Isso é CS50

Introdução do CS50 à Ciência da Computação

OpenCourseWare

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

David J. Malan (https://cs.harvard.edu/malan/)
malan@harvard.edu
(https://www.clubhouse.com/@davidjmalan) (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://www.tiktok.com/@davidjmalan) (https://davidjmalan.t.me/)
(https://twitter.com/davidjmalan)

Lab 5: Inheritance

You are welcome to collaborate with one or two classmates on this lab, though it is expected that every student in any such group contribute equally to the lab.

Simulate the inheritance of blood types for each member of a family.

```
$ ./inheritance
Child (Generation 0): blood type OO
    Parent (Generation 1): blood type AO
        Grandparent (Generation 2): blood type OA
        Grandparent (Generation 2): blood type BO
    Parent (Generation 1): blood type OB
        Grandparent (Generation 2): blood type AO
        Grandparent (Generation 2): blood type BO
```

Background

A person's blood type is determined by two alleles (i.e., different forms of a gene). The three possible alleles are A, B, and O, of which each person has two (possibly the same, possibly different). Each of a child's parents randomly passes one of their two blood type alleles to their child. The possible blood type combinations, then, are: OO, OA, OB, AO, AA, AB, BO, BA, and BB.

For example, if one parent has blood type AO and the other parent has blood type BB, then the child's possible blood types would be AB and OB, depending on which allele is received from each parent. Similarly, if one parent has blood type AO and the other OB, then the child's possible blood types would be AO, OB, AB, and OO.

Getting Started

Open VS Code (https://code.cs50.io/).

Start by clicking inside your terminal window, then execute `cd` by itself. You should find that its "prompt" resembles the below.

```
$
```

Click inside of that terminal window and then execute

```
wget https://cdn.cs50.net/2022/fall/labs/5/inheritance.zip
```

followed by Enter in order to download a ZIP called `inheritance.zip` in your codespace. Take care not to overlook the space between `wget` and the following URL, or any other character for that matter!

Now execute

```
unzip inheritance.zip
```

to create a folder called `inheritance`. You no longer need the ZIP file, so you can execute

```
rm inheritance.zip
```

and respond with "y" followed by Enter at the prompt to remove the ZIP file you downloaded.

Now type

```
cd inheritance
```

followed by Enter to move yourself into (i.e., open) that directory. Your prompt should now resemble the below.

```
inheritance/ $
```

If all was successful, you should execute

```
ls
```

and you should see `inheritance.c`.

If you run into any trouble, follow these same steps again and see if you can determine where you went wrong!

Understanding

Take a look at the distribution code in `inheritance.c`.

Notice the definition of a type called `person`. Each person has an array of two `parents`, each of which is a pointer to another `person` struct. Each person also has an array of two `alleles`, each of which is a `char` (either `'A'`, `'B'`, or `'O'`).

Now, take a look at the `main` function. The function begins by "seeding" (i.e., providing some initial input to) a random number generator, which we'll use later to generate random alleles. The `main` function then calls the `create_family` function to simulate the creation of `person` structs for a family of 3 generations (i.e. a person, their parents, and their grandparents). We then call `print_family` to print out each of those family members and their blood types. Finally, the function calls `free_family` to `free` any memory that was previously allocated with `malloc`.

The `create_family` and `free_family` functions are left to you to write!

Implementation Details

Complete the implementation of `inheritance.c`, such that it creates a family of a specified generation size and assigns blood type alleles to each family member. The oldest generation will have alleles assigned randomly to them.

- The `create_family` function takes an integer (`generations`) as input and should allocate (as via `malloc`) one `person` for each member of the family of that number of generations, returning a pointer to the `person` in the youngest generation.
  - For example, `create_family(3)` should return a pointer to a person with two parents, where each parent also has two parents.
  - Each `person` should have `alleles` assigned to them. The oldest generation should have alleles randomly chosen (as by calling the `random_allele` function), and younger generations should inherit one allele (chosen at random) from each parent.
  - Each `person` should have `parents` assigned to them. The oldest generation should have both `parents` set to `NULL`, and younger generations should have `parents` be an array of two pointers, each pointing to a different parent.

We've divided the `create_family` function into a few `TODO`s for you to complete.

- First, you should allocate memory for a new person. Recall that you can use `malloc` to allocate memory, and `sizeof(person)` to get the number of bytes to allocate.
- Next, we've included a condition to check if `generations > 1`.
  - If `generations > 1`, then there are more generations that still need to be allocated. We've already created two new `parents`, `parent0` and `parent1`, by recursively calling `create_family`. Your `create_family` function should then set the parent pointers of the new person you created. Finally, assign both `alleles` for the new person by randomly choosing one allele from each parent.
  - Otherwise (if `generations == 1`), then there will be no parent data for this person. Both `parents` of your new person should be set to `NULL`, and each `allele` should be generated randomly.

