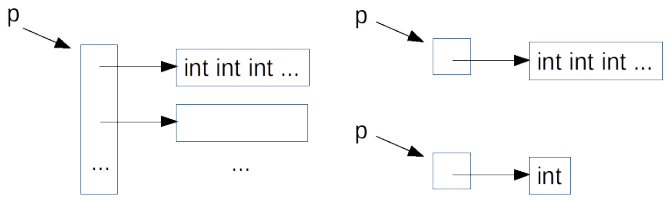
Lección 3

Test de comprensión

Contestar V/F las siguientes cuestiones.

1 [ ] La siguiente declaración: *int** p* permite crear las siguientes estructuras de datos en memoria:



2 [ ] Si la declaración del puntero anterior *p* se realiza como se muestra a continuación entonces *p* es creado como una variable residente en la pila de aplicación:

```
void f() {
  int **p;
  ...
}
```

3 [ ] Una matriz declarada como *int a[3][5]* se almacena en una zona contigua de memoria.

4 [ ] Sea *m1* una matriz declarada como *int m1[10][20]* y *m2* un puntero declarado como *int **m2* e iniciado mediante el siguiente código:

```
m2 = new int*[10];
for (int c = 0; c < 10; ++c) {
  m2[c] = new int[20];
}
```

Entonces el elemento existente en la posición *(3, 7)* se accede de igual manera con ambas estructuras de datos: *m1[3][7]* / *m2[3][7]*.

5 [ ] El siguiente código presenta memory leaks:

```
int *p = new int[1000];
for (int c = 0; c <= 1000; ++c) {
   p[c] = 0;
}
```

6 [ ] El siguiente código presenta heap overflows:

```
int *p;
for (int c = 0; c < 10; ++c) {
   p = new int[100];
}
```

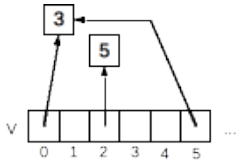
7 [ ] Una plantilla de clase instanciable para el tipo $T = int$ puede que no lo sea para el tipo $T = MiClase$, es decir, que una plantilla puede no aceptar cualquier tipo como parámetro.

8 [ ] A través el puntero *int **m* podemos manejar una matriz creada en memoria dinámica con un número de filas y columnas arbitrario.

Ejercicios

1. Implementar una función que int *bufferEnteros(int longitud, int val) que devuelva un puntero a un array en en memoria dinámica con la longitud indicada y relleno del valor dado. ¿Se borraría el array al salir la ejecución de la función?

2. Declarar un vector *v* (en la pila de la aplicación) de tamaño 100 y de tipo puntero a entero. Iniciar el vector a punteros nulos. Definir una función void escribe(int pos, int val) que permita introducir un entero en cualquier posición del vector, creando dinámicamente el entero pero sólo si no ha sido introducido previamente. En este caso el puntero apuntará al entero ya existente. La siguiente figura aclara la estructura de este vector tras las operaciones *escribe(0, 3)*, *escribe (5, 3)*, *escribe(2, 5):*



¿Que problemas plantearía el sobreescribir un valor? Por ejemplo, ejectuar escribe(0, 4) sobre el vector anterior.

3. Implementar un programa que realice reservas de memoria de 1kb hasta que agote la memoria, tras lo cual imprimirá un mensaje de error. Segunda parte: añadir el código necesario para liberar la memoria asignada al detectarse el error.

Lección 4

Test de comprensión

Contestar V/F las siguientes cuestiones.

1 [ ] Un vector dinámico es una estructura de datos básica que puede ser utilizada en la implementación de asociaciones y composiciones múltiples cuando no hay restricciones o necesidades especiales en las mismas.

2 [ ] Un dato almacenado en un vector dinámico que permanece en dicha estructura de datos a lo largo de todo su ciclo de vida nunca cambia su posición de memoria.

3 **[ ]** La siguiente definición de vector estático necesita de constructor copia y operador de asignación:

```
template<typename T> class MiVect{
  int tama;
  T *v;
public:
  MiVect(int n){ v=new T[tama=n]; }
  ...
};
```

4 [ ] La misma clase anterior debe definir así el operador corchete "[ ]" para que funcione correctamente (se obvian las comprobaciones de rango):

```
T operator[](unsigned i) { return v[i]; }
```

5 [ ] Si se ha instanciado en la clase *Biblioteca* a *Mivect* para implementar una relación de asociación con la clase *Libro* como:

```
MiVect<Libro*> estante;
```

Entonces el destructor de *Biblioteca* debe entonces eliminar estante ejecutando:

```
delete[] estante;
```

7 [ ] Es posible eliminar una posición de un vector dinámico en tiempo *O(1)* si no es necesario preservar el orden de los datos.

8 [ ] La implementación normal de un vector dinámico implementa una reducción del tamaño físico *tamf* a la mitad cuando el tamaño lógico *taml* cae por debajo de *tamf/2*.

Ejercicios

1. Partiendo de la implementación del vector estático *VEstaticoInt* que hay en las transparencias de teoría, añadir una operación *void redimensionar(int nuevoTam)* que permita modificar el tamaño del vector sin perder los datos guardados.

2. Partiendo de la implementación del vector dinámico VDinamicoInt que hay en las transparencias de teoría, añadir una operación *void aumenta(VDinamicoInt& v, int primero = -1, int ultimo = -1)* que añada los datos del vector indicado *v* al final. Pueden indicarse el primer y último elemento de *v* a copiar. El valor -1 indica desde el principio (para primero) y hasta el final (para ultimo).

