26 de marzo, 2015.

Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Estado de México
División de Diseño, Ingeniería y Arquitectura
Departamento de Tecnologías de Información y Computación

Segundo examen práctico de Programación avanzada

Profesor: Ariel Ortiz Ramírez Clave y grupo: Tc2025.1

Nombre: _____ Matrícula: _____

Apegándome al Código de Ética de los Estudiantes del Tecnológico de Monterrey, me comprometo a que mi actuación en este examen esté regida por la honestidad académica. En congruencia con el compromiso adquirido con dicho código, realizaré este examen de forma honesta y personal, para reflejar, a través de él, mi conocimiento y aceptar, posteriormente, la evaluación obtenida.

Firma: _____

NOTA MUY IMPORTANTE: Durante el examen no debes estar conectado a la red, usar teléfono celular, hablar ni utilizar material de otras personas. Cualquier indicio de copia, trampa o fraude será penalizado con DA (Deshonestidad Académica) en la calificación del primer parcial. Se penalizará tanto al copiado como al copiador.

Instrucciones generales

El archivo `examen2.zip` proporcionado tiene como contraseña: **void*p** . Contiene dos archivos fuente (`lsearch.c` y `arbol.c`) que se utilizarán como base en cada uno de los dos problemas que se describen más adelante.

NOTA IMPORTANTE: Agrega a tus archivos fuente tu nombre y matrícula dentro de un comentario en la parte superior. Se penalizarán 10 puntos si entregas algún archivo sin estos datos.

Una vez que termines el examen, deberás empaquetar los dos archivos fuente en un archivo TGZ usando el siguiente comando desde la terminal:

```
tar cvzf A0MMMMMMM.tgz lsearch.c arbol.c
```

en donde *A0MMMMMMM* es tu matrícula. Copia este archivo a la unidad de memoria USB provista por el profesor.

Tiempo límite: 60 minutos.

Problemas

1. **(50%)** En el archivo `lsearch.c`, escribe el código de la función `lsearch()`. Su prototipo es el siguiente:

   ```
   void * lsearch(const void * key, const void * base, size_t nelem, size_t size,
                  int (* cmp)(const void *, const void *));
   ```

   Esta función realiza una búsqueda lineal (secuencial) sobre un arreglo y devuelve la dirección del primer elemento del arreglo que resulta igual a la llave de búsqueda *key* (si es que existe), de otra forma regresa un apuntador nulo. El arreglo consiste de un total de *nelem* elementos, cada uno de *size* bytes de tamaño, comenzando con el elemento cuya dirección es *base*. Para hacer las comparaciones, la función `lsearch()` debe usar *cmp* de la misma manera que lo hacen las funciones `qsort()` y `bsearch()` de la biblioteca estandar de C.

   El programa debe compilar sin errores ni advertencias (*warnings*) bajo el estándar C99 (usando la opción `-std=c99` al momento de compilar). Al correr el programa, la salida debe ser:

   ```
   Se encontró 42 en el índice 2.
   No se encontró 9.
   Se encontró "Thorin" en el índice 12.
   No se encontró "Tyrion".
   ```

2. **(50%)** El archivo `arbol.c` contiene la implementación (incompleta) de una estructura de datos que combina una lista lineal sencillamente encadenada y un árbol binario de búsqueda. En esencia, cada nodo creado dinámicamente forma parte simultáneamente de la lista y el árbol. Estudia cuidadosamente el código para entender cómo funciona.

   Usando `valgrind` para correr el programa en la versión provista obtenemos el siguiente mensaje:

   ```
   ==8001== LEAK SUMMARY:
   ==8001==    definitely lost: 48 bytes in 1 blocks
   ==8001==    indirectly lost: 864 bytes in 18 blocks
   ```

   La fuga detectada se debe a que no se ha liberado la memoria que se obtuvo como resultado de llamar `malloc()`. La función `zape()` es la que debe encargarse de realizar las liberaciones correspondientes. Implementa dicha función, que tiene el siguiente prototipo:

   ```
   void zape(arbol_ligado_t * al);
   ```

   Como ya se indicó, su responsabilidad es liberar todos los nodos que conforman la lista/árbol. Al correr el programa modificado, la salida esperada es:

   ```
   Antes del zape:
     [enero, febrero, marzo, abril, mayo, junio]
     [abril, enero, febrero, junio, marzo, mayo]
   Después del zape:
     []
     []
   Antes del zape:
     [Dwalin, Balin, Kili, Fili, Dori, Nori, Ori, Oin, Gloin, Bifur, Bofur, Bombur, Thorin]
     [Balin, Bifur, Bofur, Bombur, Dori, Dwalin, Fili, Gloin, Kili, Nori, Oin, Ori, Thorin]
   Después del zape:
     []
     []
   ```

   Al usar `valgrind` para correr el programa modificado se debe reportar que ya no existe fuga de memoria alguna:

   ```
   ==8002== All heap blocks were freed -- no leaks are possible
   ```