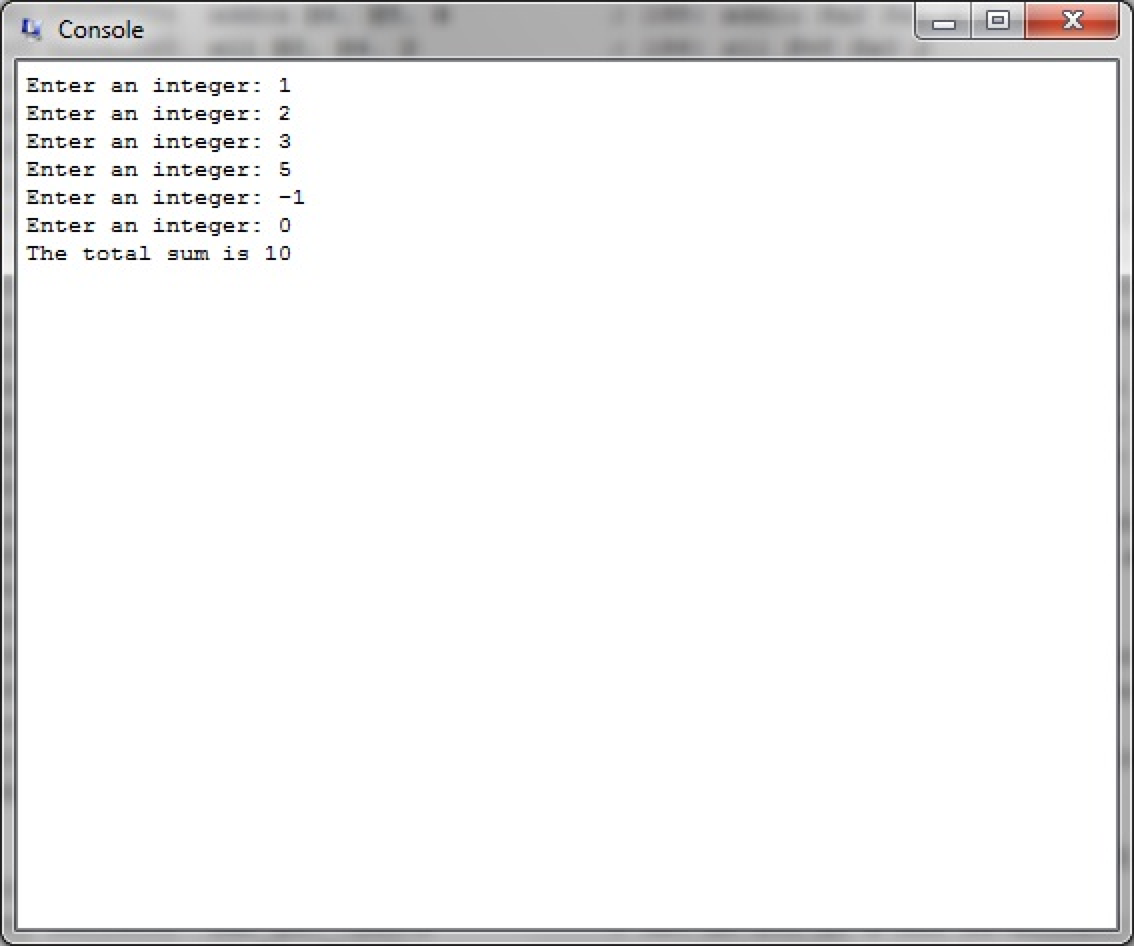
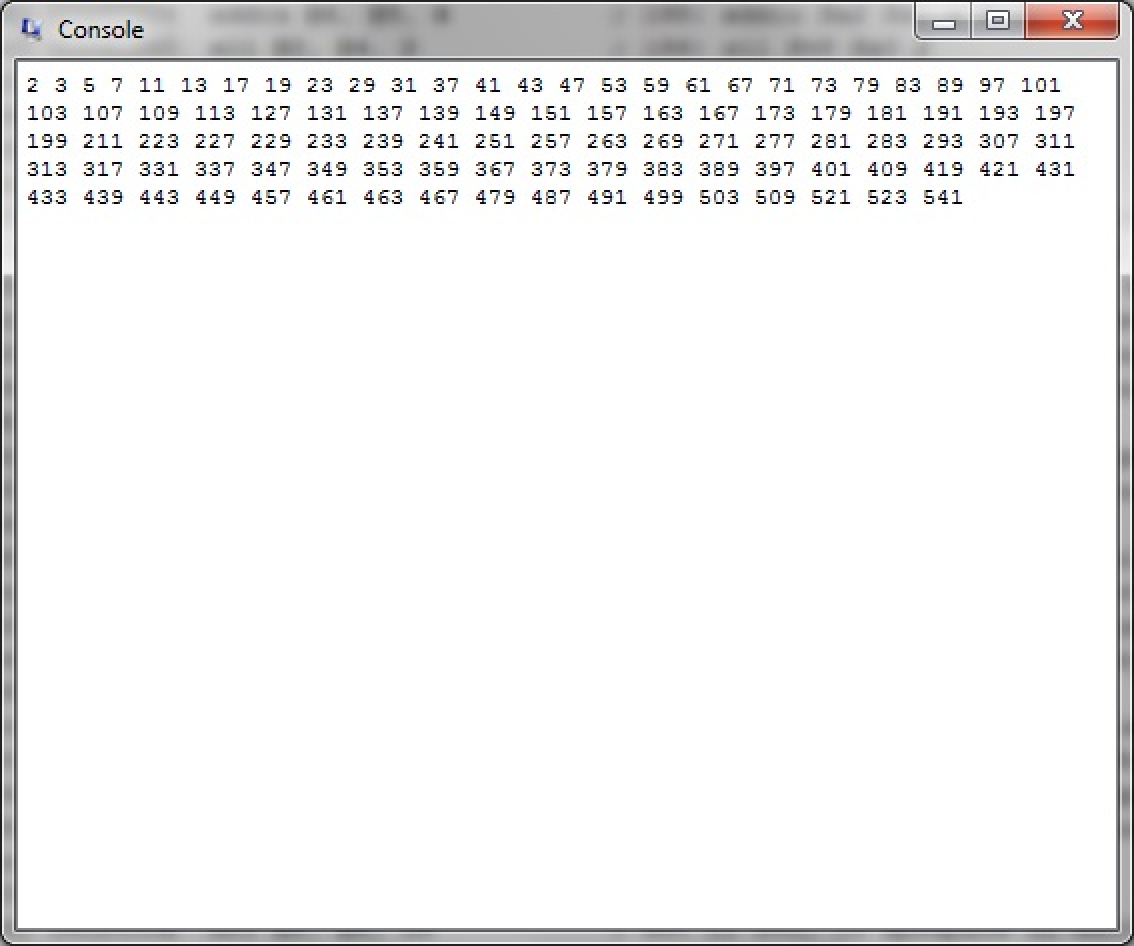
```
# Title: Project 1 Part I               Filename: Project 1 Part I.s
# Author: Dan Cassidy                   Date: 2015-02-17
# Description: This program will take a number of integers, add
#              them up, and display the sum.
# Input: A series of integers
# Output: The sum of the integers
################# Data segment #####################
.data
inputMsg:   .asciiz     "Enter an integer: "
outputMsg:  .asciiz     "The total sum is "

################# Code segment #####################
.text
.globl  main
