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 #2

por **Jorge Daniel Muchnik & José María Sola** 1.2.0.20041030

Objetivos

- Diseñar las especificaciones y las implementaciones en ANSI C de los TADs *Código Ecológico (CE)* y *Lote de Códigos Ecológicos (LCE)*, en el contexto de una institución que realiza de *Investigaciones sobre Ecología*.
- Construir la implementación de ambos TADs en ANSI C y generar una biblioteca por cada TAD.
- Construir en ANSI C una aplicación que utilice y pruebe los *casos más representativos* de ambos TADs, haciendo uso de las dos bibliotecas previamente generadas.
- Aplicar las operaciones de intersección y complemento de AFDs para el diseño de un AFD que formará parte de la implementación del TAD CE.
- Capturar las salidas de la aplicación de prueba de los TADs.

Situación

Una Institución realiza Investigaciones sobre Ecología y una de las tareas que debe efectuar es el procesamiento de lotes de códigos que recibe desde distintas filiales.

Código Ecológico – CE

Los CE representan un código relevado por la filial y de valor semántico para la institución.
Los CEs tienen el formato α `@` β `#`, dónde α comienza con exactamente dos vocales (minúsculas o mayúsculas) y termina con dos o más consonantes en el rango [`B`, `M`] (todas en mayúscula o todas en minúscula), donde `@` es un separador, donde β está formada por una secuencia de tres o más dígitos que no contiene exactamente dos dígitos seis ('`6`') consecutivos y donde `#` es el terminador.
Ejemplo: `AeBBuCMD@436776666898#`
Los valores de los CE deben ser implementados mediante un par ordenado con la cadena que representa al CE y con un número entero que, si es positivo, representa la longitud de la cadena y, si es negativo, actúa como indicador de error.

Operaciones a Especificar e Implementar

1. `CE_Crear`. *Crear: Cadena → CE*. Creación. Adquiere los recursos necesarios para sustentar un CE; realiza las acciones de preparación y de inicialización; haciendo uso de la operación privada *EsCE*, verifica que la cadena dato utilizada para la inicialización represente un CE, reflejando el resultado en el número entero del par ordenado que implementa el valor del CE. Notar que no se establece una precondición en cuanto a la longitud máxima ni de la cadena dato ni del propio CE.
2. `CE_Destruir`. *Destruir: CE → Ø*. Destrucción. Libera cualquier recurso adquirido para el CE durante su operación de creación.
3. `CE_HuboError`. *HuboError: CE → Boolean*. Consulta de error. Accede al indicador de error del CE. Informa si la cadena dato de la operación de creación representaba un CE.
4. `CE_GetComoCadena`. *GetComoCadena: CE → Cadena*. Obtención del CE pero como una cadena, para posteriores tratamientos, tal como el despliegue por pantalla. Notar que no hay una operación específica para desplegar el CE por pantalla, sino, una operación más genérica que es la base para mostrar el CE por pantalla, o para escribirlo en un archivo o para cualquier otro tratamiento. El TAD no provee la operación de despliegue, por lo tanto es una tarea de la aplicación que, para lograrla, hará uso de la operación GetComoCadena. En el caso que el CE dato de la operación contenga un error, es decir, que la operación HuboError sobre ese CE retorne verdadero ¿cómo deberá actuar GetComoCadena? ¿Qué debería retornar?

5. `CE_EsIgualA`. *EsIgualA: CE x CE → Boolean*. Determina si el primer CE es igual al segundo.
6. `EsCE`. *EsCE: Cadena → Boolean*. Validación, *operación privada* que verifica si una cadena representa un CE. El encabezado de la función que implementa esta operación comenzará con la palabra clave `static` para denotar la naturaleza privada de la función. La implementación de esta función hará uso de un AFD que debe ser diseñado a partir de operaciones de complemento e intersección.

Lote de Códigos Ecológicos – LCE

Los LCE representan un conjunto de CEs enviado por filiales.
El conjunto de CEs debe ser implementado mediante un archivo de texto.
Los LCE deben ser implementados mediante una 4-upla compuesta por el nombre completo del archivo que contiene el conjunto de CEs, el "manejador" que permite el flujo de datos desde el archivo hacia el programa (i.e. una variable `FILE*`), un puntero a un objeto con tamaño suficiente para contener la línea de mayor longitud del archivo, y un número entero que cumple la función de indicador de estado ó error con tipo de dato `LCE_Estado`.
Los posibles valores de este indicador deben ser implementados mediante un `enum` con, *por lo menos*, los siguientes identificadores

```
typedef enum {
    SinErrores = 0,
    ArchivoInexistente, MemoriaInsuficiente, ArchivoVacio,
    ErrorDeLectura, LineaDemasiadoLarga, FormatoDeArchivoIncorrecto,
    ComienzoDeLote, FinDeLote
} LCE_Estado;
```

El archivo de texto que implementa el contenido del lote está compuesto por líneas, con los CEs separados por uno o varios blancos y/o caracteres de tabulación ('\t'). La primer línea de este archivo contiene la longitud máxima posible de las siguientes líneas.

