Tarea 1 - Estructuras de Datos (INF-134)

Publicación: Jueves 14 de Agosto
Entrega: Viernes 29 de Agosto - 23:59h vía Aula

2025-2

1 Reglas

- La tarea debe realizarse en grupos de 2 personas. Cualquier situación excepcional debe ser aprobada por el ayudante coordinador (Juan Daniel Alegría).
- Debe usarse el lenguaje de programación C++. Al realizar la evaluación, la tarea será compilada usando el compilador g++ en Linux. En esta Tarea, la compilación será hecha desde consola ó a través de un archivo Makefile que se encargará de realizar la compilación de su programa. Las opciones son entonces:

```
g++ tarea1.cpp -o tarea1 -Wall
```

ó

```
make
```

  No se aceptarán entregas realizadas con MinGW en Windows o compiladores online como Dev-C++, Replit, GDB, etc. En caso de usar Mac, se recomienda usar XCode, y de ser posible, testear la tarea también en Linux.
- No se permite usar la biblioteca STL, así como ninguno de los contenedores y algoritmos definidos en ella (e.g. vector, list, etc.). En general no se permite importar ninguna estructura de datos, salvo por string. Ante cualquier duda de alguna librería consulte en el foro correspondiente a la tarea, así la duda queda resuelta para todos.
- Una tarea que no compile tendrá nota 0. En dicha situación, el/la estudiante puede apelar con el ayudante coordinador.
- La tarea puede entregarse con máximo 1 día de atraso, en cuyo caso la nota máxima será 50.

- Solo una de las tres tareas del semestre puede tener nota inferior a 30. De todas maneras recuerde que es condición aprobatoria del curso que el promedio de las tres tareas sea mayor o igual a 55.
- Si se detecta **COPIA** o abuso de herramientas de generación automática de código (ChatGPT) la tarea tendrá nota 0.
- Una cierta cantidad grupos serán seleccionados aleatoriamente para interrogación, por parte de los profesores, respecto al desarrollo de la tarea.
- Los warnings significan descuento en la nota final.
- Los despliegues por pantalla no solicitados significan descuento en la nota final.

2 Problema a resolver: Caos en el final de semestre

Se acerca el fin de semestre y las notas de los alumnos deben estar en SIGA antes de la fecha de cierre oficial. Pero ha ocurrido una accidente, la base de datos que contiene la información de los estudiantes en los cursos impartidos en el semestre ha sido borrada por un practicante. Por suerte existía un archivo de recuperación que contiene la información perdida, aunque no todo son buenas noticias, este archivo está escrito en formato binario y, lamentablemente, cuando se escribió sufrió un desperfecto que alteró los datos. Como la Universidad sabe que cuenta con un excelente departamento de informática, ha decidido encargarle a usted y a su compañero que recuperen la información y salven el semestre de un final desastroso. Si bien el programa debe recuperar la data, debe también ser capaz de ser un sistema que sirva de parche mientras se recupera todo el sistema principal, por lo que su tarea no se reduce solo a recuperar la data, sino a operar en ella. La poca información que le han dado sobre el archivo de recuperación es la siguiente:

- El archivo que contiene la información se llama datos.dat
- El archivo está en un repositorio de GitHub `https://GitHub.com/Vindemiatrixx/T1_EDD_2025_02`. Durante la evaluación de la Tarea se ejecutará su código con archivos con diferente contenido.
- La información del archivo comienza con un numero entero positivo del tipo `unsigned long`, que llamaremos N, que indica la cantidad de registros dentro. Seguido de N registros de una `struct` cuya estructura se detalla a continuación:
- El archivo sufrió un error y lamentablemente contiene data duplicada.