main:                                   #main program entry
        xor     $s0, $s0, $s0           #initialize sum to 0

loop:   li      $v0, 4                  #prepare to print string
        la      $a0, inputMsg           #choose string inputMsg
        syscall                         #print inputMsg
        li      $v0, 5                  #prepare to read input number
        syscall                         #read input number
        add     $s0, $s0, $v0           #add the input number to sum
        bne     $v0, $zero, loop        #if (input!=0) continue loop

        li      $v0, 4                  #prepare to print string
        la      $a0, outputMsg          #choose string outputMsg
        syscall                         #print outputMsg
        li      $v0, 1                  #prepare to print sum
        addi    $a0, $s0, 0             #set output to sum
        syscall                         #print sum

        li      $v0, 10                 #prepare to exit program
        syscall                         #exit program
```



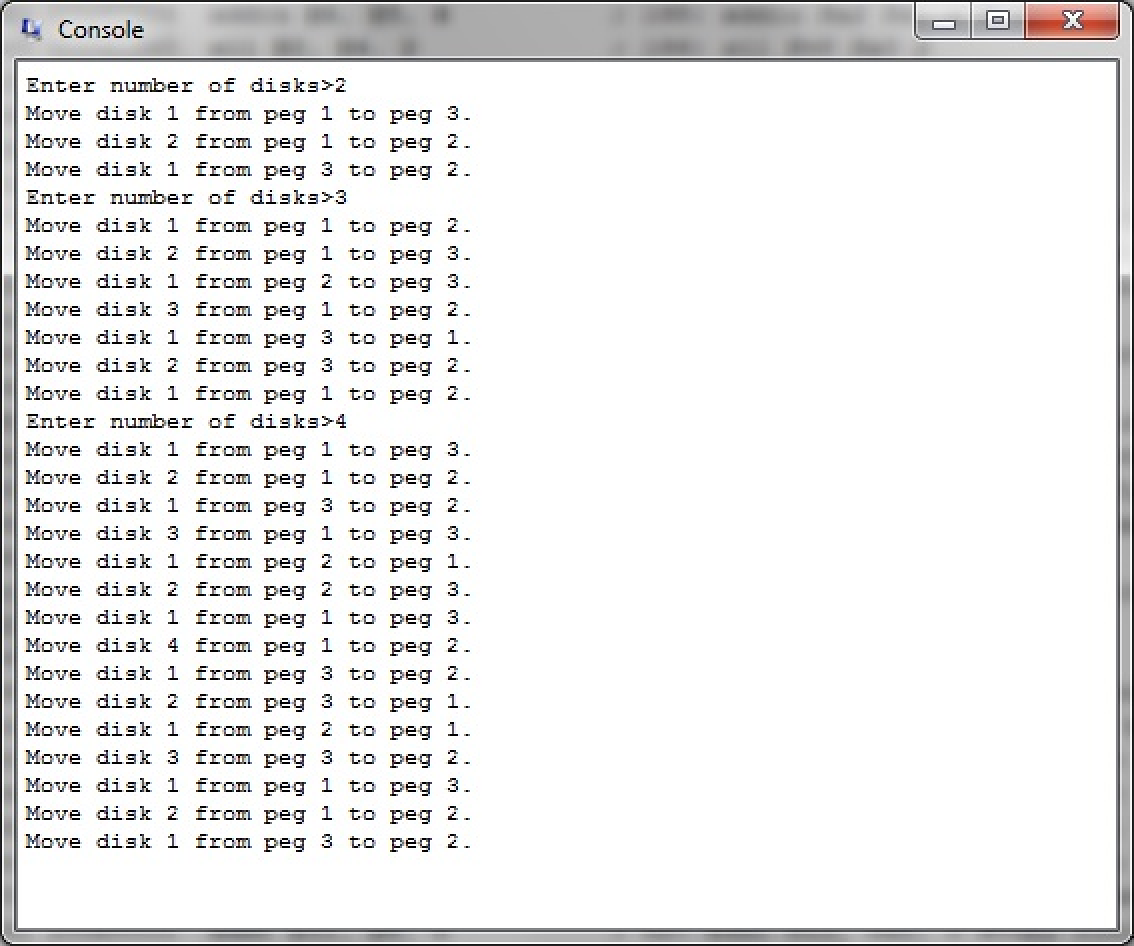
```
Enter an integer: 1
Enter an integer: 2
Enter an integer: 3
Enter an integer: 5
Enter an integer: -1
Enter an integer: 0
The total sum is 10
```

