TAREA Nº3

Programación en C: Aplicación de Árboles Binarios.

Fecha de envío: domingo 20 de noviembre, a las 23:55 hrs.
Modalidad: Trabajo en grupos de a lo más **dos** personas

I. OBJETIVOS.

El objetivo del presente laboratorio es evaluar tu capacidad para:

1. Llevar a cabo un programa completo en el lenguaje de programación C, utilizando todos los elementos básicos que provee el lenguaje, y sus tipos de datos básicos.
2. Aplicar las buenas prácticas en programación (orden, comentarios, identificadores representativos).
3. Ser capaz de cumplir con una interfaz concreta solicitada para el programa.
4. Diseñar e implementar una estructura recursiva dinámica para resolver un problema que requiera análisis de información.

II. ENUNCIADO.

El código morse es un sistema de comunicación que sirve para transmitir mensajes a través de un telégrafo. Fue desarrollado por Alfred Vail y Samuel Morse en 1830, cuando ambos estaban trabajando en la creación de un telégrafo eléctrico.

Este sistema tiene la particularidad de permitir enviar mensajes a través de pulsos, sonidos, golpes o luz intermitentes: esto se debe a que cada letra del alfabeto, dígitos y puntuación están codificados mediante el uso de puntos y rayas (o sonidos/luces "cortos" y "largos").

(Si quieres saber más ingresa a: https://morsecw.com/)

Se necesita que implementes un programa en C que sea capaz de traducir mensajes entre el lenguaje natural y el código morse (en ambos sentidos).

La codificación de cada símbolo según el código morse, está establecido en el árbol binario mostrado en la figura 1. En este árbol, la raíz es el punto de partida. Cada vez que avances en el árbol obedecerá al uso de un punto o de un guion. Cada vez que avances hacia un hijo izquierdo se interpreta como el uso de un **punto**. Por otro lado, si avanzas hacia la derecha se interpreta como el uso de un **guion**. De esta forma recorres el árbol hasta que encuentras el símbolo que buscas. A continuación, se presentan algunos ejemplos:

Ejemplo 1: La letra `K` corresponde al código: `-.-` ("guion", "punto", "guion")

Ejemplo 2: El operador `+` corresponde al código: `.-.-.` ("punto", "guion", "punto", "guion", "punto")

