**[Lab 5: Inheritance](https://cs50.harvard.edu/x/2021/labs/5/" \l "lab-5-inheritance)**

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

$ ./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

[**When to Do It**](https://cs50.harvard.edu/x/2021/labs/5/#when-to-do-it)

By Sat, Jan 1, 2022, 6:59 AM GMT+2.

[**Background**](https://cs50.harvard.edu/x/2021/labs/5/#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**](https://cs50.harvard.edu/x/2021/labs/5/#getting-started)

Create a new directory in your IDE called lab5. In that directory, execute wget https://cdn.cs50.net/2020/fall/labs/5/inheritance.c to download the distribution code for this project.

[**Understanding**](https://cs50.harvard.edu/x/2021/labs/5/#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**](https://cs50.harvard.edu/x/2021/labs/5/#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 TODOs 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. Your function should set both parents by recursively calling create\_family. (How many generations should be passed as input to each parent?) The function should then set both alleles by randomly choosing one allele from each parent.
  + Otherwise (if generations == 1), then there will be no parent data for this person. Both parents 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 freeing the child.

[**Hints**](https://cs50.harvard.edu/x/2021/labs/5/#hints)

* You might find the rand() function useful for randomly assigning alleles. This function returns an integer between 0 and RAND\_MAX, or 32767.
  + In particular, to generate a pseudorandom number that is either 0 or 1, you can use the expression rand() % 2.
* Remember, to allocate memory for a particular person, we can use malloc(n), which takes a size as argument and will allocate n bytes of memory.
* Remember, to access a variable via a pointer, we can use arrow notation.
  + For example, if p is a pointer to a person, then a pointer to this person’s first parent can be accessed by p->parents[0].

[**How to Test Your Code**](https://cs50.harvard.edu/x/2021/labs/5/#how-to-test-your-code)

Upon running ./inheritance, your program should adhere to the rules described in the background. The child should have two alleles, one from each parent. The parents should each have two alleles, one from each of their parents.

For example, in the example below, the child in Generation 0 received an O allele from both Generation 1 parents. The first parent received an A from the first grandparent and a O from the second grandparent. Similarly, the second parent received an O and a B from their grandparents.

$ ./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

Not sure how to solve?

Execute the below to evaluate the correctness of your code using check50. But be sure to compile and test it yourself as well!

check50 cs50/labs/2021/x/inheritance

Execute the below to evaluate the style of your code using style50.

style50 inheritance.c

[**How to Submit**](https://cs50.harvard.edu/x/2021/labs/5/#how-to-submit)

Execute the below, logging in with your GitHub username and password when prompted. For security, you’ll see asterisks (\*) instead of the actual characters in your password.

submit50 cs50/labs/2021/x/inheritance