

--	--	--	--

APELLIDOS :

NOMBRE :

- Pon tu nombre en todas las hojas del examen.
- Intenta contestar a las cuestiones en el espacio reservado para ellas.

Responde de forma **razonada** a las siguientes preguntas:

1. (1 punto) Indica cuál es la representación interna del número entero $N = -666$ en las cuatro representaciones internas posibles para números enteros si $n = 12$ bits. Debes indicar sólo la representación en hexadecimal de cada una de las cuatro representaciones.

El número entero $N=666$ se representa en hexadecimal como 29A,
que se corresponde a 0010 1001 1010 en binario y 12 bits.

Al ser negativo:

Signo y Magnitud = 1010 1001 1010 en binario y A9A en hexadecimal.

Complemento a 1 = 1101 0110 0101 en binario y D65 en hexadecimal.

Complemento a 2 = 1101 0110 0110 en binario y D66 en hexadecimal.

Sesgo con 12 bits = $2^{(n-1)} = 2^{11} = 2048$

Notación sesgada = $N + S = 1382 = 566$ en hexadecimal

2. (Total 2,5 puntos) Tenemos un programa que trabaja con polinomios de una variable. Nuestros polinomios estarán formados por una colección de términos ($coef_i, grado_i$), ordenados por su grado en orden creciente.

Para ello utiliza la siguiente definición:

```
{Declaración de tipos}
polinomio = registro de
    grado_max: numérico; {máximo grado válido del polinomio}
    num_terminos: numérico; {entero; número de términos no nulos del polinomio}
    coef, grado: vector[1..100] de numérico; {reales y entero; términos del polinomio}
fin registro
```

- a) (0,3 puntos) ¿Cuál es la codificación en C de estas declaraciones de tipos? ¿Y si declaramos dos variables $P1, P2$: *POLINOMIO*; en código algorítmico, cuál sería su equivalente en C?

```
typedef struct {
    int grado_max;
    int num_termino;
    float coef[100];
    int grado[100];
} polinomio;
polinomio p1, p2;
```

- b) (0,3 puntos) Con esta representación tenemos limitado el número máximo de términos válidos del polinomio ¿Por qué? ¿Cómo podríamos solucionarlo para que cada polinomio tuviese un número variable de pares (coeficiente, grado)?

Porque tanto coef como grado son dos vectores con un máximo de 100 elementos.

Si usásemos punteros y memoria dinámica,

```
float *coef;
```

```
int *grado;
```

se podría reservar memoria de forma dinámica para los pares (coef, grado)

necesarios, ya que conocemos el número de términos válidos: num_termino.

Esto quedaría como:

```
coef = (float *) calloc (num_termino, sizeof(float));
```

```
grado = (int *) calloc (num_termino, sizeof(int));
```

- c) Queremos realizar un algoritmo que sume dos polinomios P1, P2, dando como resultado un nuevo polinomio P3:

Algoritmo Suma_polinomios (P1, P2, P3, ok) es

P1, P2: Polinomio; {par. dato}

P3: Polinomio; {par. resultado}

ok: lógico; {par. resultado; indica si se pudo realizar la operación}

- 1) (0,3 puntos) ¿Suponiendo que P1 y P2 sean polinomios correctamente definidos, qué caso ó casos pueden ocurrir para que ok tome el valor falso?

Que cuando se esté realizando la suma para generar P3, nos encontremos con que el número de términos de P3 es mayor que 100 (máximo permitido).

- 2) (1,6 puntos) Analiza y explica la secuencia necesaria para realizar la suma de los dos polinomios. Indica cuál sería el tratamiento en curso, el tratamiento final y en qué consistiría obtener el primer elemento y el elemento siguiente. Indica qué esquema de tratamiento secuencial de los vistos en la asignatura sería el más apropiado para solucionar este problema o si ya hay algún esquema similar que me ayude a solucionar el problema.

El problema es equivalente a la fusión de dos vectores (equivalente a fusión de dos ficheros secuenciales).

