INF-134 Estructuras de Datos, 2022-1

Tarea 2

Profesores: Juan Pablo Castillo, Roberto Díaz, Javier Robledo
juan.castillog@sansano.usm.cl, roberto.diazu@usm.cl, javier.robledo@usm.cl

Ayudantes:
Domingo Benoit (domingo.benoit@sansano.usm.cl)
Joaquín Gatica (joaquin.gatica@sansano.usm.cl)
Diana Gil (diana.gil@sansano.usm.cl)
Martín Ruiz (martin.ruiz@usm.cl)
Martín Salinas (martin.salinass@sansano.usm.cl)
Betrice Valdes (beatrice.valdes@sansano.usm.cl)

Fecha de entrega: 06 de junio, 2022.
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. 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 listas y árboles.

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: El Teclado Descompuesto

Suponga que está tecleando un texto largo con un teclado que está descompuesto. El problema es que a veces las teclas "Atrás" (que retrocede el cursor un carácter), "Adelante" (que avanza el cursor un carácter), "Inicio" (que retrocede el cursor al comienzo del texto) y "Fin" (la cual mueve el cursor al final del texto) se presionan automáticamente, por una falla electrónica en el teclado. Usted no se ha dado cuenta de esto, dado que por preocuparse del texto ni siquiera ha mirado el monitor. Dado un texto de entrada, en el que se indica en qué momento se presionaron automáticamente las teclas Atrás, Adelante, Inicio y Fin, usted debe encontrar cuál es el texto resultante. Debe resolver el problema de manera eficiente **empleando listas enlazadas**. Su programa también debe hacer uso eficiente de la memoria.

3.1.1. Entrada

La entrada de datos se hará a través del archivo `teclado-entrada.txt`. El archivo contiene varios casos de prueba, cada uno en una única línea del archivo que contiene al menos 1 y a lo más 1.000.000 de letras (mayúsculas y minúsculas), guiones bajos ('_'), y los caracteres especiales '`<`', '`>`', '`[`' y '`]`'. Los caracteres '`<`' y '`>`' indican que se presionaron las teclas Atrás y Adelante respectivamente en ese momento. Los caracteres '`[`' y '`]`' indicando que se presionaron las teclas Inicio y Fin respectivamente. El archivo es terminado con `EOF`. Un ejemplo de entrada es el siguiente:

```
This_is_an_[example]_text
This<<_is_another<<_exam<<>ple
Yet_]anot[her_exa[mple
<<A_final>>_example
```

3.1.2. Salida

La salida de datos se hará a través del archivo `teclado-salida.txt`. Para cada caso de entrada, se debe imprimir el texto resultante que se observa en pantalla.

La salida correspondiente al ejemplo planteado anteriormente es el siguiente:

```
exampleThis_is_an__text
Th_is_anoth_exaplemeris
mpleher_exaYet_anot
A_final_example
```

3.2. Problema 2: ABB con Operación Rank

Debe implementar un Árbol Binario de Búsqueda (ABB) que soporte al menos las siguientes operaciones:

- `void insert(tElem x)`: inserta el elemento `x` en el árbol.
- `bool find(tElem x)`: busca el elemento `x` en el árbol y retorna un valor verdadero de encontrarse en el árbol.
- `int rank(tElem x)`: dado un elemento `x`, no necesariamente almacenado previamente en el ABB, retorna la cantidad de elementos que son menores o iguales a `x`.

Asuma la siguiente definición de `tElem` para el tipo de datos de los elementos:

```
typedef int tElem;
```

La implementación del operador `rank` debe ser eficiente, es decir, si buscar el elemento `x` en el ABB toma l comparaciones la operación `rank` debe calcularse en ese mismo tiempo. De ser necesario, modifique la estructura del árbol y/o los nodos del árbol para resolver esta operación de forma eficiente.

3.2.1. Consideraciones

El ABB debe implementarse como un TDA (una clase de C++) llamado `RankedABB` que tenga como mínimo las operaciones descritas y un constructor sin parámetros que inicializa el ABB como vacío. Puede extender el árbol e incluir otras operaciones o funciones que estime conveniente.

Notar que para este problema como mínimo será necesario entregar un archivo de implementación `ranked-abb.cpp` del TDA ABB. Evite definir una función `main` en este archivo.

Opcionalmente, puede entregar otro archivo `main.cpp` que contenga una función `main` que haga uso del TDA y que demuestre su buen funcionamiento. Sin embargo, debe considerar que durante la revisión es posible que se verifique su solución haciendo uso de otros potenciales `main.cpp`.

Adicionalmente, se puede entregar un archivo de cabecera `ranked-abb.hpp` y un archivo `makefile` si estima que facilitará la revisión de la tarea.

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 06 de junio, 2022, 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).

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**.