Proyecto 5a: Herencia

Simule la herencia de tipos de sangre para cada miembro de una familia.

Contexto

El tipo de sangre de una persona está dado por dos alelos (diferentes formas de un gen). Los alelos posibles son A, B y O, de los cuales cada persona tiene dos. Cada padre pasa aleatoriamente uno de sus dos alelos de tipo sanguíneo a su hijo. Las posibles combinaciones de tipos de sangre son: OO, OA, OB, AO, AA, AB, BO, BA y BB. Si uno de los padres tiene el tipo de sangre AO y el otro tiene el tipo de sangre BB, los posibles tipos de sangre del niño serían AB y OB, según el alelo que se reciba de cada padre. Y si uno de los padres tiene el tipo de sangre AO y el otro OB, los posibles tipos de sangre del niño serían AO, OB, AB y OO.

Empezando

Puedes hacer clic en el siguiente enlace **https://github.com/Davatec/Proyecto5a/herencia.c** donde podrás descargar la carpeta que contine el archivo **`herencia.c`**.

Comprensión

Observa la definición de un tipo llamado `persona`. Cada persona tiene un array de dos `padres`, cada uno de los cuales es un puntero a otra estructura `persona`. Cada persona tiene dos `alelos`, cada uno de los cuales es un carácter ('A', 'B' u 'O'). La función `main` brinda una entrada inicial a un generador de números aleatorios, que usaremos para generar alelos aleatorios. `main` llama a la función `crear_familia` para simular la creación de estructuras de personas para una familia de 3 generaciones (una persona, sus padres y sus abuelos). Luego llama a `imprimir_familia` para imprimir cada miembro de la familia y sus tipos de sangre. Finalmente, llama a `liberar_familia` para liberar cualquier memoria asignada previamente con `malloc`. Las funciones `crear_familia` y `liberar_familia` son las que escribirás.

Detalles de Implementación

Completa la implementación de `herencia.c`, para crear una familia de un tamaño generacional y asigna alelos de tipo de sangre a cada miembro de la familia. Para la generación más antigua asignar alelos al azar.

La función `crear_familia` toma como entrada un número entero (generaciones) y debe asignar una `persona` por cada miembro de la familia de ese número de generaciones, devolviendo un puntero a la `persona` de la generación más joven.

- Así, `crear_familia(3)` debería devolver un puntero a una persona con dos padres, donde cada padre también tiene dos padres.
- Cada persona debe tener asignados `alelos`. La generación más antigua debe tener alelos elegidos al azar, llamando la función `alelo_aleatorio`, y las generaciones más jóvenes deben heredar un alelo (elegido al azar) de cada padre.
- Cada `persona` debe tener `padres` asignados. La generación más antigua debe tener ambos padres configurados en NULL, y las generaciones más jóvenes deben tener padres como un array de dos punteros, cada uno apuntando a un padre diferente.

La función `crear_familia` está dividida en secciones “Por hacer” para que las completes.

- Primero, debes asignar memoria para una nueva persona. Puedes usar `malloc` para asignar memoria y `sizeof(persona)`para obtener la cantidad de bytes a asignar.
- Si `generaciones > 1`, entonces hay más generaciones que deben asignarse. Tu función debería establecer ambos `padres` llamando recursivamente a `crear_familia`. (¿Cuántas `generaciones` se

deben pasar como entrada a cada padre?) La función debe establecer ambos `alelos` eligiendo aleatoriamente un alelo de cada padre.

- De lo contrario, si `generaciones == 1`, no habrá datos de los padres de esta persona. Ambos padres deben establecerse en NULL y cada `alelo` debe generarse aleatoriamente.
- Finalmente, tu función debería devolver un puntero para la `persona` que fue asignada.

La función `liberar_familia` debe aceptar como entrada un puntero a una `persona`, liberar memoria para esa persona y liberar memoria de forma recursiva para todos sus antepasados.

- Al ser una función recursiva, primero debe manejar el caso base. Si la entrada a la función es NULL, no hay nada que liberar, y tu función puede retornar inmediatamente.
- De lo contrario, debes liberar recursivamente con `free` a ambos padres de la persona antes de liberar con `free` al hijo.

Consejos

- Puedes usar la función `rand()`para asignar alelos aleatoriamente. Esta función devuelve un número entero entre `0` y `RAND_MAX`, o `32767`.
    - Para generar un número pseudoaleatorio que sea `0` o `1`, puedes usar `rand() % 2`.
- Para asignar memoria para una persona, puedes usar `malloc(n)`, que toma un tamaño como argumento y asignará `n` bytes de memoria.
- Para acceder a una variable mediante un puntero, puedes utilizar la notación de flechas.
    - Así, si `p` es un puntero a una persona, `p->padres[0]` puede acceder a un puntero al primer padre de esta persona.

Cómo probar tu código

Ejecuta: `>./herencia`. El hijo debe tener dos alelos, uno de cada padre. Cada padre debe tener dos alelos, uno de cada uno de sus padres. En el siguiente ejemplo, el hijo de la `Generacion 0` recibió un alelo O de ambos padres de la `Generacion 1`. El primer padre recibió una A del primer abuelo y una O del segundo abuelo. Asimismo, el segundo padre recibió una O y una B de sus abuelos.

```
>./herencia

Generacion 0, tipo de sangre OO

      Generacion 1, tipo de sangre AO

      Generacion 2, tipo de sangre OA

      Generacion 2, tipo de sangre BO

Generacion 1, blood type OB

      Generacion 2, tipo de sangre AO

      Generacion 2, tipo de sangre BO
```