```
# Title: Project 1 Part II              Filename: Project 1 Part II.s
# Author: Dan Cassidy                   Date: 2015-03-02
# Description: This program outputs the first 100 prime numbers.
# Input: Nothing
# Output: The first 100 prime numbers.
# Variables:
#   main: $s0 = numPrimes, $s1 = potentialPrime
#   test_prime: $a0 = n, $t1 = halfN, $t2 = i
################# Data segment #####################
.data

################# Code segment #####################
.text
.globl  main
main:                                   #main program entry
        addi    $s7, $zero, 2           #load 2 because it's used a lot

        addi    $s0, $zero, 100         #set the number of primes to find (numPrimes)

        #2 is the only even prime number, so output that separately,
        #then only odds have to be checked for primeness
        addi    $s0, $s0, -1            #decrement numPrimes because 2 is first prime
        li      $v0, 1                  #prepare to output 2
        addi    $a0, $s7, 0             #set output to 2
        syscall                         #output 2
        li      $v0, 11                 #prepare to output a space
        addi    $a0, $zero, 32          #set output to a space
        syscall                         #output a space

        addi    $s1, $zero, 1           #initialize potentialPrime to 1
loop_m: addi    $s1, $s1, 2             #increment potentialPrime by 2
        addi    $a0, $s1, 0             #load argument for test_prime
        jal     test_prime              #call test_prime to test potentialPrime
        beq     $v0, $zero, loop_m      #if (test_prime returns 0), jump to loop_m
        addi    $s0, $s0, -1            #otherwise, decrement numPrimes (one less to find)
        #the following two statements aren't needed due to the way
        #values line up; they are kept in for reference only
        #li     $v0, 1                  #prepare to output potentialPrime
        #addi   $a0, $s1, 0             #set output to potentialPrime
        syscall                         #output potentialPrime
        li      $v0, 11                 #prepare to output a space
        addi    $a0, $zero, 32          #set output to a space
        syscall                         #output a space
        bne     $s0, $zero, loop_m      #if (numPrimes != 0), jump to loop_m

exit_m: li      $v0, 10                 #prepare to exit program
        syscall                         #exit program



# Function: test_prime
# Description: Tests a number and determines whether it is prime.
# Input:
#   $a0, holds the number to be tested, must be odd and >= 3
# Output:
#   $v0, holds 1 if the number is a prime and 0 if not
# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
test_prime:                             #test_prime function entry
        div     $a0, $s7                #divide n by 2
        mflo    $t1                     #get n / 2

        addi    $t2, $zero, 3           #set i to 3
        slt     $t0, $t1, $t2           #set if (halfN < i)
        bne     $t0, $zero, exit_t      #if (halfN < i)[i <= halfN], jump to exit_t
loop_t: div     $a0, $t2                #divide n / i
        mfhi    $t0                     #get n % i
        bne     $t0, $zero, skip_t      #if (n % i != 0), jump to skip_t
        addi    $v0, $zero, 0           #set return value to false
        jr      $ra                     #return to main
skip_t: addi    $t2, $t2, 2             #increment i by 2
        slt     $t0, $t1, $t2           #set if (halfN < i)
        beq     $t0, $zero, loop_t      #if (i <= halfN), jump to loop_t

exit_t: addi    $v0, $zero, 1           #set return value to true
        jr      $ra                     #return to main
```



```
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101
103 107 109 113 127 131 137 139 149 151 157 163 167 173 179 181 191 193 197
199 211 223 227 229 233 239 241 251 257 263 269 271 277 281 283 293 307 311
313 317 331 337 347 349 353 359 367 373 379 383 389 397 401 409 419 421 431
433 439 443 449 457 461 463 467 479 487 491 499 503 509 521 523 541
```

