ORGANIZACIÓN DE COMPUTADORAS
Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur
Segundo Cuatrimestre de 2022

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 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 Cola Con Prioridad

Implementar un *TDA Cola Con Prioridad* en lenguaje C, 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 debe proveer las siguientes operaciones:

1. `TColaCP crear_cola_cp(int (*f)(TEntrada, TEntrada))` Crea y retorna una cola con prioridad vacía. El orden en que las entradas deben ser retiradas de la cola estará dado por la función de prioridad `int f(TEntrada, TEntrada)`. Se considera que la función `f` devuelve -1 si la clave de la entrada como primer argumento tiene menor prioridad que la clave de la entrada del segundo argumento, 0 si la prioridad es la misma, y 1 si la prioridad es mayor.

2. `int cp_insertar(TColaCP cola, TEntrada entr)` Agrega la entrada `entr` en la cola. Retorna verdadero si procede con éxito, falso en caso contrario.

3. `TEntrada cp_eliminar(TColaCP cola)` Elimina y retorna la entrada con mayor prioridad (esto es, menor clave) de la cola. Reacomoda la estructura heap de forma consistente. Si la cola es vacía, retorna `ELE_NULO`.

4. `int cp_cantidad(TColaCP cola)` Retorna la cantidad de entradas de la cola.

5. `void cp_destruir(TColaCP cola, void (*fEliminar)(TEntrada) )` Elimina todas las entradas y libera toda la memoria utilizada por la cola `cola`. Para la eliminación de las entradas, utiliza la función `fEliminar`.

En los casos anteriormente indicados, sin considerar la operación `crear_cola_cp`, si la cola parametrizada no está inicializada, se debe abortar con *exit status* `CCP_NO_INI`.

Para la implementación, se debe considerar que los tipos `TColaCP`, `TNodo`, `TEntrada`, `TClave` y `TValor` están definidos de la siguiente manera:

```
typedef struct cola_con_prioridad {
   int cantidad_elementos;
   TNodo raiz;
   int (*comparador)(TEntrada, TEntrada);
} * TColaCP;

typedef struct nodo {
   TEntrada entrada;
   struct nodo * padre;
   struct nodo * hijo_izquierdo;
   struct nodo * hijo_derecho;
} * TNodo;

typedef struct entrada {
   TClave clave;
   TValor valor;
} * TEntrada;

typedef void * TClave;
typedef void * TValor;
```

2. Programa Principal

Implementar una aplicación de consola que reciba como argumento por línea de comandos el nombre de un archivo de texto con el formato indicado en el *Ejemplo 1* y ofrezca 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 indicar 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 respetar la siguiente sintaxis 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 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 este TDA.

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

Ejemplo 1

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

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.

Del ejemplo se puede deducir que el usuario se encuentra 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 $< X, Y > = < 0, 3 >$.

Constantes a utilizar

Se deberán definir las siguientes constantes:

Constante	***Valor***	***Significado***
`FALSE`	`0`	Valor lógico falso.
`TRUE`	`1`	Valor lógico verdadero.
`CCP_NO_INI`	`2`	Intento de acceso inválido sobre Cola CP sin inicializar.
`POS_NULA`	`NULL`	Posición nula.
`ELE_NULO`	`NULL`	Elemento nulo.

Sobre la implementación

- Los archivos principales se deben denominar **colacp.c** y **planificador.c** respectivamente. También se debe adjuntar el archivo de encabezados colacp.h.
- 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 los nombres de tipos y encabezados de funciones especificados en el enunciado. Los proyectos que no cumplan esta condición automáticamente recibirán por calificación una **E**.
- La compilación debe realizarse con el *flag* `-Wall` habilitado. El código debe compilar **sin advertencias** de ningún tipo. Los proyectos que no cumplan esta condición automáticamente recibirán por calificación una **E**.
- La copia o plagio del proyecto es una falta grave. Quien incurra en estos actos de deshonestidad académica desaprobará el proyecto y **el cursado de la materia**.

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 interna del código fuente asociado 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.

Entrega

- Las comisiones estarán conformadas por 2 alumnos.
- La entrega del código fuente se realizará a través de un archivo comprimido **zip** o **rar**, denominado ***P-Comision-XX*** (el valor XX debe modificarse por el número de comisión correspondiente) donde se deben incorporar los archivos fuente ".c" y ".h" además de la documentación, donde se debe incorporar el informe del proyecto ".pdf" (ningún otro).
- El archivo comprimido debe subirse a la plataforma Moodle, en el apartado *Proyecto de programación*, dentro de la tarea Proyecto [actividad]. Para esto, tenga en cuenta las siguientes consideraciones:
  - **Fecha límite entrega:** jueves 03/11/22, 20:00 hs. Considere que la plataforma Moodle de forma **automática inhabilitará** la opción de entrega a partir de esta hora, por lo que se recomienda no esperar a último momento para realizar la entrega.
  - **Intentos de entrega:** la plataforma Moodle **no permitirá** realizar más de una entrega de los archivos, por lo que debe asegurarse de que el archivo comprimido cuente con todos los archivos requeridos antes de confirmar el envío. No se aceptará ninguna entrega fuera de la plataforma Moodle.
  - El archivo comprimido debe subirlo **sólo uno** de los integrantes de la comisión.
  - La cátedra no se responsabiliza por cualquier inconveniente surgido a partir del envío del proyecto a último momento. Es importante **no esperar hasta último momento** para realizar el envío evitando así todo tipo de situación excepcional que pudiese ocurrir (sobrecarga o caída del sistema, fallas en la conexión a Internet, etc.).

Corrección

- La cátedra evaluará tanto el **diseño**, **implementación**, **documetanción**, **presentación** y cumplimiento de **todas** las condiciones de entrega.