Operaciones a Especificar e Implementar

1. `LCE_Crear`. Crear: Cadena → LCE. Creación. Adquiere los recursos necesarios para sustentar un LCE; realiza las acciones de preparación y de inicialización. La cadena dato representa el nombre del archivo donde se implementa el contenido del lote. Luego de la creación, si fue satisfactoria y el lote no está vacío, el LCE resultante está listo para leer el primer CE del lote y el estado del lote es ComienzoDelLote.
2. `LCE_Destruir`. *Destruir: LCE → Ø*. Destrucción. Libera cualquier recurso adquirido para el LCE durante su operación de creación.
3. `LCE_GetNombreDelArchivo`. *GetNombreDelArchivo: LCE → Cadena*. La cadena resultado informa cual es el nombre del archivo que contiene los CEs.
4. `LCE_GetEstado`. *GetEstado: LCE → LCE_Estado*. Obtiene el estado del LCE dato. El conjunto de llegada es el conjunto de estados o posibles de un LCE.
5. `LCE_ContarCEsCorrectos`. *ContarCEsCorrectors: LCE → Entero x LCE.* (1)
6. `LCE_ContarCEsIgualesA`. *ContarCEsIgualesA: LCE x CE → Entero x LCE.* (1)
7. `LCE_ContarElementosNoCEs`. *ContarElementosNoCEs: LCE → Entero x LCE.* (1)

(1) Las operaciones de conteo actúan sobre el LCE dato de forma íntegra, contempla todos los elementos del lote (proceso secuencial). En el caso de que se haya podido realizar el conteo, las tres operaciones tienen como *postcondición* del LCE resultado GetEstado = ComienzoDelLote, es decir que luego de una operación de conteo, el LCE resultado queda listo para realizar otra operación de conteo; por lo tanto la *precondición* del LCE dato para las tres operaciones es que, si es un LCE sin errores, este LCE dato se encuentra al comienzo del lote.

Consideraciones y Restricciones

- Cualquier decisión que el equipo tome que no esté en el enunciado o no esté del todo aclarado deberá formar parte de la especificación y/o ser agregada como hipótesis de trabajo.
- Considerar el tratamiento de los errores en ambos TADs. ¿Cómo implementarlo? Si existen diferentes posibilidades seleccionar una y agregar una hipótesis de trabajo que precise la opción elegida.
- En el TAD LCE se enumeran 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.
- Cada TAD debe implementarse en una biblioteca independiente. La aplicación de prueba debe hacer uso de ambas bibliotecas.

- Luego de ser construidas con la herramienta de desarrollo elegida por el equipo, las implementaciones de las bibliotecas y la aplicación de prueba deben ser verificadas 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.
- Los identificadores para las funciones que implementan las operaciones deben ser los indicados en el comienzo de las anteriores descripciones de las operaciones.
- El set de pruebas que se utiliza en la aplicación que prueba del TAD debe ser constante, y no ingresado por el usuario a través del teclado, *salvo* los nombres de los archivos, que sí son ingresados por teclado y luego correctamente validados. La lectura de estos nombre debe ser realizada mediante la función `fgets` con el flujo `stdin` como argumento.
- Deben usarse obligatoriamente las funciones `malloc` (ó `calloc`), `free`, `fgets`, `strtok`, `rewind` y cualquier otra función ANSI que facilite la construcción de la implementación.

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 la 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 **12** 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 CE

1.1. Especificación

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

1.2. Implementación

1.2.1. Diseño del Autómata

Definición formal del AFD y Diagrama de transiciones del AFD. Se deben incluir las operaciones de complemento e intersección utilizadas para el diseño del AFD.

1.2.2. Biblioteca que implementa el TAD

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

1.2.3. Salidas

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

2. TAD LCE

2.1. Especificación

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

2.2. Implementación

2.2.2. Biblioteca que implementa el TAD

2.2.2.1. Listado de código fuente del archivo encabezado, **LCE.h**.
2.2.2.2. Listado de código fuente del la definición de la Biblioteca, **LCE.c**.

2.2.3. Salidas

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

3. Aplicación Prueba del TAD

3.1. Código Fuente

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

3.2. Salidas

3.2.1. Captura impresa de la salida de la aplicación de prueba. Utilizar fuente de ancho fijo.
3.2.2. Captura impresa de la salida del proceso de traducción (**BCC32**). Utilizar fuente de ancho fijo.

Otros Componentes de la Entrega

4. Copia Digitalizada

CD ó disquette con copia de **solamente** los 5 archivos de código fuente (**CE.h**, **CE.c**, **LCE.h**, **LCE.c** y **IIEAplicacion.c**). No se debe entregar ningún otro archivo.

5. Formulario de Seguimiento de Equipo

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

Tiempo

La entrega se realizará aproximadamente dos semanas después de la presentación del enunciado. La fecha de entrega se definirá en clase.