Universidad Tecnológica Nacional • Facultad Regional Buenos Aires
Ingeniería en Sistemas de Información
Sintaxis y Semántica de Lenguajes
Director de Cátedra **Jorge Daniel Muchnik**

Trabajo Práctico #3

por **Jorge Daniel Muchnik & José María Sola**

1.0.0.20041106

Objetivos

- Aplicar en ANSI C el concepto de *Argumentos en Línea de Comando (Programa Comando).*
- Realizar un procesamiento básico de archivos binarios y de texto.
- Aplicar el concepto de registro de longitud variable.
- Diseñar la especificación y la implementación en ANSI C del TAD *Condición Académica de un Estudiante (CAE),* en el contexto de una *Universidad Nacional.*
- Construir la implementación del TAD CAE en ANSI C y generar una biblioteca que la contenga.
- Construir en ANSI C un programa comando que, en base a sus argumentos en línea de comando, procese archivos de texto y binarios haciendo uso del TAD CAE.
- Capturar las salidas de los procesos de traducción y del Programa Comando.

Situación

La Universidad Nacional en cuestión es un organismo de grandes proporciones. Los diferentes departamentos de procesamiento informático distribuidos en las diferentes facultades y dependencias de la universidad conforman un ambiente heterogéneo.
Es por eso que, para el procesamiento de la información que compone la *condición académica* de los estudiantes, se necesita cumplir ciertos requerimientos, por un lado, la *portabilidad* y *compatibilidad* de la información compartida entre los diferentes sectores (i*nter-sector*) y, por otro, la *eficiencia* tanto en volumen como en tiempo de procesamiento dentro de un mismo sector (*intra-sector*).
Para resolver esta problemática se debe proporcionar un mecanismo común, que pueda ser utilizado en todos los sectores de la Universidad, que permita convertir la estructuración de la información que compone la condición académica de cada alumno de un modo independiente pero no tan eficiente a un modo más eficiente pero dependiente. Se desarrollará una herramienta que permita que los archivos compartidos con la información de los alumnos sean convertidos de texto a binario y viceversa. Para que la propia herramienta sea *portable*, se la deberá desarrollar en ANSI C.

TAD Condición Académica de un Estudiante -- CAE

Los valores de los CAE son capaces de contener el legajo, los nombres y los apellidos de un alumno, el código de asignatura y la calificación de las últimas de evaluaciones y el promedio general del alumno.
De esta forma, los valores CAE *pueden* definirse como la siguiente 6-upla (*número de legajo*, *nombres*, *apellidos*, *cantidad de evaluaciones*, *últimas evaluaciones*, *promedio general*), donde *últimas evaluaciones* es una secuencia de pares ordenados de la forma (*código de asignatura*, *calificación*). El *código de asignatura* es una palabra de exactamente tres *mayúsculas o números* (por ejemplo AM1 ó AGA y no am2) y la *calificación* es un número natural entre 1 y 10. El n*úmero de legajo* es una código de exactamente 10 caracteres; los n*ombres* y a*pellidos*, cada uno, no superan los 20 caracteres, pero pueden contener espacios para los nombres compuestos y para los apellidos dobles o más complejos; la *cantidad de evaluaciones* es un número natural que indica cuantos pares ordenados tiene la secuencia de evaluaciones; y por último, el *promedio general* es un número real entre 1 y 10.
Ejemplo: (*"5469817221", "Ritchie", "Dennis M.", 2,* ((*"AED", 9*), (*"SSL", 10*)), *9.5*).
La Universidad en cuestión considera un *promedio general aprobado* a aquel que es igual o mayor a cuatro.
Dentro del conjunto de valores posibles del TAD CAE, existe un valor especial, conocido como *LaCaeNula*, que sirve para indicar situaciones excepcionales. Como 6-upla, *LaCaeNula* se representa de la siguiente forma (*ε*, *ε*, *ε*, *0*, *∅*, *0.0*).
A modo de *guía* se presenta una base de lo que podrían ser las estructuras que implementen los valores del TAD. Contrastar las diferencias entre la especificación y la implementación de los valores.

```
#define LONG_CODIGO_ASIGNATURA  3
#define LONG_NUMERO_LEGAJO     10
#define MAX_LONG_NOMBRES       20
#define MAX_LONG_APELLIDOS     20

typedef struct {
     char             elCodigoDeAsignatura [ LONG_CODIGO_ASIGNATURA + 1 ];
     unsigned int     laCalificacion;
} CAE_Evaluacion;

typedef struct {
     char             elNumeroDeLegajo [ LONG_NUMERO_LEGAJO + 1 ];
     char             losNombres [ MAX_LONG_NOMBRES + 1 ];
     char             losApellidos [ MAX_LONG_APELLIDOS + 1 ];
     unsigned int     laCantidadDeEvaluaciones;
     CAE_Evaluacion*  lasEvaluaciones;
     float            elPromedioGeneral;
} CAE;

CAE LaCaeNula = { "", "", "", 0, NULL, 0.0 };
```

