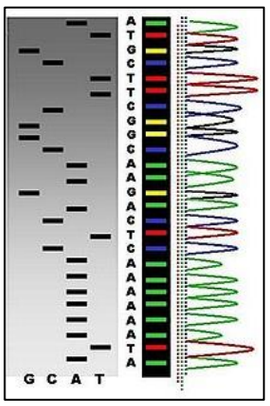
LISTAS Y ÁRBOLES

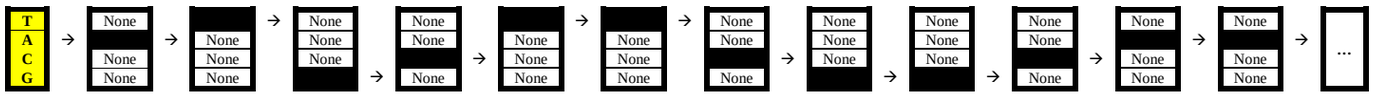
EJERCICIO 1

La **secuenciación del ADN** es un conjunto de métodos y técnicas bioquímicas cuya finalidad es la determinación del orden de los nucleótidos (A, C, G y T) en un oligonucleótido. Un **oligonucleótido** es una secuencia corta de ADN o ARN y constituye la información genética heredable que forma la base de los programas de desarrollo de los seres vivos. Así pues, determinar la secuencia de ADN es útil en el estudio de la investigación básica de los procesos biológicos fundamentales, así como en campos aplicados, como la investigación forense. Además, se puede utilizar la secuenciación del ADN para conocer las mutaciones somáticas, como las sustituciones de bases, generadas entre distintos organismos.



En un laboratorio especializado en análisis de ADN, dentro de los servicios que ofrece está el de certificación de paternidad. Para ello, mediante la técnica **Secuenciación por Terminador Fluorescente**, someten a estudio 2 muestras de ADN que son comparadas y si la secuencia ordenada de los nucleótidos coincide por sobre un 90%, se concluye a favor de la paternidad.

Para este certamen cada nucleótido es representado por un nodo dentro de una lista enlazada dinámicamente. Este nodo consta de 4 campos llamados G, C, A, T los que guardan la dirección de memoria del siguiente nodo (nucleótido). De esta forma, según la imagen de la derecha, la lista enlazada tendría la siguiente apariencia para los primeros 11 nucleótidos:



La base de esta lista, corresponde a una variable cuya estructura de registro es la misma que la de cada nodo. En la figura anterior aparece pintada de amarillo.

SE PIDE hacer un programa que compare la secuencia de 2 muestras y determine si se concluye a favor la paternidad o no. Para ello, cada secuencia es almacenada en una lista enlazada dinámicamente, en donde cada nodo corresponde a un nucleótido. La cantidad de nucleótido es desconocida, por lo que la comparación debe considerar esta situación. Cada nucleótido guarda la dirección de memoria del siguiente nucleótido.

Para este programa, asuma que ambas secuencia ya se encuentran alojadas en memoria del computador, en donde la ubicación del primer nucleótido de la primera secuencia es almacenada en una variable base llamada ADN1 y la ubicación del primer nucleótido de la segunda secuencia es almacenada en la variable base ADN2.

El programa deberá desplegar como resultado el porcentaje de coincidencia y la palabra CONFIRMADA PATERNIDAD o RECHAZADA PATERNIDAD según corresponda.

Además, el programa deberá desplegar en pantalla 3 columnas con la información de cada cromosoma de ambas secuencia, señalando cuáles no coinciden, por ejemplo:

SECUENCIA 1	SECUENCIA 2	COINCIDENCIA
A	A	COINCIDE
T	T	COINCIDE
G	G	COINCIDE
C	C	COINCIDE
T	T	COINCIDE
T	A	NO COINCIDE
C	G	NO COINCIDE
G	G	COINCIDE
	T	NO COINCIDE
	T	NO COINCIDE
	A	NO COINCIDE
	...	

EJERCICIO 2

Existe cierta organización en que los datos de los empleados se mantienen en una estructura de árbol binario, cuya clave es el código del empleado (COD_EMPLEADO). La estructura de cada nodo es la siguiente:

CAMPO	TIPO	DESCRIPCION
COD_EMPLEADO	NUMÉRICO	Valor numérico entre 1 y 10000 que permite individualizar a cada empleado.
NOMBRE	TEXTO	Texto de hasta 30 caracteres.
EDAD	NUMÉRICO	Valor numérico entre 18 y 80.
SEXO	TEXTO	Carácter, en donde F: Femenino y M: Masculino.
IZQ	PUNTERO	Localización en donde se encuentra el siguiente nodo cuyo código del empleado es menor.
DER	PUNTERO	Localización en donde se encuentra el siguiente nodo cuyo código del empleado es mayor.

