INF-134 Estructuras de Datos, 2021-1

Tarea 1

Profesores: Diego Arroyuelo B., Roberto Díaz U.
darroyue@inf.utfsm.cl, roberto.diazu@usm.cl

Ayudantes:
Anastasiia Fedorova anastasiia.fedorova@sansano.usm.cl,
Gonzalo Fernandez gonzalo.fernandezc@sansano.usm.cl,
Tomás Guttman tomas.guttman@sansano.usm.cl,
Héctor Larrañaga hector.larranaga@sansano.usm.cl,
Martín Salinas martin.salinass@sansano.usm.cl.

Fecha de entrega: 27 de mayo, 2020.
Plazo máximo de entrega: 5 días.

1. Reglas del Juego

La presente tarea debe hacerse en grupos de 3 personas. Toda excepción a esta regla debe ser conversada con los ayudantes (usando los correos indicados en el encabezado de esta tarea). No se permiten de ninguna manera grupos de más de 3 personas. Debe usarse el lenguaje de programación C. Al evaluarlas, las tareas serán compiladas usando el compilador `gcc`, usando la línea de comando `gcc archivo.c -o output -Wall`. Alternativamente, se aceptan variantes o implementaciones particulares de `gcc`, como el usado por MinGW (que está asociado a la IDE code::blocks). Se deben seguir los tutoriales que están en Aula USM, cualquier alternativa explicada allí es válida. Recordar que una única tarea en el semestre puede tener nota menor a 30. El no cumplimiento de esta regla implica reprobar el curso.

2. Objetivos

Comprender y familiarizarse con las estructuras y tipos de datos básicos que provee el Lenguaje de Programación C. Entre los conceptos mas importantes, se encuentran:

- Paso de parámetros por valor.
- Paso de parámetros por referencia.
- Asignación de memoria dinámica.
- Manipulación de punteros.
- Manejo de Archivos.
- Recursividad

Además se fomentará el uso de las buenas prácticas y el orden en la programación de los problemas correspondientes.

3. Problemas a Resolver

En esta sección se requiere que implementen varias funciones en C. Cada una de éstas debe estar en archivos `.c` separados en su entregable, con la correspondiente función `main`.

Problema 1: Búsqueda de Strings

Dado un string $z[1 \dots n]$, se define como prefijo de z a todo string $z[1 \dots i]$, tal que $i = 1, \dots, n$. Por ejemplo, para $z[1 \dots 11]$ = `mississippi`, tenemos 11 prefijos posibles: `m`, `mi`, `mis`, `miss`, `missi`, `missis`, `mississ`, `mississi`, `mississip`, `mississipp`, y `mississippi`.

Para una aplicación de búsqueda de strings, se necesita implementar la siguiente funcionalidad: dado un conjunto de strings S y un string P, queremos encontrar todos los strings de S que tengan a P como prefijo. Para esto se pide implementar la función:

```
char ** buscar_str(char ** S, int n, char * P),
```

que recibe como parámetro un arreglo `S` de `n` strings, y un string `P`. La función retorna un arreglo con todos los strings de `S` que tienen a `P` como prefijo. En caso de ser necesario, puede agregar parámetros adicionales a la función.

Por ejemplo, dado un arreglo que contiene los strings {`"flor"`, `"hola"`, `"fracción"`, `"flotante"`}, y dado el string `P` = `"flo"`, la función debe retornar un arreglo con los strings `"flor"` y `"flotante"`, ya que ambas tienen a `"flo"` como prefijo.

Para la implementación, se pueden usar funciones de manipulación de strings y/o alguna otra función necesaria de la librería standard de C.

Bonus: se dará 20 puntos de bonus a los tres grupos que logren el menor tiempo de ejecución total (y además respondan correctamente).

Formato de Entrada

Para probar la función `buscar_str`, su programa deberá leer strings almacenados en dos archivos:

- Archivo `S.txt`: almacena los strings correspondientes al conjunto `S`.
- Archivo `P.txt`: almacena strings que serán usados para buscar en el conjunto `S`. Es decir, por cada string almacenado en este archivo, debemos invocar a la función `buscar_str`.

Ambos archivos tiene un único string por línea, y son terminados por `EOF`. La longitud máxima de cada string en el archivo es 200 chars, y puede haber hasta 1 millón de strings en `S.txt`, y hasta 1000 strings en `P.txt`.

Un ejemplo de archivo `S.txt` es el siguiente:

```
flor
hola
fracción
flotante
florencia
frecuente
floración
```

Un ejemplo de archivo `P.txt` es el siguiente:

```
flor
fr
```

Esto significa que, en este ejemplo, la función `buscar_str` deberá invocarse 2 veces, una por cada string de `P.txt`.

Formato de Salida

Por cada string del archivo `P.txt`, se debe generar un archivo cuyo nombre es igual al string, y cuya extensión es ".out". El archivo debe contener los strings obtenidos al invocar la función `buscar_str` con el conjunto de strings del archivo `S.txt`, y el string del archivo `P.txt`. En dichos archivos, debe escribir un único string por línea.

