Este é o CS50x

OpenCourseWare

Doar (https://cs50.harvard.edu/donate)

David J. Malan (https://cs.harvard.edu/malan/)
malan@harvard.edu
(https://www.facebook.com/dmalan) (https://github.com/dmalan)
(https://www.instagram.com/davidjmalan/) (https://www.linkedin.com/in/malan/)
(https://orcid.org/0000-0001-5338-2522)
(https://www.quora.com/profile/David-J-Malan)
(https://www.reddit.com/user/davidjmalan) (https://twitter.com/davidjmalan)

Laboratório 5: herança

Você está convidado a colaborar com um ou dois colegas de classe neste laboratório, embora seja esperado que todos os alunos em qualquer um desses grupos contribuam igualmente para o laboratório.

O GitHub agora requer que você use SSH ou um token de acesso pessoal em vez de uma senha para fazer login, mas você ainda pode usar `check50` e `submit50`! Consulte **cs50.ly/github (https://cs50.ly/github)** para obter instruções, caso ainda não o tenha feito!

Simule a herança de tipos sanguíneos para cada membro de uma família.

```
$ ./inheritance
Generation 0, blood type OO
    Generation 1, blood type AO
        Generation 2, blood type OA
        Generation 2, blood type BO
    Generation 1, blood type OB
        Generation 2, blood type AO
        Generation 2, blood type BO
```

Quando fazer

No sábado, 1º de janeiro de 2022, 1h59 GMT-3 (https://time.cs50.io/2021-12-31T23:59:00-05:00).

Fundo

O tipo sanguíneo de uma pessoa é determinado por dois alelos (isto é, diferentes formas de um gene). Os três alelos possíveis são A, B e O, dos quais cada pessoa tem dois (possivelmente iguais, possivelmente diferentes). Cada um dos pais de uma criança passa aleatoriamente um de seus dois alelos de tipo sanguíneo para o filho. As combinações possíveis de tipo sanguíneo, então, são: OO, OA, OB, AO, AA, AB, BO, BA e BB.

Por exemplo, se um dos pais tem o tipo sanguíneo AO e o outro tem o tipo sanguíneo BB, os possíveis tipos sanguíneos da criança seriam AB e OB, dependendo de qual alelo é recebido de

cada pai. Da mesma forma, se um dos pais tem tipo de sangue AO e o outro OB, os possíveis tipos de sangue da criança seriam AO, OB, AB e OO.

Começando

Crie um novo diretório em seu IDE chamado `lab5`. Nesse diretório, execute `wget https://cdn.cs50.net/2020/fall/labs/5/inheritance.c` para baixar o código de distribuição para este projeto.

Entendimento

Dê uma olhada no código de distribuição em `inheritance.c`.

Observe a definição de um tipo chamado `person`. Cada pessoa tem um array de dois `parents`, cada um dos quais é um ponteiro para outra `person` estrutura. Cada pessoa tem também um conjunto de dois `alleles`, cada um dos quais é um `char` (ou `'A'`, `'B'`, ou `'O'`).

Agora, dê uma olhada na `main` função. A função começa "semeando" (ou seja, fornecendo alguma entrada inicial para) um gerador de números aleatórios, que usaremos mais tarde para gerar alelos aleatórios. A `main` função então chama a `create_family` função para simular a criação de `person` estruturas para uma família de 3 gerações (ou seja, uma pessoa, seus pais e seus avós). Em seguida, ligamos `print_family` para imprimir cada um desses membros da família e seus tipos de sangue. Finalmente, a função chama `free_family` a `free` qualquer memória que foi anteriormente alocada com `malloc`.

As funções `create_family` e `free_family` são deixadas para você escrever!

Detalhes de Implementação

Conclua a implementação de `inheritance.c`, de modo que crie uma família de um tamanho de geração especificado e atribua alelos de tipo sanguíneo a cada membro da família. A geração mais antiga terá alelos atribuídos aleatoriamente a eles.

