

Apellidos:

Nombre:

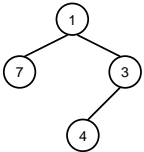
Convocatoria:

DNI:

Examen TAD/PED septiembre 2006

Modalidad 0

- Normas:**
- La entrega del test no corre convocatoria.
 - Tiempo para efectuar el test: **18 minutos**.
 - Una pregunta mal contestada elimina una correcta.
 - Las soluciones al examen se dejarán en el campus virtual.
 - **Una vez empezado el examen no se puede salir del aula hasta finalizarlo. A continuación comenzará el siguiente ejercicio.**
 - El test vale un 40% de la nota de teoría.
 - En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

	V	F	
Las operaciones constructoras generadoras de un tipo permiten obtener cualquier valor de dicho tipo.	<input type="checkbox"/>	<input type="checkbox"/>	1. V
En C++, si no se ha implementado la sobrecarga del operador asignación, se invoca automáticamente al constructor de copia.	<input type="checkbox"/>	<input type="checkbox"/>	2. F
Es posible reconstruir un único árbol binario de altura 6 a partir de un recorrido en postorden con 63 etiquetas.	<input type="checkbox"/>	<input type="checkbox"/>	3. V
La semántica de la operación nodos del tipo <i>arbin</i> vista en clase es la siguiente: $VAR i, d: arbin; x: item;$ $nodos(crea_arbin()) = 0$ $nodos(enraizar(i, x, d)) = nodos(i) + nodos(d)$	<input type="checkbox"/>	<input type="checkbox"/>	4. F
Se puede reconstruir un único árbol binario cualquiera teniendo sus recorridos en preorden y postorden.	<input type="checkbox"/>	<input type="checkbox"/>	5. F
La semántica de la operación <i>recu</i> vista en clase es la siguiente: $VAR v: vector; i, j: int; x: item;$ $recu(crear_vector(), i) = error_item()$ $recu(asig(v, i, x), j)$ $si (i == j) entonces x$ $sino FALSO$ fsi	<input type="checkbox"/>	<input type="checkbox"/>	6. F
En un árbol AVL cuyo factor de equilibrio es -2, al insertar un elemento en la rama derecha, el árbol vuelve al estado de equilibrio.	<input type="checkbox"/>	<input type="checkbox"/>	7.
Dado un árbol 2-3 de altura h con n ítems con todos sus nodos del tipo 2-Nodo: la complejidad de la operación de búsqueda de un ítem es $O(\log_2 n)$	<input type="checkbox"/>	<input type="checkbox"/>	8. V
Un árbol binario de búsqueda lleno de altura 4 es un árbol 2-3-4, pero no se puede conseguir a partir de un árbol inicialmente vacío y utilizando las operaciones de inserción y borrado de un árbol 2-3-4	<input type="checkbox"/>	<input type="checkbox"/>	9. V
Un grafo no dirigido puede tener aristas que empiecen y acaben en el mismo vértice.	<input type="checkbox"/>	<input type="checkbox"/>	10. F
El siguiente árbol es leftist mínimo: 	<input type="checkbox"/>	<input type="checkbox"/>	11. V
Un trie cumple las propiedades de un árbol general.	<input type="checkbox"/>	<input type="checkbox"/>	12. V

Examen PED septiembre 2006

- Normas:**
- ♦ Tiempo para efectuar el ejercicio: **2 horas**
 - En la cabecera de cada hoja **Y EN ESTE ORDEN** hay que poner: **APELLIDOS, NOMBRE**.
 - Cada pregunta se escribirá en hojas diferentes.
 - Se dispone de 20 minutos para abandonar el examen sin que corra convocatoria.
 - Las soluciones al examen se dejarán en el campus virtual.
 - Se puede escribir el examen con lápiz, siempre que sea legible
 - **Todas las preguntas tienen el mismo valor.** Este examen vale el 60% de la nota de teoría.
 - **Publicación notas:** 18 de septiembre. **Revisión exámenes TEORÍA:** 19 de septiembre de 9:30 a 10:30 en aula LS14I del sótano de la EPS IV
- Los alumnos que estén en 5ª o 6ª convocatoria deben indicarlo en la cabecera de todas las hojas

1. Utilizando exclusivamente las operaciones constructoras generadoras del tipo conjunto, definid la sintaxis y la semántica de la operación *subconjunto_impares* que se aplica sobre un conjunto dado de números naturales y devuelve el subconjunto formado por los números impares que existen en el conjunto original.

Ej: $C = \{1, 8, 10, 3, 12\}$

$\text{subconjunto_impares}(C) = \{1, 3\}$

Nota: Se asume que está definida la operación MOD para calcular el resto de la división entera.

2. Dada la siguiente declaración de la clase *TLista* que representa una lista ordenada (de menor a mayor) doblemente enlazada de números enteros donde no se permiten repetidos, escribe el código del método *bool Insertar(int)*, que devuelve cierto si el número se puede insertar y falso en caso contrario. Se proporciona el código del constructor y destructor de la clase *TNodo*.

