoiRecursiveSub( totalDiskI - 1, fromS, viaS, toS, bufferI )
 AddLine( Format( "Move disk", " ", totalDiskI, " ", "from peg", " ", "'", fromS, "'", " ", "to peg", " ", "'", toS, "'" ), bufferI )
 PROCProgramRunTowersofhanoiRecursiveSub( totalDiskI - 1, viaS, toS, fromS, bufferI )
END

// library: program: run: towersofhanoi: recursive <description></description> <version>1.0.0.0.6</version> <version control></version
 control> (filenamemacro=runprtre.s) [kn, ri, tu, 07-02-2012 19:40:45]
PROC PROCProgramRunTowersofhanoiRecursive( INTEGER totalDiskI, STRING fromS, STRING toS, STRING viaS )
 INTEGER bufferI = 0
 PushPosition()
 bufferI = CreateTempBuffer()
 PopPosition()
 PROCProgramRunTowersofhanoiRecursiveSub( totalDiskI, fromS, toS, viaS, bufferI )
 GotoBufferId( bufferI )
END

PROC Main()
STRING s1[255] = "4"
IF ( NOT ( Ask( "program: run: towersofhanoi: recursive: totalDiskI = ", s1, _EDIT_HISTORY_ ) ) AND ( Length( s1 ) > 0 ) ) RETURN() EN
DIF
 PROCProgramRunTowersofhanoiRecursive( Val( s1 ), "source", "target", "via" )
END
```

uBasic/4tH (/wiki/Category:UBasic/4tH)

Translation of: C

```
Proc  _Move(4, 1,2,3)                 ' 4 disks, 3 poles
End

_Move Param(4)
  If (a@ > 0) Then
    Proc _Move (a@ - 1, b@, d@, c@)
    Print "Move disk from pole ";b@;" to pole ";c@
    Proc _Move (a@ - 1, d@, c@, b@)
  EndIf
Return
```

UNIX Shell (/wiki/Category:UNIX_Shell)

Works with: bash (/wiki/Bash)

```
#!/bin/bash

move()
{
  local n="$1"
  local from="$2"
  local to="$3"
  local via="$4"

  if [[ "$n" == "1" ]]
  then
    echo "Move disk from pole $from to pole $to"
  else
    move $((n - 1)) $from $via $to
    move 1 $from $to $via
    move $((n - 1)) $via $to $from
  fi
}

move $1 $2 $3 $4
```

Ursala (/wiki/Category:Ursala)

```
#import nat

move = ~&al^& ^rlPlrrPCT/~&arhthPX ^|W/~& ^|G/predecessor ^/~&htxPC ~&zyxPC

#show+

main = ^|T(~&,' -> '--)* move/4 <'start','end','middle'>
```

Output:

```
start -> middle
start -> end
middle -> end
start -> middle
end -> start
end -> middle
start -> middle
start -> end
middle -> end
middle -> start
end -> start
middle -> end
start -> middle
start -> end
middle -> end
```

VBScript (/wiki/Category:VBScript)

Derived from the BASIC256 version.

```
Sub Move(n,fromPeg,toPeg,viaPeg)
        If n > 0 Then
                Move n-1, fromPeg, viaPeg, toPeg
                WScript.StdOut.Write "Move disk from " & fromPeg & " to " & toPeg
                WScript.StdOut.WriteBlankLines(1)
                Move n-1, viaPeg, toPeg, fromPeg
        End If
End Sub

Move 4,1,2,3
WScript.StdOut.Write("Towers of Hanoi puzzle completed!")
```

Output:

```
Move disk from 1 to 3
Move disk from 1 to 2
Move disk from 3 to 2
Move disk from 1 to 3
Move disk from 2 to 1
Move disk from 2 to 3
Move disk from 1 to 3
Move disk from 1 to 2
Move disk from 3 to 2
Move disk from 3 to 1
Move disk from 2 to 1
Move disk from 3 to 2
Move disk from 1 to 3
Move disk from 1 to 2
Move disk from 3 to 2
Towers of Hanoi puzzle completed!
```

Vedit macro language (/wiki/Category:Vedit_macro_language)

This implementation outputs the results in current edit buffer.

