Apellidos: Nombre:

2º PARCIAL y FINAL

La puntuación para preguntas con múltiples opciones es:
- ***Pregunta correcta: 1 punto***
- ***Pregunta incorrecta: -0,25 puntos***
- ***Pregunta en blanco: 0 puntos***

Las preguntas en las que se marquen varias opciones serán consideradas incorrectas.

1.- Dada la siguiente función:

```
typedef int Matriz[N][N];

void f(Matriz m)
{
    int i = 0, j = 1;

    while ( (j < N) && (m[i][j] != 0) )
    {
        if (i == N - 1)
        {
            m[i][j] = 1;
            j = j + 2;
            i = 0;
        }
        else
            i++;
    }
}
```

a.- Indicar el orden de complejidad de este algoritmo en el mejor y en el peor caso

	Mejor caso	*Peor caso*
Orden de complejidad:	**cte**	**n^2**

b.- Describir a qué situación corresponte el mejor y el peor de los casos.

El mejor de los casos será aquél en el que el elemento m[0][1] valga cero.

El peor será aquél en el que ninguno de los elementos de las columnas impares de la Matriz sea cero, por lo que tendremos que recorrer todos sus elementos.

2.- Dada la siguiente implementación del método encolar de la clase cola:

```
bool Cola::Encolar (Valor x)
{
   bool ok = true;
   Puntero p_aux;

   p_aux = new Nodo;
   p_aux->info = x;
   p_aux->sig = NULL;

   if (ini == NULL)
       ini = p_aux;
   fin->sig = p_aux;
   fin = p_aux;

   return ok;
}
```

Indicar cuál de las siguientes afirmaciones es correcta:

(a) El método es correcto.
(b) Dará un error de compilación.
(c) Contiene errores lógicos pero no producirá errores ni de compilación ni de ejecución.
(d) Generará un error de ejecución cuando la cola esté vacía.

3.- Dado el siguiente vector {7, 5, 11, -6, 7, 0} indica el número de asignaciones de elementos que se realiza para ordenar el vector en cada uno de los siguientes métodos:

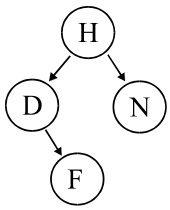
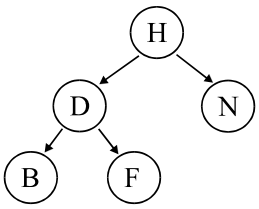
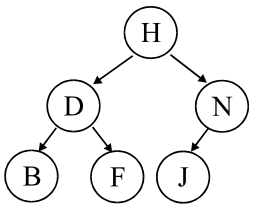
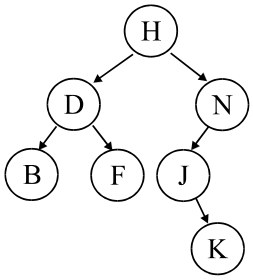
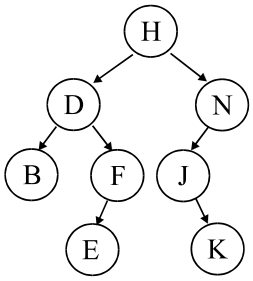
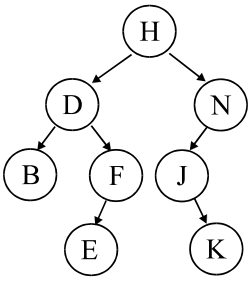
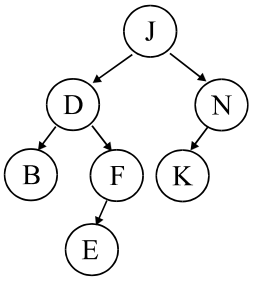
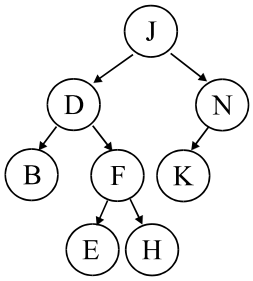
Inserción	(a) 9	(b) 12	**(c) 18**	(d) 23
Selección	**(a) 9**	(b) 12	(c) 18	(d) 23
Quick-Sort	(a) 9	(b) 12	**(c) 18**	(d) 23

