Examen de Programación 1
Instituto de Computación - Facultad de Ingeniería
Julio 2017

Leer con atención

- Todos los códigos deben ser escritos en el lenguaje **Pascal** tal como fue dado en el curso. A grandes rasgos, este es el Pascal estándar con los siguientes agregados: utilización de `else` en la instrucción `case` y evaluación por circuito corto de las operaciones booleanas (`and` y `or`).
- 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. De esta manera se restarán puntos entre otros conceptos por: mala o nula indentación, mala utilización de las estructuras de control, código confuso e innecesariamente largo, programas ineficientes, utilización de variables globales, pasaje incorrecto de parámetros, etc. No obstante, por razones prácticas, no exigimos que incluya comentarios en los códigos que escriba en la prueba.
- **Escriba su nombre completo y cédula en todas las hojas. Numere todas las hojas y escriba la cantidad total de hojas.**
- **Escriba de un solo lado de la hoja y comience cada ejercicio en una nueva hoja**.

Ejercicio 1

Dadas las siguientes declaraciones:

```
const
    MAX = ...;
type
    Dia = 1..31;
    Mes = 1..12;
    Anio = 1900..3000;
    Fecha = record
                dd: Dia;
                mm: Mes;
                aa: Anio;
            end;
    Fechas = array [1..MAX] of Fecha;
```

Parte a)

Escribir la función **`EsPosterior`** que, dadas dos fechas, determina si la primera es posterior a la segunda. Por ejemplo, la fecha (12,7,2017) es posterior a la fecha (30,6,2017).

```
function EsPosterior (f1, f2: Fecha): boolean;
```

Parte b)

Escribir la función **`IndMaxFecha`** que, dados un arreglo de fechas y una posición **`pos`** del arreglo, devuelve el índice del arreglo en el que se ubica la fecha máxima de todas las almacenadas entre las posiciones **`1`** y **`pos`** (ambas inclusive). Se considera que una fecha es mayor que otra si es posterior a ella. Asuma que no hay fechas repetidas en el arreglo.

```
function IndMaxFecha (arre: Fechas; pos: integer): integer;
```

Parte c)

Escribir el procedimiento **`OrdenarFechas`** que, dado un arreglo de fechas (sin repetidas), lo ordena cronológicamente, aplicando el algoritmo de ordenación por selección visto en el curso.

```
procedure OrdenarFechas (VAR arre: Fechas);
```

Ejercicio 2

Dadas las siguientes declaraciones:

```
const
    MAX = 10;

type
    ArregloTope = record
                      valores: array[1 .. MAX] of Integer;
                      tope: 0 .. MAX
                  end;
```

Escribir el procedimiento **normalizarArreglo** que, dado un arreglo con tope de enteros arrTope y un valor N (menor o igual que MAX), modifica arrTope de modo que contenga N valores válidos.
La modificación se realiza según el criterio siguiente:

- Si el valor del tope es mayor que N, se conservan los N últimos valores válidos, manteniendo el orden original.
- Si el valor del tope es menor que N, se conservan todos los valores válidos de arrTope y se completa hasta N con el primer valor negativo que haya en el arreglo, o con el valor 0 si no hay ningún negativo.
- Si el valor del tope es igual a N, arrTope no se modifica.

Observación: Asuma que arrTope nunca está vacío.

```
procedure normalizarArreglo (var arrTope : arregloTope, N: Integer);
```

Ejemplos para MAX=10

Valor de N	arrTope.tope original	arrTope.valores original	arrTope.tope modificado	arrTope.valores modificado
3	5	[3, -2, 1, 125, -480, ...]	3	[1, 125, -480, ...]
8	3	[3, -2, -1, ...]	8	[3, -2, -1, -2, -2, -2, -2, -2, ...]
8	3	[3, 2, 11, ...]	8	[3, 2, 11, 0, 0, 0, 0, 0, ...]

Ejercicio 3

Considere los siguientes tipos, que representan una lista de registros con información sobre los pagos de impuestos de diferentes personas:

```
type
    listaPersonas = ^Persona;
    Persona = record
                codigo : Integer;
                sig : listaPersonas;
                case tiene_saldo: boolean of
                                false: ();
                                true: (saldoIRPF : Integer;
                                       saldoIASS: Integer)
                end;
```

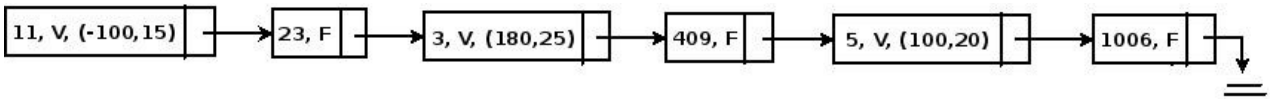
La información está compuesta por un código que identifica a la persona y un campo booleano que indica si la persona tiene o no saldo sobrante de sus impuestos pagos. Si tiene saldo, puede provenir del IRPF (sueldos) o del IASS (pasividades).

Escribir la función:

```
function cant_personas_saldo (valor: Integer; lista: listaPersonas): Integer;
```

que, dado un valor y una lista, devuelva la cantidad de personas de la lista tales que la suma de sus saldos exceda el valor.

En el ejemplo se muestran 6 elementos con campos “codigo”, el valor del campo "tiene_saldo" (F o V) y el par de saldos. Para el valor 100 la función debe devolver 2.



Ejercicio 4

Determinar la salida del siguiente programa, cuando se le da como entrada el último dígito antes del guión de su número de cédula. Por ejemplo, si su cédula es 1234567-8, la entrada será 7.

```
PROGRAM departamentos;
VAR
   total:INTEGER;

Procedure colonia(VAR a: Integer; VAR x: Integer);
VAR
   total : integer;
BEGIN
   total:= a+1;
   a:= total;
   x:= x*2
END;

Procedure lavalleja(a: Integer; x: Integer);
BEGIN
   total:= a+1;
   x:= x*2
END;

Procedure florida(a: Integer; VAR x: Integer);
BEGIN
   a:= a+1;
   x:= x*2;
   WriteLn(a,' ',x,' ',total);
   lavalleja(a,x)
END;

procedure mostrar(var valor : integer);
begin
   valor:= (total + valor) div 2;
   writeln(total)
end;

BEGIN
   readln(total);
   florida(total,total);
   mostrar(total);
   colonia(total,total);
   mostrar(total)
END.
```