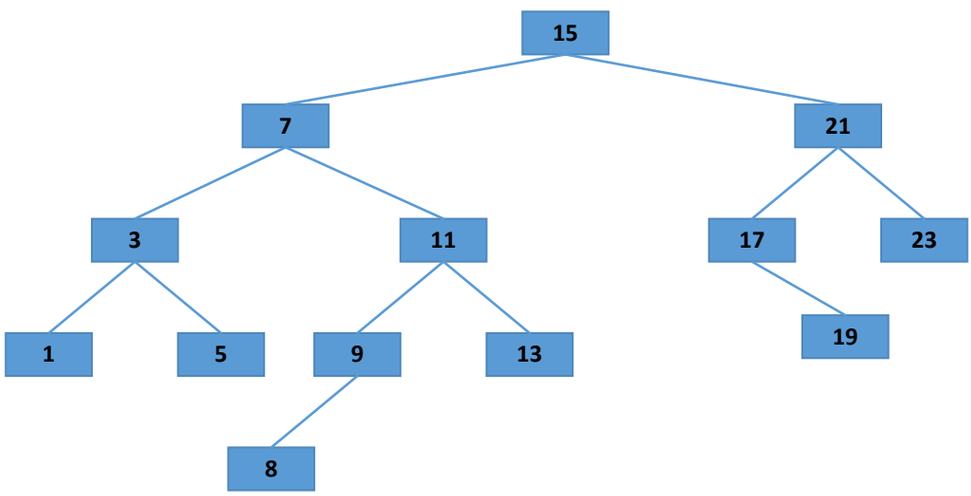
En un comienzo, el árbol se encuentra vacío. La distribución de los nodos será considerando que el código del empleado a añadir al árbol se ubicará a la izquierda si éste es menor; de lo contrario, se ubicará a la derecha.

Se pide escribir un programa que permita desplegar un menú con las siguientes alternativas:

1. **AGREGAR EMPLEADO**. El programa deberá solicitar el ingreso por teclado del código del empleado a agregar. Luego, se deberá validar si el código ya existe en el árbol. Si el código existe, se deberá desplegar un mensaje de error, de lo contrario se procederá a leer el resto de los datos y añadir el empleado. (Considere que el nombre, edad y sexo son correctamente ingresados).

2. **LISTAR**. Se deberá solicitar el ingreso por teclado de la EDAD y SEXO de los empleados a listar, en donde se deberá desplegar por pantalla todos aquellos empleados cuyo SEXO sea el ingresado y cuya edad sea superior o igual al ingresado.

3. **SALIR**. Finaliza la ejecución del programa.



EJERCICIO 3

En una empresa del rubro informático, se lleva el registro de empleados en una **lista dinámica**, en donde se almacenan los datos, de acuerdo a la siguiente definición de registro:

CAMPO	TIPO	DESCRIPCION
COD_EMPLEADO	NUMÉRICO	Valor numérico entre 10000 y 99999 que permite individualizar a cada empleado.
NOMBRE	TEXTO	Texto de hasta 20 caracteres.
APE_PATERNO	TEXTO	Texto de hasta 15 caracteres.
APE_MATERNO	TEXTO	Texto de hasta 15 caracteres.
SIGUIENTE_CODIGO	PUNTERO	Tendrá la localización del siguiente empleado (NODO) dentro de la lista, manteniendo el orden ascendente por COD_EMPLEADO
SIGUIENTE_PATERNO	PUNTERO	Tendrá la localización del siguiente empleado (NODO) dentro de la lista, manteniendo el orden ascendente por APE_PATERNO.
PRIMER_CONTACTO	PUNTERO	Tendrá la localización del primer contacto (NODO) ubicado en la lista de Contactos del Empleado (Ver imagen al final).

Además, se registran los contactos que cada empleado pueda poseer para comunicarse. Cada empleado posee su propia lista de contactos. La estructura de cada nodo es la siguiente:

CAMPO	TIPO	DESCRIPCION
TIPO	TEXTO	Carácter que representa el tipo de contacto al que se referencia en el campo CONTACTO. Sus valores pueden ser: C: Celular, E: Email, T: Teléfono Fijo.
CONTACTO	TEXTO	Texto de hasta 30 caracteres correspondiente al valor del contacto, por ejemplo: +56 9 4123 6754
SIGUIENTE	PUNTERO	Localización del siguiente nodo en la lista de contactos para el empleado.

