UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

DEPARTAMENTO DE
ELECTRONICA

ELO 320 – Estructura de Datos y Algoritmos
1er semestre 2018
Profesor P1: María José Escobar S., **Ayudante P1:** Oscar Lizama
Profesor P2: Ioannis Vourkas **Ayudante P2:** Javier Torres

TAREA Nro. 1

Indicación

Las tareas son **individuales**. Cualquier acción que pueda beneficiar de forma injusta la calificación de su tarea está prohibida, incluyendo la presentación de cualquier componente que no es de su autoría, o la facilitación de esto para otros. Por supuesto, es aceptable discutir *-en líneas generales-* los métodos y resultados con sus compañeros, pero **se prohíbe explícitamente realizar las tareas en conjunto o compartir código de programación. Utilizar código de internet que no es de su autoría, también es considerado plagio**. Por último, por favor presten atención en las instrucciones para la entrega de la tarea, que pueden incluir políticas de nombre de archivos, codificación y criterios de evaluación.

Contenidos cubiertos en esta tarea

Los contenidos principales a tratar son los que entran en el Certamen 1 de la asignatura.

Objetivos

- Practicar el desarrollo de código C que contiene punteros y en particular `strings`.
- Ejecutar programas entregando parámetros vía línea de comandos (`argc, argv`).
- Familiarizarse con el desarrollo modular/incremental de código C.
- Desarrollar y practicar el uso de estructuras de datos básicas aprendidas en clase, tales como listas enlazadas y árboles binarios.

Descripción del trabajo a realizar

Contexto

Debido a que nuestra Universidad se encuentra ubicada en una ladera del Cerro Placeres, se hace muy difícil el acceso para personas con movilidad reducida, pues como bien sabemos gran parte de nuestro tránsito diario se realiza mediante escaleras y subidas empinadas.

Un profesor de la Universidad está liderando un proyecto que pretende dar solución a este punto. No obstante, lleva meses *entrampado* en un problema que no le permite avanzar. Esta persona tiene plena confianza en que tú serás la persona indicada para resolver su problema; sabe que estas cursando ELO 320, por lo cual te invita a participar.

Este desarrollo consiste en un sistema de transporte para personas con movilidad reducida mediante sillas eléctricas autónomas, capaces de transitar por escaleras. El problema radica en el sistema de navegación: se espera que la silla pueda planificar la ruta entre dos puntos en la Universidad de manera de minimizar costos.

Avda. España 1680,
Edificio B,
Valparaíso, Chile.
Teléfono: (56 32) 2654191
www.electronica.usm.cl

La Tarea

El objetivo de esta tarea es desarrollar un programa (`rutaOptima`) en C que planifique la ruta de la silla. Para ello, Ud. pasará como argumento a su programa la ubicación de un archivo de texto (`mapa.txt`) que contiene un mapa actualizado de los alrededores de la silla[1]:

```
./rutaOptima ~/ELO320/Tarea1/mapa.txt
```

Escriba un programa que de la bienvenida al usuario y luego presenta las siguientes opciones mediante **un menú**:

1. **Cargar el mapa:** Esta opción lee y muestra el mapa almacenado en la ruta y archivo entregado como argumento a la función main, imprimiéndolo por pantalla junto al número de filas y columnas.

   En este caso se debe llamar una función: `int verificar_mapa(char **map)` que verifica que el mapa entregado se encuentra en el formato correcto (ver **Anexo 1**) y lo deja como mapa vigente. De no ser leído correctamente, se mostrara un mensaje de error por pantalla.

2. **Generar un mapa:** Esta opción genera un mapa de manera aleatoria, llamando la función: `void generar_mapa(char **map, int nfil, int ncol)`
   Dicha función recibirá como parámetro de entrada el espacio de memoria para almacenar el mapa y sus dimensiones (`nfil` filas y `ncol` columnas). El mapa creado quedará como mapa vigente y se mostrará por pantalla junto al número de filas y columnas.

3. **Ingresar y validar ruta**: Esta opción requiere que el usuario ingrese una ruta formada por un string que solamente contenga caracteres `'^', `'v', `'<' y `'>', que en código ASCII corresponden a los números 60, 62, 94 y 118, respectivamente. Se deberá llamar la función: `int verificar_ruta(char *ruta)`
   y luego informar por pantalla tanto *el costo total de la ruta* como también *si llega o no al destino* (ver **Anexo 1**).