Por otro lado, se le dejó una lista con las descripciones de las funcionalidades que se le piden implementar en su programa, para así ayudar a los profesores que necesitan una solución:

1. Función que permita leer el archivo *datos.dat* y cargarlo en memoria dinámica.

```
struct Dato{
    unsigned int Id;       //Id único por estudiante
    char Nombre[50];       //Arreglo de char para nombre y apellido
    char Fecha_Nacimiento[11]; //Fecha de nacimiento del estudiante
    int Notas[3];          //Arreglo con las notas de cada evaluación
    char Curso[7];         //Sigla del curso, en formato INF134 por
                           //ejemplo.
    unsigned short VTR;    //Veces que alguien ha rendido el ramo
                           //(1-3).
};
```

Figura 1: Definición estructura

2. Función que permita eliminar los alumnos duplicados de los cursos.

3. Función que permita agregar un estudiante a un curso, pidiendo rellenar los campos requeridos por la struct.

4. Función que permita obtener el promedio de notas de cierta asignatura para un estudiante.

5. Función que permita obtener el promedio por evaluación de una asignatura (Como el arreglo es de tamaño 3, existen 3 evaluaciones y la evaluación 1 corresponde a Notas[0]).

6. Función que permita saber todos los cursos en los que está un estudiante.

7. Función que permita saber qué alumnos de un curso (ingresado por consola) con un VTR (ingresado por consola) están reprobando la asignatura.

8. Función que permita generar un archivo .txt por cada curso, junto a su lista de estudiantes, además de información relevante del curso, esta "información relevante" está detallada en la definición de la función.

2.1 Preguntas Teóricas (Bonus)

Para cada una de estas preguntas se deben citar fuentes confiables. Se considera una fuente fidedigna a cualquier publicación, libro o documentación que tenga un moderado apoyo comunitario, fama o renombre [1]. En caso de no existir una fuente única con la información, citar aquellas fuentes que permitieron derivar la respuesta.

Las respuestas a estas preguntas deben adjuntarse en un archivo llamado RESPUESTAS.md, y debe utilizar correctamente la sintaxis de markdown. El puntaje de la pregunta teórica se asignará unicamente si, además de responder en palabras, se implementa lo solicitado, es decir, si se le pregunta por alguna funcionalidad, la respuesta teórica no basta

[1] Use las directrices especificadas en `https://biblioteca.usm.cl/busqueda/uso_etico_informacion` para listar sus citas. Los modelos de lenguaje, como ChatGPT, no son fuentes, si bien pueden proveerlas.

si no está implementada en su entrega. Recordar que las respuestas deben ser completas y bien desarrolladas, no se aceptarán respuestas del tipo "Si" o "No" sin la explicación correspondiente.

Pregunta 1 (3 pts)

¿Por qué es importante la implementación del archivo README.md en su entrega? ¿Qué utilidad le puede dar a alguien que no estuvo involucrado con el proyecto desde el inicio?

Pregunta 2 (3 pts)

¿Cuál es la utilidad de implementar un archivo MAKEFILE en su entrega? ¿Qué utilidad tiene para otras personas? ¿Es posible usarlo en más lenguajes de programación? ¿Cuáles? ¿Es necesario tener consideraciones especiales?

Pregunta 3 (3 pts)

¿Qué utilidad tiene utilizar tecnologías como git, en especial una plataforma como GitHub como repositorio para su proyecto? ¿De qué manera agrega valor al desarrollo en conjunto de 3 ó más personas?

Pregunta 4 (3 pts)

¿Es útil documentar las funciones que usted implementa en su código? ¿Por qué? ¿De qué manera impacta no documentar el código?

Pregunta 5 (3 pts)

¿Cuál es la relevancia de la utilización de estas cuatro herramientas/tecnologías/métodos para su proyecto? ¿Se complementa bien para un mejor desarrollo de su código y que sea más amigable con usuarios externos a su proyecto?

En esta Tarea, el uso de README con markdown, MAKEFILE y GitHub es opcional, pero otorgando bonus a la nota de su entrega. En las siguientes Tareas será obligatorio y sin bonus a la nota.

2.2 Desarrollo

Se pide implementar operaciones que le resultarán de utilidad para entender más del lenguaje `C++`, así como buenas prácticas que le servirán en cursos posteriores. Primero se detallan los requerimientos del código y luego las buenas prácticas que se le pedirá implementar.

Para manejar el despliegue por consola se ha dejado un código de C++ en el repositorio de Github que se encarga de esto, copie la función y la inclusión de la librería requerida

en su propio código. Cualquier cambio a la forma de mostrar los datos por consola será penalizado.