Se pide diseñar e implementar un programa que permita mantener los datos de la lista de EMPLEADOS junto con sus respectivos CONTACTOS. Se deberá desplegar un menú con las opciones:

1. **AGREGAR: Permite añadir un empleado a la lista.**
   - Solicitar el ingreso por teclado del código del empleado.
   - Validar si el código del empleado existe en la lista.
   - Si existe, se deberá mostrar un mensaje de error.
   - Si no existe, se deberá solicitar el ingreso por teclado del resto de los datos personales del empleado.
   - A continuación, se deberá preguntar si desea ingresar contactos. De ser afirmativa la respuesta a la pregunta anterior, se deberá proceder a ingresar los datos de contacto (Tipo y Contacto) en un proceso iterativo hasta que el usuario desee finalizarlo (emplee una lista dinámica auxiliar).
   - Finalmente, se deberá preguntar si el usuario desea **Guardar** los datos. Si la respuesta es afirmativa, se procederá a guardar los datos en las respectivas listas. De lo contrario, los datos serán desechados.

2. **MODIFICAR: Permite modificar los datos de cada empleado, excepto el código del empleado.**
   - En primer lugar, se deberá validar que haya al menos un empleado en la lista de empleados.
   - Luego, solicitar el ingreso del código del empleado a modificar.

- Validar si el código del empleado existe en la lista de empleados.
- Si no existe, se deberá mostrar un mensaje de error.
- Si existe, se deberá desplegar el resto de los datos del empleado, inclusive los de contacto. Anteponga un número a cada dato editable, de tal forma que sea presentado como un menú en donde el usuario elegirá qué datos desea modificar. Lo anterior implica que el proceso se deberá realizar iterativamente y deberá finalizar cuando se digite un cero como dato a modificar. Si se digita un -1 (en vez de elegir un dato a modificar) implicará que el usuario podrá añadir un nuevo contacto. Si se digita un valor negativo distinto a -1, implicará que el usuario desea eliminar el contacto, en tal caso el valor absoluto del número digitado corresponderá al contacto a eliminar, verifique que el valor digitado corresponde a un dato de contacto. Las modificaciones a los contactos se deben realizar en una lista auxiliar.
- Finalmente, se deberá preguntar si el usuario desea **Guardar** las modificaciones realizadas. Si la respuesta es afirmativa, se procederá a guardar los datos en las respectivas listas. De lo contrario, los datos ingresados serán desechados.

3. **ELIMINAR: Permite eliminar todo registro de un empleado.**
   - En primer lugar, se deberá validar que haya al menos un empleado en la lista de empleados.
   - Se deberá solicitar el ingreso del código del empleado.
   - Validar si el código del empleado existe en la lista de empleados.
   - Si no existe, se deberá mostrar un mensaje de error.
   - Si existe, se deberá desplegar el resto de los datos del empleado, inclusive los de contacto. Luego, el programa deberá preguntar al usuario si está seguro de eliminar los datos del empleado. De ser afirmativo, se deberá proceder a la eliminación de todos los datos en cada lista.