Nota: se tiene que escribir el código de todos los métodos auxiliares que se empleen.

```
class TLista {
public:
    TLista();
    ...
    ~TLista();
    ...
    bool Insertar(int);
    ...
private:
    TNodo *primero, *ultimo;
};

class TNodo {
public:
    TNodo();
    ...
    ~TNodo();
    ...
private:
    int etiqueta;
    TNodo *anterior, *siguiente;
};

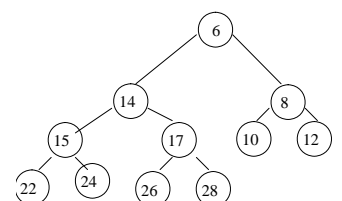
TLista::TLista() {
    primero = NULL;
    ultimo = NULL;
}

TNodo::TNodo() {
    etiqueta = 0;
    anterior = NULL;
    siguiente = NULL;
}

TNodo::~TNodo() {
    etiqueta = 0;
    anterior = NULL;
    siguiente = NULL;
}
```

3. Dado el siguiente árbol Leftist mínimo (izquierdista mínimo):

- Insertar los siguientes elementos en el orden que se indica: 13, 16, 20, 11, 4, 2, 30, 1
- Sobre el Leftist resultante, realizar el borrado de los 3 ítems mínimos.



4. Dado el **grafo no dirigido** representado por la siguiente diagonal superior de una matriz de adyacencia, tal y como se ha visto en clase:

	1	2	3	4	5	6	7	8
1					•			•
2							•	•
3					•			
4					•	•	•	•
5								•
6								
7								•
8								

- Realizar el recorrido en profundidad y en anchura empezando por 1.
- Obtener el bosque extendido en profundidad y clasificación de arcos.
- Obtener razonadamente la complejidad en el peor caso de la operación *obtenerAdyacencia(v)* utilizando la representación interna del grafo no dirigido mostrado en este ejercicio.

NOTA: Se seguirá el criterio de recorrer la adyacencia de cada vértice ordenada de menor a mayor vértice.

Examen PED septiembre 2006. Soluciones

1.

Sintaxis

subconjunto_impares: conjunto \rightarrow conjunto

Semántica

```
Var C: conjunto; x: natural;
subconjunto_impares(crear_conjunto()) = crear_conjunto()
subconjunto_impares(Insertar(C,x))=
    si (x MOD 2 ==1) entonces Insertar(subconjunto_impares(C),x)
    si no subconjunto_impares (C)
```

2.

```
bool
TLista::Insertar(int item) {
    if(primerero == NULL)
    {
        primero = new TNode;
        if(primerero == NULL)
            return false;

        ultimo = primero;
        primero->etiqueta = item;
        return true;
    }
    else
    {
        TNode *aux, *nuevo;

        if(item < primero->etiqueta)
        {
            nuevo = new TNode;
            if(nuevo == NULL)
                return false;

            nuevo->etiqueta = item;
            nuevo->siguiente = primero;
            primero->anterior = nuevo;
            primero = nuevo;
            return true;
        }

        aux = primero;
        while(aux)
        {
            if(item < aux->etiqueta)
            {
                nuevo = new TNode;
                if(nuevo == NULL)
                    return false;

                nuevo->etiqueta = item;
                nuevo->anterior = aux->anterior;
                nuevo->siguiente = aux;
                aux->anterior->siguiente = nuevo;
                aux->anterior = nuevo;
                return true;
            }
            else if(item == aux->etiqueta)
                return false;
            aux = aux->siguiente;
        }

        nuevo = new TNode;
        if(nuevo == NULL)
            return false;

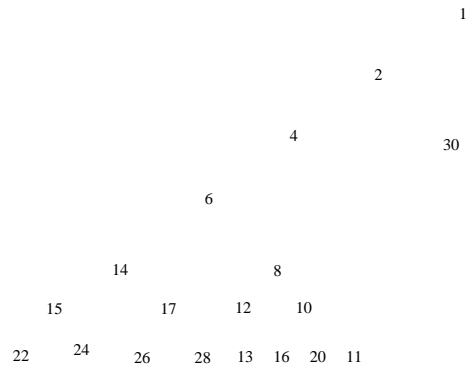
        nuevo->etiqueta = item;
        nuevo->anterior = ultimo;
```

```

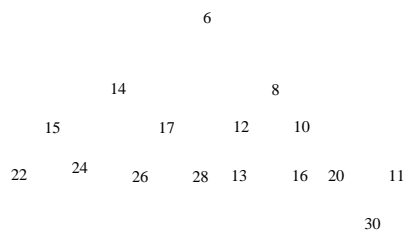
    ultimo->siguiente = nuevo;
    ultimo = nuevo;
    return true;
}
}

```

3. a)



b)



4. a) DFS: 1, 5, 3, 4, 6, 7, 2, 8
 BFS: 1, 5, 8, 3, 4, 2, 7, 6
 b)

	1	2	3	4	5	6	7	8
1					A			R
2							A	A
3					A			
4					A	A	A	R
5								R
6								
7								R
8								

c) $O(n)$, con n el número de vértices del grafo, ya que el peor caso sería el de *obtenerAdyacencia(8)* en el que nos obligaría a recorrer toda la columna del 8, o bien el caso de *obtenerAdyacencia(1)* en el que tendríamos que recorrer toda la fila del 1.