Trabajo Práctico 1
España decime qué se siente

Organización del Computador II

Segundo Cuatrimestre 2014

1. Introducción

El objetivo de este TP es implementar un conjunto de funciones sobre una estructura recursiva que representa una lista de jugadores de baloncesto para la XVII edición del Campeonato Mundial de Baloncesto a realizarse en España del 30 de agosto al 14 de septiembre de 2014 organizado por la Fédération Internationale de Basket-ball (FIBA).

Antiguamente la FIBA llevaba el registro de los jugadores manualmente, inscribían en una planilla a cada jugador. Las planillas eran luego separaban en carpetas por cada selección, agrupando los jugadores de cada país. En esas mismas carpetas llevaban la contabilidad de las estadísticas, puntos y todo lo relativo al campeonato.

Desafortunadamente para la FIBA, los rumores sobre las irregularidades en las distintas federaciones regionales comenzó a correr. Hace unas semanas, y a muy poquito del arranque del mundial, los integrantes de la llamada "generación dorada" con su capitán Scola a la cabeza y junto a Ginóbili, Prigioni, Gutiérrez, Herrmann, Delfino y Nocioni, denunciaron la situación por la que pasa hace tiempo la Confederación Argentina de Básquetbol (CABB), que derivó en su intervención y posterior renuncia del presidente[1].

La presión es grande, la FIBA no quiere más bardo, y comienzan a correrse también los rumores sobre la tecnología utilizada para la organización del evento. Por lo tanto, decidieron desarrollar un sistema no manual para administrar la información de los jugadores para el campeonato.

La estructura más común para administrar datos de "cosas" es una **lista** donde se van anexando a la misma los nuevos datos a agregar.

La forma más común de implementar una lista es mediante la concatenación de **nodos**. El nodo define la forma y contenido de los elementos de la lista. Además, el nodo define cuál es el nodo que le sigue. Así, identificando cuál es el primer nodo, una lista es un conjunto de nodos que se siguen unos a otros. La cantidad de elementos de esta lista es variable.

Por alguna razón desconocida, la lista será además **doblemente enlazada**, es decir que cada nodo sabrá quién es su siguiente y su anterior.

En la Figura 1 podemos ver un ejemplo de este tipo de listas.

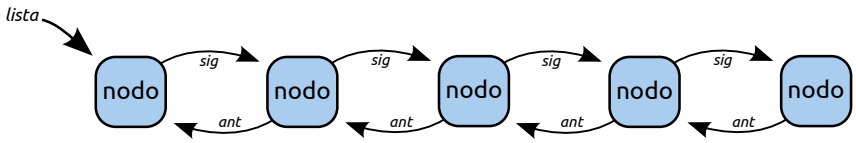

Figura 1: Ejemplo de Lista Doblemente Enlazada.

Las funciones a implementar permiten crear listas, recorrerlas, imprimirlas y borrarlas. Además de estas operaciones, se deberán programar funciones avanzadas para poder realizar ciertas operaciones sobre sus nodos y reconfigurar la información en la lista.

La mayor parte de los ejercicios a realizar para este TP estarán divididos en dos secciones:

- **Primera:** Completar las funciones que permiten manipular de forma básica una **lista**.
- **Segunda:** Realizar las funciones avanzadas sobre **lista**s.

Por último se deberá realizar un programa en `lenguaje C`, que cree determinadas **lista**s y ejecute algunas de las funciones de manipulación de las mismas.

[1] `https://www.google.com.ar/search?q=basquet+argentina+denuncia+jugadores`

1.1. Tipos lista, nodo, jugador y seleccion

La estructura de la **lista** está compuesta por dos punteros, representando el `primero` y `ultimo` nodo. Cada nodo tiene punteros al nodo anterior (`ant`) y al siguiente (`sig`), y un puntero a *void* representando los `datos`. Esto hace que la lista sea "genérica".

```
typedef struct nodo_t {
    void            *datos;
    struct nodo_t   *sig;
    struct nodo_t   *ant;
} __attribute__((__packed__)) nodo;

typedef struct lista_t {
    struct nodo_t   *primero;
    struct nodo_t   *ultimo;
} __attribute__((__packed__)) lista;
```

En este TP, los datos que se van a usar en los nodos, serán **jugador** o **seleccion**, obteniendo así listas de jugadores o de selecciones.