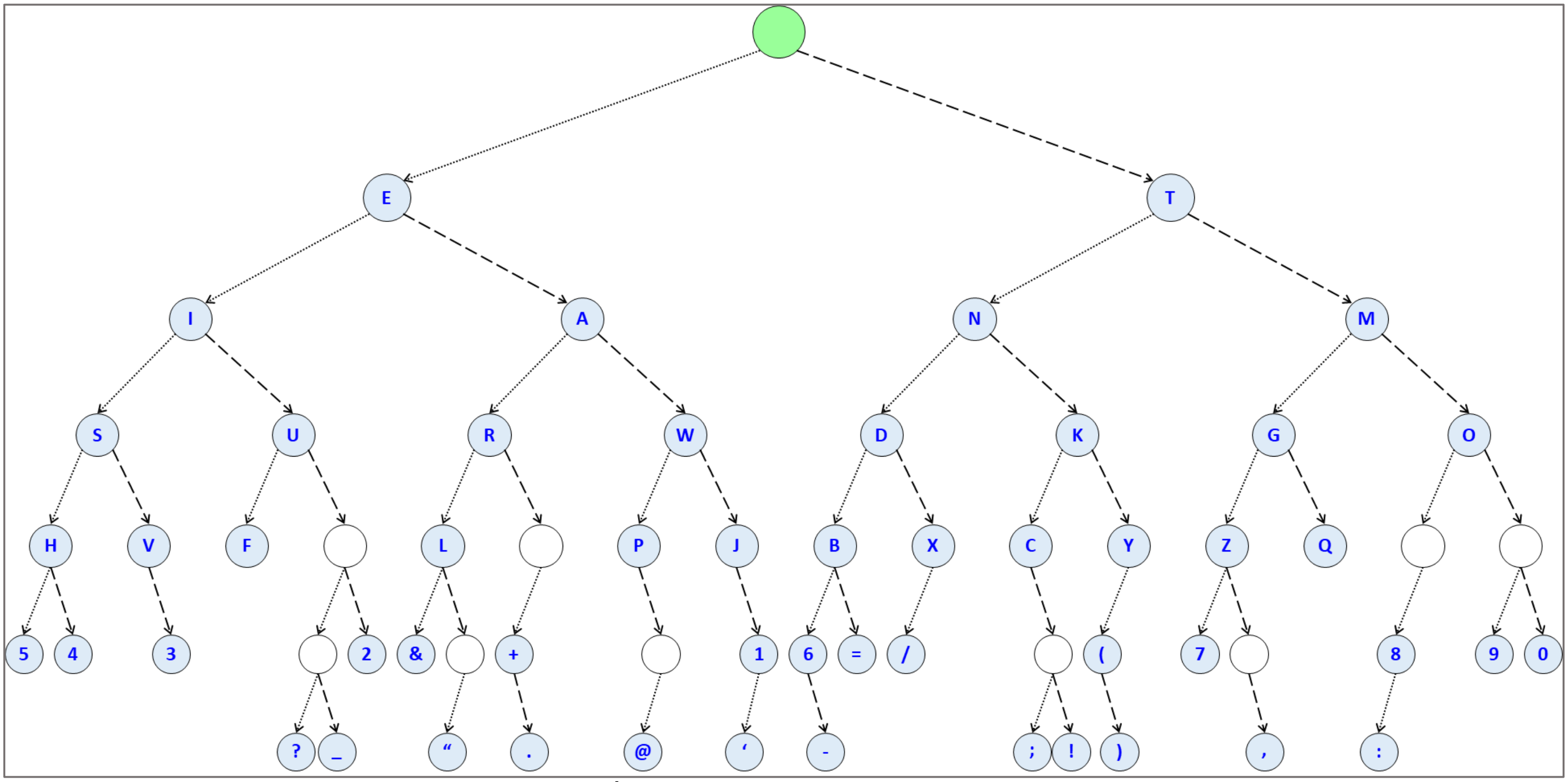


Figura 1: Árbol binario que almacena el código morse.

Observa que se puede codificar textos que incluyan puntuación, correos electrónicos, operaciones matemáticas, direcciones o cualquier expresión que se pueda armar con la unión de estos símbolos.
Para codificar una palabra, el código morse de cada símbolo se debe separar por **un espacio**. Por otro lado, para codificar una secuencia de palabras, el código morse de cada palabra se debe separar por **dos espacios**. A continuación, se presentan algunos ejemplos (los cuadros amarillos destacan el espacio entre cada código):

Ejemplo 1:

Texto: JUAN.PEREZ@UNAB.CL

Código:
.--- ..- .- -. .-.-.- .--. . .-. . --.. .--.-. ..- -. .- -... .-.-.- -.-. .-..

Ejemplo 2:

Texto: MAMA, TE QUIERO MUCHO!

Código:
-- .- -- .- --..-- - . --.- ..- .. . .-. --- -- ..- -.-. .... --- -.-.--

Ejemplo 3:

Código: . .-.. .-. . ... ..- .-.. - .- -.. --- . ... ---... .....

Texto: EL RESULTADO ES: 5

Ejemplo 4:

Código: .. -. - -- .- .. -. -.--. -.--.-

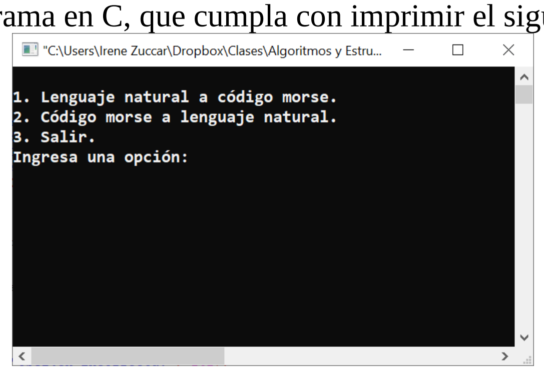
Texto: INT MAIN()

Observa que:

