Guía para la realización de la Práctica 2 – Tema 2

Programación de Sistemas y Concurrencia

Pasos

- Leer el enunciado completo de la práctica
- Crear un proyecto C en Eclipse
- Descargar los recursos asociados a la práctica y guardarlos en el proyecto
  - arbolbb.h
  - Practica2B.c
- Seguir las instrucciones de las transparencias siguientes
- Enviar las dudas que tengáis al foro creado para la práctica en el CV
- Apuntarse para tutorías virtuales en los horarios indicados en el CV

Instrucciones

- La práctica se puede dividir en tres partes:
  - Parte 1: Lectura/escritura de fichero binario
    - Hay que crear un fichero binario en el que se almacenarán N datos enteros generados de forma aleatoria. Después, el fichero binario se leerá y los datos se mostrarán por pantalla
  - Parte 2: Implementación del módulo que implementa un árbol binario de búsqueda
    - Hay que implementar un árbol binario de búsqueda mediante listas enlazadas. El fichero arbolbb.h con la definición de los tipos de datos necesarios y las funciones a implementar se proporciona como parte de los recursos de la práctica.
  - Parte 3: Uso del módulo implementado en la Parte 2 para ordenar y eliminar las repeticiones de los datos contenidos en el fichero de la Parte 1
    - Los datos del fichero creado en la Parte 1 se almacenarán en un árbol binario de búsqueda utilizando el módulo implementado en la Parte 2. Posteriormente se volcará el contenido del árbol en un fichero para obtener los datos ordenados y sin repeticiones

Instrucciones

- Parte 1: Lectura/escritura de fichero binario

Se desea generar un fichero con `TAM` números aleatorios comprendidos entre 0 y `TAM-1`. Para ello se leerá el valor de `TAM` y después se generarán de forma aleatoria los números y se guardarán en el fichero binario cuyo nombre se lee de teclado.
Los números se generan llamando a la función rand que nos devuelve un número entre 0 y `RAND_MAX` cuyo valor depende de la implementación del compilador. Basta obtener `rand()%TAM` para obtener el valor entre 0 y `TAM-1`.
También es conveniente llamar a la función que crea la semilla de los números aleatorios se la siguiente forma:

```
srand (time(NULL));
```

De esta forma, nos aseguramos que cada ejecución obtiene unos valores distintos.

Una vez generado el fichero lo cerramos y comprobamos su contenido leyéndolo y escribiendo su contenido por pantalla.

Instrucciones

- Parte 1: Lectura/escritura de fichero binario
  - Ir al fichero Practica2BB.c
  - Echad un vistazo a la primera parte del código de la función main()
    - Podéis comentar el resto del código del main(), menos el return último, para probar la primera parte

```
char nfichero[50];
printf ("Introduce el nombre del fichero binario:\n");
fflush(stdout);
scanf ("%s",nfichero);
fflush(stdin);
creafichero(nfichero);
printf("\nAhora lo leemos y mostramos:\n");
muestrafichero(nfichero);
fflush(stdout);
```

Se le pide al usuario el nombre del fichero que quiere crear

Se llama a la función para crear el fichero

Se muestra por pantalla el contenido del fichero creado anteriormente

Se llama a la función para mostrar el contenido del fichero

Instrucciones

- Parte 1: Lectura/escritura de fichero binario
  - Implementación de la función creafichero
  - Recibe el nombre del fichero como parámetro
- Mirar transparencias 131 y 132
- Mirar código de ejemplos adjuntos

```
void creafichero(char* nfichero){
    //Declarar variable para el descriptor del fichero
    //y otras necesarias

    //Abrir el fichero como binario para escribir

    //Si no se ha podido crear correctamente mostrar error

    //Si se ha creado, escribir en el fichero N numeros aleatorios
        //Pedirle al usuario la cantidad de numeros a escribir
        //Generar tantos numeros aleatorios como se indique
        /** Para crear los numeros aleatorios
            srand(time(NULL)); //crea la semilla una sola vez

            int n;
            n = rand()%TAM; //crea un numero entre 0 y TAM-1
            //Esta linea se ejecutará para cada valor generado
        **/
        /**Para cada numero aleatorio escribirlo en el fichero
           fwrite(&n, sizeof(int), 1, pf);

           n - variable que almacena el numero a escribir
           pf - descriptor del fichero
         **/
         //Crear el fichero
}
```

