

# Tarea 1

Profesores:

Diego Arroyuelo (darroyue@usm.cl)  
Natalia González (natalia.gonzalezg@usm.cl)  
Roberto Díaz (robertodiazurra@gmail.com)

Ayudantes:

Gabriel Carmona (gabriel.carmonat@sansano.usm.cl)  
María Paz Morales (maria.moralesll@sansano.usm.cl)  
Abdel Sandoval (abdel.sandoval@sansano.usm.cl)  
Alejandro Vilches (alejandrovilches@sansano.usm.cl)

Fecha de entrega: 19 de abril, 2019  
Plazo máximo de entrega (atrasos): 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. No se permiten de ninguna manera grupos de más de 3 personas. Las tareas deben compilarse en los computadores que se encuentran en el laboratorio LDS – B-032 (Fedora 17 de 64 bits). Debe usarse el lenguaje de programación C. Se recomienda compilar en el terminal usando `gcc archivo.c -o output -Wall`. 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 más importantes, se encuentran:

- Paso de parámetros por valor.
- Paso de parámetros por referencia.
- Asignación de memoria dinámica.
- Manipulación de punteros.
- Manejo de E/S.

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 requiere que implementen varias funciones en C. Cada una de éstas debe estar en archivos `.c` separados en su entregable, con la correspondiente función `main`.

#### Problema 1: Combinación de Archivos

Se requiere la implementación del programa `grupos_de_trabajo.c`, el cual obtenga desde un archivo binario `alumnos.dat` los datos de los alumnos de la universidad, y pueda crear grupos de trabajo para cada una de las asignaturas donde se requiera, respetando el número de integrantes por grupo que solicita cada ramo, indicado en el archivo binario `cursos.dat`.

Asuma las siguientes definiciones para los structs `curso` y `alumno`.

```
typedef struct {
    int id_curso;
    char sigla_curso[7];
    char nombre_curso[30];
    int integrantes_por_grupo;
} curso;

typedef struct {
    char rol_estudiante[12];
    char carrera[4];
    char nombre_completo[41];
    int numero_cursos;
    int id_cursos_inscritos[50];
} alumno;
```

Donde las siglas de cada curso están denotadas por 3 caracteres y 3 números, por ejemplo INF134.

`curso.id_curso` corresponde a un entero identificador del curso y `alumnos.id_cursos_inscritos` a los identificadores de los cursos inscritos por el alumno.

#### Formato de Entrada

La entrada de datos será a través de dos archivos: `alumnos.dat` y `cursos.dat`. El archivo `alumnos.dat` estará en **formato binario** y tendrá la siguiente estructura: un entero  $n$ , seguido por  $n$  `struct alumno`. El archivo `cursos.dat` también estará en **formato binario**, y contendrá un entero  $m$  seguido por  $m$  `struct curso`.

#### Formato de Salida

Para cada curso del archivo `cursos.dat` se debe generar un archivo `grupos-<curso>.txt` donde en `<curso>` se debe reemplazar por la sigla del curso en cuestión.

El número de integrantes por curso está indicado por el campo `integrantes_por_grupo` en el `struct curso`, el cual será un número  $\leq 5$ . Los grupos deben ser armados aleatoriamente y distintas ejecuciones del programa deberían poder entregar una combinación distinta de integrantes de los grupos. Si el número de alumnos en un curso no es un múltiplo del número `integrantes_por_grupo`, se deben dejar los alumnos sobrantes en un grupo de menor tamaño. Si, por otro lado, `integrantes_por_grupo` es un número menor o igual a cero, el curso no requiere grupos y no se debe generar su archivo de grupos correspondiente.

El formato de los archivos de salida debe ser ASCII, y su estructura debe ser: en la primera línea debe aparecer la sigla y el nombre del curso, seguido por una línea vacía, y luego los grupos, donde cada grupo debe estar numerado y seguido por los nombres de los alumnos en el grupo seguido por una línea vacía. Por ejemplo si el curso INF134 solicitaba grupos de 2 integrantes, el archivo `grupos-INF134.txt` debe ser de la forma:

Grupo 1:

Alejandro Carmona

Maria Paz Sandoval

Grupo 2:

Abdel Vilches

Nicolas Gonzalez

Grupo 3:

Gabriel Morales

### Consideraciones

Los archivos de prueba de entrada para ‘`alumnos.dat`’ y ‘`cursos.dat`’ se encontraran en la plataforma Aula. Sin embargo, se recomienda generar y probar con otros archivos el correcto funcionamiento del programa.

