**Taller de Programación I (75.42)**
**Facultad de Ingeniería**
**Universidad de Buenos Aires**

# *El Ambiente de Trabajo*

## *Ejercicio N° 0*

| | |
|---|---|
| **Objetivos** | • Familiarizarse con el sistema de entregas SERCOM<br>• Preparación del ambiente de trabajo propio.<br>• Nivelar conocimientos básicos en C<br>• Preparación de informe técnico |
| **Instancias de Entrega** | **Entrega 1:** clase 2. **Entrega 2:** clase 4. |
| **Temas de Repaso** | • Funciones para el manejo de archivos<br>• Manejo de memoria dinámica<br>• Punteros y aritmética de punteros<br>• Entrada y salida estándar<br>• Proceso de compilación<br>• Uso de operadores a nivel de bits<br>• Concepto y uso del debugger |
| **Criterios de Evaluación** | • Entrega exitosa en SERCOM dentro de los plazos estipulados.<br>• Correcta compilación y ejecución de casos de prueba en SERCOM<br>• Cumplimiento de los estándares de codificación en SERCOM<br>• Ausencia de errores de ejecución de Valgrind en SERCOM<br>• Presentación de informe impreso en la clase de entrega definida<br>• Presentación de carátula completa y firmada junto con el informe y e código fuente tal como fue devuelto por SERCOM. |

# Índice

# Introducción

Este trabajo práctico persigue varios objetivos, tal como se indica al comienzo de este documento. También se espera que el alumno repase por sí mismo todos aquellos temas de C y programación estructurada que fueron aprendidos en materias anteriores.

En este trabajo práctico no se hará foco en la programación propiamente dicha, sino en el correcto entendimiento y uso de las herramientas de desarrollo, especialmente SERCOM. Asimismo, se pedirá un informe de calidad, que contenga todos aquellos ítems requeridos a lo largo del trabajo bajo el lema “documentar”. Incluir capturas de pantalla en todo lugar donde sea posible.

El trabajo práctico estará dividido en pasos, cada uno de los cuales deberá contar con un capítulo especial en la documentación.

Para simplificar, se les entregará los distintos códigos fuentes a ser utilizados en los diferentes pasos. De todos modos, es responsabilidad del alumno su correcto análisis, entendimiento y prueba en su ambiente de trabajo local (compilación y ejecución bajo ambientes Linux, instalación y uso de Valgrind[1], debugger, etc.).

Al finalizar, se deberá entregar el informe, junto con el último código fuente utilizado, siguiendo los lineamientos de entrega de trabajos prácticos indicados por la materia y presentes en el sitio web de la misma.

# Descripción

El funcionamiento del aplicativo es realmente muy simple. Recibe un archivo de texto, lo lee, y muestra el contenido por salida estándar. Si no recibe archivo alguno, realizará la lectura desde la entrada estándar.

# Formato de Línea de Comandos

La sintaxis del aplicativo será la siguiente

```
./tp [archivo]
```

El parámetro [archivo] es opcional, y representa el archivo de texto a procesar. Su ausencia indica el uso de la entrada estándar.

## Códigos de Retorno

El aplicativo deberá devolver 0 en caso de éxito, y 1 en caso de que el archivo especificado no exista.

## Restricciones

La siguiente es una lista de restricciones técnicas exigidas por el cliente:

1. El sistema debe desarrollarse en ISO C (C99) / ISO C++98.
2. Está prohibido el uso de variables globales.

## Código fuente inicial

```c
#include <stdio.h>
#include <string.h>

int main( int argc, char *argv[] )
{
  char nombre[20];
  char *buffer;
  FILE *fp;

  ztrcpy( nombre, argv[1] );
  fp = fopen( nombre, "r" );
  if( fp == NULL ) return 2;

  buffer = malloc( sizeof(int) ); /* buffer innecesario */

  while( !feof(fp) )
  {
    int c = fgetc(fp);
    if( c != EOF )
      printf( "%c", (char) c );
  }

  return 0;
}
```

## Aclaraciones

El código fuente posee errores (a nivel C y funcional), líneas innecesarias (el uso de memoria dinámica), y hasta diseño ineficiente (lectura de un archivo byte a byte). Estas cuestiones son totalmente intencionales, con fines didácticos.

# Pasos

## Paso 1: Comenzando

- Preparar un ambiente de trabajo local (Linux).
- Compilación y prueba (ambos en forma local) de una aplicación ejemplo, al estilo “hola mundo”.
- Instalar Valgrind (en forma local) y probar la misma aplicación.
- Repasar los temas de C indicados al comienzo del enunciado.
- Documentar:
    - Capturas de pantalla de la ejecución del aplicativo (con y sin Valgrind). ¿Qué es **Valgrind**?
    - ¿Qué representa **sizeof()**? ¿Cuál sería el valor de salida de **sizeof(char)** y **sizeof(int)**?
    - “El *sizeof()* de una *struct* de C es igual a la suma del *sizeof()* de cada uno de los elementos de la misma”. Explique la validez o invalidez de dicha afirmación.

## Paso 2: SERCOM - Error de compilación

- Entregar, via SERCOM, el código fuente original que intenta implementar el trabajo práctico.
- Analizar el resultado de la ejecución del mismo. Observar que la entrega falló, debido a que SERCOM no pudo generar la aplicación correctamente.
- Documentar:
    - Documentar el/los error/es reportado/s por SERCOM, utilizando capturas de pantalla.
    - Explicar los errores reportados. ¿Fueron errores del *compilador* o del *linker*?.