Un jugador contiene: nombre, país, número y altura (en centímetros). Está definido de la siguiente manera:

```
typedef struct jugador_t {
    char            *nombre;
    char            *pais;
    char            numero;
    unsigned int    altura;
} __attribute__((__packed__)) jugador;
```

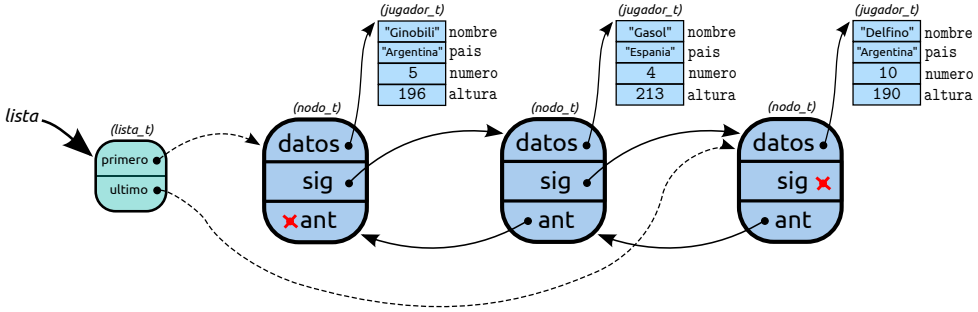

Figura 2: Ejemplo de Lista de Jugadores.

Una selección es una agrupación de jugadores del mismo país, contiene a este país, la altura promedio de los jugadores, y los jugadores en sí mediante un puntero a una lista:

```
typedef struct seleccion_t {
    char            *pais;
    double          alturaPromedio;
    struct lista_t  *jugadores;
} __attribute__((__packed__)) seleccion;
```

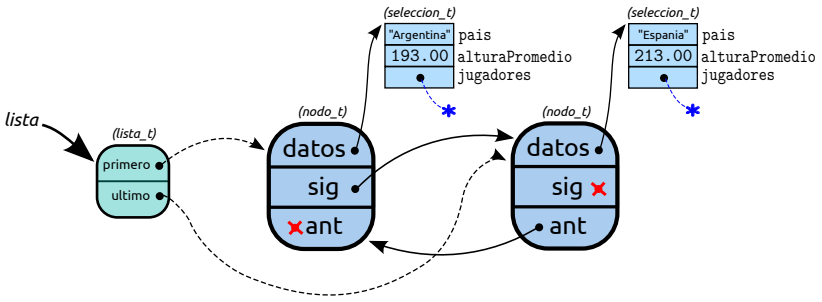

Figura 3: Ejemplo de Lista de Selecciones.

Al ser una lista genérica, es necesario contar además con funciones que especifiquen cómo borrar un dato, imprimirlo, y compararlo con otro. Por conveniencia, se proveen las siguientes definiciones de punteros a funciones:

- `typedef void (*tipo_funcion_borrar) (void*)`
  tipo_funcion_borrar toma datos (tipo `void*`) y los borra.
- `typedef void (*tipo_funcion_imprimir) (void*, FILE*)`
  tipo_funcion_imprimir toma datos (tipo `void*`) y un puntero a un archivo (tipo `FILE*` [2]) e imprime los datos en ese archivo.
- `typedef void* (*tipo_funcion_mapear) (void*)`
  tipo_funcion_mapear toma datos (tipo `void*`) y devuelve nuevos datos según algún criterio.
- `typedef bool` [3] `(*tipo_funcion_cmp) (void*, void*)`
  tipo_funcion_cmp toma dos datos (tipo `void*`) y determina si la comparación es cierta entre los datos con algún cierto criterio de comparación.

1.2. Funciones de lista y nodo

- `nodo *nodo_crear(void *datos)`
  Crea un nodo con los datos pasados por parámetro.
- `lista *lista_crear(void)`
  Crea una lista vacía.
- `void lista_borrar(lista *l, tipo_funcion_borrar f)`
  Borra la lista y todos sus nodos internos, usando la función `f` para borrar los datos dentro de cada nodo.
- `void lista_imprimir(lista *l, char *nombre_archivo, tipo_funcion_imprimir f)`
  Agrega al `nombre_archivo` pasado por parámetro los datos de `l` utilizando `f`.
  Para esto, abre el archivo en modo `append` para que `f` adicione las nuevas líneas al archivo original.