3. Construir una clase *VDinamicoIntAjustado* que herede de la clase *VDinamicoInt* estudiada en la teoría e incluya una operación *ajustar()* que sustituya el array utilizado internamente como soporte por un array con el tamaño justo para contener los elementos almacenados en el vector.

Lección 5

Test de comprensión

Contestar V/F las siguientes cuestiones.

1 [ ] El operador de asignación de la clase Matriz<T>::operator= debe siempre destruir la matriz destino de la asignación

2 [ ] El operador de la clase Matriz<T>::operator+= devuelve el objeto resultado por copia

3 [ ] La implementación de conjuntos mediante vectores realiza la intersección en tiempo lineal

4 [ ] El problema que poseen los conjuntos de bits es que el ID de un elemento debe ser un entero desde 0 hasta el tamaño del conjunto -1. Para convertirlo en un ID genérico se necesita otra EEDD.

5 [ ] Para almacenar 3841 datos en un conjunto de bits, debo crear un buffer de 480 bytes.

6 [ ] *a.intersec(b+a) == a* es correcto

7 [ ] El resultado de realizar esta operación: *char mascara = 1 << (500 % 8);* es 0000100

Ejercicios

1. Partiendo de esta definición del atributo *m* de la clase *MiEstructura*:

```
int ***m;
```

   implementar el constructor de dicha clase para que sea:

   a) una matriz (nxm) de punteros a enteros aleatorios

   b) una matriz tridimensional (nxmxp) de enteros aleatorios

   c) un puntero a una matriz (nxm) de enteros aleatorios

2. Implementar las operaciones *void Matriz2D<T>::anadirFila()* y *void Matriz2D<T>::anadirColumna()* de la siguiente clase que implementa una matriz bidimensional:

```
template<typename T>
class Matriz2D {
  int nfilas, ncolumnas;
  T** datos;

public:
  Matriz2D(int nfilas, int ncolumnas):
    nfilas(nfilas), columnas(ncolumnas) {

    datos = new T*[nfilas];
    for (int f = 0; f < nfilas; ++f) {
      datos[f] = new T[ncolumnas];
    }
  }
  ...
};
```

3. Implementar la función *zoom-* sobre la definición de la matriz Matriz2D anterior para que reduzca una imagen almacenada en dicha matriz con k sucesivas compresiones de la imagen. En cada proceso de compresión, se toma la media aritmética entera de cada 4 casillas vecinas 2x2 y se sustituye por una única casilla con ese valor medio. Se lanza una excepción si *nfilas* $\%2^k$ *!=0* ó *ncolumnas* $\%2^k$ *!=0*

```
Matriz2D<T> Matriz2D::zoom- (k veces);
```

Lección 6

Test de comprensión

Contestar V/F las siguientes cuestiones.

1 [ ] El tiempo para eliminar un dato en una posición arbitraria de una lista es lineal.

2 [ ] El tiempo para eliminar un dato en una posición apuntada por un iterador es lineal.

3 [ ] Para eliminar e insertar un dato en posiciones intermedias de una lista se necesita un puntero a la posición anterior al dato que se va a insertar/borrar.

4 [ ] Se puede realizar la operación merge_sort entre dos listas ordenadas en tiempo O(n+m), siendo n y m los tamaños respectivos de ambas listas

5 [ ] Un dato que permanece en una lista cambia su posición a veces al sufrir la lista inserciones y borrados.

Ejercicios

1. Crear el constructor de la lista a partir de un objeto de la clase Vector:

```
ListaEnlazada<T>:: ListaEnlazada(Vector<T> v);
```

2. Implementar la primitiva que invierta el orden de una lista en tiempo lineal, de manera que el dato de la cabecera esté en la cola y viceversa.

```
void ListaEnlazada<T>::invertir();
```

Lección 7

Test de comprensión

Contestar V/F las siguientes cuestiones.

1 [ ] Una lista doblemente enlazada permite realizar busquedas binarias en tiempo $O(log\, n)$ si los datos se encuentran ordenados.

2 [ ] Iterar sobre un vector dinámico es siempre más rápido que sobre una lista enlazada (simple o doble).

3 [ ] El siguiente código inserta correctamente en un caso genérico (lista con datos), un nuevo nodo apuntado por p por delante de la posición del nodo apuntado por q en una lista doblemente enlazada y circular.

```
p->siguiente = q;
p->anterior = q->anterior;
q->anterior = p;
p->anterior->siguiente = q;
```

4 [ ] Transferir todos los nodos de una lista doblemente enlazada *l1* al final de tora lista enlazada *l2* requiere tiempo $O(1)$ (nota: *l1* queda vacía tras esta operación).

Ejercicios

1. Implementar la función *void ListaDEnlazada<T>::insertar(const IteradorD<T>& iterador, const T& dato)* para insertar un dato por delante del dato apuntado por el iterador. El iterador podrá apuntar a cualquier posición de la lista: delantera, intermedia, última o nulo (el significado de este último será insertar al final). Implementar la operación sin realizar llamadas a otras operaciones de la clase.

2. Implementar la función *void ListaCircularSimple<T>::inserta(const T& dato)* que inserta un dato en una lista circular simplemente enlazada, en la posición apuntada por el único puntero *cola* (recordar que no existe el puntero *cabecera*).

Lección 8

Test de comprensión

Contestar V/F las siguientes cuestiones.

1 [F] El contenedor *list<T>* de STL implementa el operador [] para acceder directamente al dato almacenado en una posición arbitraria.

2 [V] Se ha pensado en implementar una matriz para almacenar todos los jugadores convocados en todos los partidos de todas las jornadas de la liga de fútbol. Esta sería una posible estructura de datos para crear dicha tabla: vector<vector<vector <Jugador> > > jugadores.