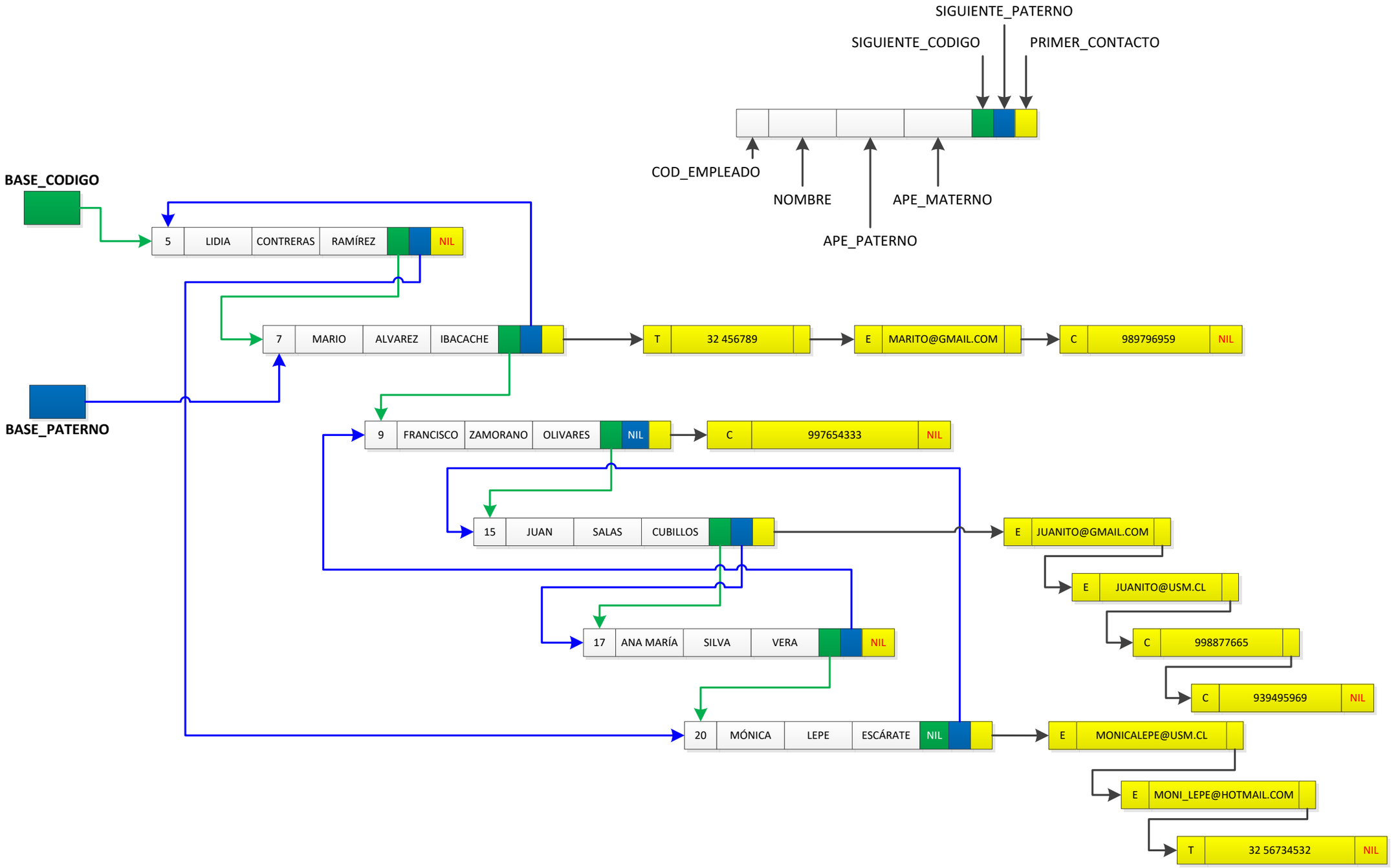
4. **CONSULTAR: Permite conocer los datos almacenados de cada empleado.**
   - En primer lugar, se deberá validar que haya al menos un empleado en la lista de empleados.
   - Se deberá solicitar el ingreso del código del empleado
   - Validar si el código del empleado existe en la lista de empleados.
   - Si no existe, se deberá mostrar un mensaje de error.
   - Si existe, se deberá desplegar el resto de los datos del empleado, inclusive los de contacto.

5. **LISTAR: Permite desplegar todos los datos de todos los empleados, inclusive los datos de contacto.**
   - En primer lugar, se deberá validar que haya al menos un empleado en la lista de empleados.
   - Ofrezca la posibilidad de desplegar el listado ordenado por Código del Empleado o por Apellido Paterno.
   - Procure que el despliegue de datos esté controlado, de tal forma hacer pausa cada vez que el despliegue complete todas las líneas de pantalla.

6. **SALIR: Da término a la ejecución del programa.**

Recuerde que el campo PRIMER_CONTACTO debe contener la dirección de memoria del primer contacto en la lista de Contactos para el empleado en cuestión. Si el empleado no posee contactos, el valor será None.

En la siguiente figura se esquematiza una situación.

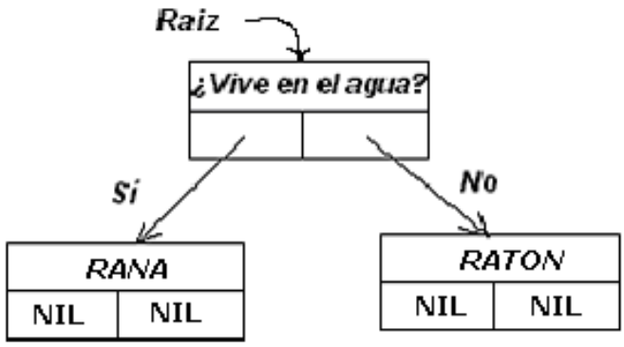

SIGUIENTE_PATERNO
SIGUIENTE_CODIGO
PRIMER_CONTACTO
COD_EMPLEADO
NOMBRE
APE_PATERNO
APE_MATERNO
BASE_CODIGO
BASE_PATERNO
5
LIDIA
CONTRERAS
RAMÍREZ
NIL
7
MARIO
ALVAREZ
IBACACHE
T
32 456789
E
MARITO@GMAIL.COM
C
989796959
NIL
9
FRANCISCO
ZAMORANO
OLIVARES
NIL
C
997654333
NIL
15
JUAN
SALAS
CUBILLOS
E
JUANITO@GMAIL.COM
E
JUANITO@USM.CL
C
998877665
17
ANA MARÍA
SILVA
VERA
NIL
C
939495969
NIL
20
MÓNICA
LEPE
ESCÁRATE
NIL
E
MONICALEPE@USM.CL
E
MONI_LEPE@HOTMAIL.COM
T
32 56734532
NIL