```
#1=1; #2=2; #3=3; #4=4          // move 4 disks from 1 to 2
Call("MOVE_DISKS")
Return

// Move disks
// #1 = from, #2 = to, #3 = via, #4 = number of disks
//
:MOVE_DISKS:
if (#4 > 0) {
    Num_Push(1,4)
        #9=#2; #2=#3; #3=#9; #4--       // #1 to #3 via #2
        Call("MOVE_DISKS")
    Num_Pop(1,4)

    Ins_Text("Move a disk from ")       // move one disk
    Num_Ins(#1, LEFT+NOCR)
    Ins_Text(" to ")
    Num_Ins(#2, LEFT)

    Num_Push(1,4)
        #9=#1; #1=#3; #3 = #9; #4--     // #3 to #2 via #1
        Call("MOVE_DISKS")
    Num_Pop(1,4)
}
Return
```

Vim Script (/wiki/Category:Vim_Script)

```
function TowersOfHanoi(n, from, to, via)
  if (a:n > 1)
    call TowersOfHanoi(a:n-1, a:from, a:via, a:to)
  endif
  echom("Move a disc from " . a:from . " to " . a:to)
  if (a:n > 1)
    call TowersOfHanoi(a:n-1, a:via, a:to, a:from)
  endif
endfunction

call TowersOfHanoi(4, 1, 3, 2)
```

Output:

```
Move a disc from 1 to 2
Move a disc from 1 to 3
Move a disc from 2 to 3
Move a disc from 1 to 2
Move a disc from 3 to 1
Move a disc from 3 to 2
Move a disc from 1 to 2
Move a disc from 1 to 3
Move a disc from 2 to 3
Move a disc from 2 to 1
Move a disc from 3 to 1
Move a disc from 2 to 3
Move a disc from 1 to 2
Move a disc from 1 to 3
Move a disc from 2 to 3
```

Visual Basic .NET (/wiki/Category:Visual_Basic_.NET)

```
Module TowersOfHanoi
    Sub MoveTowerDisks(ByVal disks As Integer, ByVal fromTower As Integer, ByVal toTower As Integer, ByVal viaTower As Integer)
        If disks > 0 Then
            MoveTowerDisks(disks - 1, fromTower, viaTower, toTower)
            System.Console.WriteLine("Move disk {0} from {1} to {2}", disks, fromTower, toTower)
            MoveTowerDisks(disks - 1, viaTower, toTower, fromTower)
        End If
    End Sub

    Sub Main()
        MoveTowerDisks(4, 1, 2, 3)
    End Sub
End Module
```

Wren (/wiki/Category:Wren)

Translation of: Kotlin

```
class Hanoi {
    construct new(disks) {
        _moves = 0
        System.print("Towers of Hanoi with %(disks) disks:\n")
        move(disks, "L", "C", "R")
        System.print("\nCompleted in %(_moves) moves\n")
    }

    move(n, from, to, via) {
        if (n > 0) {
            move(n - 1, from, via, to)
            _moves = _moves + 1
            System.print("Move disk %(n) from %(from) to %(to)")
            move(n - 1, via, to, from)
        }
    }
}

Hanoi.new(3)
Hanoi.new(4)
```

Output:

```
Towers of Hanoi with 3 disks:

Move disk 1 from L to C
Move disk 2 from L to R
Move disk 1 from C to R
Move disk 3 from L to C
Move disk 1 from R to L
Move disk 2 from R to C
Move disk 1 from L to C

Completed in 7 moves

Towers of Hanoi with 4 disks:

Move disk 1 from L to R
Move disk 2 from L to C
Move disk 1 from R to C
Move disk 3 from L to R
Move disk 1 from C to L
Move disk 2 from C to R
Move disk 1 from L to R
Move disk 4 from L to C
Move disk 1 from R to C
Move disk 2 from R to L
Move disk 1 from C to L
Move disk 3 from R to C
Move disk 1 from L to R
Move disk 2 from L to C
Move disk 1 from R to C

Completed in 15 moves
```

XPL0 (/wiki/Category:XPL0)

```
code Text=12;

proc MoveTower(Discs, From, To, Using);
int  Discs, From, To, Using;
[if Discs > 0 then
    [MoveTower(Discs-1, From, Using, To);
    Text(0, "Move from ");  Text(0, From);
    Text(0, " peg to ");  Text(0, To);  Text(0, " peg.^M^J");
    MoveTower(Discs-1, Using, To, From);
    ];
];

MoveTower(3, "left", "right", "center")
```

Output:

```
Move from left peg to right peg.
Move from left peg to center peg.
Move from right peg to center peg.
Move from left peg to right peg.
Move from center peg to left peg.
Move from center peg to right peg.
Move from left peg to right peg.
```

XQuery (/wiki/Category:XQuery)