- A `create_family` função recebe um inteiro (`generations`) como entrada e deve alocar (como via `malloc`) um `person` para cada membro da família daquele número de gerações, retornando um ponteiro para o `person` na geração mais jovem.
  - Por exemplo, `create_family(3)` deve retornar um ponteiro para uma pessoa com dois pais, onde cada pai também tem dois pais.
  - Cada um `person` deve ter `alleles` atribuído a eles. A geração mais velha deve ter alelos escolhidos aleatoriamente (como chamando a `random_allele` função), e as

    gerações mais jovens devem herdar um alelo (escolhido aleatoriamente) de cada pai.
  - Cada um `person` deve ter `parents` atribuído a eles. A geração mais velha deve ter `parents` definido como `NULL` e as gerações mais novas devem ter `parents` um array de dois ponteiros, cada um apontando para um pai diferente.

Dividimos a `create_family` função em alguns programas `TODO` para você completar.

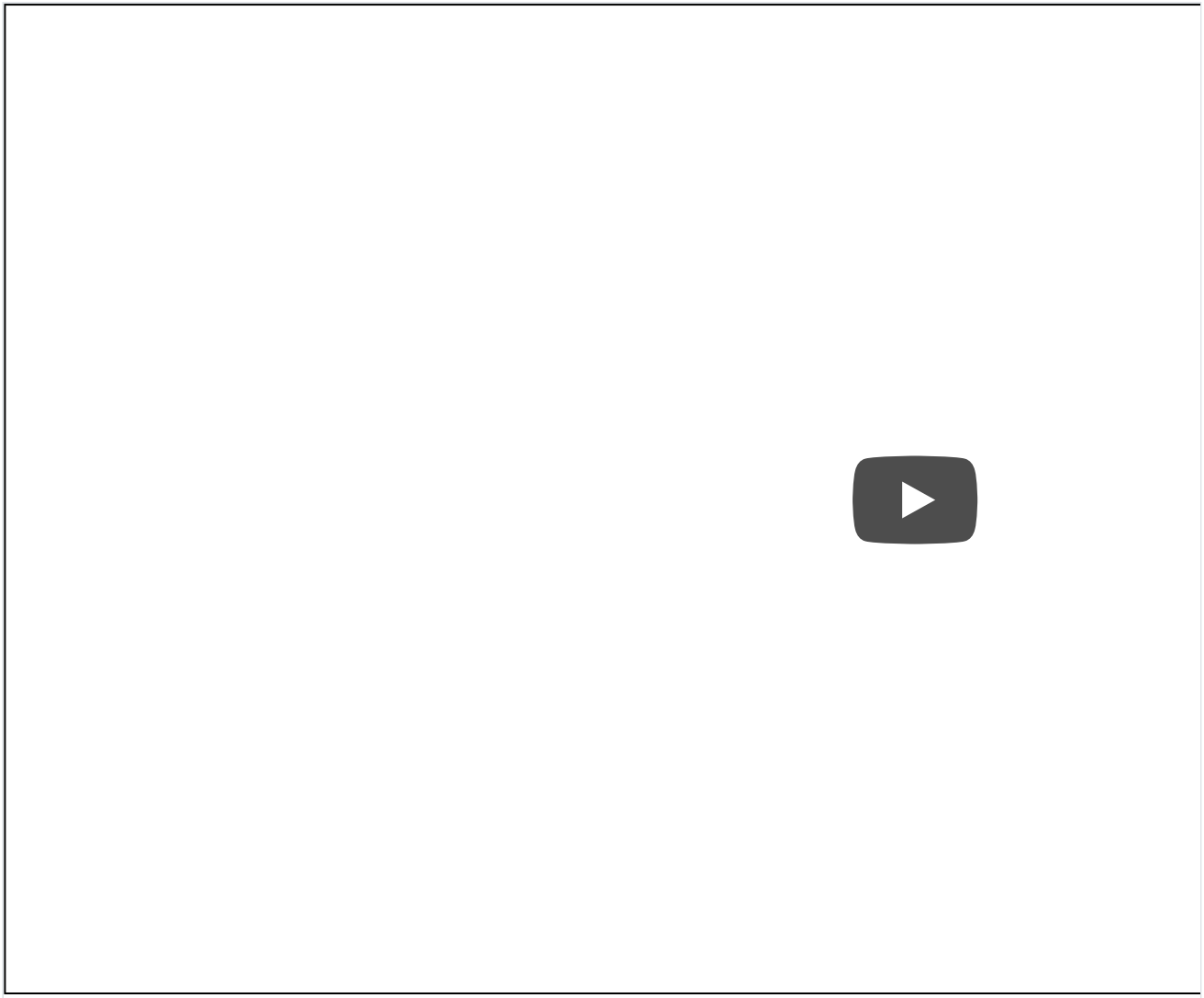
- Primeiro, você deve alocar memória para uma nova pessoa. Lembre-se de que você pode usar `malloc` para alocar memória e `sizeof(person)` para obter o número de bytes a serem alocados.
- A seguir, incluímos uma condição para verificar se `generations > 1` .
  - Se `generations > 1` , então, há mais gerações que ainda precisam ser alocadas. Sua função deve definir ambos `parents` chamando recursivamente `create_family` . (Quantos `generations` devem ser passados como entrada para cada pai?) A função deve então definir ambos `alleles` escolhendo aleatoriamente um alelo de cada pai.
  - Caso contrário (se `generations == 1`), não haverá dados dos pais para esta pessoa. Ambos `parents` devem ser definidos como `NULL` e cada um `allele` deve ser gerado aleatoriamente.
- Finalmente, sua função deve retornar um ponteiro para o `person` que foi alocado.