EJERCICIO 4

El Juego de los animales, es un caso de simulación de inteligencia en el computador. El juego consiste en lograr que el computador sea capaz de adivinar cierto animal en el que está pensando el usuario, en base a preguntas que son respondidas con un “SI” o un “NO” (por el usuario).

Como estructura de datos en la memoria del computador, se utiliza un árbol binario ordenado, en cuyos nodos van quedando almacenadas preguntas y animales.

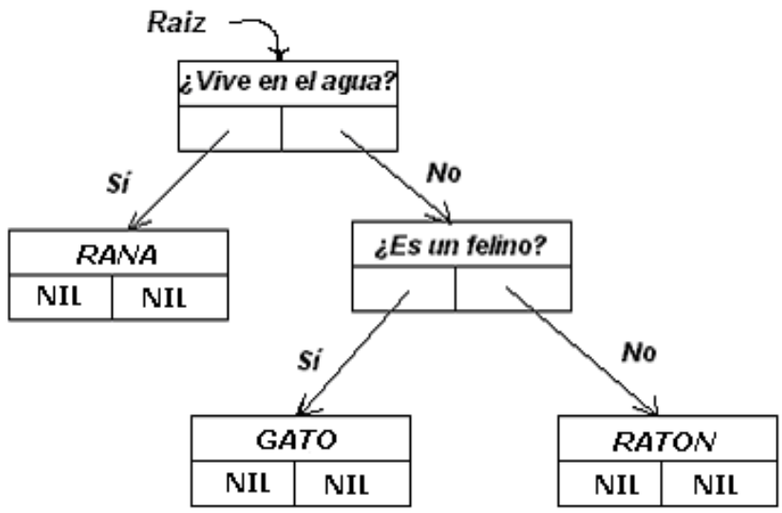
Al principio, el árbol comienza con sólo 3 nodos, los que contienen: La pregunta inicial “¿Vive en el agua?” y 2 animales (“Ratón” y “Rana”). El árbol inicial es del siguiente tipo:



En la situación inicial, se podría establecer en un diálogo como el siguiente:

```
Computador : Piense en un animal y yo trataré de adivinarlo. (El usuario piensa en un gato)
Computador : El animal que Ud. pensó, ¿Vive en el agua?
Usuario    : No
Computador : ¿Es un(a) RATON?
Usuario    : No
Computador : Lo felicito, Ud. ganó, pero debe decirme cuál era el animal.
Usuario    : GATO
Computador : Ahora, debe decirme una pregunta que diferencie Un(a) GATO de un(a) RATON.
Usuario    : ¿Es un felino?
Computador : ¿Cuál sería la respuesta correcta a esa pregunta para Un(a) GATO ?
Usuario    : Sí
Computador : Gracias, ahora conozco 3 animales. ¿Desea jugar nuevamente?
```

En ese momento, el árbol debería tomar la siguiente forma:



Note que cada vez que el computador pierde, logra conocer a un nuevo animal. Ahora, cada vez que el computador gana, se “pone muy contento”

Especificaciones:

Se debe construir un programa en lenguaje Python, que permita las siguientes opciones:

a) **JUGAR**: Debe permitir jugar como se ha descrito, comenzando con los 2 animales iniciales (la primera vez). Después, en sucesivos juegos, se irá incrementando el número de animales. Cuando se realiza la primera jugada, el programa debe preguntar si se desea Recuperar lo existente en el archivo de respaldo (si es que no se ha realizado tal opción).

b) **RESPALDAR**: Esta opción permite almacenar en un archivo en disco los datos del árbol construido hasta ese momento (se pierden los datos anteriores del archivo, si existía). Se debe realizar recorrido "Pre-orden" sobre el árbol e ir almacenando el contenido de los nodos en forma consecutiva en el archivo (el que será en formato CSV). No se debe almacenar los punteros, pero sí debe almacenarse en el registro grabado si el nodo es "pregunta" o es "animal".

c) **RECUPERAR**: Esta opción que permita restaurar el árbol desde el archivo en disco (volviendo a construir el árbol). Se debe recordar el recorrido pre-orden.

d) **LISTAR**: Esta opción sirve para revisar el contenido del árbol en recorrido pre-orden (se usa pantalla y se debe indicar si cada nodo es "pregunta" o "animal").

