Fundamentos de la programación y la informática
Examen Parcial. 2 de Diciembre de 2025
Grados en ingeniería aeroespacial. Turno de tarde
Universidad Rey Juan Carlos

Ejercicio 1 (2 puntos)

Contesta en el fichero `logica.txt`. No le cambies el nombre. Sean las expresiones

```
e1 := not (not a and not b) or not (not c and d) or not (e and f);

e2 := not (x < -2) or (y = 9);
```

1. A partir de e1, aplica De Morgan y escribe una expresión lógica equivalente, en Pascal, intentando que sea más clara, de nombre s1.
2. A partir de e2, escribe una expresión lógica equivalente, en Pascal, más legible, de nombre s2.
3. Escribe una expresión n1, de la forma más clara posible, que sea la negación de s1.
4. Escribe una expresión n2, de la forma más clara posible, que sea la negación de s2.

Solución

```
s1 := (a or b) or (c or not d) or (not e or not f);
s2 := (x >= -2) or (y = 9);
n1 := (not a and not b) and (not c and d) and (e and f);
n2 := (x < -2) and (y <> 9);
```

También es válido

```
s1 := a or b or c or not d or not e or not f;
n1 := not a and not b and not c and d and e and f;
```

Ejercicio 2 (8 puntos)

Sean dos puntos del plano $a = (x_a, y_a)$ y $b = (x_b, y_b)$

- El punto medio (x_c, y_c) entre ambos se obtiene promediando las coordenadas correspondientes:

$$x_c = \frac{x_a + x_b}{2}, \qquad y_c = \frac{y_a + y_b}{2}$$

- Si $a \neq b$, definen una recta cuya ecuación en forma general

$$\alpha x + \beta y + \gamma = 0$$

puede determinarse mediante las expresiones [1]

$$\alpha = y_a - y_b, \qquad \beta = x_b - x_a, \qquad \gamma = x_a y_b - x_b y_a.$$

[1] La ecuación general de la recta $\alpha x + \beta y + \gamma = 0$ no es única: cualquier múltiplo no nulo de (α, β, γ) describe la misma recta. Pero en este ejercicio puedes ignorarlo. Basta con aplicar las fórmulas dadas para calcular α, β y γ.

Escribe un programa en Object Pascal (el Pascal que vemos en clase) llamado **exactamente** `puntos.pas` según la siguiente especificación

1. Usará este tipo de datos (y solo este tipo de datos) para contener puntos en el plano

```
type
  TipoPunto = record
    x: Real;
    y: Real;
  end;
```

2. Tendrá una función que reciba dos puntos y devuelva el punto medio entre ambos.

3. Tendrá un procedimiento con

   - Un parámetro de entrada para el primer punto.
   - Otro parámetro de entrada para el segundo punto.
   - Un parámetro de salida, de tipo booleano, llamado *ok*. Si los puntos definen una recta (esto es, son distintos), valdrá *TRUE*. En otro caso, valdrá *FALSE*.

     Consideraremos que los puntos son distintos si sus coordenadas no son *prácticamente iguales*. Consideraremos que dos reales son prácticamente iguales si su diferencia en valor absoluto es menor que ϵ (epsilon), donde ϵ será una constante local al subprograma que lo necesite, con valor `1e-9`.
   - Tres parámetros de salida con los coeficientes de la recta: alpha, beta, gamma. Serán números reales. Si *ok* es *TRUE*, contendrán los coeficientes. Si *ok* es *FALSE*, su contenido es irrelevante.

4. Tendrá un procedimiento para escribir en pantalla todo lo relativo al punto medio.

5. Tendrá otro procedimiento para escribir en pantalla todo lo relativo a la ecuación de la recta. Naturalmente esto incluye la posibilidad de indicar que la recta no está definida.

6. Podrá tener otros subprogramas si lo consideras adecuado.

7. El programa principal:

   - Declarará las variables *punto_a* y *punto_b* de tipo *TipoPunto*.
   - Les dará un valor.
   - Llamará a la función para calcular el punto medio.
   - Llamará al procedimiento para mostrar el resultado.
   - Llamará al procedimiento para calcular la recta.
   - Llamará al procedimiento para mostrar el resultado.
   - Volverá a darles otro valor a *punto_a* y *punto_b*, y volverá a calcular y mostrar punto medio y recta.
   - Observa que en ningún momento el programa pide nada al usuario.

8. La salida será similar a esta:

```
Punto a: (-4.00, 0.00)
Punto b: (2.00, 3.00)
Punto medio: (-1.00, 1.50)
Ecuacion de la recta:
```

```
  -3.00 * x + 6.00 * y + -12.00 = 0

Punto a: (1.00, 1.00)
Punto b: (1.00, 1.00)
Punto medio: (1.00, 1.00)
Los puntos coinciden: no determinan una recta.
```

Observa que para escribir la ecuación correctamente, deberíamos omitir el operador `+` cuando un coeficiente sea negativo. Por ejemplo, no escribir `+ -12` sino `-12`. Pero olvida este defecto.