Operaciones a Especificar e Implementar

1. **`CAE_CrearDesdeFlujoBinario`**. *CrearDesdeFlujoBinario : FlujoBinario → CAE x FlujoBinario*. Utilizar las funciones `malloc` (ó `calloc`) y `fread`.
2. **`CAE_CrearDesdeFlujoDeTexto`**. *CrearDesdeFlujoDeTexto : FlujoDeTexto → CAE x FlujoDeTexto*. Utilizar las funciones `malloc` (ó `calloc`), `fgets`, `atoi`, `strcpy` y `strtok`. La función `strtok` debe ser invocada *cuatro* veces con los –diferentes– separadores **`"\t "`**, **`"\t "`**, **`","`** y **`"\n"`** para la primer línea, y para las líneas restantes, invocarla *dos* veces primero con **`"\t "`** y luego con **`"\n"`** como separadores).
3. **`CAE_Destruir`**. *Destruir : CAE → ∅*. Utilizar función `free`.

Las operaciones de *creación* son dos, una a partir de un *flujo de lectura binario* y otra a partir de un *flujo de lectura de texto*, ambas retornan un nuevo valor del TAD CAE creado a partir de la información proveniente del flujo dato, atendiendo a las diferencias en la estructuración de los datos de cada flujo. Adicionalmente retornan, el mismo flujo dato pero listo para leer el siguiente registro. Ambas operaciones, en el nivel de implementación, peticionarán memoria dinámicamente, la cual será librara en la operación *Destruir*. Si la creación no pudo concretarse, se retorna *LaCaeNula*.

4. **`CAE_EscribirEnFlujoBinario`**. *EscribirEnFlujoBinario : CAE x FlujoBinario → Boolean x FlujoBinario*. Utilizar función `fwrite`.
5. **`CAE_EscribirEnFlujoDeTexto`**. *EscribirEnFlujoDeTexto : CAE x FlujoDeTexto → Boolean x FlujoDeTexto*. Utilizar función `fprintf`.

Ambas operaciones de *escritura* se encargan de producir la secuencia de datos (bytes o caracteres) correcta para la representación correspondiente. Reciben un flujo de escritura, binario o de texto según corresponda, y retornan una copia de ese flujo listo para escribir el siguiente registro, junto con un valor lógico que indica si se pudo realizar la escritura satisfactoriamente.

6. **`CAE_GetPromedioGeneral`**. *GetPromedioGeneral : CAE → Real*.
7. **`CAE_EsPromedioGeneralAprobado`**. *EsPromedioGeneralAprobado : CAE → Boolean*.

Las operaciones 6 y 7 se utilizan para consultar la condición del estudiante.