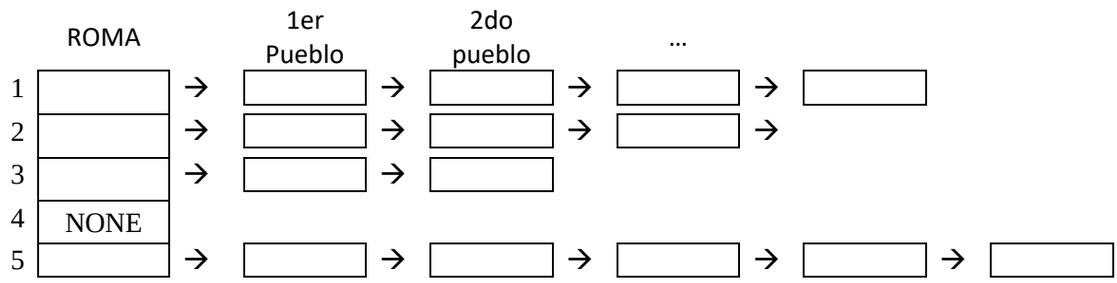
Note que en un nodo, a veces, se almacena un animal y a veces, se almacena una pregunta.

EJERCICIO 5.

La expresión "Todos los caminos conducen a Roma", proviene de la época del Imperio romano en donde se construyeron más de 400 vías -unos 70.000 kilómetros- para comunicar la capital, Roma, considerada el centro donde convergía el poder del imperio, con las provincias más alejadas. En muchas ocasiones estos caminos fueron creados de forma espontánea por las propias legiones.

Roma estará representada por un Vector que guardará una referencia al primer pueblo de cada camino (Vector de Bases de las listas). Este vector deberá permitir registrar los pueblos de hasta 5 caminos.

El primer pueblo de cada lista (primer nodo) estará dado por la primera vez que aparezca para un camino en la lectura del archivo.



Cada nodo tendrá la siguiente estructura:

CAMPO	TIPO DE DATO	DESCRIPCIÓN
Pueblo	Texto	Texto de hasta 30 caracteres correspondientes al Nombre del Pueblo
Distancia	Numérico	Distancia entre el pueblo y el siguiente de acuerdo al orden establecido en este archivo, expresado como valores enteros correspondientes a kilómetros.
Siguiente	Puntero	Localización del siguiente nodo.

El programa deberá presentar un menú con las siguientes alternativas:

1. **Añadir Pueblo**. Deberá solicitar el Nombre del pueblo a agregar (considere que los nombres de los pueblos no se pueden repetir), además deberá ingresar el nombre del pueblo que lo precede (pueblo anterior), de tal forma que el nuevo pueblo deberá quedar inserto a continuación de éste. También deberá ingresar la distancia que hay entre el nuevo pueblo y el que lo precede y la nueva distancia que hay entre el nuevo pueblo y el que lo sucede (recuerde que se ha insertado un pueblo entre dos), considere que este último dato solo se debe solicitar si el nuevo pueblo queda inserto entre dos pueblos.

2. **Listar Pueblos por Camino**. Deberá listar todos los pueblos por los que pasa un camino. En este caso se deberá solicitar el ingreso por teclado del número del camino a listar.

3. **Conocer el pueblo más distante de Roma**. Se deberá mostrar en pantalla el nombre del pueblo más distante de roma y la distancia que los separa.

4. **Finalizar**. Dará por término al programa, guardando las listas en un archivo tipificado.

Los pueblos por los que pasa cada camino se deberán almacenar en un archivo tipificado llamado PUEBLOS.DAT, el cual tiene la siguiente estructura de registro:

CAMPO	TIPO DE DATO	DESCRIPCIÓN
Camino	Numérico	Número que identifica el camino del camino y está entre el rango 1 y 5
Pueblo	Texto	Texto de hasta 30 caracteres correspondiente al Nombre del Pueblo
Distancia	Numérico	Distancia entre el pueblo y el siguiente de acuerdo al orden establecido en este archivo. Corresponde a un valor con decimales en la unidad de medida kilómetros.