Para el ejemplo mostrado anteriormente, se deben crear dos archivos, `flor.out` y `fr.out`, con el siguiente contenido:

- Archivo `flor.out`:

  ```
  flor
  florencia
  floración
  ```

- Archivo `fr.out`:

  ```
  fracción
  frecuente
  ```

En cada archivo, los strings pueden estar en cualquier orden (según su conveniencia).

Problema 2: Transacciones Bancarias

Suponga la siguiente definición de tipo que representa clientes de un banco:

```
typedef struct {
   int nroCuenta;
   int saldo;
   char nbre[51];
   char direccion[51];
} clienteBanco;
```

en donde los números de cuenta son un valor entre 1 y 10.000.000, y no son necesariamente consecutivos.

Se pide implementar la función `actualizarSaldos (char *clientes, char *transacciones)`, la cual recibe dos parámetros:

`clientes:` string que representa el nombre de un archivo binario, el cual contiene una cantidad arbitraria de structs de tipo `clienteBanco`. El archivo está ordenado de forma creciente por `nroCuenta`.

`transacciones:` string que representa el nombre de un archivo con formato ASCII, el cual contiene transacciones hechas por los clientes. Cada transacción corresponde a una línea del archivo, y tiene alguno de los siguientes formatos:

- `+ C N`: corresponde a una transacción de depósito de `N` pesos en la cuenta con número `C`.
- `- C N`: corresponde a una transacción de extracción de `N` pesos en la cuenta con número `C`.
- `> C1 C2 N`: corresponde a una transacción de transferencia de `N` pesos desde la cuenta con número `C1` a la cuenta con número `C2`.

El fin del archivo es indicado por `EOF`.

La función debe procesar las transacciones indicadas en el archivo de transacciones, y actualizar los saldos correspondientes en el archivo binario de clientes. Dado que el acceso a archivos es considerablemente lento, la actualización en el archivo binario debe realizarse con un único recorrido secuencial (es decir, desde el comienzo hasta el final) del archivo de clientes. El banco permite que el saldo final correspondiente a un cliente sea negativo. Además, el programa debe chequear el buen uso desde la línea de comando: se debe mostrar un mensaje de error si no se incluyen los nombres de archivo al ejecutar el programa, y concluir la ejecución.

Invocación a la función `actualizarSaldos`

Los parámetros a usar para la función `actualizarSaldos` serán indicados a través de la línea de comando, como parámetros de la función `main`. Suponiendo que el ejecutable para su programa es `tarea1-parte2`, un ejemplo de línea de comando es

```
./tarea1-parte2 clientes.dat transacciones.txt
```

en donde `clientes.dat` es el archivo binario de clientes, y `transacciones.txt` es el archivo de transacciones. Los nombres de archivo son sólo referenciales. El programa debe permitir usar cualquier nombre de archivo.

4. Entrega de la Tarea

La entrega de la tarea debe realizarse enviando un archivo comprimido llamado

`tarea1-apellido1-apellido2-apellido3.tar.gz`

(reemplazando sus apellidos según corresponda) a la página aula.usm del curso, a más tardar el día 27 de mayo, 2020, a las 23:59:00 hs (Chile Continental), el cual contenga:

- Los archivos con los códigos fuentes necesarios para el funcionamiento de la tarea. ¡Los archivos deben compilar!
- **nombres.txt**, Nombre, ROL, Paralelo y qué programó cada integrante del grupo.
- **README.txt**, Instrucciones de compilación en caso de ser necesarias, y la forma de compilación que usó (debe ser alguna de las indicadas en los tutoriales entregados en Aula USM).

5. Restricciones y Consideraciones

- Por cada día de atraso en la entrega de la tarea se descontarán 10 puntos en la nota.
- El plazo máximo de entrega es 5 días después de la fecha original de entrega.
- **Las tareas que no compilen no serán revisadas y serán calificadas con nota 0**.
- Debe usar **obligatoriamente** alguna de las formas de compilación indicada en los tutoriales entregados en Aula USM.
- Por cada *Warning* en la compilación se descontarán 5 puntos.
- Si se detecta **COPIA** la nota automáticamente sera 0 (CERO), para todos los grupos involucrados. El incidente será reportado al jefe de carrera.
- La prolijidad, orden y legibilidad del código fuente es obligatoria. Habrá descuentos si alguno de estos items no se cumple.

6. Consejos de Programación

El código fuente del programa debe estar estructurado adecuadamente en archivos (separados de ser necesario). Si el código fuente está desordenado, se pueden descontar hasta 20 puntos de la nota.

Cada función programada debe tener comentarios de la siguiente forma:

```
/*****
*   TipoFunción NombreFunción
******
*   Resumen Función
******
*   Input:
*       tipoParámetro NombreParámetro : Descripción Parámetro
*        .......
******
*   Returns:
*       TipoRetorno, Descripción retorno
*****/
```

Por cada comentario faltante, se restarán 5 puntos.

Por último, la indentación (1 TAB o 4 espacios), es muy importante. Por **cada bloque mal indentado, se quitarán 10 puntos**.