3 [F] Para simular una lista circular (p.e. procesos de un S.O) en el que los datos entran siempre por el mismo lugar pero salen en cualquier momento es igualmente eficiente utilizar una lista (list) que un vector de STL.

4 [V] Se ha utilizado la siguiente estructura de datos para implementar un editor de texto interactivo: *list<vector<char> > texto*. La implementación de la operación *pulsarTecla(char c)* sería la siguiente: *i->insert(i->begin() + pos, c)*, siendo i un iterador de la lista que apunta a la línea donde se encuentra el cursor y *pos* un entero positivo que indica la posición del cursor en dicha línea.

5 [V] Sean dos conjuntos a y b de palabras, implementados usando dos vectores de STL instanciados al tipo *string*. Entonces obtener el vector $c = a \cap b$ (palabras que están en ambos conjuntos) requiere un tiempo cuadrático.

6 [F] La inserción de un dato d al principio de un vector v de STL mediante *v.insert(v.begin(), d)* requiere tiempo $O(1)$.

Ejercicios

1. Implementar una operación *vector<int> primosMenores(int n)* que devuelva la lista de los números primos menores que n usando la criba de Euler. Su funcionamiento es el siguiente:

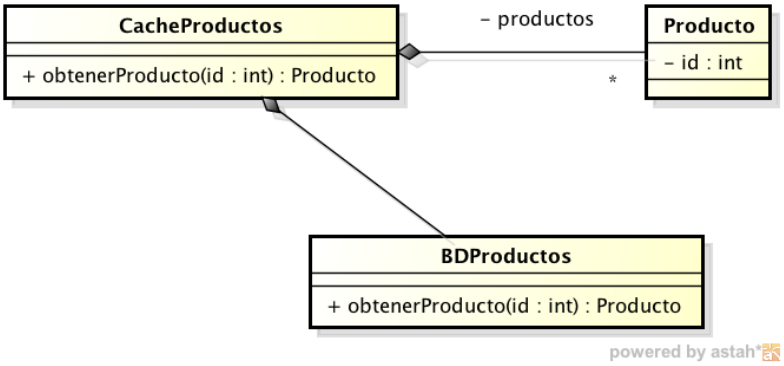
```
vector<int> primosMenores (int
num){
    list<int> l;
    for (int i=2; i< num; i++){
        l.push_back(i);
```

1. Generar una lista con los números desde 2 a n
2. El primer número de la lista es primo (añadir al resultado)
3. Multiplicar el primer número por todos los elementos de la lista y marcar los elementos resultantes
4. Eliminar el primer elemento y todos los elementos marcados
5. Volver al paso (b) si la lista no está vacía

2. Disponemos de una base de datos de productos con un número muy grande de entradas que debe ser consultada con una frecuencia muy alta para localizar productos a partir de su identificador. Para acelerar este acceso disponemos de una caché que guarda los últimos 1000 productos solicitados. El funcionamiento es el siguiente:

- El producto es solicitado a través de la operación *obtenerProducto()* de *CacheProductos*.
- Si existe en la caché, se devuelve inmediatamente.
- Si no existe en la caché, se solicita a la base de datos y se guarda en la caché por si vuelve a ser solicitado. Si la caché tiene 1000 elementos, se elimina el último que fue solicitado y se sustituye por éste. Finalmente se devuelve el producto.

Implementar la clase *CacheProductos*. La relación "productos no tiene por qué traducirse en una única estructura de datos, y pueden incluirse los atributos adicionales que se consideren necesarios.


CacheProductos
+ obtenerProducto(id : int) : Producto
- productos
*
Producto
- id : int
BDProductos
+ obtenerProducto(id : int) : Producto
powered by astah*

Lección 9

Test de comprensión

Contestar V/F las siguientes cuestiones.

1 [V] Un vector estático puede implementar eficientemente a pilas y colas estáticas.

2 [V] Una lista simplemente enlazada puede implementar eficientemente a pilas y colas dinámicas.

3 [V] Una deque puede implementar eficientemente a pilas y colas dinámicas.

4 [V] Se usa una pila para resolver aquellos procesos recursivos que agotan la pila.

5 [F] Una cola con prioridad siempre tendrá un push() en O(1) independientemente de la implementación concreta (vectores, vector de listas o listas de listas)

6 [F] Una stack o una queue se implementa por defecto sobre un std:list pero se puede cambiar en ambos casos el contenedor.

7 [V] Para hacer que una priority_queue de STL considere como dato más prioritario al mayor, se debe definir con el operator greater

8 [ ] Necesito una pila para eliminar la recursividad de la sucesión de Fibonacci (Ejercicio 1)

Ejercicios

1. Eliminar recursividad al cálculo de la sucesión de Fibonacci:

```
int fibonacci(int n){
    if(n==1 || n==2)
        return 1;
    else
        return fibonacci(n-1)+fibonacci(n-2);
}
```

2. Usar una std:stack para mostar en pantalla los datos de una std:queue en orden inverso

3. Implementar la siguiente función que evalúa una expresión postfija devolviendo su valor usando una pila. En una expresion postfija el operador ocupa la posición después de los operandos. Un ejemplo de expresión infija es: $(2+(3*4)) = x$ y su expresión postfija es $2\ 3\ 4\ *\ +\ x\ =$

```
float evaluaPostfija(string &exp)
```

4. La clase Empleado, definida a continuación, establece una jerarquía en la empresa, de manera que un empleado tiene categorías del 1 al 5 para indicar al director general (categoría 1) y al empleado de mantenimiento (categoría 5). El resto de trabajadores poseen categorías intemedias. Un empleado también tiene un número de empleado que se les asignó de forma correlativa según iban siendo contratados por la empresa desde sus inicios. Un empeado de contabilidad es el más antiguo actualmente con NumEmpl = 5 mientras que el director general es el 22.