Instrucciones

- Parte 1: Lectura/escritura de fichero binario
  - Implementación de la función muestrafichero
  - Recibe como parámetro el nombre del fichero
- Mirar transparencias 130 y 132
- Mirar código de ejemplos adjuntos

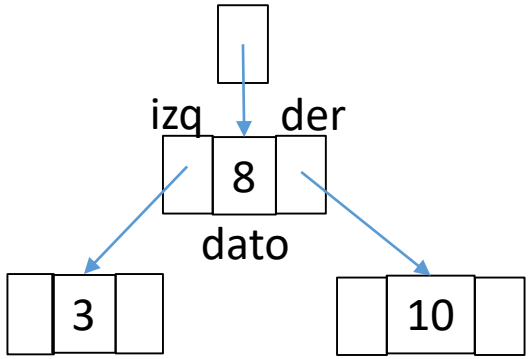
```
void muestrafichero(char* nfichero){
    //Declarar variable para el descriptor del fichero
    //y otras necesarias

    //Abrir el fichero como binario para leer

    //Si no se ha podido abrir correctamente mostrar error

    //Si se ha abierto correctamente,
    //leer mientras haya datos en el fichero con el bucle:
    while (fread(&n,sizeof(int),1,pf)==1){
        //escribir por pantalla
    }
    //n es la variable donde guardamos el dato leído
    //pf es el descriptor del fichero

    //Cerrar el fichero
}
```

Instrucciones

- Parte 2: Creación del módulo árbol binario de búsqueda

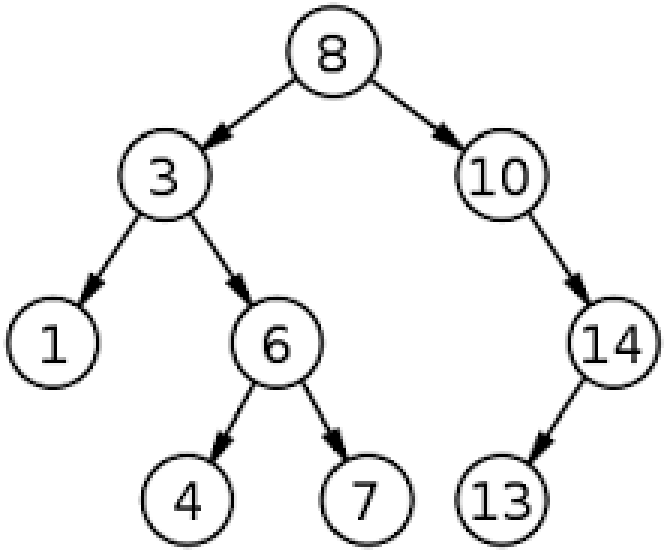
Ahora queremos volver a leerlo para obtener sus datos ordenados de menor a mayor, eliminando los valores repetidos. Una vez leído, lo guardamos ya ordenado en el fichero. Para ello vamos a usar un árbol binario de búsqueda. Se ha pensado en utilizar una estructura como la siguiente:

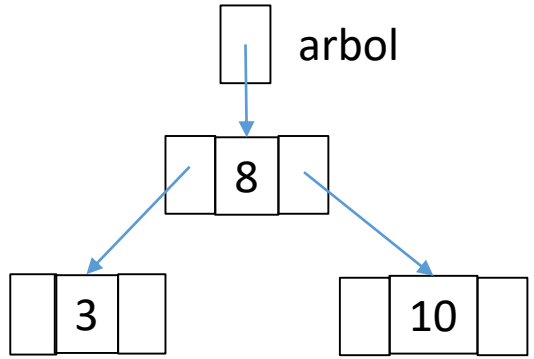
```
typedef struct T_Nodo* T_Arbol;
struct T_Nodo {
        unsigned dato;
        T_Arbol izq, der;
};
```



