INF-134 Estructuras de Datos, 2024-2

Tarea 3

Profesores: Andrés Navarro, Ricardo Salas, Juan-Pablo Castillo
andres.navarrog@usm.cl, ricardo.salas@usm.cl, juan.castillog@sansano.usm.cl

Ayudantes:
Gabriel Escalona (gabriel.escalona@usm.cl)
Tomás Barros (tomas.barros@sansano.usm.cl)
Jaime Inostroza (jinostroza@usm.cl)
Juan Alegría (juan.alegria@usm.cl)
Carlos Lagos (carlos.lagosc@usm.cl)
Damian Rojas (damian.rojas@usm.cl)

Fecha de entrega: 24 de Noviembre, 2024.
Plazo máximo de entrega: 1 día.

1. Reglas

- La presente tarea debe hacerse en equipos de dos personas. Toda excepción a esta regla está sujeta a autorización del ayudante coordinador Gabriel Escalona.
- Debe usarse el lenguaje de programación **C++**. Al realizar la evaluación, 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** (normalmente ocupado en **Dev C++**). Se deben seguir los tutoriales disponibles en Aula USM.
- No se permite usar la biblioteca STL, así como ninguno de los contenedores 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.
- Recordar que una única tarea en el semestre puede tener nota menor a 30. El incumplimiento de esta regla implica reprobación del curso.

2. Objetivos de aprendizaje

Entender el funcionamiento del hashing

- Aprender como crear funciones de hashing eficientes.
- Aprender como gestionar el manejo de colisiones dentro de una tabla hash.
- Aprender a redimensionar una tabla hash.

3. Problema a resolver

Un restaurante local en rápido crecimiento necesita una solución para registrar y gestionar eficientemente sus pedidos. Los pedidos pueden ser para servir o llevar. El sistema debe permitir almacenar, modificar y eliminar pedidos. Cada pedido puede incluir varios platos, y se debe calcular el total a pagar, tanto con propina como sin ella (10 %).

3.1. Requisitos del sistema

Para almacenar los pedidos se debe utilizar una tabla de hashing cerrado. La llave en la tabla de hashing es una combinación del identificador del pedido y el tipo de pedido. Los pedidos para servir tiene un identificador asociado al número de mesa (1 a n, siendo n el total de mesas) y los pedidos para llevar tienen asociado un identificador único que comienza en 1 y aumenta secuencialmente.

Se debe mantener un factor de carga menor o igual a 0.8 en el sistema. En caso de sobrepasar el factor de carga, se debe copiar los datos a una nueva tabla de tamaño 50 % mayor al tamaño actual.

Se solicita que el sistema provea las siguientes funcionalidades:

- Registro de pedidos.
  - **Guardar** cada pedido.
  - **Agregar** platos a un pedido.
  - **Finalizar** el pedido, mostrando el detalle con los platos y el total a pagar, incluyendo un cálculo opcional de propina.
- Eliminación de pedidos: Una vez que se paga el pedido, este debe eliminarse del registro.

Cada pedido puede registrar un máximo de 25 platos.

3.2. Especificación de clases

Se deben implementar las siguientes clases y estructuras:

```
struct Plato{
    std::string nombre;
    int precio;
};

class Pedido{
    private:
    Plato* platos; // arreglo de platos en el pedido, tamaño inicial 25
    bool servir; // true para servir, false para llevar
    size_t cant_platos;

    public:
    Pedido();
    ~Pedido();
    void agregar_plato(Plato* plato); // agrega un plato al pedido
    int precio_total(); // retorna la suma del precio de todos los platos del pedido
};

class Registro{
    private:
    Pedido* pedidos; // arreglo de pedidos, tamaño inicial n (cantidad de mesas)
    size_t size;
    void ajustar_arreglo(); //ajusta el tamaño de la tabla de hashing
    int ganancias;

    public:
    Registro();
```

```
    ~Registro();
    void agregar_pedido(Pedido* pedido);
    Pedido* get_pedido(int id, bool tipo); // Retorna el pedido según id y tipo (servir true
                                           // llevar false)
    Pedido* eliminar_pedido(int id, bool tipo); // Elimina el pedido según id y tipo
};
```

3.3. Ejecución

El programa comienza leyendo un archivo llamado `Menu.txt`, que contiene la información de los platos. La primera línea indica el número total de platos del Menú, y las líneas siguientes presentan cada plato con su nombre y precio, separados por un guion. A continuación se presenta la estructura del archivo `Menu.txt`.

```
num_platos
nombre_plato1-precio_plato1
nombre_plato2-precio_plato2
...
nombre_plato_n-precio_plato_n
```

Al iniciar el programa, el usuario debe ingresar el número de mesas, que determina el tamaño inicial del arreglo `pedidos`.

