Apellidos:
Nombre:
Convocatoria:
DNI:

Examen TAD/PED junio 2003
Modalidad 0

Normas:
- La entrega del test no corre convocatoria.
- Tiempo para efectuar el test: **20 minutos**.
- Una pregunta mal contestada elimina una correcta.
- Las soluciones al examen se dejarán en la página web de la asignatura. Además se dejará disponible en el sistema AWAM en http://gplsi.dlsi.ua.es/awam/.
- **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** se el verdadero se corresponderá con la **A**, y el falso con la **B**.

	V	**F**	
La extensión de un TAD se obtiene añadiendo nuevos tipos sobre los ya creados	❑	❑	V
En C++, el constructor de copia recibe como argumento un objeto del mismo tipo pasado por referencia o por valor.	❑	❑	F
En las pilas las inserciones y borrados se realizan por el mismo extremo.	❑	❑	V
El grado de un nodo es el máximo número de etiquetas de dicho nodo más uno.	❑	❑	V
En la inserción de un árbol AVL el equilibrado se realiza de las hojas hacia la raíz.	❑	❑	V
La operación de Rotación en un árbol 2-3 se da cuando el hermano del nodo donde se efectúa el borrado es del tipo 2-Nodo.	❑	❑	F
En la operación de inserción en un árbol 2-3-4 las reestructuraciones se realizan desde las hojas hacia la raíz.	❑	❑	F
Un árbol rojo-negro es un árbol binario de búsqueda que representa a un árbol 2-3.	❑	❑	F
En un árbol B tiene que haber el mismo número de claves en el hijo izquierdo de la raíz que en el hijo derecho.	❑	❑	F
La especificación algebraica de la siguiente operación indica que se devolverá el número de elementos del conjunto multiplicado por 3 (C: Conjunto; x: Ítem): Operación(Crear) ⇔ 1 Operación (Insertar(C, x)) ⇔ 3 + Operación(C)	❑	❑	F
En la dispersión cerrada se pueden producir colisiones entre claves sinónimas y no sinónimas	❑	❑	V
El TAD Cola de Prioridad en el que no se permiten elementos repetidos, representado por una lista desordenada, tendrá coste O(n) para la inserción, con n el número de elementos del TAD.	❑	❑	V
En un árbol Leftist, el camino mínimo del nodo raíz siempre es mayor que 1.	❑	❑	F
Un bosque extendido en profundidad de un grafo dirigido al que se le añaden los arcos de cruce y avance es un grafo acíclico dirigido.	❑	❑	V
En C++, si un objeto se sale de ámbito entonces se invoca automáticamente al destructor de ese objeto.	❑	❑	V

Examen TAD/PED junio 2003

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 folios diferentes.
- Se dispone de 20 minutos para abandonar el examen sin que corra convocatoria.
- Las soluciones al examen se dejarán en la página web de la asignatura.
- 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 de notas de exámenes y prácticas**: 20 de junio. **ÚNICA fecha de revisión de exámenes:** 27 de junio, revisión de 17:00 a 19:00 horas en el laboratorio OIN01 del edificio EPSA IV. **Revisión de prácticas:** del 23 al 27 junio se colgará una lista en la puerta del despacho de Jesús Peral, para solicitar la revisión.
- **Los alumnos que estén en 5ª o 6ª convocatoria deben indicarlo en la cabecera de todas las hojas**

1. Utilizando exclusivamente las operaciones *crear* y *encolar* definir la sintaxis y la semántica de la operación *segundo* que actúa sobre una cola y devuelve el elemento situado en la segunda posición de la cola, es decir, el elemento situado a continuación de la cabeza.

2. El siguiente fragmento de código define una clase en C++ que representa una lista enlazada simple de enteros:

```
struct TNodo{
  int dato;
  TNodo *sig;
};
class TLista{
  public:
    TLista();
    TLista(int);
    ~TLista();
    ...
    ...

  private:
    TNodo *lista;
};
```

```
TLista::TLista(){
  lista = NULL;
}
TLista::TLista(int i){
  lista = new TNodo;
  lista->dato = i;
  lista->sig = NULL;
}
TLista::~TLista(){
  TNodo *borrar = lista;
  while(lista != NULL){
    lista = lista->sig;
    delete borrar;
  }
}
```

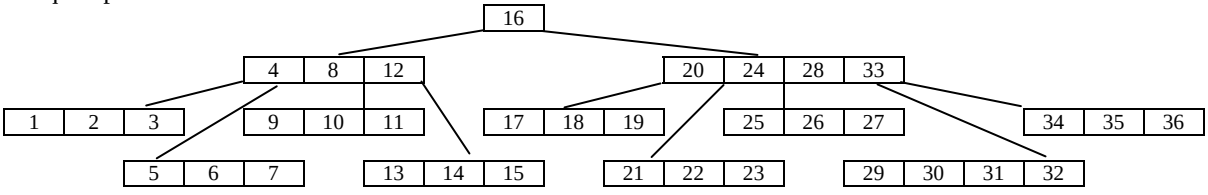
El siguiente fragmento de código define el método Longitud(), que devuelve la longitud de una lista (0 si la lista está vacía). Este código contiene una serie de errores (pueden ser de distinto tipo: en el planteamiento del algoritmo, lógicos y sintácticos); se pide indicar para cada uno de los errores los siguientes datos: línea, causa del error y código correcto. Se considera mal contestado: no detectar un error real, indicar un error que no existe y explicar incorrectamente un error real. Analizar el código línea por línea y escribir la solución por orden de aparición de los errores.

```
1   TLista::Longitud(void)
2   {
3     int i;
4     TNodo *aux;
5
6     aux = lis;
7     While(aux != NULL);
8     {
9       aux = aux.sig;
10      i++;
11    }
12    return i;
13  }
```