4.- Supuesto un problema en el que se necesita utilizar una lista de enteros cuyo tamaño máximo estimado es de 1000 elementos, calcular el coste espacial cuando la lista tiene 600 elementos, en las siguientes implementaciones:

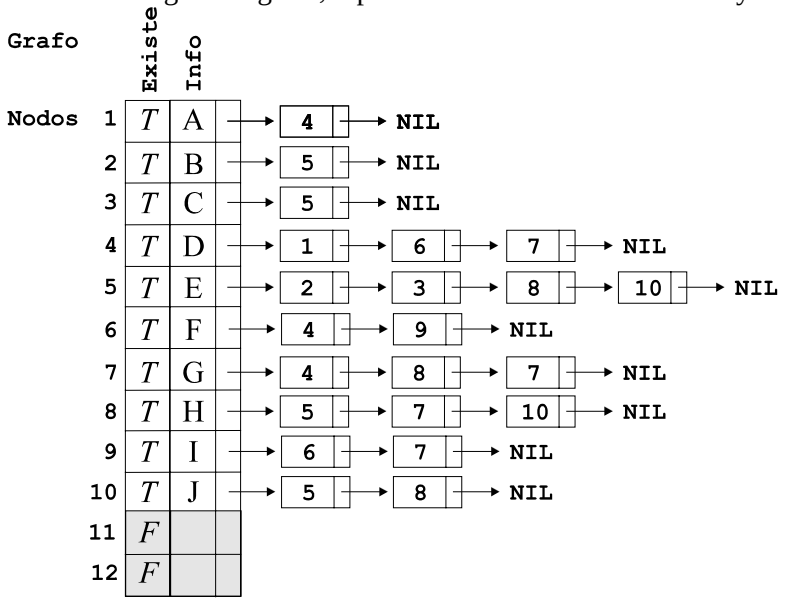
a.- Implementación estática. **2 * sizeof (int) + 1000 * sizeof (int) =** **= 2 * 4 + 1000 * 4 = 4008 bytes**	**b.-** Implementación dinámica doblemente enlazada, no circular y sin nodo cabeza. **3 * sizeof (puntero) +** **+ 600 * (sizeof (int) + 2 * sizeof (puntero)) =** **= 3 * 4 + 600 * (4 + 2 * 4) = 8 + 600 * 12 =** **= 7212**

Información: Suponed que tanto los enteros como los punteros ocupan 4 bytes.

5.- Dado el siguiente árbol binario de búsqueda, representa el árbol tras cada una de las operaciones siguientes:

Estado inicial	Insertar (B)	Insertar (J)	Insertar (K)
			
Insertar (E)	**Insertar (D)**	**Eliminar (H)**	**Insertar (H)**
			

6.- Dado el siguiente grafo, representado mediante listas de adyacencia:



Di cuál de los siguientes recorridos es un recorrido BFS válido desde el nodo '1'.

(a) 1 4 6 7 9 8 5 10 2 3
(b) 1 4 7 8 10 5 3 2 6 9
(c) 1 4 7 6 8 10 9 5 2 3
(d) 1 4 6 7 9 8 5 2 3 10

7.- Sea un montículo de mínimos representado mediante un *array*. Di cuál será el aspecto del vector tras cada una de las siguientes operaciones:

Estado inicial

num:	4	info:	3	5	5	6							

Insertar (4)

num:	5	info:	3	4	5	6	5						

Insertar (4)

num:	6	info:	3	4	4	6	5	5					

Insertar (7)

num:	7	info:	3	4	4	6	5	5	7				

Eliminar_Minimo

num:	6	info:	4	5	4	6	7	5					
			2	4	5	6	5	7					

Insertar (2)

num:	7	info:	2	5	4	6	7	5	4				
			2	4	4	6	5	7	5				

SÓLO 2º PARCIAL

8.- En un tipo abstracto de datos, los axiomas...

(a) ...indican la manera exacta de implementar las operaciones.
(b) ...indican las operaciones que se pueden realizar sobre este tipo de dato.
(c) ...no forman parte de la especificación.
(d) ...indican las propiedades que cumplen las operaciones del tipo.