Definid la función de comparación *CategorizarEmpleado* para la gestión de los pedidos mensuales de material de oficina de modo que priorice en primer lugar a la categoría profesional y en segundo lugar al número de empleado.

```
class Empleado{
    int CatProf;
    int NumEmpl;
};
priority_queue <Empleado, CategorizarEmpleado> pedidos;
```

Lección 10

Test de comprensión

Contestar V/F las siguientes cuestiones.

1 [ ] Conocer al altura de un ABB da información sobre el número de datos que contiene

2 [ ] Dos árboles ABB equivalentes pueden tener diferente altura, raíz y hojas.

3 [ ] Un árbol binario que representa expresiones matemáticas se resuelve mediante un recorrido en postorden.

4 [ ] Para obtener los datos ordenados de un ABB se hace un recorrido en preorden

5 [ ] Todo proceso recursivo que opere sobre un ABB se puede resolver de forma iterativa mediante una pila de punteros a nodos de tipo ABB

6 [ ] Al insertar la siguiente secuencia en un ABB: {4, 3, 7, 12, 2, 6, 5, 13}, el borrado del 12 implica una llamada a la función borraMin()

7 [ ] Para recorrer un árbol binario por niveles se necesita una cola. Este recorrido pasaría primero por la raíz, luego por raíz-> izq, luego por raiz->der, raiz->izq->izq, raíz->izq->der, etc...

8 [ ] Los recorridos recursivos Preorden, Inorden y Postorden permiten iterar sobre los árboles hacia delante y detrás.

Ejercicios

1. Listar los datos por niveles de un árbol binario tal y como se explica arriba en el test y devolverlos en un vector:

```
vector<T> recorreNiveles ();
```

2. Implementar una función que devuelva la altura de un ABB:

```
int  ABB::altura ();
```

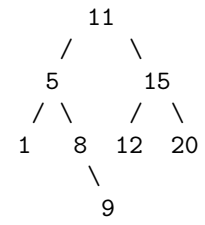
3. Implementar la versión iterativa de la búsqueda de un dato en un árbol ABB

```
bool Abb<T>::buscarNR (T &ele){
```

Lección 11

Test de comprensión Contestar V/F las siguientes cuestiones.

1 [ ] Dado el siguiente árbol AVL, la inserción de 10 requiere una rotación doble a derecha.



2 [ ] Si se introducen datos ordenados de forma ascendente en un AVL, el tipo de rotación que se realizaría siempre sería el caso 4.

3 [ ] En un árbol AVL tras una inserción el proceso de ajuste requiere a lo sumo una única rotación simple o doble.

4 [ ] En un árbol AVL, tanto el borrado como la inserción requieren la localización de algún nodo hoja durante el proceso.

5 [ ] Se pueden listar los datos de un árbol AVL o ABB en orden inverso a su definición sin necesidad de añadir un puntero al padre.

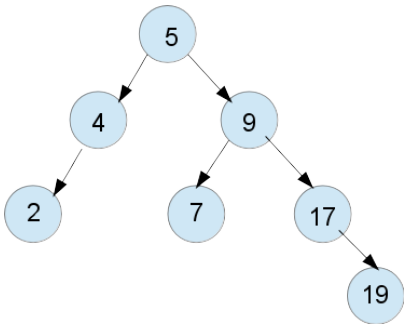
6 [ ] No es posible que un árbol AVL tenga un nodo hoja a una profundidad 4 y otra a profundidad 8.

7 [ ] En los árboles AVL las rotaciones garantizan que el número de descendientes por la izquierda y derecha de un nodo difiere a lo sumo en 1.

8 [ ] Es posible que exista una secuencia de datos que al ser insertarda en un árbol AVL no provoque rotaciones

Ejercicios 1. Dado el siguiente árbol AVL, dibujad las modificaciones de dicho árbol tras realizar las siguientes inserciones (I) y borrados (B):

I(6), I(13), I(10), B(19), I(1)



2. En el control de acceso a una tienda Web se desea mantener una estructura de datos con la lista de de usuarios que han entrado en el sistema. Dicha lista se almacena en un vector dinámico pero para acelerar las búsquedas se usan dos árboles AVL auxiliares, ordenados por nombre de usuario y por fecha/hora de último acceso (en un long con formato AAAAMMDDHHMMSS). Las clases almacenadas en el vector y en los dos árboles AVL son las siguientes:

```
/* EEDD Vector */
class Usuario {
public:
   string nombre;
   long ultimoAcceso;
   string e-mail;
};
```

```
/* EEDD AVL usuario por nombre */
class UsuarioPorNombre {
public:
   string nombre;
   Usuario *usuario;

   bool operator<(const UsuarioPorNombre& u) {
```

```
      return nombre < u.nombre;
   }
};
```

```
/* EEDD AVL usuario por fecha último acceso */
class UsuarioPorAcceso {
public:
   long ultimoAcceso;
   Usuario *usuario;

   bool operator<(const UsuarioPorAcceso& u) {
      return ultimoAcceso < u.ultimoAcceso;
   }
};
```

Mostrar las estructuras de datos descritas durante las siguientes inserciones (agrupar pasos siempre que no se dificulte la lectura del ejercicio):

I(Usuario("pruiz", 20161205101402, "pruiz@gmail.com")), I(Usuario("alopez", 20170103122000, alopez@gmail.com)), I(Usuario("igomez", 20161110091523, "igomez@hotmail.com")), I(Usuario("mhernandez", 20161122173045, "mhernandez@yahoo.es")), I(Usuario("yperez", 20170120193215, "yperez@hotmail.com"))

Lección 12

Test de comprensión Contestar V/F las siguientes cuestiones.