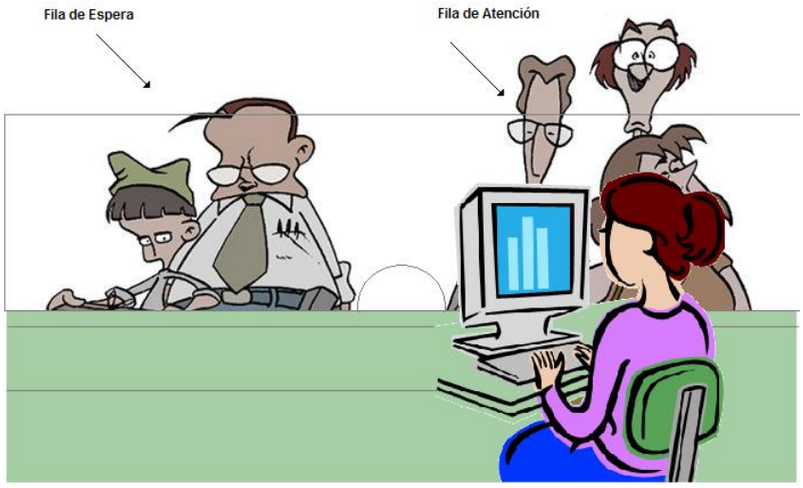
Con el archivo PUEBLOS.DAT, se deberá construir las listas enlazadas respectivas cuando se inicie el programa.

El cálculo del camino más largo se deberá obtener a partir del recorrido y análisis de cada lista enlazada, **NO DESDE LA LECTURA DE LOS DATOS DESDE EL ARCHIVO.**

EJERCICIO 6.

En la atención de clientes en una caja de un banco, se pueden apreciar 2 tipos de listas que se van formando en la medida que los clientes llegan: Fila de Atención y Fila de Espera.

Cuando un cliente llega al banco, éste se incorpora a la Fila de Atención. Sin embargo, cuando el cliente es atendido puede ser que tenga que esperar por algún trámite interno (verificación de cheque, consulta a sucursales, etc.); en tal caso, el cliente deberá esperar a un costado de la caja hasta que se solucione el problema. Esta situación le puede suceder a más de un cliente, por lo que se irá formando una fila al costado de la caja, la que llamaremos Fila de Espera. En la siguiente figura se ilustra dicha situación:



SE PIDE hacer un programa que, empleando estructuras de datos, pueda **SIMULAR** esta situación. Para ello, se debe contar con 2 arreglos: Espera y Atención, ambos de 5 elementos. El programa deberá desplegar un menú en pantalla con las siguientes alternativas:

1. **INCORPORAR A LA FILA DE ATENCIÓN**. Se deberá ingresar, por teclado, solamente la parte numérica del RUN del cliente, este deberá quedar almacenado en el arreglo ATENCIÓN, según corresponda.

2. **ATENDER CLIENTE**. Implica sacar el cliente correspondiente de la fila de Atención. Se deberá mostrar en pantalla el RUN del cliente a ser atendido.

3. **PASAR CLIENTE EN ATENCIÓN A FILA DE ESPERA**. Se deberá agregar a la fila de espera el cliente que está siendo atendido.

4. **ATENDER CLIENTE EN ESPERA**. Se deberá sacar de la fila de espera al cliente correspondiente según el orden de atención. Además deberá desplegar el RUN del cliente.

5. **LISTAR ARREGLO ATENCION**. Desplegar los valores almacenados en el arreglo correspondientes a la lista.

6. **LISTAR ARREGLO ESPERA**. Desplegar los valores almacenados en el arreglo correspondientes a la lista.

7. **SALIR.** Finalizar la ejecución del programa.

EJERCICIO 7.

El reino animal se clasifica en diferentes grupos de clases, subclases, especies y órdenes. Un modelo muy simplificado de clasificar al reino animal, sería dividiéndolos en dos Clases: Vertebrados e Invertebrados. Los Vertebrados se divide a su vez en Aves y Mamíferos (hay otras más que no serán consideradas) y los Invertebrados en Artrópodos y Moluscos (hay otras más que no serán consideradas). Dentro de las Aves se pueden distinguir entre aquellas que vuelan (Halcón, Cóndor, etc.) y las que no vuelan (Pingüino, Ñandú, etc.); de los Mamíferos, los Terrestres (León, Caballo, etc.) y Acuáticos (Ballena, Delfín, etc.); de los Artrópodos, los Arácnidos (Araña, Escorpión, etc.) e Insectos (Mosca, Hormiga, etc.); de los Moluscos, los Cefalópodos (Pulpo, Calamar, etc.) y Bivalvos (Ostra, Almeja, etc.). En la Figura de la siguiente hoja, se puede apreciar la clasificación antes hecha.