Solución

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program puntos;

type
  TipoPunto = record
    x: Real;
    y: Real;
  end;

function calcula_punto_medio(a, b: TipoPunto): TipoPunto;
begin
  result.x := (a.x + b.x) / 2;
  result.y := (a.y + b.y) / 2;
end;

function practicamente_iguales(a, b: Real): Boolean;
const
  Epsilon = 1e-9;
begin
  result := abs(a - b) < Epsilon;
end;

procedure calcula_recta(a, b: TipoPunto;
                        var ok: Boolean;
                        var alpha, beta, gamma: Real);
begin
  if practicamente_iguales(a.x, b.x) and practicamente_iguales(a.y, b.y)
     then
    ok := False // alpha, beta, gamma irrelevantes en este caso
  else begin
    ok := True;
    alpha := a.y - b.y;
    beta  := b.x - a.x;
    gamma := a.x * b.y - b.x * a.y;
  end;
end;

procedure muestra_punto_medio(a, b, c: TipoPunto);
begin
  writeln('Punto a: (', a.x:0:2, ', ', a.y:0:2, ')');
  writeln('Punto b: (', b.x:0:2, ', ', b.y:0:2, ')');
```

```
  writeln('Punto medio: (', c.x:0:2, ', ', c.y:0:2, ')');
end;

procedure muestra_recta(ok: Boolean; alpha, beta, gamma: Real);
begin
  if ok then begin
    writeln('Ecuacion de la recta: ');
    write( alpha:0:2, ' * x + ');
    write( beta:0:2, ' * y + ');
    writeln(gamma:0:2, ' = 0');
  end
  else begin
    writeln('Los puntos coinciden: no determinan una recta.');
  end;
  writeln;
end;

var
  punto_a, punto_b, punto_c: TipoPunto;
  ok: Boolean;
  alpha, beta, gamma: Real;

begin
  punto_a.x := -4.0;  punto_a.y := 0.0;
  punto_b.x := 2.0;  punto_b.y := 3.0;

  punto_c := calcula_punto_medio(punto_a, punto_b);
  muestra_punto_medio(punto_a, punto_b, punto_c);

  calcula_recta(punto_a, punto_b, ok, alpha, beta, gamma);
  muestra_recta(ok, alpha, beta, gamma);

  punto_a.x := 1.0;  punto_a.y := 1.0;
  punto_b.x := 1.0;  punto_b.y := 1.0;

  punto_c := calcula_punto_medio(punto_a, punto_b);
  muestra_punto_medio(punto_a, punto_b, punto_c);

  calcula_recta(punto_a, punto_b, ok, alpha, beta, gamma);
  muestra_recta(ok, alpha, beta, gamma);
end.
```

Observaciones:

1. El enunciado dice que el cuerpo principal declarará las variables con los puntos y les dará valor. Esto es:

   ```
   var
     punto_a, punto_b: TipoPunto;
     [...]

   begin
     punto_a.x := -4.0;  punto_a.y := 0.0;
     punto_b.x := 2.0;  punto_b.y := 3.0;
   ```

   Y aunque no sería necesario indicar nada más, el enunciado aclara que no se debe preguntar nada al usuario. Por tanto, los ejercicios que piden valores al usuario, están mal. Y los ejercicios que usan constantes para los valores ejemplo, están mal.

2. El enunciado exige almacenar los puntos siempre en un registro. El enunciado exige que las funciones que calculen reciban dos puntos. Por tanto esto sería incorrecto:

```
function calculo_punto_medio(xa,ya,xb,yb:real):TipoPunto; //MAL
```

   Porque esta función, aunque devuelve un TipoPunto, no recibe 2 TipoPuntos. Recibe 4 reales. No es lo mismo.

3. En el cuerpo principal, primero llamamos a las funciones que calculan, obtienen los resultados y luego llamamos a los procedimientos que los escriben. En algunos ejercicios, el cuerpo principal llama al procedimiento que escribe, y este a su vez llama a la función que calcula. Esto no es un buen diseño y no es lo que pide el enunciado.

4. Un subprograma con varios *else* suele ser complejo y propenso a errores. Se puede mejorar la claridad evitando los *else*. Esto es, en vez de escribir

```
if condicion1 then
    sentencia1
else
    sentencia2;
```

   Podemos usar la estructura

```
if condicion1 then
    sentencia1;

if condicion2 then
    sentencia2;
```

   En este programa, como solo hay un *else*, y es muy sencillo, se puede usar *else*, no es necesario evitarlo. Pero si a pesar de ello queremos usar dos *if*, condición2 debe ser la negación de condicion1. Es frecuente que para escribir esta negación, haya que aplicar De Morgan. Y aquí es donde muchos estudiantes fallan y escriben código incorrecto similar al siguiente:

```
    if abs(a.x - b.x)<Epsilon and abs(a.y -b.y)<Epsilon then
          ok := false;

    // ¡MAL! Esta condición no es la negación de la anterior.
    if abs(a.x - b.x)>=Epsilon and abs(a.y -b.y)>=Epsilon then
          ok := true;
          alpha := a.y - b.y;
          beta := b.x - a.x;
          gamma := a.x*b.y - b.x*a.y;
    end;
```

   La primera sentencia es correcta (aunque sería preferible usar una función para comparar los números). Pero la segunda no lo es, porque la condición no es la negación de la anterior. Tal y como está escrito, para dar el punto por bueno exige que sus cordenadas x sean diferentes y que sus coordenadas y también lo sean. Es un error. No es necesario que ambas lo sean. Por ejemplo los puntos (4,3) y (4,9) son diferentes.

   Lo correcto sería aplicar *De Morgan* y escribir

```
    if abs(a.x - b.x)>=Epsilon or abs(a.y -b.y)>=Epsilon then
```