1 [ ] Implementar un heap mediante nodos y punteros al igual que el resto de árboles binarios tiene dos graves inconvenientes: consume mucha más memoria y la inserción en la siguiente posición libre del último nivel (durante los push) o el borrado de la última posición del último nivel (durante los pop) no podría implementarseen tiempo constante.

2 [ ] La operación pop() es más eficiente en un heap que en una cola con prioridad montada mediante una lista de listas.

3 [ ] La siguiente tabla referente a la eficiencia de las distintas implementaciones de una cola con prioridad es correcta (n es el número de datos y p el número de valores de prioridad distintos):

Implementación	**push**	**pop**
vector de listas	$O(1)$	$O(p)$
lista de listas	$O(p)$	$O(1)$
heap	$O(\log n)$	$O(\log n)$

4 [ ] Un heap permite obtener el dato con menor prioridad en O(1).

5 [ ] Un heap es un árbol binario equilibrado en altura.

6 [ ] La unión de dos conjuntos disjuntos de tamaños n y m puede llevarse a cabo mediante una operación en O(1)

7 [ ] La operación busca() en conjuntos disjuntos es $O(n)$ y $\Omega(1)$

Ejercicios 1. Mostrar la evolución de un heap tras la realización de las siguientes operaciones (mostrar sólo la representación compacta en el vector de soporte): push(8), push(7), push(1), push(2), push(9), push(5), pop(), pop(), push(3). Menor valor numérico implica mayor prioridad.

2. Dada la siguiente definición de un heap, implementar la operación *push()*. Se considerarán con mayor prioridad los dato de tipo T con menor valor al ser comparados con el operador $<$.

```
template <typename T>
class Heap {
  T *arr; // El contenedor
  int tamal, tamaf;   // tamaños  logico y físico del contenedor
  public:
    Heap(int aTamaf = 500) {
      arr = new T[tamaf = aTamaf];
      tamal = 0;
    }
    ~Heap() { delete[] arr; }

    void push(const T &t); // Implementar
};
```

3. Definir la interfaz de la clase *ConjuntoDisjunto* que representa a una colección de conjuntos disjuntos con la estructura que hemos estudiado en la teoría. Implementar la función de búsqueda de un dato en el conjunto:

```
int ConjuntoDisjunto::busca(int dato);
```

Lección 10

Test de comprensión

Contestar V/F las siguientes cuestiones.

1 [ ] Si la clase ClassA tiene el siguiente atributo: map<string, *ClassB> atributo; entonces necesariamente la relación entre ClassA y ClassB es de asociación.

2 [ ] Si esta sentencia es válida: Entonces micontainer puede tener esta definición: map <int, vector<int> >micontainer;

```
int nuevoValor = 7;
it=micontainer.find(clave);
(*it).second[i] = &nuevoValor;
```

Entonces *micontainer* puede tener esta definición: *map <int, vector<int> >micontainer;*

3[ ] El objecto mc se define como: map <int, miClase> mc; ¿se podría realizar la siguiente operación sobre mc?

```
map<int, miClase>:: iterator it = mc. begin(); *(it).first = 5;
```

4 [ ] El contenedor de STL más adecuado para albergar las reservas de un restaurante para tener acceso a éstas por fecha es un mapa de la siguiente forma: multimap <Fecha, Reserva>

5 [ ] La correspondiente definición de una matriz dispersa en STL según la Lección 7 es: vector <list <int> > matrizDis;

6 [ ] La siguiente sentencia: v["María"] = dato; es válida si v representa a un deque.

7 [ ] Un map definido como: <int, ClaseA> puede sustituirse por un set<ClaseA> si ClaseA tiene sobrecargado el operator< y la clave entera forma parte de la clase ClaseA

8 [ ] Si en el caso anterior el operator< ya está usándose para otro tipo de ordenación sobre ClaseA, entonces se puede usar la definición: set<ClaseA, comparaClaseA>, siendo comparaClaseA una clase de comparación para ClaseA

Ejercicios

1. Crear una rutina que genere un multset<int> con 10.000 números aleatorios entre 0 y 999. Para comprobar si la distribución de números aleatorios es uniforme contar el número mínimo y el máximo de repeticiones de un número y mostrarlo por pantalla. Realizar esta operación en O(n).
2. Crear un set<int> a partir de los datos del multiset anterior y comprobar si tiene un tamaño de 1000.
3. Crear un set a partir de las palabras del diccionario de las prácticas. Investigar cual de las funciones de búsqueda es más adecuada find/lower_bound/upper_bound para encontrar todas las palabras que empiezan por una letra (por ejemplo la m) y listarlas en pantalla.
4. Crear una clase de comparación *compFecha* para la clase Fecha de la lección y realizar las siguientes operaciones sobre una posible estructura de datos para mantener las reservas: map<Fecha, vector<Reserva>, compFecha> reservas; siendo la clase Reserva

```
class Reserva{
    Fecha f;
    Cliente c;
    int numPersonas;
public:
... };
```

   Y realizar las siguientes operaciones:

   a) Devolver todas las reservas de una fecha
   b) Devolver todas las reservas entre dos fechas f1 y f2 (f1 < f2)
   c) Disponibilidad en una fecha, es decir, el núemero de personas que caben hasta completar aforo (Suponemos una constante asociada MAX_COMENSALES y que se sirve un único turno de comidas)

Lección 14

Test de comprensión

Contestar V/F las siguientes cuestiones.

