INF-134 Estructuras de Datos, 2023-1

Tarea 3

Profesores: Gabriel Carmona, Juan Pablo Castillo, Roberto Díaz, Hubert Hoffmann
gabriel.carmonat@sansano.usm.cl,
juan.castillog@sansano.usm.cl,
roberto.diazu@usm.cl
hoffmann@inf.utfsm.cl

Ayudantes San Joaquín:
Diego Debarca
diego.debarca@usm.cl
Gabriel Escalona
gabriel.escalona@usm.cl
Joaquín Gatica
joaquin.gatica@sansano.usm.cl
Juan Cucurella
juan.cucurella@usm.cl
Rodrigo Flores
rodrigo.floresf@sansano.usm.cl

Ayudantes Casa Central:
Catalina Cortez
catalina.cortez@usm.cl
Benjamin Cayo
benjamin.cayo@usm.cl
Sebastián Torrealba
sebastian.torrealba@usm.cl
Tomás Barros
tomas.barros@sansano.usm.cl

Fecha de entrega: 30 de junio, 2023.
Plazo máximo de entrega: 5 horas.

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 `g++`, usando la línea de comando `g++ archivo.cpp -o output -Wall`. **No se aceptarán variantes o implementaciones particulares de g++, como el usado por MinGW**. 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. No se permite usar la biblioteca *stl*, así como ninguno de los *containers* y algoritmos definidos en ella (e.g. `vector`, `list`, etc.). Está permitido usar otras funciones de utilidad definidas en bibliotecas estándar, como por ejemplo `math.h`, `string`, `fstream`, `iostream`, etc.

2. Objetivos

Entender y familiarizarse con la implementación y utilización de estructuras de datos tipo hashing y colas de prioridad. 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 describen los problemas a resolver implementando programas, funciones o clases en C++. Se debe entregar el código para cada uno de los problemas en archivos `.cpp` separados (y correspondientes `.hpp` de ser necesario).

3.1. Problema 1: Users Login

Hoy en día el inicio de sesión es la tarea más común que se realiza en cualquier sitio web o aplicación. Dado esto los han contactado para implementar un *login* que utilizará una aplicación que manipulará grandes cantidades de usuarios, donde cada usuario está identificado por: un nombre de usuario (un string de longitud máxima 32), y este usuario tiene asociada una clave (un string de 8 caracteres como mínimo).

Para resolver este problema usted deberá implementar el TDA `Login` que implemente las siguientes operaciones como mínimo:

- `bool iniciar_sesion(string usuario, string clave);`
  - Dado el nombre `usuario` debe verificar si el usuario está registrado, en caso de no estarlo imprime por pantalla "El usuario no se encuentra registrado" y retornar `false`
  - Si el usuario está registrado y la contraseña no coincide imprime "La clave ingresada no coincide" y retornar `false`.
  - Si el usuario está registrado y la contraseña coincide imprime "Sesion iniciada con exito" y retornar `true`.
- `bool crear_nuevo_usuario(string usuario, string clave);`
  - Debe verificar si `usuario` ya está registrado, en caso de estarlo debe imprimir por pantalla "El usuario ya se encuentra registrado" y retornar `false`.
  - En caso de que el usuario no esté registrado debe verificar que la clave tiene como mínimo 8 caracteres, si no cumple esta restricción de imprimir "La clave debe tener al menos 8 caracteres" y retornar `false`.
  - En otro caso registra al usuario, imprime "Usuario registrado con exito" y retornar `true`.
- `bool cambiar_clave(string usuario, string nueva_clave);`
  - Debe verificar si `usuario` ya está registrado, en caso de no estarlo debe imprimir por pantalla "El usuario no se encuentra registrado" y retornar `false`.
  - En caso de encontrarse registrado debe verificar que la nueva clave cumpla con las restricciones. Si la nueva clave no tiene un mínimo de 8 caracteres imprime "La clave debe tener al menos 8 caracteres" y retornar `false`.
  - En otro caso actualiza la clave, imprime "Clave actualizada con exito" y retorna `true`.

Para resolver las consultas de forma eficiente sobre el TDA `Login`, deberá ser implementado utilizando las técnicas de hashing cerrado vistas en el curso. El tamaño inicial de la tabla de hashing debe ser de 32 (o algo cercano a 32). Además, no debe superar un factor carga $\alpha = 0{,}7$. Cuando la tabla sobrepase el factor de carga de 0,7, debe crear una nueva tabla duplicando[1] el tamaño de la tabla anterior y copiar los valores de la antigua tabla hacia la nueva tabla[2].

[1] Puede ser algo cercano al doble del tamaño de la tabla anterior si le es útil.

[2] Debe tener cuidado de volver a insertar los usuarios utilizando la función de hashing nuevamente.

3.1.1. Entrada

La entrada se recibirá mediante la entrada estándar. Las operaciones se reciben con tres parámetros separados por un espacio:

1. Tipo de operación, las cuales pueden ser: `INICIAR_SESION`, `REGISTRAR` o `ACTUALIZAR`, que se definieron en la sección anterior.
2. Nombre del usuario.
3. Contraseña.