```
# Title: Project 1 Part III             Filename: Project 1 Part III.s
# Author: Dan Cassidy                   Date: 2015-03-03
# Description: This program will take the given number of disks and
#   solve the Towers of Hanoi puzzle using a recursive function.
# Input: The number of disks to use in the puzzle.
# Output: The steps taken to solve the puzzle.
# Variables:
#   main: $a0 = n
#   hanoi: $a0 = n, $a1 = start, $a2 = finish, $a3 = extra
#       *note: start, finish, and extra move around some
################# Data segment #####################
.data
getDisks:   .asciiz     "Enter number of disks>"
moveDisk:   .asciiz     "Move disk "
fromPeg:    .asciiz     " from peg "
toPeg:      .asciiz     " to peg "
endLine:    .asciiz     ".\n"

################# Code segment #####################
.text
.globl  main
main:                                   #main program entry
        li      $v0, 4                  #prepare to output getDisks
        la      $a0, getDisks           #set output to getDisks
        syscall                         #output getDisks

        li      $v0, 5                  #prepare to input n
        syscall                         #input n

        addi    $a0, $v0, 0             #load n into argument 1 of hanoi
        addi    $a1, $zero, 1           #load start into argument 2 of hanoi
        addi    $a2, $zero, 2           #load finish into argument 3 of hanoi
        addi    $a3, $zero, 3           #load extra into argument 4 of hanoi

        jal     hanoi                   #call hanoi

        li      $v0, 10                 #prepare to exit program
        syscall                         #exit program




# Function: hanoi
# Description: Given inputs, will solve 'towers of hanoi' recursively
# Input:
#   $a0, holds the disk number
#   $a1, holds the number designator of the starting peg
#   $a2, holds the number designator of the final peg
#   $a3, holds the number designator of the extra peg
# Output:
#   A single step in the process of solving the puzzle.
# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
hanoi:                                  #function entry
        beq     $a0, $zero, end_h       #if (n==0), jump to end_h
        addi    $sp, $sp, -20           #make room in stack
        sw      $ra, 16($sp)            #save $ra to stack
        sw      $a0, 12($sp)            #save n to stack
        sw      $a1, 8($sp)             #save start to stack
        sw      $a2, 4($sp)             #save final to stack
        sw      $a3, 0($sp)             #save extra to stack

        addi    $a0, $a0, -1            #decrement n (load n-1 into argument 1)
        addi    $t3, $a2, 0             #copy finish to a temp location
        addi    $a2, $a3, 0             #load extra into argument 3
        addi    $a3, $t3, 0             #load finish into argument 4
        jal     hanoi                   #hanoi(n-1, start, extra, finish)
        lw      $s0, 12($sp)            #restore n from stack
        lw      $a1, 0($sp)             #restore extra from stack into argument 2
        lw      $a2, 4($sp)             #restore finish from stack into argument 3
        lw      $a3, 8($sp)             #restore start from stack into argument 4
        addi    $sp, $sp, 16            #move $sp back but keep $ra saved

        li      $v0, 4                  #prepare to output moveDisk
        la      $a0, moveDisk           #set output to moveDisk
        syscall                         #output moveDisk
        li      $v0, 1                  #prepare to output n
        addi    $a0, $s0, 0             #set output to n
        syscall                         #output n
        li      $v0, 4                  #prepare to output fromPeg
        la      $a0, fromPeg            #set output to fromPeg
        syscall                         #output fromPeg
        li      $v0, 1                  #prepare to output start
        addi    $a0, $a3, 0             #set output to start
        syscall                         #output start
        li      $v0, 4                  #prepare to output toPeg
        la      $a0, toPeg              #set output to toPeg
        syscall                         #output toPeg
        li      $v0, 1                  #prepare to output final
        addi    $a0, $a2, 0             #set output to final
        syscall                         #output final
        li      $v0, 4                  #prepare to output endLine
        la      $a0, endLine            #set output to endLine
        syscall                         #output endLine

        addi    $a0, $s0, -1            #load n-1 into argument 1
        jal     hanoi                   #hanoi(n-1, extra, finish, start)
        lw      $ra, 0($sp)             #restore $ra from stack
        addi    $sp, $sp, 4             #move $sp; all memory reclaimed
end_h:  jr  $ra                         #return from function
```



```
Enter number of disks>2
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.
Enter number of disks>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.
Enter number of disks>4
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.
Move disk 4 from peg 1 to peg 2.
Move disk 1 from peg 3 to peg 2.
Move disk 2 from peg 3 to peg 1.
Move disk 1 from peg 2 to peg 1.
Move disk 3 from peg 3 to peg 2.
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.
```