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

// Struct onde cada pessoa tem 2 pais e 2 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)
{
    // Gera numeros aleatórios.
    srand(time(0));

    // Cria uma nova familia com 3 gerações.
    person *p = create_family(GENERATIONS);

    // Imprime a familia e seus respectivos tipos sanguineos.
    print_family(p, 0);

    // Libera a memória.
    free_family(p);
}

// Cria a familia.
person *create_family(int generations)
{
    // Aloca memória para uma nova pessoa.
    person *new = malloc(sizeof(person));

    // Se não for a primeira geração.
    if (generations > 1)
    {
        // Crie o pai e a mãe dessa pessoa, reiniciando essa função 1
geração a menos.
        person *pai = create_family(generations - 1);
        person *mae = create_family(generations - 1);
```

```
        // Após criar os pais, adiciona cada um a 1 lugar no array.
        new->parents[0] = pai;
        new->parents[1] = mae;

        // Gera um número aleatório que define qual allelo de cada pai
deve ser escolhido.
        new->alleles[0] = new->parents[0]->alleles[rand() % 2];
        new->alleles[1] = new->parents[1]->alleles[rand() % 2];
    }

    // Se for a primeira geraçao
    else
    {
        // Define os pais dessa criança como nulo, pois não existe outra
geração.
        new->parents[0] = NULL;
        new->parents[1] = NULL;

        // Escolhe o tipo sanguineo desse ser aleatóriamente.
        new->alleles[0] = random_allele();
        new->alleles[1] = random_allele();
    }

    // Retorna a pessoa criada para a malha de informações.
    return new;
}

// Devolve a memória alocada.
void free_family(person *p)
{
    // Se "p" for nullo, retorna.
    if(p == NULL)
    {
        return;
    }

    // Chama essa função para cada pai, até achar a ultima geração e
apaga de tras para frente.
    free_family(p->parents[0]);
    free_family(p->parents[1]);

    // Devolve a memória da criança em questão.
    free(p);

}

// Imprime a família, função já pronta.
void print_family(person *p, int generation)
```

```c
{
    // Handle base case
    if (p == NULL)
    {
        return;
    }

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

    // Print person
    printf("Generation %i, blood type %c%c\n", generation, p->alleles[0],
p->alleles[1]);
    print_family(p->parents[0], generation + 1);
    print_family(p->parents[1], generation + 1);
}

// Aleatóriamente escolhe o tipo sanguineo, função já pronta.
char random_allele()
{
    int r = rand() % 3;
    if (r == 0)
    {
        return 'A';
    }
    else if (r == 1)
    {
        return 'B';
    }
    else
    {
        return 'O';
    }
}
```