```
declare function local:hanoi($disk as xs:integer, $from as xs:integer,
    $to as xs:integer, $via as xs:integer) as element()*
{
  if($disk > 0)
  then (
    local:hanoi($disk - 1, $from, $via, $to),
    <move disk='{$disk}'><from>{$from}</from><to>{$to}</to></move>,
    local:hanoi($disk - 1, $via, $to, $from)
  )
  else ()
};

<hanoi>
{
  local:hanoi(4, 1, 2, 3)
}
</hanoi>
```

Output:

```
<?xml version="1.0" encoding="UTF-8"?>
<hanoi>
   <move disk="1">
      <from>1</from>
      <to>3</to>
   </move>
   <move disk="2">
      <from>1</from>
      <to>2</to>
   </move>
   <move disk="1">
      <from>3</from>
      <to>2</to>
   </move>
   <move disk="3">
      <from>1</from>
      <to>3</to>
   </move>
   <move disk="1">
      <from>2</from>
      <to>1</to>
   </move>
   <move disk="2">
      <from>2</from>
      <to>3</to>
   </move>
   <move disk="1">
      <from>1</from>
      <to>3</to>
   </move>
   <move disk="4">
      <from>1</from>
      <to>2</to>
   </move>
   <move disk="1">
      <from>3</from>
      <to>2</to>
   </move>
   <move disk="2">
      <from>3</from>
      <to>1</to>
   </move>
   <move disk="1">
      <from>2</from>
      <to>1</to>
   </move>
   <move disk="3">
      <from>3</from>
      <to>2</to>
   </move>
   <move disk="1">
      <from>1</from>
      <to>3</to>
   </move>
   <move disk="2">
      <from>1</from>
      <to>2</to>
   </move>
   <move disk="1">
      <from>3</from>
      <to>2</to>
   </move>
</hanoi>
```

XSLT (/wiki/Category:XSLT)

```
<xsl:template name="hanoi">
<xsl:param name="n"/>
<xsl:param name="from">left</xsl:param>
<xsl:param name="to">middle</xsl:param>
<xsl:param name="via">right</xsl:param>
  <xsl:if test="$n &gt; 0">
    <xsl:call-template name="hanoi">
      <xsl:with-param name="n"    select="$n - 1"/>
      <xsl:with-param name="from" select="$from"/>
      <xsl:with-param name="to"   select="$via"/>
      <xsl:with-param name="via"  select="$to"/>
    </xsl:call-template>
    <fo:block>
      <xsl:text>Move disk from </xsl:text>
      <xsl:value-of select="$from"/>
      <xsl:text> to </xsl:text>
      <xsl:value-of select="$to"/>
    </fo:block>
    <xsl:call-template name="hanoi">
      <xsl:with-param name="n"    select="$n - 1"/>
      <xsl:with-param name="from" select="$via"/>
      <xsl:with-param name="to"   select="$to"/>
      <xsl:with-param name="via"  select="$from"/>
    </xsl:call-template>
  </xsl:if>
</xsl:template>
```

```
<xsl:call-template name="hanoi"><xsl:with-param name="n" select="4"/></xsl:call-template>
```

Yabasic (/wiki/Category:Yabasic)

```
sub hanoi(ndisks, startPeg, endPeg)
    if ndisks then
        hanoi(ndisks-1, startPeg, 6-startPeg-endPeg)
        //print "Move disk ", ndisks, " from ", startPeg, " to ", endPeg
        hanoi(ndisks-1, 6-startPeg-endPeg, endPeg)
    end if
end sub

print "Be patient, please.\n\n"
print "Hanoi 1 ellapsed ... ";

t1 = peek("millisrunning")
hanoi(22, 1, 3)
t2 = peek("millisrunning")
print t2-t1, " ms"


sub hanoi2(n, from, to_, via)
    if n = 1 then
        //print "Move from ", from, " to ", to_
    else
        hanoi2(n - 1, from, via , to_ )
        hanoi2(1    , from, to_ , via )
        hanoi2(n - 1, via , to_ , from)
    end if
end sub

print "Hanoi 2 ellapsed ... ";
hanoi2(22, 1, 3, 2)
print peek("millisrunning") - t2, " ms"
```

zkl (/wiki/Category:Zkl)

Translation of: C

```
fcn move(n, from,to,via){
   if (n>0){
      move(n-1, from,via,to);
      println("Move disk from pole %d to pole %d".fmt(from, to));
      move(n-1, via,to,from);
   }
}
move(3, 1,2,3);
```

Output:

