INF-134 Estructuras de Datos, 2021-1

Tarea 1

Profesores: Diego Arroyuelo B., Roberto Díaz U.
darroyue@inf.utfsm.cl, roberto.diazu@usm.cl

Ayudantes:
Diego Beltrán (diego.beltran@sansano.usm.cl)
Martin Crisóstomo (martin.crisostomo@sansano.usm.cl)
Joaquín Gatica (joaquin.gatica@sansano.usm.cl)
Diana Gil (diana.gil@sansano.usm.cl)
Martín Salinas (martin.salinass@sansano.usm.cl)
Sebastián Sepúlveda (sebastian.sepulvedab@sansano.usm.cl)
Daniel Tapia (daniel.tapiara@sansano.usm.cl)

Fecha de entrega: 14 de mayo, 2021.
Plazo máximo de entrega: 5 días.

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`. Alternativamente, se aceptan variantes o implementaciones particulares de `g++`, como el usado por MinGW (que está asociado a la IDE code::blocks). 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.

2. Objetivos

Comprender y familiarizarse con las estructuras y tipos de datos básicos que provee el Lenguaje de Programación C++. Entre los conceptos mas importantes, se encuentran:

- Paso de parámetros por valor.
- Paso de parámetros por referencia.
- Asignación de memoria dinámica.
- Manipulación de punteros.
- Manejo de Archivos.

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 deben implementarse funciones y clases en C++, de acuerdo a las siguientes descripciones. Se debe entregar cada uno de los problemas en archivos `.cpp` separados (y correspondientes `.hpp` de ser necesario). Para cada problema, entregue una función `main` adecuada que permita probar sus programas. Sin embargo, tenga en cuenta que durante la corrección se probarán otras funciones `main`.

3.1. Problema 1: Aforo Máximo

Un supermercado necesita mantener un registro de la cantidad de personas que se encuentran dentro del local a cada minuto, de forma de cumplir con las normas sanitarias vigentes de aforo máximo. Para ello se han instalado dispositivos automáticos que permiten contar personas en las entradas principales de público general en cada minuto del día. Llamaremos cuenta-personas a dichos dispositivos. Por otro lado, para mantener el número de empleados y funcionarios del supermercado, se usan los huelleros biométricos utilizados para el control de asistencia. Estos permiten detectar la entrada y salida de los 100 empleados actualmente contratados. La tarea consiste en implementar la función `cantidadPersonas` que permite determinar el número de personas (tanto público como empleados) que se encuentran en el local a una hora específica. El prototipo de la función a implementar es el siguiente:

```
int cantidadPersonas(string hora);
```

El parámetro `hora` es un string que indica la hora en formato *hh:mm* de 24 horas, y el retorno de la función es un entero indicando la cantidad de personas dentro del local en esa hora.

Archivos. Ambos dispositivos (es decir, el cuenta-personas y el huellero) mantienen su información en archivos. La función `cantidadPersonas` debe extraer información de ellos y calcular la cantidad de personas desde estos archivos. Los *cuenta-personas* mantienen el flujo de gente en un archivo binario llamado `flujo-publico.dat` que contiene registros tipo `FlujoNeto` de la forma:

```
struct FlujoNeto {
    int hora;
    int minuto;
    int personas;
};
```

El entero `hora` puede tomar un valor entre 0 y 23, mientras que el entero `minuto` puede tomar un valor entre 0 y 59. Ambos campos indican una hora del día. Finalmente, el entero personas indica el cambio en el número de personas dentro del local, un número positivo indica que aumentó la cantidad de gente mientras un valor negativo indica que disminuyó.

Por ejemplo, si se definiera un registro de la siguiente forma:

```
FlujoNeto flujo = {15, 57, 10}
```

en este caso, la variable `flujo` indica que a las 15:57 se incrementó en 10 el número de personas en el local. Esto es, la diferencia entre personas que entraron y salieron del local es 10. Si una hora no aparece registrada en el archivo, se asume que no hubo cambio en el número de personas. Ocurre lo mismo si el entero `personas` es igual a cero.

Los *huelleros* registran las horas de entrada y salida de un empleado en particular, identificado por su RUT. Esta información es almacenada en un archivo de texto llamado `asistencia.txt`, en el que cada línea contiene los siguientes datos (separados entres sí) por espacios:

- Tipo de evento: un único carácter que puede ser '`E`' (para indicar una entrada) o '`S`' (para indicar una salida).
- RUT del empleado: un string, en formato *12345678-9*.

- Hora del evento: en formato *hh:mm* de 24 horas.

Cabe notar que por decisiones de diseño, un empleado puede marcar su entrada o salida múltiples veces seguidas, sin embargo, se considerará dentro del local solo después su primera entrada y hasta su primera salida. Un ejemplo de este archivo es el siguiente:

```
E 6379250-0 08:00
E 6379250-0 08:04
E 21056194-3 12:00
S 6379250-0 12:30
S 21056194-3 16:00
```

Aquí hay 2 empleados. El empleado con RUT 6379250-0 entró a las 08:00 y se retiró a las 12:30. Notar que registró una entrada adicional a las 08:04, que debe ser ignorada. El empleado con RUT 21056194-3 entró a las 12:00 y se retiró a las 16:00.