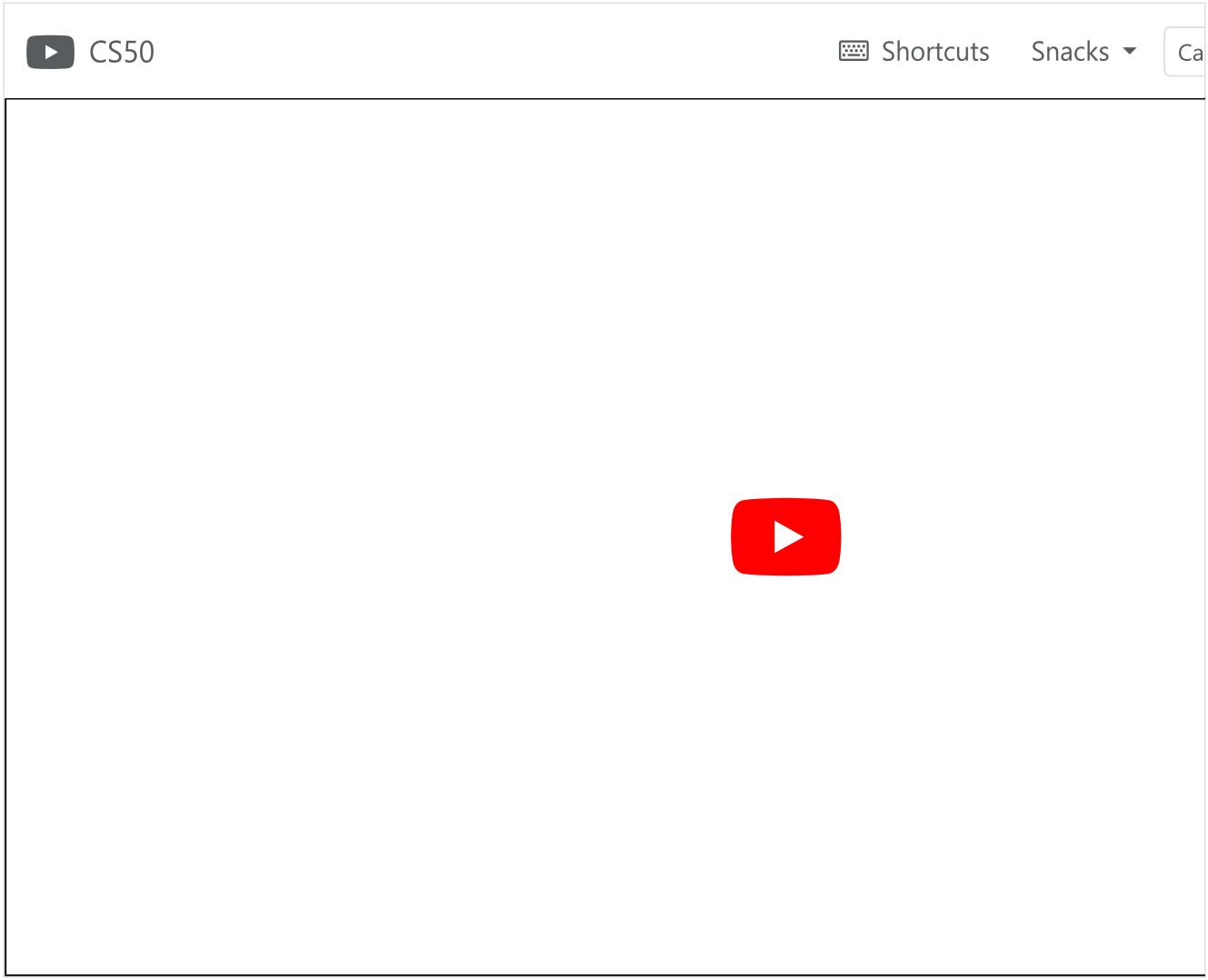
- Finally, your function should return a pointer for the `person` that was allocated.

The `free_family` function should accept as input a pointer to a `person`, free memory for that person, and then recursively free memory for all of their ancestors.

- Since this is a recursive function, you should first handle the base case. If the input to the function is `NULL`, then there's nothing to free, so your function can return immediately.
- Otherwise, you should recursively `free` both of the person's parents before `free`ing the child.

Walkthrough

This video was recorded when the course was still using CS50 IDE for writing code. Though the interface may look different from your codespace, the behavior of the two environments should be largely similar!



Hints

- Você pode achar a `rand()` função útil para atribuir alelos aleatoriamente. Esta função retorna um número inteiro entre `0` e `RAND_MAX`, ou `2147483647`.
  - 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 determinada pessoa, podemos usar `malloc(n)`, que recebe 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]`.

▶ **Não sabe como resolver?**

Como testar seu código

Ao executar `./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 dos 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
Child (Generation 0): blood type OO
    Parent (Generation 1): blood type AO
        Grandparent (Generation 2): blood type OA
        Grandparent (Generation 2): blood type BO
    Parent (Generation 1): blood type OB
        Grandparent (Generation 2): blood type AO
        Grandparent (Generation 2): blood type BO
```

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

```
check50 cs50/labs/2023/x/inheritance
```

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

```
style50 inheritance.c
```

Como enviar

Em seu terminal, execute o abaixo para enviar seu trabalho.

```
submit50 cs50/labs/2023/x/inheritance
```