  Esta función no escribe directamente en el archivo, sino que lo hace aplicando la función `f`, quien se encargará de escribir los datos.
- `void lista_imprimir_f(lista *l, FILE *file, tipo_funcion_imprimir f)`
  Agrega al archivo pasado por parámetro los datos de `l`.
  Los datos deben imprimirse siguiendo el orden que tienen en la lista y utilizando `f` para tal propósito.
  Si la lista está vacía se deberá imprimir `<vacia>` seguido del caracter especial de fin de línea (`LF = 10`).

1.3. Funciones de Jugador

- `jugador *crear_jugador(char *nombre, char *pais, char numero, unsigned int altura)`
  Crea un jugador con los datos pasados por parámetro.
  Los strings *nombre* y *pais* pasados por parámetro deben ser copiados en el nuevo jugador.
- `bool menor_jugador(jugador *j1, jugador *j2)`
  Devuelve si `j1` es menor a `j2`.
  Se considera que `j1` es menor a `j2`, si el nombre de `j1` es lexicográficamente [4] menor al de `j2`. Si tienen el mismo nombre, entonces es menor si la altura de `j1` es menor o igual a la de `j2`.

[2] típico retorno de la función `fopen` como un puntero al buffer en memoria con los datos del archivo abierto.
[3] utilizado para indicar un valor de verdad. Es un byte que vale 0 o 1 y está definido en `stdbool.h`
[4] relación de orden utilizada para comparar cadenas de caracteres, por ejemplo: **merc**ed < **merc**urio, perro < zorro y **senior** < **senior**a.

- `jugador *normalizar_jugador(jugador *j)`
  Devuelve un nuevo jugador por copia que corresponde a la normalización de `j`.
  Un nodo está normalizado con respecto al original si:
  - El nombre está convertido a su equivalente en mayúsculas.
  - Su altura está convertida a su equivalente en pies (enteros).
  - País y número se mantienen iguales.

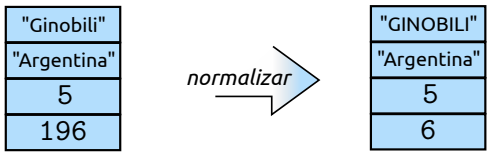

Figura 4: Ejemplo de Ginobili normalizado.

Nota: `1 pie = 30,48 centímetros`. En pies enteros, `196 cm = 6 pies`.

- `bool pais_jugador(jugador *j1, jugador *j2)`
  Devuelve si el país de `j1` es igual al país de `j2`.

- `void borrar_jugador(jugador *j)`
  Borra un jugador.

- `void imprimir_jugador(jugador *j, FILE *file)`
  Imprime los datos de `j` en `file`.
  Se deben imprimir en orden los datos del jugador: nombre, país, número y altura; con un espacio después de cada uno (incluyendo uno después de la altura). No debe haber espacios antes del nombre. Al finalizar, deberá agregarse el caracter especial de fin de línea.

  Por ejemplo, si imprimiéramos al jugador que aparece en la Figura 4, se imprimiría:

  `Ginobili` *[espacio]* `Argentina` *[espacio]* `5` *[espacio]* `196` *[espacio]* *[LF]*

1.4. Funciones de Seleccion

- `seleccion *crear_seleccion(char *pais, double alturaPromedio, lista *jugadores)`
  Crea una selección con los datos pasados por parámetro.
  El string *pais* pasado por parámetro debe ser copiado en la nueva selección.

- `bool menor_seleccion(seleccion *s1, seleccion *s2)`
  Devuelve si `s1` es menor a `s2`.
  Se considera que una selección es menor a otra si su país es lexicográficamente menor.

- `jugador *primer_jugador(seleccion *s)`
  Dada una selección `s`, devuelve una copia de su primer jugador.
  Se puede suponer que `s` siempre tiene por lo menos un jugador al momento de llamar a esta función.

- `void borrar_seleccion(seleccion *s)`
  Borra una selección.