A `free_family` função deve aceitar como entrada um ponteiro para um `person` , memória livre para essa pessoa e, em seguida, memória livre recursivamente para todos os seus ancestrais.

- Como esta é uma função recursiva, você deve primeiro lidar com o caso base. Se o

- Como esta é uma função recursiva, você deve primeiro lidar com o caso base. Se a entrada para a função for `NULL`, então não há nada para liberar, então sua função pode retornar imediatamente.
- Caso contrário, você deve recursivamente `free` os pais da pessoa antes de `free` enviar a criança.

Passo a passo



Dicas

- Você pode achar a `rand()` função útil para atribuir alelos aleatoriamente. Esta função retorna um número inteiro entre `0` e `RAND_MAX`, ou `32767`.
  - Em particular, para gerar um número pseudoaleatório que seja `0` ou `1`, você pode usar a expressão `rand() % 2`.
- Lembre-se, para alocar memória para uma pessoa em particular, podemos usar `malloc(n)`, que toma um tamanho como argumento e alocará `n` bytes de memória.
- Lembre-se, para acessar uma variável por meio de um ponteiro, podemos usar a notação de seta.
  - Por exemplo, se `p` for um ponteiro para uma pessoa, um ponteiro para o primeiro pai dessa pessoa pode ser acessado por `p->parents[0]`.

Como testar seu código

Ao ser executado `./inheritance`, seu programa deve seguir as regras descritas em segundo plano. A criança deve ter dois alelos, um de cada pai. Cada um dos pais deve ter dois alelos, um de cada um de seus pais.

Por exemplo, no exemplo abaixo, a criança na Geração 0 recebeu um alelo O de ambos os pais da Geração 1. O primeiro pai recebeu um A do primeiro avô e um O do segundo avô. Da mesma forma, o segundo pai recebeu um O e um B de seus avós.

```
$ ./inheritance
Generation 0, blood type OO
    Generation 1, blood type AO
        Generation 2, blood type OA
        Generation 2, blood type BO
    Generation 1, blood type OB
        Generation 2, blood type AO
        Generation 2, blood type BO
```

▶ **Não sabe como resolver?**

Execute o seguinte para avaliar a exatidão do seu código usando `check50`. Mas certifique-se de compilar e testar você mesmo!

```
check50 cs50/labs/2021/x/inheritance
```

Execute o seguinte para avaliar o estilo do seu código usando `style50`.

```
style50 inheritance.c
```

Como enviar

Execute o seguinte para enviar seu trabalho.

```
submit50 cs50/labs/2021/x/inheritance
```