9.- En cuál de las siguientes situaciones es menos conveniente utilizar una pila:

(a) Como estructura auxiliar para invertir una cola.
(b) Como estructura auxiliar para invertir una pila
(c) Gestionar el sistema de llamadas a funciones dentro de un programa.
(d) Implementar una lista.

10.- Contesta las siguientes preguntas referidas a un árbol binario de 40 nodos.

a.- ¿Cuál es el número mínimo de hojas?	**1**
b.- ¿Cuál es el número máximo de hojas?	**20**
c.- ¿Cuál es el número máximo de subárboles vacíos?	**41**
d.- ¿Cuál es el número mínimo de subárboles vacíos?	**41**

SÓLO FINAL

8.- Dada la siguiente declaración de tipos y variables:

```
typedef float * Ptr;

typedef Ptr Vector [3];

struct Registro
{
    Ptr punt;
    Vector info;
};

typedef Registro VecReg[3];

VecReg x;
```

Di si son correctas y que se obtiene en las siguientes expresiones. O si son incorrectas y por qué.

`x.punt[1]` _ **MAL_X_es_un_vector,_no_un_registro**

`*x->punt` _ **float** _____

`x[1].info[2]` _ **Ptr** _____

`*x[1].info[2]` _ **float**_____

`*(*x).info` _ **Ptr** _____

`x[1].punt` _ **Ptr** _____

9.- Sea la siguiente declaración de tipos y de variables:

```
typedef int Vect1[3];

struct Reg1
{
    Vect1 a1;
    float b1;
};

typedef Reg1 Vect2[10];

struct Reg2
{
    Vect2 a2;
    string b2;
};

Reg2 x;
```

Sabiendo que la variable '**x**', comienza en la posición de memoria 1000, que el tamaño de un int es de 2 bytes, el de un float es de 4 bytes y el de un puntero es de 7 bytes, determina la posición de memoria en la que se encontrará la siguiente información:

```
x.a2[3].a1[2]
```

$$m = m_0 + 3 * sizeof(Reg1) + 2 * sizeof(int) = 1000 + 3 * 10 + 2 * 2 = 1034$$

$$sizeof(Reg1) = 3 * sizeof(int) + sizeof(float) = 3 * 2 + 4 = 10$$

10.- Dada la siguiente declaración de variables:

```
float a = 4.99;
int b = 5;
double c = 2.5;
unsigned short d = 3;
```

Y sabiendo que el prototipo de la función 'sqrt' es:

```
double sqrt (double);
```

Evalua paso a paso la siguiente expresión:

```
int (a) / (b / sqrt (4) * int (c) ) <= 4 / 5 || 3 + 5 % d – 5 > 0

4   / ( 5 /  2.0    *   2    ) <=   0   || 3 + 5 % 3 - 5 > 0

4   / (  2.5        *   2    ) <=   0   || 3 +   2    - 5 > 0

4   /           5.0             <=   0   ||        0       > 0

          0.8                   <=   0   ||       false

                                false      ||       false

                                        false
```

2º PARCIAL y FINAL

P.1.- Realiza una función que acepte como parámetro un árbol binario y nos devuelva una lista que contenga la información de los nodos de grado uno del árbol. El prototipo de la función es:

```
Lista NodosGradoUno (const Arbol &);
```

Escribe el **prototipo** de todos los métodos que vayas a utilizar, tanto de árboles como de listas, para realizar esta función.

P.2.- Supongamos una Cola representada de forma dinámica. Realizar un nuevo método sobre la cola, al que le pasemos un cierto valor 'x', que elimine de la cola todos aquellos elementos mayores que el valor 'x'.

Se valorará negativamente la utilización de otros métodos o estructuras auxiliares en la realización de este método.