Pregunta 6 (100 pts)

Se pide implementar una función para cada una de las tareas que se especificaron anteriormente, y crear el sistema parche solicitado. Su código debe ser capaz de interactuar con la persona que lo esté usando via consola, para esto deberá crear un código capaz de manejar la opción de entrada de datos. Las opciones posibles de entrada están listadas en la función implementada en **print.cpp** del repositorio compartido. **Es importante** que no todas las funciones a implementar son para interactuar de manera dinámica con la persona que esté usando el programa, las funciones a interactuar son de la 3 en adelante.

Inicialmente su programa debe cargar los datos mediante la función `Leer_Archivo`, para luego eliminar los duplicados mediante la función `Eliminar_Duplicados`. Una vez listo este proceso se despliega el menú de interacción con el usuario, no antes de todo esto. **Por favor seguir el orden solicitado.** Las funciones prototipo a implementar se detallan a continuación:

```
Dato* Leer_Archivo(const string &nombre_archivo){};

void Eliminar_Duplicados(Dato*& datos, int& cantidad_datos){};

void Agregar_Alumno(Dato*& datos, Dato Alumno_nuevo){};

int Calcular_Promedio_Estudiante(Dato* datos, unsigned int id_alumno, char
   * asignatura_a_buscar){};

int* Calcular_Promedio_Asignatura(Dato* datos, char* asignatura_a_buscar)
   {};

void Listar_Cursos_Estudiante(Dato* datos, char* asignatura_a_buscar){};

int* Listar_Reprobados_VTR(Dato* datos, unsigned short VTR_a_buscar, char*
    asignatura_a_buscar){};

void Generar_Informe(Dato* datos){};
```

Las funciones están declaradas en el mismo orden en el que se listaron anteriormente, para evitar confusiones.

Los argumentos de las funciones pueden ser modificados, pero la finalidad de la función no, y recuerde **SIEMPRE** documentar las funciones. Se incluirá un ejemplo de la descripción de una función, usted puede copiarlo o crear/copiar otro formato, siempre y cuando cumpla con la idea de documentar una función y que su propósito sea el de informar.

2.3 Funciones

Se les ha proporcionado dos funciones ya listas, una que permite mostrar los primeros cinco elementos de la colección y otra que permite mostrar el menú de interacción. Deben usar estas funciones y no modificarlas.

La figura 2 muestra un ejemplo de despliegue de la función `Mostrar_Datos_Alumnos`. Los datos mostrados no corresponden a datos de datos.dat.

ID	NOMBRE	CURSO	FECHA NAC.	VTR	NOTA 1	NOTA 2	NOTA 3
259	FRANCISCO JAVIER GOMEZ	INF221	1998/05/06	2	51	27	80
1079	MARIA ISABEL FERNANDEZ	INF221	2000/05/27	1	46	10	65
216	JAVIER ALVAREZ	INF221	1999/02/13	1	37	51	95
80	FRANCISCO LOPEZ	INF221	2005/08/23	3	23	46	93
116	JUAN ALVAREZ	INF221	2002/07/11	2	53	97	67

Figura 2: Ejemplo de despliegue de datos de la función `Mostrar_Datos_Alumnos`

La figura 3 muestra el menú de interacción con el usuario:

```
1) Agregar un alumno.
2) Obtener promedio de un estudiante.
3) Obtener por evaluación de una asignatura.
4) Obtener los cursos en los que está inscrito un estudiante.
5) Obtener los reprobados de un curso con cierto VTR.
6) Generar informe de los cursos.
0) Salir del programa.
Ingrese su opción:
```

Figura 3: Ejemplo de despliegue de datos de la función `Menu`

2.3.1 `Leer_Archivo`

Esta función debe ser capaz de cargar el archivo **datos.dat**. Nótese que recibe como argumento el nombre del archivo. En este caso **el argumento no debe ser modificado**. Para cargar deberán usar memoria dinámica para el arreglo/puntero a crear. No usar memoria dinámica significa penalización en la nota. El retorno corresponde entonces al puntero al arreglo de los datos.

2.3.2 `Eliminar_Duplicados`

La función debe ser capaz de eliminar **todos** los datos duplicados contenidos en el parámetro `datos`. Considere que, la data duplicada **SIEMPRE** está adyacente y se considera data duplicada a datos que coinciden en los campos **Id** y **Curso** de la struct Dato. Como notará, esta función implica un cambio en el tamaño del arreglo, por lo que usted deberá re-dimensionar con memoria dinámica el arreglo.

