# Soluciones Segundo Parcial - Programación 1
# Instituto de Computación - Facultad de Ingeniería
# Julio 2021

## Ejercicio 1 (30 pts)

Las siguientes declaraciones representan una cónica cerrada.

```
type
 Punto = record          TipoConica = (circunferencia,
           x,y: Real;                  elipse);
         end;

Conica = record case clase: TipoConica of
            circunferencia : (radio: Real; centro: Punto);
            elipse : (foco1, foco2 : Punto);
          end;

 TipoResul = record
                 figura : TipoConica;
                 enCentro : Boolean
             end;
```

**parte a) (15 pts)** Escribir el procedimiento `focoEnCentro` que devuelve en el parámetro `resultado` el tipo de cónica y el valor booleano `true` si la cónica tiene alguno de sus focos en el centro de ejes cartesianos (0.0, 0.0), o `false` si no. En el caso de una circunferencia, su centro es el foco.

```
procedure focoEnCentro (conica : Conica; var resultado : TipoResul);
```

**Solución:**

```
procedure focoEnCentro (conica : Conica; var resultado : TipoResul);
begin
   resultado.figura := conica.clase;
   if conica.clase = circunferencia then
      resultado.enCentro := (conica.centro.x = 0.0) and (conica.centro.y = 0.0)
   else
      resultado.enCentro := ((conica.foco1.x = 0.0) and (conica.foco1.y = 0.0 ) or
                             (conica.foco2.x = 0.0) and (conica.foco2.y = 0.0 ))
end;
```

**parte b) (15 pts)** Dado un arreglo de cónicas

```
const MAX = (* entero positivo *)
type ArrConicas = array [1..MAX] of Conica;
```

Escribir la función `primeraCfa` que devuelve el índice del primer elemento del arreglo que es circunferencia o -1 si no hay ninguna

```
function primeraCfa (conicas : ArrConicas): Integer;
```

**Solución:**

```
function primeraCfa (conicas : ArrConicas) : Integer;
var i : Integer;
begin
   i := 1;
   while (i <= Max) and (conicas[i].clase <> circunferencia) do
       i := i + 1;
   if (i <= Max) then primeraCfa := i
                 else primeraCfa := -1
end;
```

## Ejercicio 2 (20 pts)

```
type  Lista  = ^Nodo;
      Nodo = record
                 valor: Integer;
                 sig: Lista;
             end;
```

Escribir el siguiente procedimiento que dados una lista ordenada en forma creciente y un nuevo valor, inserta el valor en la lista manteniendo el orden

```
   procedure InsertOrdenado (valor: Integer; var L: Lista);
```

**Solución:**

```
   procedure InsertOrdenado (valor : Integer; var L : Lista);
   var
     nuevo, cursor : TipoLista;
   begin
      new(nuevo);
      nuevo^.valor := valor;
      if (L = NIL) or (valor < L^.valor) then
        begin
            nuevo^.sig := L;
            L := nuevo
        end
      else
        begin
            cursor := L;
            while (cursor^.sig <> NIL) and (valor > cursor^.sig^.valor) do
                cursor := cursor^.sig;
            nuevo^.sig := cursor^.sig;
            cursor^.sig := nuevo
        end;
   end;
```

## Ejercicio 3 (12 pts)

Escribir la salida del programa cuando **m** se carga de la entrada con **el último dígito** de **su** CI (antes del dígito verificador). Por ejemplo, si su CI es 1.234.567-8, el último dígito es 7.

```
program alcance;
var m, z: Integer;
procedure proc1 (x,y : Integer; var m : Integer);
   var z : Integer;
   function fun1 (a, b : Integer) : Integer;
   begin
       if a >= 4 then fun1 := a-b
       else fun1 := b-a;
   end;
   begin
      z := x + y;
      m := fun1 (x,y);
      writeln (m, ' ', z);
   end;
begin
  readln (m); (* ultimo digito de antes del guion *);
  proc1 (m, m+1, z);
  writeln (m, ' ', z);
end.
```

**Solución:**

| m | Salida 1 | Salida 2 |
|---|---|---|
| 0 | 1 1 | 0 1 |
| 1 | 1 3 | 1 1 |
| 2 | 1 5 | 2 1 |
| 3 | 1 7 | 3 1 |
| 4 | -1 9 | 4 -1 |
| 5 | -1 11 | 5 -1 |
| 6 | -1 13 | 6 -1 |
| 7 | -1 15 | 7 -1 |
| 8 | -1 17 | 8 -1 |
| 9 | -1 19 | 9 -1 |

## Ejercicio 4 (13 pts)

Dadas las siguientes declaraciones:

```
const N = {entero positivo}
type  ArregloNum = array [1..N] of Integer;
      ArregloMul = record
                      multiplos: array[1..N] of Integer;
                      tope: 0..N;
                   end;
```

Escribir el procedimiento `indicesMultiplos` que retorna en el parámetro `resultado` los índices de los elementos del arreglo `enteros` que son múltiplos de la constante `cte`

```
procedure indicesMultiplos (enteros: ArregloNum; cte: Integer; var resultado: ArregloMul);
```

**Solución:**

```
procedure indicesMultiplos (enteros: ArregloNum; cte: Integer; var resultado: ArregloMul);
var i : Integer;
begin
    resultado.tope := 0;
    for i :=1 to N do
        if (enteros[i] mod cte = 0) then
        begin
            resultado.tope := resultado.tope + 1;
            resultado.multiplos[resultado.tope] := i;
        end;
end;
```