**Algoritmos y Estructuras de Datos**
Guía de ejercicios de preparación para el 2do examen parcial - parte 2 (versión 1)

**Temas:** Archivos. Estructura cola, pila, lista, lista doblemente enlazada. Combinación de estructuras.

**1) Integración.** Un sistema antiguo (legacy), guarda los archivos de texto con codificación ANSI y utiliza guillemets («», también conocidas como comillas de ángel o comillas francesas), en lugar de las comillas dobles “” como es habitual y esto trae problemas cuando esos archivos son persistidos en una base de datos relacional.

**Se pide**: Crear un programa que procese el archivo original (entrada) y sustituya todas las apariciones de los caracteres «» (códigos de caracter 171 y 187 respectivamente) y los reemplace por la comilla doble " (código 34). El programa deberá recibir como primer parámetro el nombre del archivo de origen y el segundo, el de salida. **Info:** Para mostrar todos los parámetros recibidos por la función ***main()***:

```
int main(int argc, char *argv[]){
        for(int i=1; i<argc; i++) {
                cout << argv[i] << endl;
        }
```

**2) Checkin.** Se desea estudiar el tiempo de espera de los pasajeros al hacer el checkin en el aeropuerto, para lo cual se debe desarrollar un sistema que permita medir el tiempo de espera de cada pasajero desde que ingresa a la fila y hasta que es atendido. Cada vez que ingresa un nuevo pasajero a la fila (única), se emite y coloca en el equipaje un código de barras autoadhesivo con el dato del momento de ingreso (id y momento de ingreso). Una vez que el pasajero es atendido, se lee este código y queda registrado el momento de salida de la fila.

**Se pide:** Crear un programa que registre el ingreso/egreso de pasajeros a la fila y muestre por pantalla el tiempo transcurrido de espera por persona. Cada pasajero es representado en el sistema por los siguientes datos:

| | |
|---|---|
| **id_pasajero** (entero) | **momento_de_ingreso** (long) |

Cada vez que se registra un nuevo pasajero, se invocará la función ***nuevoPasajero()***. Cada vez que un pasajero es atendido, se ejecutará la función ***atenderProximo()***. Luego de retirar el nodo, se deberá calcular la diferencia de tiempo invocando la función provista ***tiempoDeEspera(long momento_de_ingreso)*** que muestra por pantalla el tiempo transcurrido de cada pasajero en la fila. Crear las estructuras necesarias y las funciones no provistas. Info: Para obtener el momento actual, invocar a la función: time(NULL) de la biblioteca time.h

**3) NoSQL.** Una implementación de base de datos NoSQL organiza los documentos asociados a un identificador de objeto (OID) a través de un mapa <clave,valor>[1], donde la clave es un OID y el valor es una lista de documentos de texto (JSON, XML, CSV, etc.). Internamente, el motor de la base de datos organiza a todos los mapas en arrays de longitud máxima de 255 elementos y para hacer backup establece en su API la posibilidad de desarrollar una función para exportar todos los documentos asociados a un OID. Cada registro de documento posee los siguientes atributos:

| | | |
|---|---|---|
| **id** (entero) | **activo** (booleano) | **documento** (char [2048]) |

**Se pide:** Crear la función ***exportar()*** que recibe el array de todos los documentos del sistema y guarda en un archivo binario todos los registros asociados a un OID. La función exportar() recibe por parámetro el array, el OID (validado) y la ruta al archivo donde se desea grabar todos los registros (documentos) del elemento objetivo. Crear las estructuras necesarias.

[1] https://db-engines.com/en/article/Key-value+Stores