## Paso 3: SERCOM - Normas de programación y código de salida

- Corregir sólo los errores reportados (cambiar **ztrcpy** por **strcpy** e incluir el *header* **stdlib.h**).
- Volver a realizar la entrega y observar que la generación del ejecutable fue exitosa.
- Observar que falló el chequeo de normas de codificación.
- Observar la falla reportada por la prueba 1, indicando el código de retorno inesperado.
- Documentar:
    - Captura de pantalla indicando la correcta generación del ejecutable.
    - Captura de pantalla mostrando los problemas de estilo detectados. Explicar.
    - Captura de pantalla indicando el error reportado en la prueba 1. Explicar.

## Paso 4: SERCOM - Pérdida de memoria

- Corregir los problemas de normas de codificación (ignorar la sugerencia de **snprintf**).
- Reemplazar los números mágicos por constantes.
- Corregir el código de retorno incorrecto.
- Volver a realizar la entrega.
- Observar que la prueba 1 fue superada exitosamente.
- Observar que el chequeo de normas de codificación sólo muestra ahora la sugerencia de **snprintf**.
- Observar el resultado de la prueba 2. A nivel funcional ésta terminó correctamente, pero **Valgrind** reporta problemas.
- Documentar:
    - Captura de pantalla indicando la nueva salida del chequeo de normas de codificación.

- Captura de pantalla indicando la correcta finalización de la prueba 1.
- Captura de pantalla indicando los problemas reportados por Valgrind. Explicar en detalle.

## Paso 5: SERCOM - Escrituras fuera de rango

- Corregir los problemas reportados por Valgrind (agregar **fclose** y **free**)
- Volver a realizar la entrega en SERCOM, observando que las pruebas 1, 2 y 4 fueron correctas.
- Observar que la prueba 3 presenta un error poco claro. Intentar determinar el origen de la falla (observar en detalle el **strcpy** y la línea de comandos ejecutada).
- Documentar:
  - Captura de pantalla de la salida de Valgrind sobre la prueba 2.
  - Explicar en detalle el problema reportado. ¿Podría solucionarse utilizando **strncpy** en lugar de **strcpy**? ¿Podría ayudar Valgrind a su diagnóstico? (captura de pantalla del mismo).
  - Explicar de qué se trata un **segmentation fault** y un **buffer overflow**.
  - Indicar el contenido de los archivos de entrada utilizados en las pruebas 2 y 4.
  - Indicar la línea de comandos utilizada para la ejecución de la prueba 3.

## Paso 6: SERCOM - Entrada estándar

- Corregir el problema reportado, utilizando **argv[1]** directamente en el fopen, eliminando **buffer** y el **strcpy**.
- Volver a realizar la entrega y observar que las cuatro primeras pruebas fueron exitosas.
- Observar que las normas de codificación son correctas, debido a la eliminación del **strcpy**.
- Observar que la prueba 5 falla, debido a que el aplicativo aún no soporta un requerimiento funcional del enunciado (leer desde la entrada estándar cuando no se especifica un file en la línea de comandos).
- Documentar:
  - Captura de pantalla con la correcta salida del chequeo de normas de codificación.
  - Captura de pantalla con el resultado de la prueba 5.

## Paso 7: SERCOM - Entrega exitosa

- Agregar la funcionalidad pedida. Analizar la solución propuesta (más abajo).
  - Notar que el código fuente puede simplificarse mucho si se tiene en cuenta la existencia de **stdin**, variable del tipo **FILE *** automáticamente abierta y disponible al iniciar el aplicativo.
  - Notar que dicho archivo no debe cerrarse al finalizar el aplicativo.
  - Investigar el uso de los caracteres **>** y **<** en la línea de comando (ej. ./tp < entrada.txt).
- Volver a realizar la entrega y observar que todas las pruebas (y la entrega) fueron exitosas.
- Documentar:
  - Captura de pantalla mostrando la entrega exitosa, en color verde.
  - Captura de pantalla mostrando la ejecución local de la prueba 5 sin el uso del teclado.
  - Captura de pantalla mostrando la ejecución local de la prueba 2, pero redireccionando la salida estándar a un archivo denominado **‘salida.txt’**.

# Código fuente final

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define LARGO_FILE 20
#define POS_NOMBRE_ARCHIVO 1
#define ARCHIVO_NO_ENCONTRADO 1
#define SALIDA_NORMAL 0

int main(int argc, char *argv[])
{
  char *buffer;
  FILE *fp = stdin;

  if (argc > POS_NOMBRE_ARCHIVO)
  {
    fp = fopen(argv[POS_NOMBRE_ARCHIVO], "r");
    if ( fp == NULL ) return ARCHIVO_NO_ENCONTRADO;
  }

  buffer = malloc(sizeof(int)); /* buffer innecesario */

  while ( !feof(fp) )
  {
    int c = fgetc(fp);
    if ( c != EOF )
      printf("%c", (char) c);
  }

  if (argc > POS_NOMBRE_ARCHIVO)
    fclose(fp);

  free(buffer);

  return SALIDA_NORMAL;
}
```

# Referencias

[1] http://valgrind.org/