- `void imprimir_seleccion(seleccion *s, FILE *file)`
  Primero se imprimirá en `file` el *pais* y la *alturaPromedio*, dejando un espacio después de cada uno (incluyendo uno después de la altura).
  La *alturaPromedio* debe imprimirse con precisión de dos decimales.
  No debe haber espacio antes del país. Al finalizar, deberá agregarse el caracter especial de fin de línea. Luego se deberán imprimir todos los *jugadores*.

Por ejemplo, si imprimiéramos las selecciones que aparecen en la Figura 3 y cada selección contuviera correspondientemente a los jugadores que aparecen en la Figura 2, se imprimiría:

`Argentina` *[espacio]* `193.00` *[espacio] [LF]*
`Ginobili` *[espacio]* `Argentina` *[espacio]* `5` *[espacio]* `196` *[espacio] [LF]*
`Delfino` *[espacio]* `Argentina` *[espacio]* `10` *[espacio]* `190` *[espacio] [LF]*
`Espania` *[espacio]* `213.00` *[espacio] [LF]*
`Gasol` *[espacio]* `Espania` *[espacio]* `4` *[espacio]* `213` *[espacio] [LF]*

Nota: utilice la función `lista_imprimir_f` e `imprimir_jugador`, no repita el código de *imprimir jugadores* en `imprimir_seleccion`.

1.5. Funciones Avanzadas

- `void insertar_ordenado(lista *l, void *datos, tipo_funcion_cmp f)`
  Inserta un nodo con `datos` en `l`, manteniendo el orden dado por `f`.
  Suponiendo que la lista ya tiene sus nodos ordenados de forma creciente según `f`, el nuevo nodo se insertará respetando ese orden. Se considera que una lista vacía está ordenada.
  `f(d1,d2)` devuelve *true* si $d1 < d2$.

- `lista *filtrar_jugadores(lista *l, tipo_funcion_cmp f, nodo *cmp)`
  Dada una lista de jugadores `l`, una función de comparación `f` y un nodo de comparación `cmp`, devuelve una nueva lista tal que sólo están los elementos de `l` que cumplen `f (datosActual, datosCmp)`, donde `datosActual` son los *datos* de un nodo particular de `l`, y `datosCmp` son los *datos* del nodo de comparación.
  Los elementos de la nueva lista estarán en el mismo orden en el que se encontraban en `l`. Cada nodo contendrá datos (un jugador) que serán copias de los que estaban en la lista original. `l` no debe ser modificada.
  Esta función ya se encuentra implementada en `lenguaje C`.

- `lista *mapear(lista *l, tipo_funcion_mapear f)`
  Dada una lista de jugadores `l` y una función de mapeo `f`, devuelve una nueva lista de jugadores que equivale a aplicar `f` a los *datos* de cada nodo de `l`. Los elementos de la nueva lista respetan el orden en el que se encontraban en `l`. `l` no debe ser modificada.

- `lista *generar_selecciones(lista *l)`
  Dada una lista de jugadores `l`, devuelve una nueva lista de selecciones que contiene correctamente a los jugadores agrupados por selección. Cada selección tiene bien definidos sus valores de *pais* y *alturaPromedio* (en pies) de sus jugadores.

  Los jugadores de cada selección se encuentran ordenados (según `menor_jugador`) y normalizados (según `normalizar_jugador`).

  Además la lista de selecciones resultado está ordenada (según `menor_seleccion`) y no contiene países repetidos.

  Por ejemplo, si imprimiéramos la lista de selecciones resultante de aplicar `generar_selecciones` a la lista de jugadores de la Figura 2, se imprimiría:

  `Argentina` *[espacio]* `6.00` *[espacio] [LF]*
  `DELFINO` *[espacio]* `Argentina` *[espacio]* `10` *[espacio]* `6` *[espacio] [LF]*
  `GINOBILI` *[espacio]* `Argentina` *[espacio]* `5` *[espacio]* `6` *[espacio] [LF]*
  `Espania` *[espacio]* `6.00` *[espacio] [LF]*
  `GASOL` *[espacio]* `Espania` *[espacio]* `4` *[espacio]* `6` *[espacio] [LF]*

  Nota: no repita código, piense cómo utilizar las funciones de *filtrar*, *mapear* e *insertar ordenado*.