2.3.3 `Agregar_Alumno`

Se debe solicitar por consola los campos requeridos para llenar la struct. Es su deber implementar la interacción mostrada en la figura 4 sin cambios. El prototipo de la función

```
Ingrese los datos del estudiante:
ID (Número mayor a 1500): 1600
Primer nombre y primer apellido: Balatro Balatrez
Fecha de nacimiento (formato YYYY/MM/DD): 2024/02/24
Notas de las 3 evaluaciones (separadas por espacio): 50 60 72
Sigla del curso (ejemplo: INF134): INF134
VTR (1-3): 1

Alumno ingresado con éxito.
```

Figura 4: Ingreso de alumno

```
Ingrese los datos del estudiante:
ID (Número mayor a 1500): 1500
La Id ingresada no cumple el requisito.
```

Figura 5: Id no cumple requisito

`Agregar_Alumno` puede modificarse dependiendo su implementación. En esta función también deberá re-dimensionar el arreglo con memoria dinámica. Además, al ingresar un alumno nuevo se debe revisar que la inserción no cree un par id-curso duplicado. Si sucede, el programa deberá indicar que no fue posible insertar con un mensaje por consola como el que se muestra en la figure 6. En este caso, si ya se ingresó a Balatro Balatrez con Id: 1600, al Curso: BLT100, y se trata de ingresar otro alumno con la misma información esto correspondería a un caso de duplicado.

Por simplicidad considere que la id a ingresar del nuevo alumno debe ser **siempre** un valor mayor a 1500, esto corresponderá a alumnos nuevos que no existan ya en los datos que usted tiene. La figura 5 muestra un ejemplo de interacción.

Solo la Id y el Curso son considerados al comparar. Los demás campos de la struct no se revisan al comparar.

```
Ingrese los datos del estudiante:
ID (Número mayor a 1500): 1600
Primer nombre y primer apellido: Balatro Balatrez
Fecha de nacimiento (formato YYYY/MM/DD): 2024/02/24
Notas de las 3 evaluaciones (separadas por espacio): 50 60 72
Sigla del curso (ejemplo: INF134): BLT100
VTR (1-3): 1

Alumno ya ingresado en ese curso, no es posible ingresarlo otra vez.
```

Figura 6: Ingreso duplicado

Como puede notar, se usó la asignatura BLT100, esta asignatura no existe en el archivo proporcionado. No es relevante para el programa si la asignatura existe o no.

2.3.4 `Calcular_Promedio_Estudiante`

En esta función, el prototipo puede cambiar, pero recuerde que la interacción con el usuario no debe verse afectada por la elección de implementación. Lo solicitado es buscar el alumno por Id y curso en el arreglo datos, calcular su promedio de notas (promedio aritmético de las tres evaluaciones) y retornar el valor de este promedio calculado. Note que el retorno es de tipo int, esto no lo debe cambiar. Debe redondear el valor resultante del promedio, este valor será de tipo int. La función se encarga de calcular el promedio más no de imprimir su valor.

A continuación se listan ejemplos de interacción. La idea es que tenga una noción de como se debe comportar el programa. Recordar no cambiar la manera de interacción, usted debe escribir su código para que esta interacción por consola sea **igual**.

```
Ingrese los datos del estudiante:
ID: 1600
Curso: BLT100
El promedio del estudiante es: 61
```

```
Ingrese los datos del estudiante:
ID: 1650
Curso: BLT100
No existe un estudiante de ID 1650 en el curso BLT100.
```

```
Ingrese los datos del estudiante:
ID: 1600
Curso: BLT350
No existe el curso BLT350.
```

```
Ingrese los datos del estudiante:
ID: 1650
Curso: BLT350
No existe un estudiante de ID 1650 en el curso BLT350.
```

2.3.5 `Calcular_Promedio_Asignatura`

Esta función deberá calcular el promedio de cada evaluación para una asignatura ingresada por consola. Retorna un puntero/arreglo con el promedio para cada evaluación en sus respectivos índices. Los promedios son de tipo int y no float, por lo que deberá redondear el resultado del promedio por evaluación. El arreglo de notas de la struct contiene la nota del alumno para cada evaluación, siendo el índice 0 el que corresponde a la nota del estudiante en la evaluación 1 de la asignatura correspondiente. Una vez calculado y entregado el arreglo/puntero usted deberá imprimir por consola el resultado.