Supuestos. Para su desarrollo considere lo siguiente:

- El inicio del día es a las 00:00.
- El fin del día es a las 23:59.
- Al inicio del día habrán cero personas dentro del local.
- El flujo neto de personas de cada archivo es independiente del otro. El archivo `flujo-publico.dat` es exclusivo para público, mientras que el de `asistencia.txt` es exclusivo para empleados.
- Ambos archivos estarán ordenados por hora.
- Cada empleado que registre al menos una entrada tendrá registrada al menos una salida.
- Un empleado puede tener múltiples turnos, es decir, entrar y salir múltiples veces en un día.
- El cambio en número de personas se considera desde el minuto indicado en adelante.

A continuación se muestran ejemplos del número de personas en determinadas horas del día para el ejemplo de `FlujoNeto` y el archivo `asistencia.txt` presentados previamente:

- A las **00:00** hay **0** personas.
- A las **07:59** hay **0** personas.
- A las **08:00** hay **1** persona.
- A las **11:59** hay **1** persona.
- A las **12:00** hay **2** personas.
- A las **12:30** hay **1** persona.
- A las **15:56** hay **1** persona.
- A las **15:57** hay **11** personas.

3.2. Problema 2: El TDA Arreglo Extensible

Un *arreglo extensible* es una estructura de datos que funciona básicamente como un arreglo, pero que además puede crecer agregando elementos al final del mismo, y decrecer eliminando el último elemento. En esta parte de la tarea se pide implementar el TDA `arr_extensible` en C++ usando una clase (class). La misma debe implementar la siguiente funcionalidad sobre un arreglo extensible `A`$[0..n-1]$ que almacena valores enteros:

- `bool A.setValue(unsigned long i, int v)`: asigna el valor entero `v` a `A[i]`, para $0 \leq$ `i` $< n$, y retorna `true`. Si `i` $\geq n$, la operación no tiene ningún efecto sobre el arreglo, y retorna `false`.
- `int A.getValue(unsigned long i)`: retorna el valor entero de `A[i]`, para $0 \leq$ `i` $< n$. Si `i` $\geq n$, la operación no tiene efecto sobre el arreglo, pero debe imprimir el mensaje "`Error de acceso al arreglo`" en el stream `cerr`, y finalizar la ejecución del programa con `exit(1)`.
- `void A.append(int v)`: agrega una nueva componente `A`$[n]$ con valor entero `v`, haciendo crecer el arreglo.
- `void A.remove()`: elimina `A`$[n-1]$ del arreglo, haciendo que éste decrezca en una componente.
- `unsigned long A.size()`: devuelve n, la cantidad actual de componentes del arreglo.

Existen 3 maneras de inicializar un arreglo extensible:

1. Como un arreglo vacío, que no tiene ninguna componente.
2. Como un arreglo de un tamaño inicial n (de tipo `unsigned long`), con componentes sin inicializar.
3. Como un arreglo de un tamaño inicial n (de tipo `unsigned long`) y componentes inicializadas con un valor entero v.

En su implementación, debe usar un arreglo base (al que denotaremos B) para almacenar el arreglo extensible $A[0..n-1]$. Dicho arreglo base B debe ser manejado de forma dinámica, dentro de la clase. El arreglo A crecerá y decrecerá sobre el arreglo B. Eso implica que el arreglo B siempre tiene una mayor (o la misma) cantidad de componentes que A. Por ejemplo, si en un momento dado el arreglo base es de la forma $B[0..15]$ (es decir, tiene 16 componentes) y el arreglo A tiene 10 elementos, entonces estos serán almacenados en la porción $B[0..9]$ de B. En particular, si el arreglo extensible A tiene n componentes, entonces el arreglo base B tendrá 2^k componentes, tal que $2^{k-1} < n \leq 2^k$, para $k \geq 0$. En otras palabras, el arreglo base B será de tamaño 2^k, para el menor k tal que se cumple $n \leq 2^k$.

Para mantener la anterior invariante respecto al tamaño de B, note que el arreglo B debe crecer cuando A tiene el mismo tamaño que B y se hace `A.append(v)`. Si en ese momento tanto A como B tienen 2^k componentes y ocurre una operación `append`, el arreglo base B debe hacerse crecer a 2^{k+1} componentes antes de agregar el nuevo elemento. Este procedimiento debe implementarse de forma manual, pidiendo memoria dinámica para un nuevo arreglo de 2^{k+1} componentes, luego copiando los elementos del antiguo arreglo de tamaño 2^k, y finalmente liberando el espacio del antiguo arreglo B. No está permitido usar funciones de biblioteca que hagan este trabajo, tipo `realloc` de C.

De forma similar, si en un momento dado el arreglo A tiene $2^k + 1$ componentes y B tiene 2^{k+1} componentes, y ocurre una operación `A.remove()`, el arreglo base B debe pasar a tener 2^k componentes. El procedimiento a usar debe ser similar al caso anterior.

Su programa debe hacer uso responsable de la memoria dinámica. Se recomienda un manejo responsable de la misma en los constructores y el destructor de la clase a implementar, así como en las funciones `append` y `remove`.

4. Entrega de la Tarea

Entregue la tarea enviando un archivo comprimido `tarea1-apellido1-apellido2-apellido3.zip` o `tarea1-apellido1-apellido2-apellido3.tar.gz` (reemplazando sus apellidos según corresponda) a la página aula.usm del curso, a más tardar el día 14 de mayo, 2021, a las 23:59:00 hs (Chile Continental), el cual contenga:

- 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 (debe ser alguna de las indicadas en los tutoriales entregados en Aula USM).

5. Restricciones y Consideraciones

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