Organización de los Archivos

Archivos con Formato de Texto

Organizado de a líneas, cada registro de CAE ocupa por lo menos dos. La primer línea comienza con la cantidad de evaluaciones que contiene el registro, seguida por uno o más blancos o tabulados (**`' '`**, **`'\t'`**), por el número de legajo, uno o más blancos o tabulados, el o los apellidos (separados por un espacio), una coma (**`','`**) y el o los nombres (también separados por un espacio). Las restantes líneas contienen las evaluaciones, tres caracteres para el código de la asignatura, uno o más blancos o tabulados, y el número natural para la calificación. Ejemplo con tres registros:

```
2 5469817221 Ritchie,Dennis M.
AED  9
SSL 10
3 4845612979 Pérez García,María Clara
SSL  7
PPP  8
GDD  9
4 6481451937 Einstein,Albert
AM1  9
FI1 10
AM2  9
FI2 10
```

Archivos con Formato Binario

Los CAEs en estos archivos están almacenados en registros de longitud variable, la variabilidad está dada por la cantidad de evaluaciones de cada estudiante. La secuencia de los campos es la siguiente:

...| leg | nom | ape | cant. eval. | cod. asig. | calif. | cod. asig. | calif. | cod. asig. | calif.| ...

El formato de los campos es el siguiente:

...|←-- *Longitud fija* --→ |←-- *Longitud variable* --→ |...
...| 10B | 20B | 20B | unsigned int | 3B | 1B | 3B | 1B | 3B | 1B |...

Notar la inversión de la secuencia nombre-apellido en comparación con los archivos de texto; y que en el caso de las cadenas legajo, nombre, apellido y código de asignatura *no* se almacena el carácter nulo cuando la cadena llegó al máximo.

Siguiendo el ejemplo del archivo de texto, el primer registro, en el cuarto campo contendrá el entero 2 y luego le seguirán dos pares de campos (código de asignatura y calificación); el segundo registro contendrá el entero 3 en el cuarto campo y le seguirán tres pares de campos; por último, el tercer registro contendrá el valor 4 en el cuarto campo y será seguido por cuatro pares de campos.

El Programa Comando – Transformar.c

El Programa Comando *Transformar*, haciendo uso del TAD CAE, recorre un archivo de entrada que posee un determinado formato y genera un archivo de salida en el formato opuesto. A su vez, genera un archivo de texto con un *resumen de los datos procesados*.

La sintaxis *general* del programa comando es la siguiente

`transformar` *sentido-transformación nombre-archivo-resumen nombre-archivo-in nombre-archivo-out*?

- *sentido-transformación* indica el sentido de transformación. Son dos posibles, Texto→Binario y Binario→Texto, debe ser de dos caracteres con la forma "TB" ó "BT" ó cualquier combinación de esas letras en mayúsculas o minúsculas.
- *nombre-archivo-resumen* es el nombre del archivo de texto que contendrá el resumen del procesamiento.
- *nombre-archivo-in* es el nombre del archivo que actuará como entrada.
- *nombre-archivo-out*, opcional, es el nombre del archivo que actuará como salida.

En caso de faltar el argumento *nombre-archivo-out*, el parámetro *nombre-archivo-in* se considerará *nombre-base-archivo* y se construirán los nombres del archivo de entrada y de salida anexando al las extensiones correspondientes al *nombre-base-archivo*. Ver ejemplos.

Formato del archivo resumen

Compuesto por cinco líneas. Las líneas 3 y 4 están precedidas por un tabulado ('\t'). Las letras *n*, *a* y *d* representan números naturales y *x* un número real.

```
Alumnos procesados: n
Promedios generales
     mayores o iguales al nivel de aprobación: a
     por debajo del nivel de aprobación: d
Promedio del lote: x
```

Invocaciones Posibles

Existen tres formas correctas de invocar a *Transformar*.