4. **Encontrar todas las rutas posibles (listas):** Esta opción debe encontrar todos las rutas que unan al punto `A` con el punto `B` indicados en el mapa. Esta opción debe almacenar todos los caminos posibles en *listas doblemente enlazadas*. Considere cada una de las rutas posibles como una lista independiente.

5. **Encontrar todas las rutas posibles (árboles):** Esta opción debe encontrar todos las rutas que unan al punto `A` con el punto `B` indicados en el mapa. Todos los caminos posibles deben almacenarse utilizando *Árboles* de manera de no replicar tramos comunes entre dos rutas. Puede haber más de un Árbol.

6. **Desplegar rutas almacenadas:** Esta opción permite visualizar en pantalla todas las rutas encontradas que unen el punto `A` con el punto `B` indicados en el mapa.

---

[1] En el **Anexo 1** encontrará una descripción detallada del formato de este archivo. Dentro del mapa, considere que la silla puede moverse solo en las direcciones norte, sur, este u oeste.

7. **Encontrar la mejor ruta posible:** Esta opción permite encontrar la ruta entre el punto de inicio y final que tenga un menor costo. Para esto debe recorrer las estructuras de datos utilizadas para almacenar las rutas y determinar el costo de cada una de ellas según la información mostrada en la **Tabla 1**. Notar que puede haber más de una ruta óptima. La o las rutas óptimas deben visualizarse en pantalla.
8. **Salir del programa:** Después de terminar la ejecución de cualquiera opción entregada por el usuario, su programa **debe volver al menú inicial** y así permitir al usuario salir del programa.

NOTA: En los casos 6) y 7), la ruta encontrada deberá ser mostrada por pantalla como un string con los caracteres anteriormente mencionados: `^', `v', `<' y `>'. En caso de no existir solución alguna, para los puntos 4)-7), deberá informar al usuario por pantalla.

Anexo 1

El mapa corresponde a un archivo de texto que contiene N filas y M columnas como el que se presenta a continuación (N=5 y M=6), en la que cada una de sus celdas contiene uno de los siguientes caracteres:

```
+ + + O O O
A S S + S +
+ O + O + +
+ E + O O +
O O + E B +
```

El significado de cada letra del mapa, así como también los costos asociados se encuentran en la **Tabla 1**.

Tabla 1: Nomenclatura utilizada en el mapa y sus costos respectivos

Símbolo	Significado	Costo
O	Obstáculos, es decir, celdas por la que la silla **no puede pasar**	--
E	Escaleras	3
S	Subidas empinadas	2
+	Piso normal	1
A	Punto de partida	--
B	Punto de llegada	--

Los costos anteriormente descritos **deberán definirse globalmente** mediante directivas `#define` de modo que se pueda adaptar el código fácilmente modificando solo unas constantes.

Para describir un mapa en un archivo de texto, este último debe contener en cada una de sus líneas una fila del mapa, con tantos caracteres como las columnas que tenga el mapa. Notar que, aunque no se especifican los saltos de línea (`\n'), ellos estarán presentes al final de cada línea. Además, el documento termina con un salto de línea, por lo que habrá una línea vacía extra.

Consideraciones y Formato de entrega

- Utilice comentarios para describir lo que se hace en cada etapa de su código.
- El código deberá estar escrito según el estándar de codificación GNU.
- El código escrito debe estar perfectamente *indentado* con tabuladores, no espacios.
- Solamente se aceptarán y revisarán tareas con las siguientes condiciones:
  - Toda tarea debe ser correctamente compilada y ejecutada en el servidor Aragorn.
  - La tarea se entrega vía AULA dentro del plazo establecido: **viernes 4 de mayo, 23:55hrs**
  - Para la entrega envíe **un solo archivo** (`.zip, .rar, .tar.gz`, etc.) que contenga sus archivos `.c, .h` utilizados más un archivo `README.txt`, donde se especifique qué es cada archivo de su programa, cómo debe ser compilado y cómo debe ser ejecutado. NO OLVIDE explicar cualquier particularidad que pueda tener su programa.