La clase cola tendrá el siguiente aspecto.

```
class Cola
{
   public:
      Cola ();
      Cola (const Cola&);
      ~Cola ();
      const Cola& operator= (const Cola&);
      bool Encolar (Valor);
      bool Desencolar ();
      bool PrimeroCola (Valor &);
      bool ColaVacia ();
      void EliminarMayores (Valor);
   private:
     struct Nodo;
     typedef Nodo * Puntero;
     struct Nodo
     {
           Valor info;
           Puntero sig;
     };
     Puntero inicio, fin;
     void Vaciar();
     bool Copiar (const Cola&);
};
```

SÓLO FINAL

P.3.- Dado el siguiente programa, realiza la traza:

```
#include <iostream.h>

int F (int);

int main(void)
{
    int a, b, r;

1   b = 2;

2   a = 2 + F (b, r);
3   r = F (a, b);

    return 0;
}
```

```
int F (int a, int & b)
{
    int r;

4   b = 3;
    if (a > 2)
    {
5       b = b - 2;
6       r = F (b, b);
    }
    else
7       r = a;

    return r;
}
```

P.1.- Realiza una función que acepte como parámetro un árbol binario y nos devuelva una lista que contenga la información de los nodos de grado uno del árbol. El prototipo de la función es:

```
Lista NodosGradoUno (const Arbol &);
```

Escribe el **prototipo** de todos los métodos que vayas a utilizar, tanto de árboles como de listas, para realizar esta función.

OPCIÓN 1:

```
Lista NodosGradoUno (const Arbol & arb)
{
    Lista l1, l2;

    // Si el arbol no esta vacio
    if (!arb.ArbolVacio () )
    {
        // Miramos los resultados que tenemos en los hijos
        NodosGradoUno (arb.HijoIzdo (), l1);
        NodosGradoUno (arb.HijoDcho (), l2);

        // Pasamos la informacion de la lista dos a la lista uno
        l2.IrAInicio ();
        while (l2.Consultar (x) )
        {
            l1.Insertar (x);
            l2.Eliminar ();
        }

        // Miramos el nodo en el que estamos y si es de grado uno
        // ponemos su informacion tambien en la lista
        if (EsGradoUno (arb) )
        {
            arb.Informacion (x);
            l1.Insertar (x)
        }
    }
    // Si el arbol estaba vacio la lista uno quedara vacia

    return l1;
}

bool EsGradoUno (const Arbol & arb)
{
    bool esgradouno;

    if ( (arb.HijoIzdo ().ArbolVacio () && !arb.HijoDcho ().ArbolVacio () ) ||
         (!arb.HijoIzdo ().ArbolVacio () && arb.HijoDcho ().ArbolVacio () ) )
        esgradouno = true;
    else
        esgradouno = false;

    return esgradouno;
}
```

Prototipos de los métodos utilizados:

```
class Lista
{
    public:
        ...
        bool Insertar (Valor);
        bool Eliminar (void);
        bool Consulta (Valor &);
        void IrAInicio (void);
        ...
    private:
        ...
};
```

```
class Arbol
{
    public:
        ...

        bool ArbolVacio (void);
        bool Informacion (Valor &);
        Arbol &HijoIzdo (void);
        Arbol &HijoDcho (void);
        ...
    private:
        ...
};
```

P.1.- Realiza una función que acepte como parámetro un árbol binario y nos devuelva una lista que contenga la información de los nodos de grado uno del árbol. El prototipo de la función es:

```
Lista NodosGradoUno (const Arbol &);
```

Escribe el **prototipo** de todos los métodos que vayas a utilizar, tanto de árboles como de listas, para realizar esta función.

OPCIÓN 2:

```
Lista NodosGradoUno (const Arbol & arb)
{
    Lista l;

    NodosGradoUnoRec (arb, l);

    return l;
}

void NodosGradoUnoRec (const Arbol & arb, Lista & l)
{
    Valor x;

    if (!arb.ArbolVacio () )
    {
        if (EsGradoUno (arb) )
        {
            arb.Informacion (x);
            l.Insertar (x);
        }
        NodosGradoUnoRec (arb.HijoIzdo (), l);
        NodosGradoUnoRec (arb.HijoDcho (), l);
    }
    return;
}

bool EsGradoUno (const Arbol & arb)
{
    bool esgradouno;

    if ( (arb.HijoIzdo ().ArbolVacio () && !arb.HijoDcho ().ArbolVacio () ) ||
         (!arb.HijoIzdo ().ArbolVacio () && arb.HijoDcho ().ArbolVacio () ) )
        esgradouno = true;
    else
        esgradouno = false;

    return esgradouno;
}
```