Instrucciones

- Parte 2: Creación del módulo árbol binario de búsqueda
  - Crear el fichero arbolbb.c
  - Copiar la cabecera de todas las funciones que aparecen en arbolbb.h e implementarlas
    - Si nos fijamos en un nodo concreto del árbol:
      - Todos los elementos a la izquierda del nodo son menores que él
      - Todos los elementos a la derecha del nodo son mayores que él
    - No se añadirá un elemento al árbol si ya existe
    - El recorrido enOrden del árbol accederá a los elementos del árbol de forma ordenada





Instrucciones

- Parte 2: Creación del módulo árbol binario de búsqueda
  - Crear el árbol


arbol

  - Destruir el árbol
    - Libera la memoria de todos los nodos del árbol
  - Insertar nodo en el árbol
    - Si el valor ya existe no se inserta
  - Mostrar el árbol
    - Mostrar en orden:
      - Parte izquierda del árbol
      - Nodo actual
      - Parte derecha del árbol
  - Salvar el árbol
    - Igual que mostrar pero guardando a un fichero

Todos se pueden realizar de forma recursiva

Instrucciones

- Parte 2: Creación del módulo árbol binario de búsqueda
  - Crear el árbol


arbol

```
void Crear(T_Arbol* arbol){
        //El árbol se crea vacío
}
```

Instrucciones

- Parte 2: Creación del módulo árbol binario de búsqueda
  - Destruir el árbol: función recursiva

```
// Destruye la estructura utilizada.
void Destruir(T_Arbol* arbol){
    if (*arbol!=NULL){
        Destruir(&((*arbol)->izq));
        Destruir(&((*arbol)->der));
        free(*arbol);
        *arbol =NULL;
    }
}
```

Parámetro por referencia

Caso base: el árbol está vacío

Necesario porque debemos pasar el parámetro por referencia

El árbol debe ser correcto: nodo a NULL (árbol vacío)

Instrucciones

- Parte 2: Creación del módulo árbol binario de búsqueda
  - Insertar un nodo en el árbol
    - Podéis implementarlo de forma iterativa o recursiva
    - En la solución que os daremos después tendréis las dos soluciones

```
void Insertar(T_Arbol* arbol, unsigned num){
   //Si el árbol está vacío crear nodo e insertar
   //Si no está vacío y el valor del nodo es num no hacer nada
   //Si no está vacío y el valor del nodo es mayor que num insertar a la izquierda
   //Si no está vacío y el valor del nodo es menor que num insertar a la derecha
}
```

Instrucciones

- Parte 3: Uso del módulo implementado en la Parte 2 para ordenar y eliminar las repeticiones de los datos contenidos en el fichero de la Parte 1
  - Ir al fichero Practica2BB.c
  - Implementar la función cargarFichero:

```
void cargaFichero(char* nfichero, T_Arbol* miarbol)
```

  - Abrir el fichero pasado como primer parámetro para leer de él en binario
  - Cada dato leído del fichero se insertará en el árbol binario pasado como segundo parámetro

Instrucciones

- Parte 3: Uso del módulo implementado en la Parte 2 para ordenar y eliminar las repeticiones de los datos contenidos en el fichero de la Parte 1
  - Echad un vistazo a la segunda parte del código de la función main()

```
printf ("\nAhora lo cargamos en el arbol\n");
T_Arbol miarbol;
Crear (&miarbol);
cargaFichero(nfichero,&miarbol);
printf ("\nY lo mostramos ordenado\n"
Mostrar(miarbol);
fflush(stdout);
printf("\nAhora lo guardamos ordenado\n");
FILE * fich;
fich = fopen (nfichero, "wb");
Salvar (miarbol, fich);
fclose (fich);
printf("\nY lo mostramos ordenado\n");
muestrafichero(nfichero);
Destruir (&miarbol);
```

Creamos el árbol binario

Leemos del fichero e insertamos en el árbol

Mostramos el árbol binario por pantalla

Salvamos el árbol a un fichero

Mostramos el fichero

Liberamos la memoria del árbol