Secuencia (i, j, k, p1, p2, p3)

1 <= i <= P1.num_terminos

1 <= j <= P2.num_terminos

1 <= k <= P3.num_terminos

p3.num_terminos se calculará dentro del algoritmo

{Primer elemento}

i:=1; j:= 1; k:= 1;

{Ultimo elemento}

Bien i = P1.num_termino ó j = P2.num_termino ó k =100

{Obtener elemento siguiente}

si P1.grado(i) = P2.grado(j) entonces se suman y avanzan i y j y k

si P1.grado(i) < P2.grado(j) entonces sólo avanza i y k

si P1.grado(i) > P2.grado(j) entonces sólo avanza j y k

{Tratamiento en curso: obtener término k de P3 a partir de los términos i de P1 y j de P2, según $P_x.\text{grado}(y)$ }

Iría dentro de un bucle del tipo:

```
mientras i <= P1.num_termino and j <= P2.num_termino and k <= 100 hacer
  {Tratamiento en curso}
  si P1.grado(i) = P2.grado(j) entonces {se suman y avanzan i y j y k}
    P3.coef(k) := P1.coef(i) + P2.coef(j);
    P3.grado(k) := P1.grado(i);
    {obtener elemento siguiente}
    k := k + 1;
    i := i + 1;
    j := j + 1;
  sino
    si P1.grado(i) < P2.grado(j) entonces
      P3.coef(k) := P1.coef(i);
      P3.grado(k) := P1.grado(i);
      {obtener elemento siguiente}
      k := k + 1;
      i := i + 1;
    sino
      P3.coef(k) := P2.coef(j);
      P3.grado(k) := P2.grado(j);
      {obtener elemento siguiente}
      k := k + 1;
      j := j + 1;
    fin si
  fin si
```

{Tratamiento final: sólo quedan términos de P1 ó P2 -->
obtener término k de P3 a partir de P1 o bien obtener término k de P3 a partir de P2}

```
mientras i <= P1.num_termino and k <= 100 hacer
  P3.coef(k) := P1.coef(i);
  P3.grado(k) := P1.grado(i);
  k := k + 1;
  i := i + 1;
fin mientras;
```

```
mientras j <= P2.num_termino and k <= 100 hacer
  P3.coef(k) := P2.coef(j);
  P3.grado(k) := P2.grado(j);
  k := k + 1;
  j := j + 1;
fin mientras;
```

3. (Total 2,5 puntos) Disponemos de un vector de números ordenados: V . Queremos implementar un esquema de búsqueda en dicho vector de un elemento X que puede estar o no en el vector.

Algoritmo Búsqueda (V , N , X , $resp$, pos) es
 V : vector[1..TAM] de numérico; {vector ordenado de enteros; parámetro dato}
 N : numérico; {parámetro dato; número de datos en V }
 X : numérico; {parámetro dato; entero; número a buscar en V }
 $resp$: lógico; {parámetro resultado; cierto si X está en V ;
falso en caso contrario}
 pos : numérico; {parámetro resultado; entero; posición en la que se encuentra
 X en V , si está; -1 en caso contrario}

- a) ¿Cuál sería la codificación en C de esta cabecera? Justifica tu elección del tipo de la función elegida (0,5 puntos)

```
int busqueda (int V[], int N, int X, int *pos)
```

Como hay dos parámetros resultado, uno debe usarse como tipo de la función, en este caso el valor de $resp$, lógico, que se transforma en entero (0/1), y el otro parámetro resultado debe pasarse por referencia (int *pos).

- b) Explica, sin dar su código, el tipo de esquemas de búsqueda que podríamos utilizar y cuál sería el más indicado en este caso. (0,5 puntos)

Se puede usar la búsqueda secuencial o la binaria.

La binaria es más eficiente para un vector ordenado, ya que en promedio recorre menos elementos del vector (hace $\log N$ comparaciones frente a $N/2$ comparaciones en promedio de la secuencial).

- c) Implementa en código algorítmico el esquema elegido (1,5 puntos)

```
{variables locales}
inf, sup, med: numérico;
Inicio
    inf := 1;
    sup := N;
    mientras inf < sup hacer
        med := (inf + sup) div 2;
        si V(med) > X entonces
            inf := med + 1;
        sino
            sup := med;
        fin si;
    fin mientras;
    si X = V(inf) entonces
        pos := inf;
    resp := cierto;
    sino
        pos := -1;
        resp := falso;
    fin si;
Fin
```