1 [ ] Toda función de dispersión debe acabar con %tamaTabla

2 [ ] El djb2 no es una función de dispersión de cadenas

3[ ] Si la tabla A tiene un lambda $\lambda = 0.5$ y en la tabla B, $\lambda = 0.75$, entonces la tabla B tiene más datos que la A

4 [ ] Si la tabla A tiene un lambda $\lambda = 0.5$ y en la tabla B, $\lambda = 0.75$, entonces la tabla B está porcentualmente más llena que la A

5 [ ] En STL la dispersión abierta se define como un list< list <Entrada <T> > >

6 [ ] En dispersión abierta se sabe que la función de dispersión es buena conociendo el tamaño de las listas de entradas

7 [ ] El djb2 diferencia las posiciones de las letras de CASA y SACA mediante desplazamientos a nivel de bits con la cadena entrante

Ejercicios

1. Instanciar la clase DispAbierta<String> con las palabras del diccionario. Crear una función de dispersión abie que utilice djb2 y un lambda = 0.7. Crear la función de búsqueda de una palabra.

2. Crear una rutina que busque un conjunto de 10000 palabras del corpus y comparar los tiempos con respecto a hacer lo mismo usando el AVL<String>. Nota: no incluir el tiempo de lectura del fichero.

Lección 15

Test de comprensión

Contestar V/F las siguientes cuestiones.

1 [ ] Los agrupamientos secundarios se producen cuando claves que son dispersadas a posiciones diferentes siguen la misma secuencia de búsqueda para localizar una posición disponible.

2 [ ] Una tabla de dispersión cerrada con cubetas disminuye el riesgo de colisiones, pero aun así necesita una estrategia de resolución de colisiones.

3[ ] Para evitar tanto agrupamientos primarios como secundarios es perferible utilizar dispersión cuadrática que dispersión doble.

4 [ ] Una tabla de dispersión cerrada construida correctamente permite localizar un dato por su clave de manera más eficiente que un árbol AVL.

5 [ ] La técnica de dispersión doble permite evitar agrupamientos primarios pero no secundarios.

6 [ ] Las posiciones vacías y disponibles (contuvieron un dato en el pasado pero fue borrado) se manejan de igual forma a la hora de insertar un dato en una tabla de dispersión cerrada.

7 [ ] Es posible que sea necesario seguir el proceso de búsqueda en una tabla hash con dispersión cerrada y cubetas de tamaño 5 cuando se llega a una cubeta con 3 datos.

Ejercicios

1. Dibujar el estado de una tabla de dispersión tras realizar las siguientes operaciones de inserción (I) y borrado (B) usando la función de dispersión: $h(x) = (x + i^2)\,\%t$, para un tamaño de tabla 11. Emplead las marcas de borrado correspondientes: I(7), I(17), I(24), I(13), I(18), I(9), I(28), B(7), I(11), B(13), I(31)

2. Definid una función de dispersión doble con h1(x)=x %11 y h2(x)=x %7 para una tabla con dispersión cerrada de tamaño igual a 11 en el que se inserten/borren los siguientes datos enteros:

   a) I(3), I(24), I(13), I(10), I(40), B(13), I(2), I(6), I(9), B(24), I(25)

3. Implementar la función *bool DispCerrada<T>::buscar (long clave, T &dato);* que realiza la búsqueda de un dato a partir de su clave en la tabla de dispersión cerrada con exploración lineal que está definida a continuación. El dato deberá ser devuelto sobreescibiendo el propio parámetro *dato*, y el retorno de la función será *true* si la clave existe o *false* en caso contrario. Codificar la clave con valores negativos para representar los estados de borrado y disponible.

```
template<class T>
class Entrada {
   long clave;
   T dato;
   Entrada(long aClave, T aDato): clave(aClave), dato(aDato) {}
};

template<class T>
class DispCerrada {
   vector<Entrada> tabla;
   long fDisp(long clave);
public:
   ...
   bool buscar (long clave, const T &dato);
};
```

Lección 16

Test de comprensión

Contestar V/F las siguientes cuestiones.

1 [ ] Un grafo que representa carreteras nacionales puede considerarse un grafo ponderado, no dirigido y posiblemente cíclico.

2 [ ] Como un árbol es un grafo conexo sin ciclos entonces un recorrido en anchura (BFS) empezando en la raíz es un reorrido del árbol por niveles

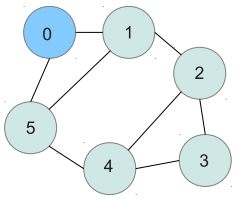
3 [ ] Averiguar las aristas de entrada a un nodo en un grafo dirigido implementado mediante listas invertidas es más eficiente que implementado sobre una matriz de adyacencia.

4 [ ] El recorrido en profundidad de un grafo usa normalmente una cola mientras que en anchura su implementación es con pila o recursiva.

6 [ ] Un árbol binario es un grafo normalmente dirigido, conexo y libre de ciclos donde el grado de salidad de todos los nodos es menor o igual a 2.

Ejercicios

1. Listar el conjunto de nodos cuando en el siguiente grafo se realiza un recorrido en profundidad (DFS) y en anchura (BFS), partiendo siempre del nodo 0.