```
Move disk from pole 1 to pole 2
Move disk from pole 1 to pole 3
Move disk from pole 2 to pole 3
Move disk from pole 1 to pole 2
Move disk from pole 3 to pole 1
Move disk from pole 3 to pole 2
Move disk from pole 1 to pole 2
```

Categories (/wiki/Special:Categories): Programming Tasks (/wiki/Category:Programming_Tasks)
| Classic CS problems and programs (/wiki/Category:Classic_CS_problems_and_programs) | Recursion (/wiki/Category:Recursion) | Games (/wiki/Category:Games)
| 11l (/wiki/Category:11l) | 360 Assembly (/wiki/Category:360_Assembly) | 8080 Assembly (/wiki/Category:8080_Assembly)
| 8086 Assembly (/wiki/Category:8086_Assembly) | 8th (/wiki/Category:8th) | ActionScript (/wiki/Category:ActionScript) | Ada (/wiki/Category:Ada)
| Agena (/wiki/Category:Agena) | ALGOL 68 (/wiki/Category:ALGOL_68) | ALGOL-M (/wiki/Category:ALGOL-M) | ALGOL W (/wiki/Category:ALGOL_W)
| AmigaE (/wiki/Category:AmigaE) | APL (/wiki/Category:APL) | AppleScript (/wiki/Category:AppleScript) | ARM Assembly (/wiki/Category:ARM_Assembly)
| Arturo (/wiki/Category:Arturo) | AutoHotkey (/wiki/Category:AutoHotkey) | AutoIt (/wiki/Category:AutoIt) | AWK (/wiki/Category:AWK) | BASIC (/wiki/Category:BASIC)
| BASIC256 (/wiki/Category:BASIC256) | Batch File (/wiki/Category:Batch_File) | BBC BASIC (/wiki/Category:BBC_BASIC) | BCPL (/wiki/Category:BCPL)
| Befunge (/wiki/Category:Befunge) | Bracmat (/wiki/Category:Bracmat) | Brainf*** (/wiki/Category:Brainf***) | C (/wiki/Category:C) | C sharp (/wiki/Category:C_sharp)
| C++ (/wiki/Category:C%2B%2B) | Clojure (/wiki/Category:Clojure) | COBOL (/wiki/Category:COBOL) | CoffeeScript (/wiki/Category:CoffeeScript)
| Common Lisp (/wiki/Category:Common_Lisp) | D (/wiki/Category:D) | Dart (/wiki/Category:Dart) | Dc (/wiki/Category:Dc) | Delphi (/wiki/Category:Delphi)
| Dyalect (/wiki/Category:Dyalect) | E (/wiki/Category:E) | EasyLang (/wiki/Category:EasyLang) | Eiffel (/wiki/Category:Eiffel) | Ela (/wiki/Category:Ela)
| Elena (/wiki/Category:Elena) | Elixir (/wiki/Category:Elixir) | Emacs Lisp (/wiki/Category:Emacs_Lisp) | Erlang (/wiki/Category:Erlang) | ERRE (/wiki/Category:ERRE)
| Excel (/wiki/Category:Excel) | Ezhil (/wiki/Category:Ezhil) | F Sharp (/wiki/Category:F_Sharp) | Factor (/wiki/Category:Factor) | FALSE (/wiki/Category:FALSE)
| Fermat (/wiki/Category:Fermat) | FOCAL (/wiki/Category:FOCAL) | Forth (/wiki/Category:Forth) | Fortran (/wiki/Category:Fortran) | FreeBASIC (/wiki/Category:FreeBASIC)
| Frink (/wiki/Category:Frink) | FutureBasic (/wiki/Category:FutureBasic) | Fōrmulæ (/wiki/Category:F%C5%8Drmul%C3%A6) | GAP (/wiki/Category:GAP)
| Go (/wiki/Category:Go) | Groovy (/wiki/Category:Groovy) | Haskell (/wiki/Category:Haskell) | HolyC (/wiki/Category:HolyC) | Icon (/wiki/Category:Icon)
| Unicon (/wiki/Category:Unicon) | Inform 7 (/wiki/Category:Inform_7) | Io (/wiki/Category:Io) | Ioke (/wiki/Category:Ioke) | IS-BASIC (/wiki/Category:IS-BASIC)
| J (/wiki/Category:J) | Java (/wiki/Category:Java) | JavaScript (/wiki/Category:JavaScript) | Joy (/wiki/Category:Joy) | Jq (/wiki/Category:Jq) | Jsish (/wiki/Category:Jsish)