- La primera, con 4 argumentos, recibe los nombres exactos del archivo de entrada y del de salida.
  `transformar` *sentido-transformación nombre-archivo-resumen nombre-archivo-in nombre-archivo-out* ↵
  Ejemplos:
  `transformar tb totales.lst Catedra1.2004 CatedraUno2004.dat` ↵
  `transformar bt inidicadores.log Homogeneas.b Comunes.t` ↵
- La segunda, con 3 argumentos, *construye* los nombres del archivo de entrada y del de salida anexando las extensiones ".txt" y ".bin" según corresponda.
  `transformar` *sentido-transformación nombre-archivo-resumen nombre-base-archivo* ↵
  Ejemplo que abre el archivo *SegundoSemestre.bin* y genera *SegundoSemestre.txt*:
  `transformar bt Resumen1.txt SegundoSemestre` ↵
  Ejemplo que abre el archivo *Ingenieria.txt* y genera *Ingenieria.bin*:
  `transformar tb Resumen2.txt Ingenieria` ↵
- La última, sin argumentos, despliega un mensaje de ayuda y descripción apropiado.
  `transformar` ↵

Sobre las Estructuración del Programa Comando y las Funciones que Invoca

- La función `main` debe estar correctamente estructurada. *Algunas* de las tareas que esta función debe realizar son:
  1. Validar y procesar los argumentos de línea de comando. Funciones `DesplegarAyuda`, `EsSentidoDeTransformacion` y `DesplegarMensajeDeError`
  2. Si corresponde, crear los nombres de los archivos con las extensiones adecuadas utilzando las funciones `malloc` y `strcpy`.
  3. Abrir los flujos de datos –con el formato correspondiente– y cerrarlos. Un flujo (de texto o binario) desde el archivo de entrada, un flujo (binario o de texto) hacia el archivo de salida y un flujo de texto hacia el archivo resumen. Utilizar las funciones `fopen` y `fclose`. Las variables deberán ser declaradas de la siguiente forma:
  ```
  FILE* laEntrada;
  FILE* laSalida;
  FILE* elResumen;
  ```
  4. Decidir cual *ciclo de transformación* ejecutar, en función del primer argumento de línea de comando, y ejecutarlo. El siguiente *pseudocódigo* sirve como *guía* para la estructuración del ciclo de lectura:
  ```
  CAE unaCae;
  while( ( unaCae = CAE_CrearDesdeFlujoBinario( laEntrada ) ) != LaCaeNula ) {
      if( ! CAE_EscribirEnFlujoDeTexto( unaCae, laSalida ) ) {
          DesplegarMensajeDeError( ErrorDuranteLaEscritura );
          Terminar programa comando;
      }
      CAE_Destruir(unaCae);
  }
  Determinar si se salió del ciclo por un error de lectura, ó por tratar de leer pasado
  el fin de archivo, ó por memoria insuficiente para la CAE, y actuar en consecuencia;
  ```
  5. Determinar la razón de la salida del ciclo de transformación (e.g. *no hay memoria suficiente*, *no se pudo leer desde la entrada* ya sea porque hubo un error o porque se llegó al fin del archivo, y *no se pudo escribir en la salida*). Funciones `ferror` y `feof`.
  6. Informar al usuario, mediante la función `DesplegarMensajeDeError`, de cualquier excepción que haya ocurrido.
- Se deberá desarrollar la función `int EsSentidoDeTransformacion( const char * );` que valida el primer argumento y retorna un valor lógico. Implementar con las funciones `toupper` ó `tolower`.
- Se deberá desarrollar la función `void DesplegarMensajeDeError( Error );` Implementar con llamadas a la función `puts`. El argumento con el cual se invocará a la función es del tipo `Error`. La declaración, ubicada antes de la definición de la función `main`, debe ser similar a la siguiente:
```
typedef enum {
    NoEsSentidoDeTransformacion = 1,
    CantidadDeArgumentosIncorrecta,
    MemoriaInsuficiente,
    ErrorDuranteLectura,
    ErrorDuranteEscritura,
    NoSePudoAbrirElArchivoDeEntrada,
    NoSePudoAbrirElArchivoDeSalida,
    NoSePudoAbrirElArchivoResumen
} Error;
```
- Se deberá desarrollar la función `void DesplegarAyuda( void );` que será invocada cuando el programa comando haya sido iniciado sin argumentos. Función `puts`.

