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

// DONOT edit this
int randint(int min, int max) {
    return rand() % (max - min)  + min;
}

long factorial(short n);

// DONOT edit this
bool test_factorial() {
    short n = randint(1, 25);
    if (factorial(n) != n*factorial(n-1)) {
        fprintf(stderr, "Fail for factorial(%d)\n", n);
        return false;
    } else
        return true;
}

// Please EDIT this. DONOT change the function signature
// TODO: Write this recursive function in C
long factorial(short n) {
    if(n >= 1){ //Base case
      return n*factorial(n-1);
    }
    else{
      return 1;
    }


}

int main(int argc, char** argv) {
    if (factorial(0) != 1)
      fprintf(stderr, "Fail for factorial(0) == 1\n");
    if (factorial(3) != 6)
      fprintf(stderr, "Fail for factorial(3) == 6\n");
    if (factorial(5) != 120)
      fprintf(stderr, "Fail for factorial(5) == 120\n");

    short i;
    for (i = 0; i < 10; ++i) {
        if (test_factorial())
          printf("%d: pass\n", i);
        else
          fprintf(stderr, "%d: fail\n", i);
    }
}
```

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

// DONOT edit this
int randint(int min, int max) {
    return rand() % (max - min)  + min;
}

// EDIT this function. DONOT edit the signature of the function
bool is_prime(long n, long* factor_p) {
    //Special case for 1 and 2
    if (n <= 2){
        return (n == 2) ? true : false;
    }
    //Static flag variable for recursion
    static int started = 0;
    if (!started){
        //New n being checked, start factor at 2
        *factor_p = 2;
        started = 1;
    }
    //Divisor with no remainder, that's a factor! Composite!
    if (n % *factor_p == 0){
        started = 0;
        return 0;
    }
    //Only check factors below sqrt(n)
    if (*factor_p * *factor_p > n){
        started = 0;
        return 1;
    }
    //Increment factor
    ++*factor_p;
    return is_prime(n, factor_p);
}

// DONOT edit this function
bool test_prime() {
    long factor;
    int n = rand();
    if (is_prime(n, &factor)) {
        if (n % factor == 0) {
            fprintf(stderr, "Fail for is_prime(%d, %ld)\n", n, factor);
            return false;
        } else {
            return true;
        }
    } else {
        int i;
        // Try 10 random factors
        for (i = 0; i < 20; ++i){
            factor = randint(2, n/2);
            if (n % factor == 0) {
                fprintf(stderr, "Fail for is_prime(%d, %ld)\n", n, factor);
                return false;
            }
        }
        return true;
    }
}

int main(int argc, char** argv) {
  long factor;
#define ARRSIZE 10
  long primes[ARRSIZE] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29 };
  short i;
  for (i = 0; i < ARRSIZE; ++i)
```

```
      if (is_prime(primes[i], &factor) != true)
          fprintf(stderr, "Fail for is_prime(%ld)\n", primes[i]);
      else
          printf("Pass\n");

  long composites[ARRSIZE] = {1, 4, 6, 8, 10, 14, 21, 25, 27, 33 };
  for (i = 0; i < ARRSIZE; ++i)
      if (is_prime(composites[i], &factor) != false)
          fprintf(stderr, "Fail for is_prime(%ld)\n", composites[i]);
      else
          printf("Pass\n");

  for (i = 0; i < 10; ++i) {
      if (test_prime())
          printf("%d: pass\n", i);
      else
          fprintf(stderr, "%d: fail\n", i);
  }
}
```

```
// Name: Landyn Francis
// Class: ECE490
// File: days.c
// Purpose: The main function calls the days() function which takes a
// date, passed a struct with 3 member ints representing month, day,
// and year, and determines the number of days since
// The Turn of the Century(1/1/2000). This program uses the assumption
// that each year has 360 days, and each month has 30 days.

#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>

//Date Struct
struct date {
    int month;
    int day;
    int year;
};

long days(struct date testDate);

int main(){
    //Test today
    struct date Test;
    Test.year = 2023;
    Test.month = 1;
    Test.day = 20;
    printf("Days since today = %ld\n",days(Test));

    // Test every day in the century
    // Start with the turn of the century
    Test.year = 2000;
    Test.month = 1;
    Test.day = 1;
    //Keep track of true days
    long count = 0;
    while (Test.year != 2100){
        Test.day++; //Increment day
        count++;
        if (Test.day == 31){ //Increment month
            Test.month++;
            Test.day = 1;
        }
        if (Test.month == 13){ //Increment year
            Test.year++;
            Test.month = 1;
        }
        if (count == days(Test)){
            printf("Date: %d/%d/%d\tdays(Test): %ld\n",Test.month,Test.day,Test.year,days(T
est));
            continue;
        }
        else{
            printf("Failed at %d/%d/%d\nCount=%ld \t days(Test)=%ld\n",Test.month,Test.day,
Test.year,count,days(Test));
            return -1;
        }


    }
    printf("Pass!\n");
}
//Determine the number of days from the turn of the century for any date within the century
 (2000<=Year<3000)
long days(struct date testDate){
    //Make a struct for the turn of the century
    struct date turn;
```

```
    turn.month = 1;
    turn.day = 1;
    turn.year = 2000;
    long days = 0;
    //Determine how many years, months, and day difference there are between the dates, and
 add them up.
    days += (testDate.year - turn.year)*360;
    days += (testDate.month - turn.month)*30;
    days += (testDate.day - turn.day);
    return days;
}
```