Finalmente, el programa termina cuando recibe la operación `FINALIZAR`. Un ejemplo de una secuencia de operaciones es la siguiente:

```
REGISTRAR user1 password1
REGISTRAR user2 password2
REGISTRAR user4 1234
INICIAR_SESION user1 password
INICIAR_SESION user2 password2
ACTUALIZAR user1 password
ACTUALIZAR user2 1234
ACTUALIZAR user3 1234
INICIAR_SESION user1 password
REGISTRAR user2 password
FINALIZAR
```

3.1.2. Salida

La salida será mediante la salida estándar. La salida correspondiente al ejemplo planteado anteriormente es el siguiente:

```
Usuario registrado con exito
Usuario registrado con exito
La clave debe tener al menos 8 caracteres
La clave ingresada no coincide
Sesion iniciada con exito
Clave actualizada con exito
La clave debe tener al menos 8 caracteres
El usuario no se encuentra registrado
Sesion iniciada con exito
El usuario ya se encuentra registrado
```

3.2. Problema 2: Comandos

Unos dispositivos usados para distintos propósitos pueden ser administrados y manipulados remotamente por un servidor central por medio de *comandos push*. Los dispositivos solicitan del servidor uno o más comandos para ser ejecutados para cambiar su configuración o datos.

Su tarea es crear la lógica del servidor que decide cual son los siguientes comandos a enviar a un dispositivo. Los comandos deberán guardarse en una estructura tipo heap mientras no sean consumidos.

Cada comando tendrá tres componentes, un **identificador** que se considerará único, un entero que indica su **prioridad** (mientras más bajo, antes deberá enviarse al dispositivo, y nunca habrá dos comandos con la misma prioridad) y un string arbitrario que indica la **instrucción** a ejecutar por el dispositivo.

Así el servidor tres operaciones las que serán recibidas como strings:

1. `PUSHCOMMAND ID PRIORITY INSTRUCTION`: que crea un nuevo comando a ser enviado a los dispositivos donde `ID` es el identificador, `PRIORITY` su prioridad e `INSTRUCTION` la instrucción que deberá ejecutarse.

2. `GET N`: en que el servidor deberá entregar `N` comandos para que sean ejecutados por el dispositivo. `N` será un número entero arbitrario mayor a cero.

3. `TERMINATE`: que limpia los recursos y finaliza el programa del servidor indicando cuantos comandos se enviaron del total de los encolados.

Cuando un nuevo comando es creado, deberá colocarse en memoria principal.

3.2.1. Entrada

Cada instrucción será recibida por entrada estándar siendo delimitadas por el salto de línea. A continuación se muestra un ejemplo de entrada:

```
PUSHCOMMAND WNWSM4 1 TZ 1 0 86400
PUSHCOMMAND AJDoMI 3 TZ 3 61200 32400
PUSHCOMMAND wsOvMk 2 TZ 2 32400 100000
GET 2
GET 1
GET 1
PUSHCOMMAND 9HmMas 8 USER 926 Diego 1 2 0
PUSHCOMMAND JotUAw 5 USER 861 Juan 1 1 0
PUSHCOMMAND hIuRAh 7 USER 456 Roberto 1 1 0
PUSHCOMMAND hjCbZ7 6 USER 486 Gabriel 1 1 0
GET 3
TERMINATE
```

3.2.2. Salida

La salida del programa será por salida estándar mostrando un output distinto por cada operación:

- `PUSHCOMMAND`: la cantidad de comandos pendientes por enviar seguidos del string `PENDING`.
- `GET N`: mostrar en una el entero `N` seguido de los identificadores de los comandos a enviar y luego, en las `N` líneas siguientes, la instrucción de cada comando. Si no hay comandos que enviar se muestra un cero.
- `TERMINATE`: mostrar cuantos comandos fueron enviados seguido del string `SENT` y luego cuantos comandos fueron generados seguido del string `CREATED`.

A continuación se muestra la salida estándar para el ejemplo de entrada de la sección anterior:

Salida Estándar:

```
1 PENDING
2 PENDING
3 PENDING
2 WNWSM4 wsOvMk
TZ 1 0 86400
TZ 2 32400 100000
1 AJDoMI
TZ 3 61200 32400
0
```

```
1 PENDING
2 PENDING
3 PENDING
4 PENDING
3 JotUAw hjCbZ7 hIuRAh
USER 861 Juan 1 1 0
USER 486 Gabriel 1 1 0
USER 456 Roberto 1 1 0
6 SENT 7 CREATED
```

4. Entrega de la Tarea

Entregue la tarea enviando un archivo comprimido `tarea3-apellido1-apellido2-apellido3.zip` o `tarea3-apellido1-apellido2-apellido3.tar.gz` (reemplazando sus apellidos según corresponda) a la página aula.usm del curso, a más tardar el día 30 de junio, 2023, a las 23:59:00 hs (Chile Continental), el cual contenga como mínimo:

- 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 para cada problema de la tarea (debe ser alguna de las indicadas en los tutoriales entregados en Aula USM).
- Los nombres de los archivos con el main para ejecutar cada problema deben ser: `problema1.cpp` y `problema2.cpp`

5. Restricciones y Consideraciones

- Por cada hora de atraso en la entrega de la tarea se descontarán 10 puntos en la nota.
- El plazo máximo de entrega es 5 horas 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 indicadas 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**.