Considerando el conjunto de datos de la figura 7. Este modo de despliegue es solo un ejemplo, está adaptado para ser más legible en este documento. La función de despliegue

```
ID   | NOMBRE             | CURSO  | FECHA NAC. | VTR | N1 | N2 | N3
259  | FRANCISCO GOMEZ    | INF221 | 1998/05/06 |  3  | 51 | 27 | 80
1079 | MARIA FERNANDEZ    | INF221 | 2000/05/27 |  1  | 46 | 10 | 65
216  | JAVIER ALVAREZ     | INF221 | 1999/02/13 |  1  | 37 | 51 | 95
80   | FRANCISCO LOPEZ    | INF221 | 2005/08/23 |  3  | 23 | 46 | 93
116  | JUAN ALVAREZ       | INF221 | 2002/07/11 |  2  | 53 | 97 | 67
```

Figura 7: Ejemplo de datos

```
El promedio por evaluación de la asignatura INF221 es: 42 46 80
```

Figura 8: Ejemplo despliegue

del programa se presenta en la figura 2. El resultado esperado por consola se presenta en la figura 8. Este formato de salida es obligatorio.

2.3.6 `Listar_Cursos_Estudiante`

La función deberá solicitar por consola la Id del estudiante a buscar y mostrar por consola el listado de todas las asignaturas donde está presente el estudiante.

```
Ingrese el Id del alumno a buscar: 1234
Las asignaturas inscritas por el estudiante son: INF134 INF152 FIS110
```

```
Ingrese el Id del alumno a buscar: 1650
No existe registro para el estudiante con Id 1650.
```

2.3.7 Lista_Reprobados_VTR

La función debe retornar la cantidad de alumnos reprobados con un VTR en específico para un curso en específico. Además de retornar las Ids de los alumnos que cumplan con el requisito. Considerando el conjunto de datos de la figura 9, las figuras 10 y 11 muestran dos posibles interacciones.

En caso de que existan más de 1 reprobados, entonces se imprime cada id separada por un espacio en blanco.

```
ID   | NOMBRE             | CURSO  | FECHA NAC. | VTR | N1 | N2 | N3
259  | FRANCISCO GOMEZ    | INF221 | 1998/05/06 |  3  | 51 | 27 | 80
1079 | MARIA FERNANDEZ    | INF221 | 2000/05/27 |  1  | 46 | 10 | 65
216  | JAVIER ALVAREZ     | INF221 | 1999/02/13 |  1  | 37 | 51 | 95
80   | FRANCISCO LOPEZ    | INF221 | 2005/08/23 |  3  | 23 | 46 | 93
116  | JUAN ALVAREZ       | INF221 | 2002/07/11 |  2  | 53 | 97 | 67
```

Figura 9: Ejemplo de datos

```
Ingrese el VTR: 1
Ingrese el curso: INF221
Cantidad de reprobados: 1
Ids: 1079
```

Figura 10: Ejemplo interacción

```
Ingrese el VTR: 2
Ingrese el curso: INF221
Cantidad de reprobados: 0
Ids:
```

Figura 11: Ejemplo interacción

Nótese que debe retornar la cantidad y el listado de reprobados. La cantidad de estudiantes reprobados corresponde al primer campo del arreglo retornado, y luego deben ir las ids de cada estudiante en dicha condición.

2.3.8 `Generar_Informe`

Deberá crear un archivo de salida por cada curso disponible en los datos. Por ejemplo, para el curso INF221, se creará un archivo *INF221.txt*. Cada uno de estos archivos debe contener la siguiente información y en el orden que se especifica a continuación:

1. Información de cada estudiante inscrito, y su promedio final.
2. Cantidad de estudiantes del curso.
3. Cantidad de aprobados y reprobados por evaluación.
4. Porcentaje de aprobación del curso (considerando promedio aritmético de las tres evaluaciones).
5. Nota más alta por evaluación.
6. Nota más baja por evaluación.
7. Cantidad de estudiantes con VTR = 3 que reprobaron la asignatura.
8. Ids de los estudiantes que cumplan el punto 7.