3.4. Comandos disponibles en consola

- `registrar <tipo><mesa>`: Inicia un nuevo pedido del tipo especificado (servir o llevar). Si se trata de un pedido para servir, se debe indicar el número de mesa y en caso de un pedido para llevar se asigna un identificador automáticamente. Cada pedido debe tener por lo menos un plato asociado. Si se intenta registrar una mesa que ya tiene un pedido asociado, se debe mostrar por mencionar que la mesa ya está ocupada. Luego se pueden ejecutar los siguientes comandos:
  - `agregar <nombre_plato>`: Agrega un plato al pedido en curso.
  - `pedir`: Finaliza el pedido y muestra el tipo y el identificador del pedido en consola.

  Ejemplo de pedido para servir:

  ```
  >registrar mesa 1
  >agregar Pad Thai
  >agregar Pastel de Choclo
  >pedir
  mesa 1 Registrado
  ```

  Ejemplo de pedido para llevar:

  ```
  >registrar llevar
  >agregar Gohan
  >agregar Pho
  >pedir
  llevar 1 Registrado
  ```

- `info <tipo_pedido><id_pedido>`: Muestra la información de un pedido específico.

  ```
  >info mesa 2
  Pho
  precio total: 10500
  ```

- `pagar <tipo_pedido><id_pedido>`: Elimina el pedido del registro, liberando la mesa y muestra el detalle del pedido con el total (con y sin propina) y el factor de carga actual de la tabla. En este ejemplo el factor de 0.2 corresponde a dos casillas ocupadas por `llevar 1` y `mesa 2` que quedan aún registradas luego de la eliminación de `mesa 1` desde una tabla de 10 de capacidad.

```
>pagar mesa 1
Pad Thai - 10000
Pastel de Choclo - 10400
total: 20400
propina: 2040
total + propina: 22440
Factor de carga: 0.2
```

- `cerrar`: Muestra por pantalla los pedidos pendientes, y el total de las ganancias del día (Suma de los pedidos pagados con propina).

```
>cerrar
Pedido pendientes:
mesa 2
llevar 1
Ganancias: 22440
```

3.5. Ejemplo completo

Para el siguiente archivo `Menu.txt`

```
5
Pad Thai-10000
Pho-10500
Lomo a lo Pobre-9500
Pastel de Choclo-10400
Gohan-8000
```

Considerando n=10, se pueden ejecutar los siguientes comandos:

```
>10
>registrar mesa 1
>agregar Pad Thai
>agregar Pastel de Choclo
>pedir
mesa 1 Registrado
>registrar llevar
>agregar Gohan
>agregar Pho
>pedir
llevar 1 Registrado
>registrar mesa 2
>agregar Pho
>pedir
mesa 2 Registrado
>info mesa 2
- Pho
precio total: 10500
>pagar mesa 1
Pad Thai - 10000
Pastel de Choclo - 10400
total: 20400
propina: 2040
total + propina: 22440
Factor de carga: 0.2
>cerrar
Pedido pendientes :
llevar 1
mesa 2
Ganancias: 22440
```

4. Entrega de la Tarea

Entregue la tarea enviando un archivo comprimido denominada `tarea3-apellido1-apellido2.zip` (reemplazando sus apellidos según corresponda), en cuyo interior debe estar la tarea dentro de una carpeta también llamada `tarea3-apellido1-apellido2`, a la página `aula.usm.cl` del curso, el cual contenga:

- Los archivos con los códigos fuentes necesarios para el funcionamiento de la tarea. ¡Los archivos deben compilar!
- El archivo **nombres.txt** que debe contener Nombre, ROL y Paralelo de cada miembro del equipo detallando qué programó cada uno.
- **README.txt** con las instrucciones de compilación en caso de ser necesarias, y la forma de compilación que usó (debe ser alguna de las indicadas en los tutoriales entregados en Aula USM).

Considere además que:

- El **apellido1** es el primer apellido de uno de los integrantes y el **apellido2** es el primer apellido del otro integrante. Esto significa que el **apellido1** y **apellido2 no son los dos apellidos de alguno de los integrantes**.
- Solo una persona del grupo debe subir la tarea al aula, no hacer esto, puede conllevar un descuento en la nota de la tarea.

5. Entrega mínima

La entrega mínima permite obtener una nota 30 si cumple con los siguientes requisitos:

- Leer archivo menu.txt e implementar el posicionamiento de pedidos en la tabla.

6. Restricciones y Consideraciones

- Se permite un único día de atraso en la entrega de la tarea. La nota máxima que se puede obtener en caso de entregar con un día de atraso es nota 50.
- Las tareas que no compilen no serán revisadas y deberán ser re-corregidas por el ayudante coordinador.
- 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 en la nota.
- Si se detecta **COPIA** la situación será revisada por ayudante y profesor coordinador.
- La prolijidad, orden y legibilidad del código fuente es obligatoria. Habrá descuentos si alguno de estos ítems no se cumple.

7. Directrices de programación

Las directrices corresponden a buenas prácticas de programación, las cuales serán tomadas en consideración para la evaluación de la tarea. 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**.