Prototipos de los métodos utilizados:

```
class Arbol
{
    public:
        ...
        bool ArbolVacio (void);
        bool Informacion (Valor
&);
        Arbol & HijoIzdo (void);
        Arbol & HijoDcho (void);
        ...
    private:
        ...
};
```

```
class Lista
{
    public:
        ...
        bool Insertar (Valor);
        ...
    private:
        ...
};
```

P.2.- Supongamos una Cola representada de forma dinámica. Realizar un nuevo método sobre la cola, al que le pasemos un cierto valor ‘x’, que elimine de la cola todos aquellos elementos mayores que el valor ‘x’.

OPCIÓN 1:

```
void Cola::EliminarMayores (Valor x)
{
    Puntero aux, anterior, siguiente;

    anterior = NULL;
    aux = ini;

    // Recorremos todos los elementos de la cola
    while (aux != NULL)
    {
        // Si el elemento estudiado es mayor que la referencia
        // deberenos eliminarlo
        if (aux->info > x)
        {
            siguiente = aux->sig;

            // Si es el primero de la lista movemos ‘ini’
            if (ant == NULL)
                ini = siguiente;
            // Si no lo es, existira un elemento anterior y nos limitamos
            // a saltarnos el que deseamos eliminar
            else
                anterior->sig = siguiente;

            // Si el que deseamos eliminar es el marcdo como ‘fin’
            // movemos la marca antes de eliminar el elemento
            if (aux == fin)
                fin = anterior;

            // Liberamos el elemento estudiado
            delete aux;
        }
        // Si el elemento no es mayor
        // actualizamos las marcas antes de avanzar al siguiente
        else
            anterior = aux;

        // Avanzamos al siguiente elemento de la cola
        aux = siguiente;
    }
    return;
}
```

P.2.- Supongamos una Cola representada de forma dinámica. Realizar un nuevo método sobre la cola, al que le pasemos un cierto valor ‘x’, que elimine de la cola todos aquellos elementos mayores que el valor ‘x’.

OPCIÓN 2:

```
void Cola::EliminarMayores (Valor x)
{
    Puntero p_aux;

    p_aux = ini;
    while (p_aux != NULL)
    {
        if (p_aux->info > x)
        {
            // Si no hay ninguno detras
            if (p_aux == fin)
            {
                // Si es el unico
                if (p_aux == ini)
                {
                    delete p_aux;
                    ini = NULL;
                    fin = NULL;
                }
                else
                {
                    aux = inicio;
                    //buscar el anterior al 'fin'
                    while ( aux->sig != fin )
                        aux = aux->sig;
                    fin = aux;
                    fin->sig = NULL;
                    delete p_aux;
                }
                p_aux = NULL;
            }
            else
            {
                //caso normal
                aux = p_aux->sig;
                *p_aux = *aux;
                if (aux == fin)
                    fin = p_aux;
                delete aux;
            }
        }
        else
            p_aux = p_aux->sig;
    }
    return;
}
```

P.3.- Dado el siguiente programa, realiza la traza:

```
#include <iostream.h>

int F (int);

int main(void)
{
    int a, b, r;

1   b = 2;

2   a = 2 + F (b, r);
3   r = F (a, b);

    return 0;
}
```

```
int F (int a, int & b)
{
    int r;

4   b = 3;
    if (a > 2)
    {
5       b = b - 2;
6       r = F (b, b);
    }
    else
7       r = a;

    return r;
}
```

	a	*b*	*r*						
1	?	2	?	a_1	b_1/r	r_1			
2	?	2	?	2	?	?			
4_1	?	2	3	2	3	?			
7_1	?	2	3	2	3	2			
2	4	2	3	a_2	b_2/b	r_2			
3	4	2	3	4	2	?			
4_2	4	3	3	4	3	?			
5_2	4	1	3	4	1	?	a_3	b_3/b_2	r_3
6_2	4	1	3	4	1	?	1	1	?
4_3	4	3	3	4	3	?	1	3	?
7_3	4	3	3	4	3	?	1	3	1
6_2	4	3	3	4	3	1			
3	4	3	1						