1.6. Funciones Auxiliares Obligatorias

- `double altura_promedio(lista *l)`
  Dada una lista de jugadores `l`, devuelve el promedio de sus alturas.

- `lista *ordenar_lista_jugadores(lista *l)`
  Dada una lista de jugadores `l`, devuelve una nueva lista de jugadores que contiene los mismos jugadores que `l` pero ordenados según la función `menor_jugador`.
  No hay que implementar ningún algoritmo de ordenamiento particular, simplemente utilizar las funciones `insertar_ordenado` y `menor_jugador`.

1.7. Funciones Auxiliares Sugeridas

A continuación se presentan funciones auxiliares sugeridas para facilitar el desarrollo del trabajo práctico.

- `char *string_copiar(char *s)`
  Devuelve una nueva copia de `s`.

- `bool string_iguales(char *s1, char *s2)`
  Indica si `s1` y `s2` son iguales, caracter a caracter.

- `int string_comparar(char *s1, char *s2)`
  Compara lexicográficamente a `s1` y `s2`. Devuelve un valor negativo si `s1` $<$ `s2`; un *cero* si `s1` $=$ `s2`; y un valor positivo si `s1` $>$ `s2`.

- `void insertar_antes_de(lista *l, nodo *nuevo, nodo *n)`
  Inserta el `nuevo` nodo antes del nodo `n`, en la lista `l`.

- `void insertar_ultimo(lista *l, nodo *nuevo)`
  Inserta el `nuevo` nodo al final de la lista `l`.
  Esta función ya se encuentra implementada en `lenguaje C`.

Importante:

a) Las funciones:

   - `crear_jugador`
   - `crear_seleccion`

   devuelven nuevas estructuras, reciben como `char*` los parámetros de *nombre* y *pais*.

   En la estructura devuelta **debe copiarse el contenido de los strings *nombre* y *pais***, y no simplemente copiar la referencia a los mismos. Sugerencia: usar la función auxiliar `copiar_string`.

b) Algo similar sucede con las funciones:

   - `normalizar_jugador`
   - `primer_jugador`
   - `filtrar_jugadores`
   - `mapear`
   - `generar_selecciones`
   - `ordenar_lista_jugadores`

   Éstas **deben crear una nueva estructura y copiar todo el contenido resultado en ella**, es decir, no devuelven referencias sino copias. Los `char*` de *nombre* y *pais* también deben ser copiados.

2. Enunciado

A lo largo del trabajo se deberán implementar un conjunto de funciones en `lenguaje ensamblador` y otras en `lenguaje C`.

2.1. Las funciones a implementar en lenguaje ensamblador son:

- `nodo *nodo_crear(void *datos)` *[15]*
- `lista *lista_crear(void)` *[10]*
- `void lista_borrar(lista *l, tipo_funcion_borrar f)` *[25]*
- `void lista_imprimir(lista *l, char *nombre_archivo, tipo_funcion_imprimir f)` *[25]*
- `void lista_imprimir_f(lista *l, FILE *file, tipo_funcion_imprimir f)` *[35]*
- `jugador *crear_jugador(char *nombre, char *pais, char numero, unsigned int altura)` *[30]*
- `bool menor_jugador(jugador *j1, jugador *j2)` *[30]*
- `jugador *normalizar_jugador(jugador *j)` *[35]*
- `bool pais_jugador(jugador *j1, jugador *j2)` *[20]*
- `void borrar_jugador(jugador *j)` *[15]*
- `void imprimir_jugador(jugador *j, FILE *file)` *[20]*
- `seleccion *crear_seleccion(char *pais, double alturaPromedio, lista *jugadores)` *[25]*
- `bool menor_seleccion(seleccion *s1, seleccion *s2)` *[15]*
- `jugador *primer_jugador(seleccion *s)` *[15]*
- `void borrar_seleccion(seleccion *s)` *[15]*
- `void imprimir_seleccion(seleccion *s, FILE *file)` *[20]*
- `void insertar_ordenado(lista *l, void *datos, tipo_funcion_cmp f)` *[45]*
- `double altura_promedio(lista *l)` *[25]*
- `lista *ordenar_lista_jugadores(lista *l)` *[35]*
- `lista *mapear(lista *l, tipo_funcion_mapear f)` *[30]*

2.2. Las funciones a implementar en lenguaje C son:

- `lista *generar_selecciones(lista *l)` *[20]*

Las funciones auxiliares sugeridas

Para implementar todas las funciones anteriores de una manera concisa y fácil de entender se pueden implementar y utilizar también las funciones auxiliares sugeridas. Las mismas son opcionales. Queda a criterio de cada uno cómo utilizarlas para implementar las funciones anteriores y agregar todas las que considere.