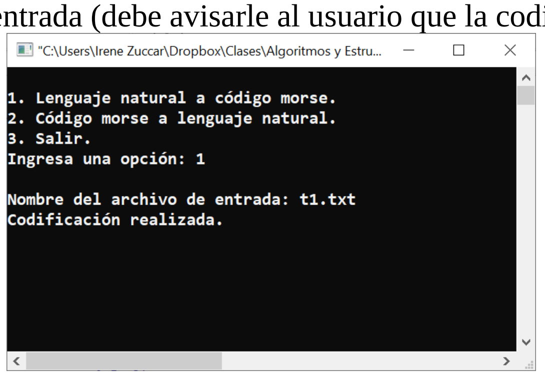
- La codificación solo trabaja con letras mayúsculas.
- Los nodos del árbol binario que están "vacíos" corresponden a símbolos que no pertenecen a nuestro lenguaje.
- No se incluyeron vocales con tilde, ni la Ñ.

Se pide:

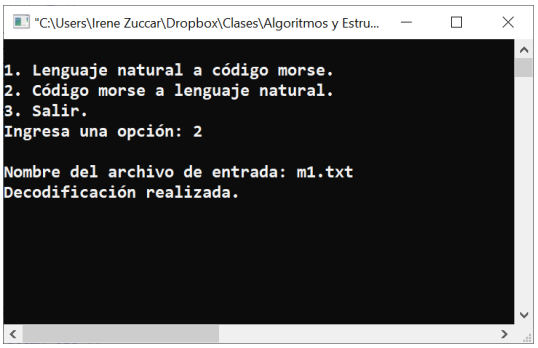
Debes construir un programa en C, que cumpla con imprimir el siguiente menú:



Para la opción 1 el programa debe solicitar el nombre del archivo de texto con el texto en lenguaje natural que se desea codificar. El resultado de la codificación debe ser almacenado en el mismo archivo de entrada (debe avisarle al usuario que la codificación se realizó):

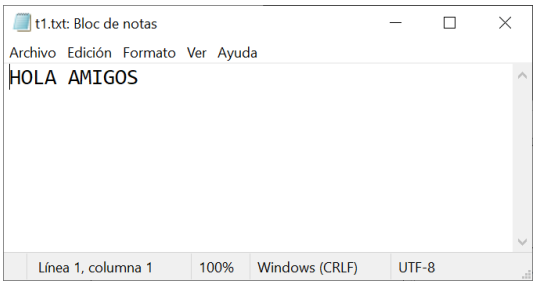


Para la opción 2 el programa debe solicitar el nombre del archivo de texto con el código morse que se desea decodificar a lenguaje natural. El resultado de la decodificación debe ser almacenado en el mismo archivo de entrada (debe avisarle al usuario que la codificación se realizó):

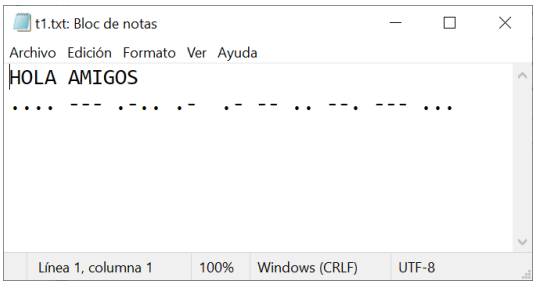


Recuerda que un menú se debe volver a mostrar luego de realizar el trabajo de cada opción, mientras el usuario no escoja la opción 3 de "**Salir**".

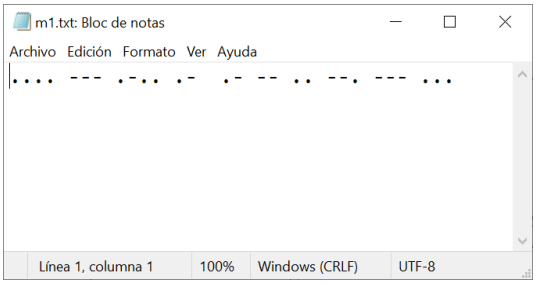
A continuación, se muestran un ejemplo de la modificación que debe hacer tu programa al archivo de entrada, para la opción y para la opción 2:



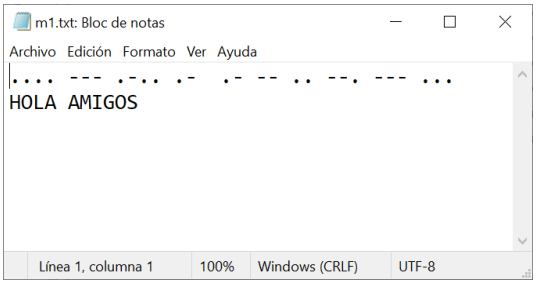
Archivo de entrada "t1.txt" para la opción 1, **antes** de la ejecución de tu programa.