A continuación se presenta un ejemplo de despliegue de Informe considerando los datos de la figura 9.

```
259 FRANCISCO GOMEZ INF221 1998/05/06 3 51 27 80 53
1079 MARIA FERNANDEZ INF221 2000/05/27 1 46 10 65 40
216 JAVIER ALVAREZ INF221 1999/02/13 1 37 51 95 61
80 FRANCISCO LOPEZ INF221 2005/08/23 3 23 46 93 54
116 JUAN ALVAREZ INF221 2002/07/11 2 53 97 67 72
5
Aprobados: 2
Reprobados: 3
Porcentaje de aprobación: 40%
Maximos: 53 97 95
Mínimos: 23 10 65
VTR3 reprobados: 2
Ids: 259 80
```

En caso de que no existan VTR3 reprobados, el Informe deberá desplegar las dos últimas líneas como se muestra a continuación.

```
VTR3 reprobados: 0
Ids:
```

3 Formato de Entrega

Se debe entregar vía aula un archivo llamado `tarea1-apellido1-apellido2.zip`[2] que contenga:

1. `main.cpp` : El archivo que contiene su código principal.
2. Un archivo README.md (opcional en esta Tarea que sea md, sino debe ser txt), que contenga los integrantes, así como **TODA** la información relevante de su proyecto, como características técnicas, tales como sistema operativo, versión de `C++`, versión del compilador, etc. Características de funcionamiento, como funciona su código, como se compiló su código, etc. En caso de escribir el archivo README con markdown (.md) debe estar escrito con las sintaxis correcta, el link a continuación contiene la sintaxis básica de este tipo de archivos. `https://www.markdownguide.org/basic-syntax/`.
3. Un archivo MAKEFILE (opcional en esta Tarea) que contenga los comandos de compilación, borrado y ejecución para su código. Debe contener el comando make, make clean y make run. `https://makefiletutorial.com/`
4. Un archivo Repositorio.txt (opcional en esta Tarea) que contenga la URL al repositorio de GitHub donde está su tarea. Se revisará que la entrega y el repositorio contengan lo mismo, así también como que el último commit/push también cumpla con el plazo de entrega. Es importante que su repositorio sea privado e incluya a su compañero/a y a los ayudantes como colaboradores, nadie más puede tener acceso aparte de los mencionados, si es público se arriesga a que su nota sea 0. Se pide al menos tres commit por cada miembro al git.

[2]apellido1 y apellido2 corresponden al primer apellido de cada integrante.

4 Directrices de programación

Las directrices corresponden a buenas prácticas de programación, las cuales serán tomadas en consideración para la evaluación de la tarea. Si el código fuente está desordenado, se pueden descontar hasta 20 puntos de la nota.

Cada función de código cuya función no sea inmediatamente aparente, debe tener un comentario de la siguiente forma:

```
/*****
*   TipoFunción NombreFunción
******
*   Resumen Función
******
*   Input:
*       tipoParámetro NombreParámetro : Descripción Parámetro
*        .......
******
*   Returns:
*       TipoRetorno, Descripción retorno
*****/
```

Además,

- Se deben utilizar nombres de variables descriptivos.
- Se deben seguir buenas prácticas respecto al uso de funciones. Es decir, se deben utilizar funciones para evitar redundancia y para modularizar las componentes de su código.
- Se debe considerar la indentación correcta del código.

5 Contacto

Profesores

- Elizabeth Montero (**Coordinadora**) (elizabeth.montero@usm.cl)
- José Miguel Cazorla (jcazorla@usm.cl)
- Ricardo Salas (ricardo.salas@usm.cl)
- Juan Pablo Castillo (juan.castillog@sansano.usm.cl)

Ayudante Coordinador

- Juan Daniel Alegría (**Coordinador**) (juan.alegria@usm.cl)

Ayudantes (CC)

- Franco Cerda (franco.cerda@usm.cl)
- Bastián Jiménez (bjimenez@usm.cl)

Ayudantes (SJ)

- Damián Rojas (damian.rojas@usm.cl)
- Javier Matamala (javier.matamalag@usm.cl)