- `char *string_copiar(char *s)` *[15]*
- `bool string_iguales(char *s1, char *s2)` *[10]*
- `int string_comparar(char *s1, char *s2)` *[30]*
- `void insertar_antes_de(lista *l, nodo *nuevo, nodo *n)` *[15]*

Nota:

En cada función a implementar se indica, con la referencia *[N]*, la cantidad aproximada de líneas de código que fueron necesarias para resolver la misma según la solución de la cátedra. Su utilidad es tener una idea del tamaño relativo que tienen las distintas funciones. De ninguna manera es necesario realizar implementaciones de las funciones que cumplan o superen esa cota.

2.3. Compilación y Testeo

Para compilar el código y poder correr las pruebas cortas deberá ejecutar `make main` y luego `./pruebacorta.sh`. Para compilar el código y correr las pruebas intensivas deberá ejecutar `./prueba.sh`.

2.3.1. Pruebas cortas

Deberá construirse un programa de prueba (`main.c`) que realice las acciones detalladas a continuación. La idea del ejercicio es verificar incrementalmente que las funciones que se vayan implementando funcionen correctamente. El programa puede correrse con `./pruebacorta.sh` para verificar que no se pierde memoria ni se realizan accesos incorrectos a la misma. Recordar siempre borrar las listas luego de usarlas.

1- **Crear una lista vacía, imprimirla y borrarla.** Para esto deberá implementar las funciones `lista_crear`, `lista_imprimir` y `lista_borrar` en ensamblador. No es necesario implementar completamente `lista_imprimir` y `lista_borrar`, ya que se trabaja con una lista vacía, puede dejar eso para el siguiente punto. Ya que `lista_imprimir` y `lista_borrar` necesitan funciones, se pueden probar con una función que no haga nada, por ejemplo, definiendo `f` y `g` de la siguiente manera:

```
void f (void*) {}
void g (void*, FILE*) {}

lista_imprimir(l, g);
lista_borrar(l, f);
```

2- **Crear una lista nueva, agregarle un jugador, imprimir la lista y borrarla.** Deberá implementar `nodo_crear`, `insertar_ordenado` y `crear_jugador`. Ahora sí debe completar `lista_imprimir` y `lista_borrar`. Notar que esto implica también implementar las funciones `borrar_jugador` e `imprimir_jugador`. No es necesario implementar completamente `insertar_ordenado`, puede simplemente utilizar `insertar_ultimo` que ya está implementada en `C` para probar insertando los nodos siempre al final, ya que aún no importa el orden de la lista. Además, será necesario realizar *casteos* (conversión de tipos) para poder imprimir y borrar. Por ejemplo:

```
lista_borrar(l, (tipo_funcion_borrar) borrar_jugador);
lista_imprimir(l, "salida.txt", (tipo_funcion_imprimir) imprimir_jugador);
```

3- **Terminar la función `insertar_ordenado`, e implementar `menor_jugador`.** Probar `insertar_ordenado` con una función de comparación que de siempre verdadero o siempre de falso para asegurarse de que los casos bordes andan correctamente. Luego probarla con `menor_jugador`. Por ejemplo:

```
bool h (void*, void*) {return true;}

insertar_ordenado(l, jugador, h);
insertar_ordenado(l, jugador, (tipo_funcion_cmp) menor_jugador);
```

4- Crear selecciones e insertarlas ordenadamente

   a) **Crear una lista nueva, agregarle una selección con una lista vacía de jugadores, imprimir la lista y borrarla.** Para esto debe implementar `crear_seleccion`, `borrar_seleccion` e `imprimir_seleccion`. Puede implementar versiones incompletas de las funciones ya que no hace falta que tengan en cuenta a los jugadores de una selección.

   b) **Crear una lista nueva, agregarle una selección con una lista NO vacía de jugadores, imprimir la lista y borrarla.** Debe finalizar las implementaciones de las funciones anteriores para tener en cuenta la lista de jugadores. Considerar las funciones de jugadores ya realizadas para no repetir código.

   c) **Crear una lista, agregarle una selección de forma ordenada, imprimir la lista y borrarla.** Para esto debe utilizar `insertar_ordenado` e implementar `menor_seleccion` y `altura_promedio`.