SE PIDE diseñar, codificar e implementar un programa que permita armar un árbol binario con los datos señalados en la figura y en base a ello realizar las siguientes consultas:

1. Conocer de qué Clase, Sub-Clase y Tipo pertenece un animal.

   Se deberá ingresar por teclado el nombre del animal, buscarlo en el árbol, determinar si existe o no y desplegar uno de los dos siguientes mensajes:

   - Si no existe el animal, desplegar: “Error, No existe el animal buscado.”
   - Si existe el animal, desplegar: La Clase, Sub-Clase y Tipo a la cual pertenece.

2. Listar todos los animales que pertenecen a la Clase, Sub-Clase o Tipo.

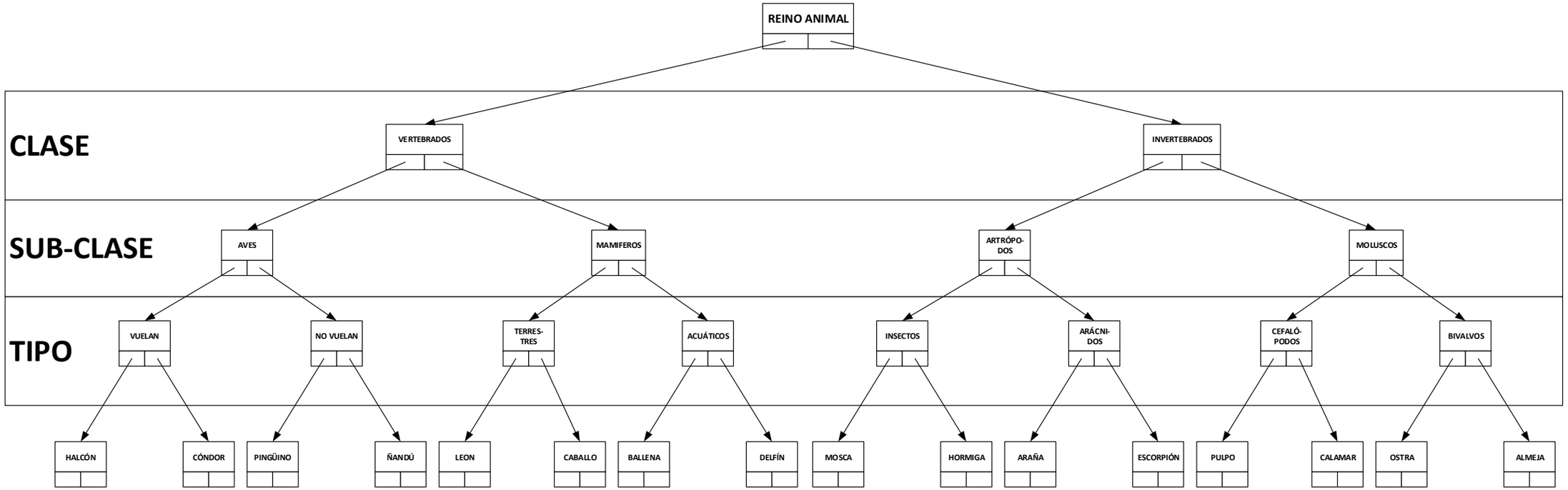
   Se deberá ingresar por teclado el nombre de la Clase, Sub-Clase o Tipo a listar, buscarlo en el árbol, determinar si existe o no y desplegar uno de los siguientes mensajes:

   - Si no existe, desplegar: “Error, No existe la Clase, Sub-Clase o Tipo buscado.”
   - Si existe, desplegar: Un listado de animales a los cuales pertenece.

Observaciones:

- Deberá implementar un procedimiento que permita crear el árbol con los mismos valores que aparecen en la figura (no haga un programa que los ingrese por teclado, simplemente escríbalos por código). La estructura de cada nodo debe tener al menos 3 campos: DATO, IZQ y DER, puede agregar otro(s) si lo estima necesario.

- Implemente un menú que ofrezca 3 alternativas: 1. Conocer Clasificación, 2. Conocer animales de una Clase, Sub-Clase o Tipo y 3. Salir


REINO ANIMAL
CLASE
SUB-CLASE
TIPO
VERTEBRADOS
INVERTEBRADOS
AVES
MAMIFEROS
ARTRÓPO-DOS
MOLUSCOS
VUELAN
NO VUELAN
TERRES-TRES
ACUÁTICOS
INSECTOS
ARÁCNI-DOS
CEFALÓ-PODOS
BIVALVOS
HALCÓN
CÓNDOR
PINGÜINO
ÑANDÚ
LEON
CABALLO
BALLENA
DELFÍN
MOSCA
HORMIGA
ARAÑA
ESCORPIÓN
PULPO
CALAMAR
OSTRA
ALMEJA