### Problema 2: Prefijo Común Más Largo

Dado un string  $z[1\dots n]$ , se define como prefijo de  $z$  a todo string  $z[1\dots i]$ , tal que  $i = 1, \dots, n$ . Por ejemplo, para  $z[1\dots 11] = \text{mississippi}$ , tenemos 11 prefijos posibles: `m`, `mi`, `mis`, `miss`, `missi`, `missis`, `mississ`, `mississi`, `mississip`, `mississipp`, y `mississippi`.

Dados dos strings  $z$  y  $w$ , el prefijo común mas largo entre ellos es el string  $v$  más largo tal que  $v$  es prefijo tanto de  $z$  como de  $w$ . Por ejemplo, dados los strings `mississippi` y `miseria`, el prefijo común más largo entre ambos es `mis`, de largo 3.

Una aplicación para detectar plagios en textos necesita la siguiente funcionalidad. Dado un conjunto de strings, se quiere encontrar aquel par de strings del conjunto que tenga el prefijo común más largo. Implementar la función `int pCML (char **S, int n)`, la cual recibe como parámetro un conjunto  $S$  de  $n$  strings, y devuelve el mayor entero  $x$  tal que dos strings  $z$  y  $w$  en  $S$  tienen prefijo común más largo de longitud  $x$ . Por ejemplo, dado el conjunto de strings `flor`, `hola`, `fracción`, `flotante`, la función debe retornar 3, el cual corresponde al prefijo `flo` entre los strings `flor` y `flotante`.

Dado que la función se usará para procesar conjuntos grandes de strings (hasta 10 millones de strings), es importante que su solución sea eficiente. Una solución ineficiente en cuanto a tiempo de ejecución (por ejemplo, una que chequee todo los posibles pares de strings para buscar el máximo) podría ser evaluada con el 50 % de los puntos. Para la implementación, se pueden usar funciones de manipulación de strings y/o alguna otra función necesaria de la librería standard de C.

### Formato de Entrada

La entrada de datos será a través del archivo ‘`strings.txt`’, el cual contiene un conjunto de strings a ser procesados. El archivo tiene un string por línea, y es terminado por EOF. La longitud máxima de cada string en el archivo es 200 chars, y puede haber hasta 10 millones de strings en un archivo.

Por ejemplo:

```
flor
hola
fracción
flotante
```

## Formato de Salida

Debe imprimirse en el archivo ‘`salida-1.txt`’ el resultado de invocar a la función `pCML` con los datos provistos en el archivo de entrada.

La salida correspondiente al ejemplo anterior es la siguiente:

3

## Consideraciones

Se recomienda procesar los datos del archivo `palabras.dat`, de tal manera que la respuesta a las consultas sea rápida. Es tarea de cada grupo desarrollar una manera de procesar los datos. Para el procesamiento se permite usar algoritmos adicionales de las librerías standard de C. Las 8 tareas que resuelvan este problema con menor tiempo de ejecución, recibirán un bonus de 15 puntos. Dicho bonus podrá ser usado en cualquier otra tarea del curso durante el semestre 2018-1. Aquellos grupos que resuelvan el problema de forma notoriamente lenta, pueden llegar a recibir descuentos.

## 4. Entrega de la Tarea

La entrega de la tarea debe realizarse enviando un archivo comprimido llamado

`tarea1-apellido1-apellido2-apellido3.tar.gz`

(reemplazando sus apellidos según corresponda) al moodle del curso, a más tardar el día 19 de abril, 2019, a las 23:55:00 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.

## 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 deben compilar en los computadores que se encuentran en los laboratorios LDS (Fedora 17 de 64 bits). **Las tareas que no compilen no serán revisadas y serán calificadas con nota 0.**
- 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.**