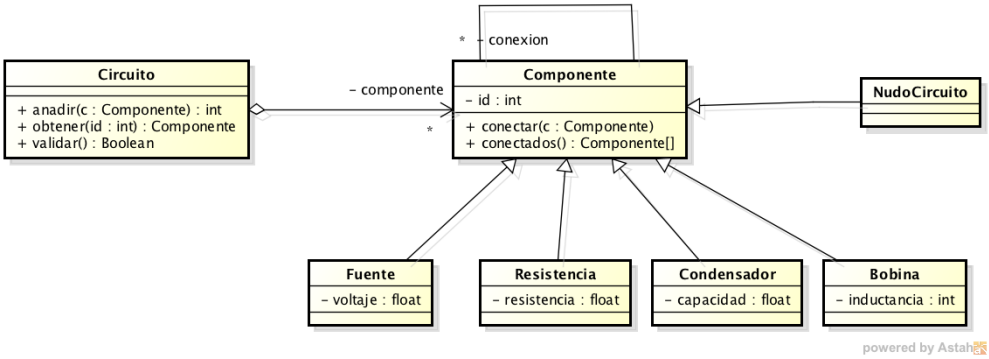
2. Dado un grafo implementado mediante una lista invertida, implementar una operación *int Grafo::numComponentesConexas(*
que devuelva el número de componentes conexas del grafo. Para ello realizar una exploración en anchura o profundidad empezando en un nodo cualquiera, marcando todos los nodos visitados. Si quedan nodos sin marcar, tomar uno de estos nodos y repetir el procedimiento anterior. Devolver el número de recorridos realizados (número de componentes).

3. Una aplicación de diseño de circuitos eléctricos necesita una estructura de datos para organizar los distintos componentes y la topología del circuito. El diagrama UML adjunto describe las clases de objetos necesarias.

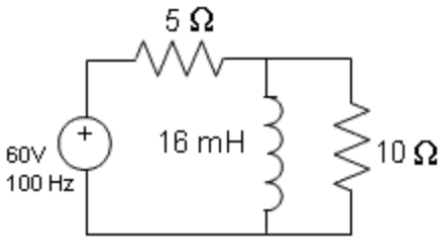
   - *Fuente*, *Resistencia*, *Condensador* y *Bobina* representan los componentes que admite la aplicación en su versión inicial. Cada uno de estos tres componentes viene definido por sus características eléctricas (voltaje, intensidad, resistencia, capacidad, inductancia). Estas clases heredan de *Componente*, que contiene un identificador único por componente y asociaciones a los otros dos componentes a los que está conectado. La operación *conectar()* permite conectar un componente con otro. Esta conexión debe ser siempre bidireccional: al conectar el componente A con B automáticamente se conecta B con A. La operación *conectados()* devuelve la lista de componentes conectados. Existe un componente especial denominado *NudoCircuito* que sirve para ayudar a crear bifurcaciones (aunque no tiene ninguna propiedad eléctrica). Este componente especial sí puede interconectar un número arbitrario de componentes (más de dos).
   - Circuito es la clase principal que organiza los componentes del circuito. Dispone de una operación para añadir un componente al circuito (*anadir()*) devolviendo el id del mismo (debe generar un identificador único para cada componente). De forma similar para obtener un componente a partir de su id usaremos *obtener()*. La operación más importante es *validar()*, que comprueba que el circuito está cerrado. Para ello deben verificarse las siguientes propiedades, en cuyo caso se devuelve true:

     a) Todos los componentes ordinarios tienen 2 conexiones (para evitar circuitos abiertos). Los nudos tienen 2 o más.

     b) Empezando en un componente y siguiendo las conexiones se pueden alcanzar todos los demas componentes (para evitar que haya varios circuitos separados).



Implementar las clases *Circuito*, *Componente*, *NudoCircuito* y *Fuente* (considerar las otras tres clases implementadas, eligiendo las estructuras de datos adecuadas e implementando los algoritmos descritos. El siguiente es un ejemplo de creación y validación del circuito mostrado en la figura (este código debería funcionar con la implementación que se pide):



Lección 17

Test de comprensión

Contestar V/F las siguientes cuestiones.

1 [ ] Encontrar el punto más cercano a otro en una malla regular puede suponer recorrer más de 20 casillas.

2 [ ] Una malla regular en 3D tiene requiere más tiempo de acceso para localizar un punto que una en 2D.

3 [ ] Si la malla regular albergara una simulación de partículas en el espacio que se mueven despacio (menos del tamaño de una celda por quantum de tiempo), entonces localizar hacia donde se dirige una partícula del instante t_1 al t_2 es un tiempo constante.

4 [ ] Se permite que un quadtree esté desequilibrado, es decir, que unas ramas sean mucho más largas que otras.

5 [ ] Dada una codificación del nodo del modo: *00-10-11-01-11-10*, se puede conocer exactamente el área del plano a la que representa.

6 [ ] Un octree es la representación 3D del quadtree y en este caso cada nodo tiene (4x4) nodos hijos.

Ejercicios

1. Definid la clase MallaRegular3D<T> y su operación de búsqueda de un dato.
2. Realizar sobre la definición de malla anterior, la función MallaRegular3D<Particula>::mueve(Particula *p, float incrX, float incrY, float incrZ), siendo los valores dados como parámetro la propia partícula, y el incrementeo del desplazamiento que sufre en (x,y,z) del instante t_i al t_{i+1}. Asumimos que este desplazaminto será siempre menor que el tamaño de una casilla. La casilla por tanto puede acabar en otra casilla vecina.
3. Definid la clase Quadtree<T> con puntero al padre.
4. Realizar la operación de inserción sobre la clase Quadtree anterior.

Lección 19

Test de comprensión

Contestar V/F las siguientes cuestiones.

1 [ ] El siguiente orden es correcto en relación a la velicidad de acceso de los dispositivos: DVD, CD, memoria flash, HHD, SSD.