| Julia (/wiki/Category:Julia) | K (/wiki/Category:K) | Klingphix (/wiki/Category:Klingphix) | Kotlin (/wiki/Category:Kotlin) | Lambdatalk (/wiki/Category:Lambdatalk)
| Lasso (/wiki/Category:Lasso) | Liberty BASIC (/wiki/Category:Liberty_BASIC) | Lingo (/wiki/Category:Lingo) | Logo (/wiki/Category:Logo) | Logtalk (/wiki/Category:Logtalk
| LOLCODE (/wiki/Category:LOLCODE) | Lua (/wiki/Category:Lua) | M2000 Interpreter (/wiki/Category:M2000_Interpreter) | MAD (/wiki/Category:MAD)
| Maple (/wiki/Category:Maple) | Mathematica (/wiki/Category:Mathematica) | MATLAB (/wiki/Category:MATLAB) | MiniScript (/wiki/Category:MiniScript)
| MIPS Assembly (/wiki/Category:MIPS_Assembly) | MK-61/52 (/wiki/Category:%D0%9C%D0%9A-61/52) | Modula-2 (/wiki/Category:Modula-2)
| Modula-3 (/wiki/Category:Modula-3) | Monte (/wiki/Category:Monte) | Nemerle (/wiki/Category:Nemerle) | NetRexx (/wiki/Category:NetRexx)
| NewLISP (/wiki/Category:NewLISP) | Nim (/wiki/Category:Nim) | Objeck (/wiki/Category:Objeck) | Objective-C (/wiki/Category:Objective-C)
| OCaml (/wiki/Category:OCaml) | Octave (/wiki/Category:Octave) | Oforth (/wiki/Category:Oforth) | Oz (/wiki/Category:Oz) | PARI/GP (/wiki/Category:PARI/GP)
| Pascal (/wiki/Category:Pascal) | Perl (/wiki/Category:Perl) | Phix (/wiki/Category:Phix) | PHL (/wiki/Category:PHL) | PHP (/wiki/Category:PHP)
| PicoLisp (/wiki/Category:PicoLisp) | PL/I (/wiki/Category:PL/I) | PlainTeX (/wiki/Category:PlainTeX) | Pop11 (/wiki/Category:Pop11) | PostScript (/wiki/Category:PostScript)
| PowerShell (/wiki/Category:PowerShell) | Prolog (/wiki/Category:Prolog) | PureBasic (/wiki/Category:PureBasic) | Python (/wiki/Category:Python)
| VPython (/wiki/Category:VPython) | Quackery (/wiki/Category:Quackery) | Quite BASIC (/wiki/Category:Quite_BASIC) | R (/wiki/Category:R)
| Racket (/wiki/Category:Racket) | Raku (/wiki/Category:Raku) | Rascal (/wiki/Category:Rascal) | Raven (/wiki/Category:Raven) | REBOL (/wiki/Category:REBOL)
| Retro (/wiki/Category:Retro) | REXX (/wiki/Category:REXX) | Ring (/wiki/Category:Ring) | Ruby (/wiki/Category:Ruby) | Run BASIC (/wiki/Category:Run_BASIC)
| Rust (/wiki/Category:Rust) | SASL (/wiki/Category:SASL) | Sather (/wiki/Category:Sather) | Scala (/wiki/Category:Scala) | Scheme (/wiki/Category:Scheme)
| Seed7 (/wiki/Category:Seed7) | Sidef (/wiki/Category:Sidef) | SNOBOL4 (/wiki/Category:SNOBOL4) | Standard ML (/wiki/Category:Standard_ML)
| Stata (/wiki/Category:Stata) | Swift (/wiki/Category:Swift) | Tcl (/wiki/Category:Tcl) | TI-83 BASIC (/wiki/Category:TI-83_BASIC) | Tiny BASIC (/wiki/Category:Tiny_BASIC)
| Toka (/wiki/Category:Toka) | True BASIC (/wiki/Category:True_BASIC) | TSE SAL (/wiki/Category:TSE_SAL) | UBasic/4tH (/wiki/Category:UBasic/4tH)
| UNIX Shell (/wiki/Category:UNIX_Shell) | Ursala (/wiki/Category:Ursala) | VBScript (/wiki/Category:VBScript)
| Vedit macro language (/wiki/Category:Vedit_macro_language) | Vim Script (/wiki/Category:Vim_Script) | Visual Basic .NET (/wiki/Category:Visual_Basic_.NET)
| Wren (/wiki/Category:Wren) | XPL0 (/wiki/Category:XPL0) | XQuery (/wiki/Category:XQuery) | XSLT (/wiki/Category:XSLT) | Yabasic (/wiki/Category:Yabasic)
| Zkl (/wiki/Category:Zkl) | Puzzles (/wiki/Category:Puzzles)