Archivo de "t1.txt" para la opción 1, **después** de la ejecución de tu programa.



Archivo de entrada "m1.txt" para la opción 2, **antes** de la ejecución de tu programa.



Archivo de entrada "m1.txt" para la opción 1, **después** de la ejecución de tu programa.

NOTA 1: Junto a este enunciado está a tu disposición el archivo **morse.c** que posee la **creación del árbol binario de la figura 1.**

NOTA 2: En la página https://morsedecoder.com/ puedes ingresar textos y códigos morse y los codifica y decodifica respectivamente. (Cuando codifica a morse aparece un "/" para separar palabras, solo debes **borrarlo**)

III. CONSIDERACIONES EN LA REVISIÓN.

Sobre el código:

1. **Debes usar árboles binarios dinámicos para resolver esta tarea. Si ocupas otro tipo de solución, la nota será 1.0.**

2. Tu programa **debe seguir exactamente** las **reglas e interfaces** explicadas en este documento.

3. Debes construir **una función para cada tarea que realice tu código**, cuidando los tipos de datos de las entradas y de las salidas.

4. Debes usar nombres representativos para tus variables, parámetros de entrada y funciones.

5. Debes comentar cada una de tus funciones, estructuras y tipos de datos que definas, indicando una descripción de la labor que lleva a cabo, sus entradas, y sus salidas. En la figura 2 aparece un ejemplo de cómo debes hacerlo.

```
/*
Entrada: número entero que se revisará.
Salida: número entero que corresponde al número de divisores.
Proceso: función que recibe un entero y calcula y retorna el total
         de divisores que posee.
*/
int numeroDivisores(int n)
{
    int i=2, cont=0;

    while (i < n)
    {
        if (n%i == 0)
        {
            cont++;
        }
        i++;
    }
    return cont+2;
}
```

Figura 2: Ejemplo de cómo debes comentar una función en esta tarea.

6. Tu código debe estar correctamente *indentado* (uso de sangrías para cada sub-bloque de instrucciones), esto incluye el correcto alineamiento de las llaves ("**{**" y "**}**") que enmarcan tales bloques. Se penalizarán códigos desordenados en este aspecto.

7. Tu código no puede presentar más de 1 línea en blanco. Se penalizarán códigos que no cumplan con este punto.

8. Puedes trabajar con el IDE y compilador de C, que más te acomode. No obstante, lo anterior, tu código debiera poder ser compilado y ejecutado sin problemas en Windows.

9. El sistema debe ser **robusto**, se penalizarán los errores no manejados, de cualquier tipo.

Sobre la entrega, atrasos y faltas a la ética.

1. Debes subir tu trabajo **al aula virtual de tu curso de cátedra** dentro de la plataforma **https://unab.blackboard.com/,** en una casilla que se habilitará especialmente para esto.

2. **NO SE ACEPTARÁN TRABAJOS ATRASADOS.**

3. Tu trabajo debe ser subido por **SOLO UNO de los integrantes del grupo**.

4. Debes subir solo el código fuente.

5. El nombre del código fuente debe ser "**rutcompleto1_rutcompleto2.c**", indicando el rut de cada integrante del grupo. Por ejemplo, si tu rut es 19.000.111-3, debieras ingresarlo sin puntos ni guión, pero sí con el dígito verificador: o sea 190001113.

6. Si el programa no se puede ejecutar, tendrá la nota mínima: **1.0**. Si su programa funciona, se evaluará su ejecución, y también el código fuente.

7. Ante el escenario de existir sospecha de copia (con otros compañeros, o desde internet), será interrogado acerca de su trabajo, para aclarar dudas de su entendimiento y autoría. Si se confirma la sospecha, el trabajo será evaluado con nota **1.0**.

8. Las consultas las debe realizar directamente a los profesores de taller, presencialmente en clases o a los correos y en los horarios que ellos establezcan.