;; fibonacci sequence

allocate-registers n

allocate-registers zero one two

allocate-registers sp comparison

allocate-registers ifOne ifTwo end loop

allocate-registers val

read n

li one 1

li two 2

li zero 0

li sp 0

li ifOne ifOneL

li ifTwo ifTwoL

li end endL

li loop loopL

st n sp

add sp sp one

loopL:

jeqz sp end

sub sp sp one

ld n sp

sne comparison n one

jeqz comparison ifOne

sne comparison n two

jeqz comparison ifTwo

sub n n one

st n sp

add sp sp one

sub n n one

st n sp

add sp sp one

j loop

ifOneL:

add val val zero

j loop

ifTwoL:

add val val one

j loop

endL:

write val

halt

;; recursive power of 2

allocate-registers one two n sp

allocate-registers baseCase end store load cont val

li one 1

li two 2

read n

li sp 0

li baseCase baseCaseL

li end endL

li store storeL

li load loadL

li cont endL

storeL:

jeqz n baseCase ; if n is 0

st two sp

add sp sp one

st cont sp

add sp sp one

sub n n one

li cont loadL

j store

loadL:

sub sp sp one

ld cont sp

sub sp sp one

ld two sp

mul val val two

j cont

baseCaseL:

li val 1

j cont

endL:

write val

halt

;;ifPrime1

allocate-registers n one zero loop ifPrime ifNotPrime a comparison remainder first-loop stop end val

read n

li one 1

li zero 0

li stop stopL

li loop loopL

li ifPrime ifPrimeL

li ifNotPrime ifNotPrimeL

li end endL

li first-loop first-loopL

first-loopL:

li a 0

sub n n one

sgt comparison n one

jeqz comparison stop

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

add a a n

loopL:

sub a a one

sgt comparison a one

jeqz comparison ifPrime

rem remainder n a

sne comparison remainder zero

jeqz comparison ifNotPrime

j loop

ifPrimeL:

li val 1

j end

ifNotPrimeL:

li val 0

j end

endL:

jeqz val first-loop

write n

j first-loop

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

stopL:

halt

//This program takes a DNA string and turn it into an RNA string

fun main (args: Array<String>){

var DNA = args[0]

var RNA = ""

for (i in DNA){ //loop through the character in DNA

when (i){

'G' -> RNA += 'C' //convert the character to RNA sequence

'C' -> RNA += 'G'

'T' -> RNA += 'A'

'A' -> RNA += 'U'

else -> { //if read in a weird letter, then return a message

println ("Not a DNA string")

System.exit(-1)

}

}

}

println(RNA)

}

// LeapYear

var n = 0

fun main (args: Array<String>){

if (args.isEmpty()){

println("Please provide a year")

}

else{

n = args[0].toInt()

when{

n%400 == 0 -> println("Leap Year")

n%100 == 0 -> println("Not Leap Year")

n%4 == 0 -> println("Leap Year")

else -> println("Not Leap Year")

}

}

}

//DreamCatcher

import processing.core.PApplet

import kotlin.math.\*

var n = 0

var angle = 0F

var prob = 0F

var rd = 0F

fun main(args:Array<String>){

n = args[0].toInt()

prob = args[1].toFloat()

PApplet.main("DreamCatcher")

}

class DreamCatcher : PApplet(){

override fun settings(){

size(400,400)

}

override fun setup(){

background(255F)

}

var A1 = FloatArray(2\*n+1)

override fun draw(){

translate(200F,200F)

strokeWeight(5F)

for (i in 1..n){

point(100F\*cos(angle), 100F\*sin(angle))

A1[i] = 100F \* cos(angle)

A1[n+i] = 100F \* sin(angle)

angle = angle + TWO\_PI/n

}

for (i in 1..n){

for (j in (i+1)..n){

rd = random(0F,1F)

if (rd < prob){

stroke(200F)

strokeWeight(1F)

line(A1[i],A1[i+n],A1[j],A1[j+n])

}

}

}

// PolarCoordinator

class PolarCoordinate1 : PApplet(){

override fun settings(){

size(400,400)

}

override fun setup(){

background(255F)

}

override fun draw(){

translate(200F,200F)

for (i in 0 until 50000){

angle = angle + TWO\_PI/50000F

r = sin(n\*angle)

x = r \* sin(angle)

y = r \* cos(angle)

point(100F\*x,100F\*y)

}

noLoop()

}

}

//range and average

import kotlin.math.\*

fun main (args:Array<String>){

var max = args[0].toInt()

var min = args[0].toInt()

var sum = 0.0

for (i in 0 until args.size){

val a = args[i].toInt()

max = max(a,max)

min = min(a, min)

sum += a

}

val average = sum/args.size

println("range = $min..$max")

println("average = $average")

}

// hexagon

import processing.core.PApplet

var angle = 0F

fun main (args:Array<String>){

PApplet.main("hexagon")

}

class hexagon : PApplet(){

override fun settings(){

size(240,240)

}

override fun setup(){

background(255f,0f,0f)

}

override fun draw(){

translate(120F,120F)

val length = 100f

fill(0f,255f,0f)

beginShape()

for (i in 0 until 6){

vertex(length.toFloat() \* cos(angle), length.toFloat() \* sin(angle))

angle = angle + PI/3

}

endShape()

noLoop()

}

}