5- Por último, las funciones avanzadas de lista:

   a) **Implementar `ordenar_lista_jugadores`, `mapear`, `normalizar_jugador`, `primer_jugador` y `pais_jugador`.** Al igual que con `insertar_ordenado`, probar primero `ordenar_lista_jugadores` con listas sencillas y luego `mapear` con funciones simples (como la función identidad) para verificar el funcionamiento. Luego probarla con `normalizar_jugador`. Por último, probar `filtrar_jugadores` con `pais_jugador` y `menor_jugador`.

   b) Implementar `generar_seleccion` en `C`.

2.3.2. Pruebas intensivas (Testing)

En un ataque de bondad, hemos decidido proveer una serie de *tests* o pruebas intensivas para que pueda verificarse el buen funcionamiento del código de manera automática.

Para correr el testing se debe ejecutar `./prueba.sh`, que compilará el *tester* y correrá todos los tests de la cátedra. Un test consiste en la creación, inserción, eliminación, ejecución de funciones avanzadas e impresión en archivo de una gran cantidad de listas. Luego de cada test, el *script* comparará los archivos generados por su TP con las soluciones correctas provistas por la cátedra.

También será probada la correcta administración de la memoria dinámica.

2.4. Archivos

Se entregan los siguientes archivos:

- `lista.h`: Contiene la definición de las estructuras y de las funciones a realizar.
- `lista_asm.asm`: Archivo a completar con su código en `lenguaje ensamblador`.
- `lista_c.c`: Archivo a completar con su código en `lenguaje C`.
- `Makefile`: Contiene las instrucciones para ensamblar y compilar el código.
- `main.c`: Es el archivo principal donde escribir los ejercicios para las pruebas cortas (`pruebacorta.sh`).
- `tester.c`: Es el archivo del tester de la cátedra. No debe ser modificado.
- `pruebacorta.sh`: Es el script que corre el test simple (pruebas cortas).
- `prueba.sh`: Es el script que corre todos los test intensivos.
- `salida.caso*.catedra.txt`: Archivos de salida que se compararán sus salidas. No deben ser modificados.

Notas:

a) Todas las funciones deben estar implementadas en `lenguaje ensamblador`, salvo las explícitamente indicadas a implementar en `lenguaje C`. Cualquier otra función extra, incluyendo las auxiliares sugeridas, deben estar implementadas en `lenguaje ensamblador`.

b) Toda la memoria dinámica reservada usando la función `malloc` debe ser correctamente liberada, utilizando la función `free`.

c) Para el manejo de archivos se recomienda usar las funciones de `C`: `fopen`, `fprintf`, `fclose`, `fputs`, `fscanf`, `fseek`, `ftell`, etc.

d) Para el manejo de strings **no está permitido** usar las funciones de `C`: `strcmp` y `strdup`.

e) Para poder correr los test, se debe tener instalado *Valgrind* (En ubuntu: `sudo apt-get install valgrind`).

f) Para corregir los TPs usaremos los mismos tests que les fueron entregados. Es condición necesaria para la aprobación que el TP supere correctamente todos los tests.

3. Resolución, Informe y Forma de Entrega

Este TP es de carácter **individual**. No deberán entregar un informe. En cambio, deberán entregar un archivo comprimido con el mismo contenido que el dado para realizarlo, habiendo modificado sólo los archivos `lista_asm.asm`, `lista_c.c` y `main.c`.

Este TP deberá entregarse como máximo a las 16:59:59 hs del día Martes 9 de Septiembre de 2014. ***La reentrega del TP, en caso de ser necesaria, deberá realizarse como máximo a las 16:59:59 hs del día Jueves 2 de Octubre de 2014.***

Ambas entregas se realizarán a través de la página web. El sistema sólo aceptará entregas de trabajos hasta el horario de entrega, sin excepciones.

Ante cualquier problema con la entrega, comunicarse por mail a la lista de docentes.