3. Calcula la complejidad temporal en el caso peor de la operación de eliminar de la estructura el elemento de mayor prioridad de la cola de prioridad utilizando las estructuras que se indican. Utiliza la notación asintótica O (big-omicron). El numero de elementos le llamamos *n*. Razona brevemente tu respuesta.

a) Lista enlazada no ordenada
b) Lista enlazada ordenada.
c) Vector ordenado.
d) Vector no ordenado.
e) Árbol binario de búsqueda.
f) Árbol AVL.
g) Árbol leftist

4. Sobre el siguiente árbol B con *m=7*, realizar el borrado de las siguientes claves según el orden que se especifica: 32, 35, 15. Indica qué operaciones de combinación o rotación se han realizado.



Examen TAD/PED junio 2003. Soluciones

1) segundo: cola → item

Var a,b:item; c:cola;
segundo(crear()) = error_item
segundo(encolar(crear(),a)) = error_item
segundo(encolar(encolar(crear(),a),b)) = b
segundo(encolar(c,a)) = segundo(c)

2) Línea: 1
Causa error: No se indica el tipo del valor de retorno
Error compilador: *ISO C++ forbids declaration of `Longitud' with no type*
Código correcto: `int TLista::Longitud(void)`

Línea: 3
Causa error: No se inicializa la variable `i`
Código correcto: `int i = 0;`

Línea: 6
Causa error: La variable `lis` no existe
Error compilador: *`lis' undeclared (first use this function)*
Código correcto: `lista`

Línea: 7
Causa error: `While`, porque en C++, las palabras reservadas se escriben en minúsculas
Error compilador: *`While' undeclared (first use this function)*
Código correcto: `while`

Línea: 7
Causa error: El bucle está vacío por el punto y coma (;): es un bucle infinito si `lista != NULL`, ya que el valor de lista no va a cambiar
Código correcto: `while(aux != NULL)`

Línea: 9
Causa error: `aux` es un puntero, por lo que se tiene que emplear el operador `->`
Error compilador: *request for member `sig' in `aux', which is of non-aggregate type `TNodo*'*
Código correcto: Existen dos posibilidades:

```
aux = aux->sig;
/* o también */
aux = (*aux).sig;
/* lo siguiente no vale, porque el . tiene mayor precedencia que * */
aux = *aux.sig;
```

3)

a) Lista enlazada no ordenada.
O(n).
Para poder encontrar el elemento de mayor prioridad en una lista enlazada no ordenada, tenemos que recorrer todos los elementos de la lista en el peor de los casos, por lo que tendremos que procesar los n elementos de la lista.

b) Lista enlazada ordenada.
O(1).
Si la lista está ordenada, el elemento de mayor prioridad de la lista enlazada será el primer elemento de la lista, por lo que solamente hay que procesar un elemento independientemente del número de elementos de la lista.

c) Vector ordenado.
O(1).
En un vector ordenado solamente tendremos que procesar el primer elemento del vector independientemente del numero de elementos que tenga la cola de prioridad.

d) Vector no ordenado.
O(n).
En el peor de los casos tendremos que procesar cada uno de los elementos del vector.

e) Árbol binario.

O(N).

Un árbol binario en el peor de los casos puede estar totalmente desequilibrado hacia una rama, de forma que este árbol tenga la misma estructura que una lista. En este caso que seria el caso peor, tendríamos que recorrer todos los elementos del árbol.

f) Árbol AVL.

$O(\log_2 n)$

Un árbol AVL, debe de estar casi equilibrado al menos con una diferencia de altura de 1. Es por este motivo por el que para encontrar el elemento de mayor prioridad solamente tenemos que recorrer el árbol en profundidad buscando el elemento que se encuentre más a la izquierda o más a la derecha, dependiendo si el elemento de mayor prioridad es el menor o el mayor del conjunto. Por lo tanto la complejidad de búsqueda viene determinada por la altura (h), $n=2h \Rightarrow h = \log_2 n$

g) Árbol leftist

$O(\log_2 n) \Rightarrow$ equivalente a $O(\log_2 m * \log_2 x)$ siendo $n = m+x$, con m y x los números de claves de cada subárbol de la raíz.

El numero de transformaciones necesarias para realizar una combinación en un árbol leftist es igual a la altura del árbol original menos 1. Esto es debido a que al realizar el borrado del elemento la altura del árbol decrece en uno. Esto no es vinculante porque el calculo de la complejidad se hace tendiendo n a infinito.

Por lo tanto

$n=2h \Rightarrow h = \log_2 n$

Se puede pensar que la complejidad puede ser otra debido a que la altura de un leftist no tiene porqué ser $h = \log_2 n$, ya que si el árbol está muy desequilibrado la altura será considerablemente mayor que $\log_2 n$, pero si suponemos esto llegamos a una situación más favorable en la combinación, por lo que la complejidad seria mejor todavía y estamos pidiendo la complejidad en el caso peor. El caso peor para un árbol leftist es precisamente que el árbol esté perfectamente equilibrado en cada uno de sus nodos, o lo que es lo mismo, que sea un árbol binario completo.

4) $m=7 \Rightarrow$ Número mínimo de claves en todos los nodos excepto la raíz = 3.

Borrado del 32: directo sin rotaciones o combinaciones

Borrado del 35: 1 combinación

Borrado del 15: 2 combinaciones, el árbol decrecerá: altura 2.