# Examen - Programación 1

**Instituto de Computación**
**Diciembre 2024**

**Leer con atención:**

- Todos los programas o fragmentos de programas deben ser escritos en el lenguaje Pascal tal como fue dado en el curso.
- En todos los problemas se evaluará, además de la lógica correcta, la utilización de un buen estilo de programación de acuerdo a los criterios impartidos en el curso.
- Escribir las respuestas de un solo lado de la hoja. Entregar solamente las hojas de solución escritas a lápiz.
- En cada hoja entregada se debe incluir nombre, cédula y qué número de hoja es. En la primera hoja se debe incluir además la cantidad total de hojas entregadas.

## Ejercicio 1 (44 puntos)

Dadas las siguientes declaraciones:

```
const M = ...; { M > 0}
type
   Lista     = ^TipoCelda;
   TipoCelda = record
                 dato : integer;
                 sig  : Lista
               end;

   ArrTope   = record
                 elems : array [1..M] of integer;
                 tope  : 0 .. M
               end;
```

### a) (20 puntos)

Escriba el procedimiento:

```
procedure eliminarPrimerParLst(var lst : Lista);
```

que elimina el primer elemento de la lista `lst` que sea par. Asuma que la lista tiene al menos un elemento.

Ejemplo:

Si `lst` al momento de invocar al procedimiento es $\boxed{5} \rightarrow \boxed{1} \rightarrow \boxed{4} \rightarrow \boxed{7} \rightarrow \boxed{6} \rightarrow \boxed{3}$,
luego de la invocación será $\boxed{5} \rightarrow \boxed{1} \rightarrow \boxed{7} \rightarrow \boxed{6} \rightarrow \boxed{3}$.

**Solución:**

```
procedure eliminarPrimerParLst(var lst : Lista);

   procedure borrar(var l : Lista);
   var aux :  Lista;
   begin
      aux := l;
      l   := l^.sig;
      dispose(aux)
   end;
var
   iter :  Lista;
begin
   if lst^.dato mod 2 = 0 then
      borrar(lst)
   else
   begin
      iter := lst;
      while (iter^.sig <> NIL) and (iter^.sig^.dato mod 2 <> 0) do
         iter := iter^.sig;

      if iter^.sig <> NIL then
         borrar(iter^.sig)
   end
end;
```

**b) (24 puntos)**

Escriba el procedimiento:

```
procedure eliminarParesArr(var arr : ArrTope);
```

que elimina **todos** los elementos del arreglo con tope `arr` que sean pares.

**Sugerencia**: Para poder realizar todo en una sola recorrida utilice dos índices: uno para recorrer cada elemento y otro para implementar la eliminación.

**Solución:**

```
procedure eliminarParesArr(var arr : ArrTope);
var i, t : integer;
begin
   t := 0;
   for i := 1 to arr.tope do
      if arr.elems[i] mod 2 <> 0 then
      begin
         t := t+1;
         arr.elems[t] := arr.elems[i]
      end;
   arr.tope := t
end;
```

## Ejercicio 2 (16 puntos)

Considere el siguiente programa

```
program alcance;
var a, b: integer;

  procedure p(a: integer; var b: integer);

    function f(b: integer): integer;
    begin
          ...
    end;

  begin
    a := a * 3;
    b := f(b);
    a := a * b;
  end;

begin
  read(a, b);
  p(b, a);
  writeln('El triple del primer valor leído es: ', a)
end.
```

en el que no se incluye la implementación de la función `f`. Implemente la función `f`, con una única línea de código, para que el programa imprima el triple del primer valor leído.

Ejemplo: Si la entrada es `2 5`, el programa deberá imprimir `El triple del primer valor leído es: 6`

**Solución:**

```
    function f(b: integer): integer;
    begin
          f := b * 3
    end;
```

## Ejercicio 3 (40 puntos)

El ajedrez es un juego que se juega sobre un tablero cuadriculado de NxN casillas (en realidad siempre N es 8, pero la estructura es más general) y que juegan dos jugadores, desplazando piezas sobre él. Las piezas de uno de los jugadores son blancas y la del otro negras. Al principio del juego cada jugador tiene dieciséis piezas: un rey, una dama, dos alfiles, dos caballos, dos torres y ocho peones. Cada jugador puede capturar piezas del otro, retirándolas así del juego.

Dada la siguiente estructura para representar al juego:

```
const N = ...; { N > 0}
type
  TipoPieza = (Rey, Dama, Alfil, Caballo, Torre, Peon);
  Color     = (Blanco, Negro);

  Pieza     = record
                tipo  : TipoPieza;
                color : Color
              end;

  Casilla = record
              case ocupada : boolean of
                true  : ( p : Pieza) ;
                false : ()
              end;

  Tablero   = array [1..N,1..N] of Casilla;

  SetPiezas = array [TipoPieza] of integer;
```

### a) (15 puntos)

Implemente una función que dado un `Tablero` y un `Color`, retorne la cantidad de piezas de ese color que quedan en el tablero

```
function cantidadColor(t : Tablero; c : Color) : integer;
```

### Solución:

```
function cantidadColor(t : Tablero; c : Color) : integer;
var
   i, j, cant : integer;
begin
   cant := 0;
   for i := 1 to N do
      for j := 1 to N do
         if t[i,j].ocupada and (t[i,j].p.color = c) then cant := cant + 1;

   cantidadColor := cant
end;
```

Solución alternativa asumiendo que el tablero es válido:

```
function cantidadColor(t : Tablero; c : Color) : integer;
var
   i, j, cant : integer;
begin
   cant := 0;

   i := 1;
   while (i <= N) and (cant < 16) do
   begin
      j := 1;
      while (j <= N) and (cant < 16) do
      begin
         if t[i,j].ocupada and (t[i,j].p.color = c) then cant := cant + 1;
         j := j+1
      end;
      i := i + 1
      end;

   cantidadColor := cant
end;
```

**b) (25 puntos)**

Implemente una función que determina si un tablero tiene una configuración de piezas válida, es decir que de cada color tiene como máximo: un rey, una dama, dos alfiles, dos caballos, dos torres y ocho peones.

**Sugerencia**: Utilice un SetPiezas auxiliar para cada color, para llevar el conteo de piezas de ese color en el tablero.

```
function tableroValido(t : Tablero) : boolean;
```

**Solución:**

```
function tableroValido(t : Tablero) : boolean;
var
   i, j    : integer;
   valido  : boolean;
   maximos : array [Color] of SetPiezas;
begin
   valido := true;
   i := 1;
   j := 1;

   maximos[Negro][Rey]     := 1; maximos[Blanco][Rey]     := 1;
   maximos[Negro][Dama]    := 1; maximos[Blanco][Dama]    := 1;
   maximos[Negro][Alfil]   := 2; maximos[Blanco][Alfil]   := 2;
   maximos[Negro][Caballo] := 2; maximos[Blanco][Caballo] := 2;
   maximos[Negro][Torre]   := 2; maximos[Blanco][Torre]   := 2;
   maximos[Negro][Peon]    := 8; maximos[Blanco][Peon]    := 8;

   while valido and (i <= N) do
   begin
      while valido and (j <= N) do
      begin
         if t[i,j].ocupada then
         begin
            maximos[t[i][j].p.color, t[i][j].p.tipo] := maximos[t[i][j].p.color, t[i][j].p.tipo] - 1;
            valido := maximos[t[i][j].p.color, t[i][j].p.tipo] >= 0
         end;
         j := j + 1
      end;
      i := i + 1;
   end;

   tableroValido := valido
end;
```