Consideraciones y Restricciones

- Cualquier decisión que el equipo tome que no esté en el enunciado o no esté del todo clara deberá formar parte de la especificación y/o ser agregada como hipótesis de trabajo.
- Prestar atención al tratamiento de las excepciones y errores en el TAD CAE y en el programa comando.
- El TAD posee operaciones que retornan más de un resultado, por lo tanto el diseño de la implementación se hará con parámetros *out* (salida) ó *inout* (entrada-salida). Considerando que ANSI C solo permite el pasaje de parámetros por valor, se deberá hacer uso de punteros para poder implementar parámetros out ó inout.
- Los identificadores de las funciones que implementan las operaciones, de las variables, tipos y demás elementos deben ser los indicados a lo largo de este enunciado.
- Deben usarse las funciones de la biblioteca estándar indicadas a lo largo de este enunciado y cualquier otra función ANSI que facilite la construcción de la implementación.
- El TAD debe implementarse en una biblioteca. El programa comando debe hacer uso de esta biblioteca.
- Luego de ser construidas con la herramienta de desarrollo elegida por el equipo, la implementación de la biblioteca y el programa comando deben ser verificados mediante la herramienta de desarrollo "*Borland C++ Compiler 5.5 with Command Line Tools*" (`BCC32`) configurada tal como dicta la cátedra.
- Los procesos de compilación con `BCC32` no deben emitir *warnings* (mensajes de advertencia) ni, por supuesto, mensajes de error.
- El set de pruebas de las diferentes corridas de *Transformar* debe diseñarse correctamente.

Entrega

Forma

El TP debe presentarse en hojas A4 abrochadas en la esquina superior izquierda, no se debe entregar ni carpetas ni folios. En el encabezado de cada hoja debe figurar el título del TP, el número de curso y los apellidos de los integrantes del equipo. Las hojas deben estar enumeradas en el pie de las mismas con el formato "Hoja n de m". Cada sección del TP debe comenzar en una hoja separada, así tenga una longitud menor a la de una hoja. Por lo tanto, el TP tendrá una longitud mínima de *7* hojas.

Forma de los listados de códigos fuente

El código fuente de cada componente del TP debe comenzar con un comentario encabezado, que actuará como carátula, con todos los datos del equipo de trabajo: curso; legajo, apellido y nombre de cada integrante del equipo y fecha de última modificación. La fuente a utilizar en la impresión debe ser una fuente de ancho fijo (e.g. `Courier New`, `Lucida Console`).

1. TAD CAE

1.1. Especificación

Especificación completa, extensa y sin ambigüedades de los valores y de las operaciones del TAD.

1.2. Implementación

1.2.2. Biblioteca que implementa el TAD

1.2.2.1. Listado de código fuente del archivo encabezado, **CAE.h**.
1.2.2.2. Listado de código fuente del la definición de la Biblioteca, **CAE.c**.

1.2.3. Salidas

Captura impresa de la salida del proceso de traducción (`BCC32` y `TLIB`). Utilizar fuente de ancho fijo.

2. Programa Comando

2.1. Código Fuente

Listado del código fuente de la aplicación de prueba, **Transformar.c**.

2.2. Salidas

2.2.1. Captura impresa de la salida del proceso de traducción (`BCC32`). Utilizar fuente de ancho fijo.
2.2.2. Captura impresa de las salidas de las diferentes corridas del programa comando, *incluyendo listados de los archivos de entrada, salida y resumen*. Utilizar fuente de ancho fijo.

Otros Componentes de la Entrega

3. Copia Digitalizada

CD ó disquette con copia de **solamente** los 3 archivos de código fuente (**CAE.h**, **CAE.c**, y **Transformar.c**). No se debe entregar ningún otro archivo.

4. Formulario de Seguimiento de Equipo

Se debe entregar la copia del Formulario de Seguimiento que posee el equipo.

Tiempo

La entrega se realizará exactamente una semana después de la presentación del enunciado.