# Programación 1
# Segundo parcial 2018
# Soluciones

## Ejercicio 1

### Parte a

```
procedure leoValores (var arr : Valores);
var
   num : Integer;
begin
   arr.tope := 0;
   read(num);
   while (arr.tope < MaxDatos) and (num >= 0) do
   begin
      arr.tope := arr.tope + 1;
      arr.info[arr.tope] := num;
      read(num)
   end
end;
```

### Parte b (versión 1)

Primera propuesta de solución.

```
function sumanNPos (arr : Valores; num : Natural; pos : Integer) : Boolean;
(* suma valores desde pos hasta alcanzar o superar num, o llegar al tope *)
var i, acum : Integer;
begin
   acum := arr.info[pos];
   i := pos + 1;
   while (i <= arr.tope) and (acum < num) do
   begin
      acum := acum + arr.info[i];
      i := i + 1
   end;
   sumanNPos := acum = num
end;

function consecutivosSumanN (arr : Valores; num : Natural) : Boolean;
var i : Integer;
begin
   i := 1;
   while (i <= arr.tope) and not sumanNPos(arr, num, i) do
      i := i + 1;
   consecutivosSumanN := i <= arr.tope
end;
```

**Parte b (versión 2)**
Segunda propuesta de solución, con mejoras en la eficiencia.

```
function consecutivosSumanN (arr : Valores; num : Natural) : Boolean;
var i, j, suma : Integer;
begin
   if arr.tope = 0 then
      consecutivosSumanN := false
   else
   begin
      suma := arr.info[1];
      i := 1;
      j := 1;
      while (j < arr.tope) and (suma <> num) do
      begin
         while (j < arr.tope) and (suma < num) do
         begin
            j := j + 1;
            suma := suma + arr.info[j]
         end;
         while (i <= j) and (suma > num) do
         begin
            suma := suma - arr.info[i];
            i := i + 1
         end
      end;
      consecutivosSumanN := suma = num
   end
end;
```

## Ejercicio 2

### Parte a

```
function posteriorHora(h1, h2 : Hora) : Boolean;
begin
   posteriorHora := (h1.hora > h2.hora) or
                    ( (h1.hora = h2.hora) and
                      ((h1.min > h2.min) or ((h1.min = h2.min) and
                                             (h1.seg > h2.seg))))
end;
```

### Parte b

```
function ultimoIngreso(park : Parking) : Integer;
var i, j, indMax : Integer;
begin
   (* busco el primer lugar ocupado para inicializar indMax*)
   i := 1;
   while (i <= MAX_AUTOS) and not park[i].ocupado do
      i := i+1;
   if i <= MAX_AUTOS then
   (* si hay algun lugar ocupado, inicializo indMax y recorro el resto *)
   begin
      indMax := i;
      for j := i+1 to MAX_AUTOS do
         if park[j].ocupado and
                        posteriorHora(park[j].hora_ingreso,park[indMax].hora_ingreso) then
            indMax := j;
      ultimoIngreso := park[indMax].matricula
   end
   else
   (* no hay ningun lugar ocupado, retorno -1 *)
      ultimoIngreso := -1
end;
```

**Ejercicio 3**

Solución con un solo puntero auxiliar para iterar.

```
procedure InsPos(val : Char; pos : Integer; var l : Lista );
var p, q : Lista;
      cont : Integer;
begin
   new(p);
   p^.elem := val;
   if (pos = 1) or (l = nil) then
   begin
      p^.sig :=l;
      l := p
   end
   else
   begin
      q := l;
      cont := 2;
      while (q^.sig <> nil) and (cont < pos) do
      begin
         q := q^.sig;
         cont := cont + 1
      end;
      p^.sig := q^.sig;
      q^.sig := p
   end
end;
```

Solución con dos punteros auxiliares para iterar.

```
procedure InsPos(val : Char; pos : Integer; var l : Lista );
var p, q, h : Lista;
    cont : Integer;
begin
   new(p);
   p^.elem := val;
   if (pos = 1) or (l = nil) then
   begin
      p^.sig:=l;
      l := p
   end
   else
   begin
      h:= l;
      q:= l^.sig;
      cont := 2;
      while (q <> nil) and (cont < pos) do
      begin
         h:=h^.sig;
         q:=q^.sig;
         cont := cont + 1
      end;
      h^.sig:=p;
      p^.sig:=q
   end
end;
```

**Ejercicio 4**

El primer número es el dígito de la CI.

```
0     6 0 0 2
1    12 2 1 4
2    18 4 2 6
3    24 6 3 8
4    30 8 4 10
5    36 10 5 12
6    42 12 6 14
7    48 14 7 16
8    54 16 8 18
9    60 18 9 20
```