2 [ ] Es conveniente conocer el tamaño de bloque del ordenador para ajustar correctamente el número de campos por registro.

3 [ ] Es preferible el uso de ficheros de texto como ficheros internos a las aplicaciones y los binarios como ficheros de intercambio entre aplicaciones.

4 [ ] Los registros con longitud fija permiten acceder mediante un acceso al dato posicionado en la posición lógica k, mientras que los de longitud variable necesitan algún mecanismo de indexación.

5 [ ] Para eliminar un registro en un fichero que maneja pila de borrados se necesita un número indeterminado de accesos a disco.

6 [ ] El proceso de lectura de todos los registros de un fichero de manera secuencial no necesita mover explícitamente el apuntador del fichero.

7 [ ] El proceso de compactación de un fichero que maneja pila de borrados para borrar definitivamente los huecos, se puede llevar a cabo sin necesidad de recorrer los borrados en modo pila y sobre el mismo fichero.

Ejercicios

1. Transformar la clase FicheroLibro en la clase Fichero<T> para que sirviera para cualquier T.

   a) Para no modificar la estructura del código de esta clase FicheroLibro, ¿qué operaciones debe sobrecargar toda clase T y qué características deben tener sus atributos para ser fácilmente instanciada?

   b) Escribir la clase Libro que permitira ser instanciada correctamente en Fichero<T>

2. Definid la clase FileTHash<T> que la dota de la funcionalidad de la tabla hash cerrada (Práctica 7) pero ubicada totalmente en fichero. El tamaño de la tabla hash es en número de bytes el producto del tamaño de una Entrada (struct con el dato, la clave numérica y el entero definiendo el estado) por el tamaño lógico de la tabla (número primo) que se estima tras aplicarle un factor de carga del 0.5. En concreto las operaciones serían:

```
FileTHash<T>::FileTHash(tamTabla);
bool FileTHash<T>::insertar(long int clave, const T& dato)
bool FileTHash<T>::borrar(long int clave, const T& dato)
T FileTHash<T>::buscar(long int clave, const T& dato)
unsigned int FileTHash<T>::tamaTabla()
```

Lección 19

Test de comprensión

Contestar V/F las siguientes cuestiones.

1 [ ] Cuando un índice simple es modificado, los cambios son inmediatamente reflejados en el fichero donde se guarda el índice.

2 [ ] Si se usa un mapa de STL para representar un índice simple en memoria, entonces puede localizarse cualquier registro del fichero de datos con $O(logn)$ accesos a disco.

3 [ ] Aunque un índice simple es una técnica para acceder eficientemente a un fichero en memoria secundaria, éste se mantiene íntegramente en memoria primaria mientras está siendo utilizado.

4 [ ] Si un fichero indexado el índice en memoria se desactualiza por algún problema en la aplicación, la operación de borrado puede realizar un borrado de un registro incorrecto.

5 [ ] Al arrancar la aplicación, es necesario siempre reconstruir el índice simple a partir del correspondiente fichero de datos.

6 [ ] Al contrario que en un índice primario, en un índice secundario pueden existir múltiples claves repetidas.

Ejercicios

1. Sea un fichero de texto con un diccionario castellano de definiciones (*diccionario.txt*), con la siguiente estructura:

```
agroturismo
m. Turismo rural, especialmente el que incluye actividades agrícolas y ganaderas.
amigovio
(Fusión de amigo y novio). m. y f. coloq. Arg., Méx., Par. y Ur. Persona que mantiene con...
amniocentesis
(Del ingl. amniocentesis, y este del gr. ἀμνίον amnían 'amnios' y κέντησις kéntēsis...
...
```

Donde una línea contiene una palabra y la siguiente su definición. Implementar una clase *DiccionarioCastellano* que:

- Tenga un constructor donde procese el fichero con el diccionario y construya un índice simple en memoria, implementado mediante un mapa de STL
- Implemente una operación *string buscarDefinicion(const string& palabra)* que devuelva la definición de un término.

Lección 19

Test de comprensión

Contestar V/F las siguientes cuestiones.

1 [ ] Al igual que los índices simples, los árboles B se mantienen permanentemente en memoria. Su ventaja respecto a los índices es la mayor eficiencia en la búsqueda de una clave: $O(\log_2 n)$ de los índices frente a $O(\log_m n)$ de los árboles B, siendo m el órden del mismo y siempre mucho mayor que 2.

2 [ ] Es posible que un árbol B de orden 20 tenga menos altura que otro árbol B con orden 22 albergando exactamente los mismos datos.

3 [ ] Un árbol B utiliza rotaciones para mantener el equilibrio en altura.

4 [ ] Un árbol B de orden cinco que está indexando 500 datos puede tener altura 3 (4 niveles)

5 [ ] En el proceso de inserción de un dato en un árbol B se puede repartir la carga de datos con nodos hermanos en el caso de que dicho dato no quepa en el nodo asignado.

6 [ ] En un árbol B de orden 256 no puede ocurrir nunca que la raiz tenga menos de 128 claves/punteros a descendientes.

Ejercicios

1. Generar paso a paso el árbol B de orden 3 tras la inserción (I) y borrado (B) de los siguientes datos correspondientes a cuentas de usuario de un sistema web:

   I(dani02), I(superelias), I(mariluna), I(kiko12), I(sergiolo), I(susi17), B(mariluna), B(dani02), I(albamanecer), I(kika), I(asun), I(begood), I(bemad)

2. Dado el siguiente árbol B de orden 3, indicar la configuración del árbol tras la inserción/borrado de cada una de las siguientes claves: I(10), B(3), B(9), B(11), B(5)