Organización de computadoras
Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur
Segundo Cuatrimestre de 2021

Proyecto N° 1
Programación en lenguaje C

Propósito

El objetivo principal del proyecto es implementar en lenguaje C un programa que le permita al usuario, visualizar información referente a posibles planes de viajes para un conjunto de ciudades que desea visitar. Con este objetivo se debe implementar:

- *TDA Lista*, para almacenar elementos de tipo genérico.
- *TDA Cola Con Prioridad*, para almacenar entradas con clave y valor de tipo genérico, ordenadas en función de su prioridad. La prioridad de las entradas será determinada por una función diseñada específicamente para ese propósito.
- Un *programa principal*, el cual debe tomar como argumento por línea de comandos el nombre de un archivo de texto y a partir de este determinar la ubicación actual y las ciudades a visitar por el usuario, permitiendo luego un conjunto de operaciones sobre estos datos.

1. TDA Lista

Implementar un *TDA Lista* en lenguaje C, cuyos elementos sean punteros genéricos. La lista debe ser implementada mediante una estructura **simplemente enlazada** con **celda centinela**, utilizando el concepto de **posición indirecta**. La implementación de la misma debe respetar la operaciones y estructuras definidas en el archivo *lista.h* adjunto a este enunciado.

2. TDA Cola Con Prioridad

Implementar un *TDA Cola Con Prioridad* en lenguaje C, La cola debe ser implementada mediante una estructura *Heap*, cuyos elementos sean entradas con clave y valor como punteros genéricos. El orden en que las entradas se retiran de la cola se especifica al momento de la creación, a través de una función de prioridad. La implementación de la misma debe respetar la operaciones y estructuras definidas en el archivo *colacp.h* adjunto a este enunciado.

3. Programa Principal

Implementar una aplicación de consola que, recibiendo como argumento por línea de comandos el nombre de un archivo de texto con el formato indicado en el *Ejemplo 1*, reconozca y mantenga la ubicación actual del usuario y una lista de ciudades a visitar. El programa debe ofrecer un menú de operaciones, con las que el usuario luego puede:

1. **Mostrar ascendente:** permite visualizar el listado de todas las ciudades a visitar, ordenadas de forma ascendente en función de la distancia que existe entre la ubicación de estas ciudades y la ubicación actual del usuario.

2. **Mostrar descendente:** permite visualizar el listado de todas las ciudades a visitar, ordenadas de forma descendente en función de la distancia que existe entre la ubicación de estas ciudades y la ubicación actual del usuario.

3. **Reducir horas manejo:** permite visualizar un listado con el orden en el que todas las ciudades a visitar deben ser visitadas, de forma tal que el usuario ubicado en una ciudad de origen conduzca siempre a la próxima ciudad más cercana al origen, reduciendo las horas de manejo entre las ciudades visitadas. Finalmente, se debe listar la distancia total recorrida con esta planificación.

4. **Salir:** permite finalizar el programa liberando toda la memoria utilizada para su funcionamiento.

El programa implementado, denominado `planificador`, debe conformar la siguiente especificación al ser invocado desde la línea de comandos:

```
$ planificador <archivo_texto>
```

El parámetro `archivo_texto`, indica el archivo a partir del cual se conocerá la ubicación actual del usuario, y la lista de ciudades que desea visitar. En caso de que la invocación no sea la indicada, se debe mostrar un mensaje indicando el error, y finalizar la ejecución.

Consideraciones para el programa principal:

1. Una ciudad será representada a través de un nombre, y una ubicación $< X, Y >$. Considerar para esto el tipo `TCiudad` especificado luego.

2. La distancia entre dos ciudades deberá calcularse mediante *Distancia de Manhattan*, esto es, $|X_2 - X_1| + |Y_2 - Y_1|$.

3. Para mantener el listado de ciudades a visitar, obtenidas desde el archivo de texto, se deberá utilizar el *TDA Lista*.

4. Para implementar las operaciones **Mostrar ascendente**, **Mostrar descendente**, y **Reducir horas de manejo**, se deberá utilizar adecuadamente el listado de ciudades a visitar, junto con el *TDA Cola Con Prioridad*, especificando en cada caso la función de prioridad que permita realizar lo solicitado. No se considerará válida ninguna otra solución que no haga uso de estos TDAs.

```
typedef struct ciudad {
   char * nombre;
   float pos_x;
   float pos_y;
} * TCiudad;
```

viajes.txt:	**Mostrar ascendente:**	**Mostrar descendente:**	**Reducir horas manejo**
1;1	1. Salliqueló.	1. Bahía Blanca.	1. Salliqueló.
Salliqueló;2;2	2. Carhué.	2. Trenque Lauquen.	2. Carhué.
Bahía Blanca;4;4	3. Trenque Lauquen.	3. Carhué.	3. Bahía Blanca.
Trenque Lauquen;4;0	4. Bahía Blanca.	4. Salliqueló.	4. Trenque Lauquen.
Carhué;0;3			Total recorrido: 14.

Ejemplo 1

Considere a modo de ejemplo, el funcionamiento del programa solicitado en función a la siguiente invocación: `$ planificador viajes.txt`

Del ejemplo se puede deducir que, el usuario se encuentra actualmente en la ubicación $< 1; 1 >$, y que Salliqueló es una ciudad que el usuario desea visitar al igual que las ciudades de Bahía Blanca, Trenque Lauquen y Carhué. En particular, Carhué se encuentra en la ubicación $< X, Y > = < 0, 3 >$.

Sobre la implementación

- Los archivos fuente principales se deben denominar **lista.c**, **colacp.c** y **planificador.c** respectivamente. En el caso de las librerías, también se deben adjuntar los respectivos archivos de encabezados **lista.h y colacp.h**, los cuales han de ser incluidos en los archivos fuente de los programas que hagan uso de las mismas.
- Es importante que durante la implementación del proyecto se haga un uso cuidadoso y eficiente de la memoria, tanto para reservar (`malloc`), como para liberar (`free`) el espacio asociado a variables y estructuras.
- Se deben respetar con exactitud los nombres de tipos y encabezados de funciones especificados en el enunciado. Los proyectos que no cumplan esta condición quedarán automáticamente desaprobados.
- La compilación debe realizarse con el *flag* `-Wall` habilitado. El código debe compilar **sin advertencias** de ningún tipo.
- La copia o plagio del proyecto es una falta grave. Quien incurra en estos actos de deshonestidad académica, desaprobará automáticamente el proyecto.

Sobre el estilo de programación

- El código implementado debe reflejar la aplicación de las técnicas de programación modular estudiadas a lo largo de la carrera.
- En el código, entre eficiencia y claridad, se debe optar por la claridad. Toda decisión en este sentido debe constar en la documentación que acompaña al programa implementado.
- El código debe estar indentado, comentado, y debe reflejar el uso adecuado de nombres significativos para la definición de variables, funciones y parámetros.