```c
// Simulate genetic inheritance of blood type

#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// Each person has two parents and two alleles
typedef struct person
{
    struct person *parents[2];
    char alleles[2];
}
person;

const int GENERATIONS = 3;
const int INDENT_LENGTH = 4;

person *create_family(int generations);
void print_family(person *p, int generation);
void free_family(person *p);
char random_allele();

int main(void)
{
    // Seed random number generator
    srand(time(0));

    // Create a new family with three generations
    person *p = create_family(GENERATIONS);

    // Print family tree of blood types
    print_family(p, 0);

    // Free memory
    free_family(p);
}

// Create a new individual with `generations`
person *create_family(int generations)
{
    // TODO: Allocate memory for new person
    person *youngest = malloc(sizeof(person));

    // If there are still generations left to create
    if (generations > 1)
    {
        // Create two new parents for current person by recursively calling
create_family
        person *parent0 = create_family(generations - 1);
        person *parent1 = create_family(generations - 1);

        // TODO: Set parent pointers for current person

        youngest->parents[0] = parent0;
        youngest->parents[1] = parent1;

        // TODO: Randomly assign current person's alleles based on the alleles of
their parents
        int r = rand() % 2;
        youngest->alleles[0] = parent0->alleles[r];
        // Rerandomize the r, make it independent from previous one
```

```
        r = rand() % 2;
        youngest->alleles[1] = parent1->alleles[r];
    }

    // If there are no generations left to create
    else
    {
        // TODO: Set parent pointers to NULL
        youngest->parents[0] = NULL;
        youngest->parents[1] = NULL;

        // TODO: Randomly assign alleles
        youngest->alleles[0] = random_allele();
        youngest->alleles[1] = random_allele();

    }

    // TODO: Return newly created person
    return youngest;
    return NULL;
}

// Free `p` and all ancestors of `p`.
void free_family(person *p)
{
    // TODO: Handle base case
    if (p == NULL)
    {
        return;
    }

    // TODO: Free parents recursively
    if (p->parents[0] != NULL)
    {
        free_family(p->parents[0]);
        free_family(p->parents[1]);
    }

    // TODO: Free child
    free(p);
}

// Print each family member and their alleles.
void print_family(person *p, int generation)
{
    // Handle base case
    if (p == NULL)
    {
        return;
    }

    // Print indentation
    for (int i = 0; i < generation * INDENT_LENGTH; i++)
    {
        printf(" ");
    }

    // Print person
    if (generation == 0)
    {
        printf("Child (Generation %i): blood type %c%c\n", generation, p->alleles[0],
p->alleles[1]);
```

```
    }
    else if (generation == 1)
    {
        printf("Parent (Generation %i): blood type %c%c\n", generation, p-
>alleles[0], p->alleles[1]);
    }
    else
    {
        for (int i = 0; i < generation - 2; i++)
        {
            printf("Great-");
        }
        printf("Grandparent (Generation %i): blood type %c%c\n", generation, p-
>alleles[0], p->alleles[1]);
    }

    // Print parents of current generation
    print_family(p->parents[0], generation + 1);
    print_family(p->parents[1], generation + 1);
}

// Randomly chooses a blood type allele.
char random_allele()
{
    int r = rand() % 3;
    if (r == 0)
    {
        return 'A';
    }
    else if (r == 1)
    {
        return 'B';
    }
    else
    {
        return 'O';
    }
}
```