

# CS 137: Assignment #2

Due on Friday, Sep 27, 2024, at 11:59 PM

Submit all programs using the Marmoset Submission and Testing Server located at  
<https://marmoset.student.cs.uwaterloo.ca/>

*Victoria Sakhnini*

Fall 2024

## Notes:

- Use the examples to guide your formatting for your output. Remember to terminate your output with a newline character.
- You can use only syntax/language features we have covered so far up to the end of M4.
- You must NOT use MATH Library
- Use only `int` type of variables.

## Problem 1

This is a straightforward version of the classic game "Snake". A snake moves only from left to right, growing itself whenever it eats the bait. The snake is on a line of  $n$  discrete spaces. At each time point, the snake moves one space.

Write a C program `snake.c` that reads a natural number  $n$  (at least 5) and prints the line that the snake is on as it moves. Assume the snake has an initial body length of 2 and its head is at position 3 (indices start from 0). The line has  $n$  spaces, and there are baits on every odd space that the snake hasn't visited yet. Print the subsequent states in new lines; the game ends when the snake's head reaches the end of the line. Denote snake's head by "H", body by "X", bait by ".", empty space character by "\_".

For  $n=5$ , the output will look like the following (each line ends with `\n` character):

```
_XXH_
_XXH_
```

For  $n=15$ , the output will look like the following (each line ends with `\n` character):

```
_XXH_._._._._
_XXH._._._._
_XXXH._._._._
_XXXH._._._._
_XXXXH._._._._
_XXXXH._._._._
_XXXXH._._._._
_XXXXH._._._._
_XXXXH._._._._
_XXXXH._._._._
_XXXXH._._._._
_XXXXH._._._._
_XXXXH._._._._
_XXXXH._._._._
_XXXXH._._._._
```

## Problem 2

a) Create the file `functions.h`, which contains the following declarations:

I) `void square(int w);`

II) `void spiral(int w);`

III) `void rotation(int w);`

**Assume  $w > 0$**

b) Implement all the functions above (explained below) in the file `functions.c`

Note: You are to submit this file (along with `functions.h` file) containing only your implemented functions (that is, you must delete the test cases portion and the `main` function). However, **you must keep the required included libraries.**

c) **Submit** `functions.zip`, which contains the files `functions.c`, and `functions.h`

Here are the objectives of the three functions:

Assume  $w > 1$ .

I) `void square(int w);` prints the numbers  $1, 2, \dots, w^2$  in a square shape (of size  $w \times w$ ) on  $w$  lines of output

Examples:

`void square(2)` prints:

```
1 2
3 4
```

`void square(5)` prints:

```
1 2 3 4 5
6 7 8 9 10
11 12 13 14 15
16 17 18 19 20
21 22 23 24 25
```

`void square(6) prints:`

```
1 2 3 4 5 6
7 8 9 10 11 12
13 14 15 16 17 18
19 20 21 22 23 24
25 26 27 28 29 30
31 32 33 34 35 36
```

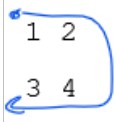
II) `void spiral(int w);` prints the numbers  $1, 2, \dots, w^2$  in a “spiral” order (compared with the square shape from part I above) on one line.

Examples:

`void spiral(2) prints:`

```
1 2 4 3
```

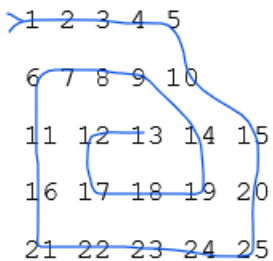
Because



`void spiral(5) prints:`

```
1 2 3 4 5 10 15 20 25 24 23 22 21 16 11 6 7 8 9 14 19 18 17 12 13
```

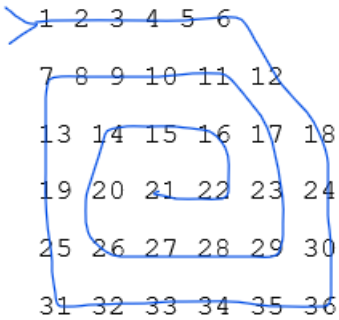
Because:



`void spiral(6) prints:`

```
1 2 3 4 5 6 12 18 24 30 36 35 34 33 32 31 25 19 13 7 8 9 10 11 17 23
29 28 27 26 20 14 15 16 22 21
```

Because:



III) `void rotation(int w);` prints the numbers  $1, 2, \dots, w^2$  in a “rotation” order on one line (compared with the square shape). Each rotation is 90 degrees for some incremental offsets.

Examples:

`void rotation(2) prints:`

```
1 2 4 3
```

`void rotation(5) prints:`

```
1 5 25 21 2 10 24 16 3 15 23 11 4 20 22 6 7 9 19 17 8 14 18 12 13
```

`void rotation(6) prints:`

```
1 6 36 31 2 12 35 25 3 18 34 19 4 24 33 13 5 30 32 7 8 11 29 26 9 17
28 20 10 23 27 14 15 16 22 21
```

Explore the pattern before you start coding

## Notes:

Each number printed is followed by a space (including the last number in each line).

Each line ends with `\n` character

### Problem 3

a) Create the file `fun.h`, which contains the following declarations:

I) `int isSophieGermainPrime(int p);`

II) `int base2nat(int bs, int num);`

III) `int nat2base(int bs, int num);`

b) Implement all the functions above (explained below) in the file `fun.c`

Note: You are to submit this file (along with `fun.h` file) containing only your implemented functions (that is, you must delete the test cases portion and the main function). However, **you should keep the required included libraries.**

c) Submit `fun.zip` which contains the files `fun.c`, and `fun.h`

I)

Definition: A *Sophie Germain<sup>1</sup> prime* is a [positive] prime number  $p$  such that  $2p + 1$  is also a prime number. For example, 2 is a Sophie Germain prime since both 2 and  $2(2) + 1 = 5$  are prime numbers.

Task: Create the function

```
int isSophieGermainPrime(int n)
```

which determines if an integer  $n$  is a Sophie Germain prime. The function should return 1 if  $n$  is a Sophie Germain prime and 0 otherwise.

Assumptions: You may assume that the values entered are valid integers such that the magnitude of  $2p + 1$  is at most  $10^9$ .

---

#### Fast Facts: Sophie Germain

**Known For:** French mathematician, physicist, and philosopher specializing in elasticity theory and number theory.

**Also Known As:** Marie-Sophie Germain

**Born:** April 1, 1776, in Rue Saint-Denis, Paris, France

**Died:** June 27, 1831, in Paris, France

**Education:** École Polytechnique

**Awards and Honors:** Number theory named after her, such as Sophie Germain prime, Germain curvature, and Sophie Germain's identity. The Sophie Germain Prize is awarded annually by the *Foundation Sophie Germain*.

1

## II)

When you see a number such as 734, it is generally assumed that you are using the base 10 number system (also known as the decimal system). That is:

$$734 = 7 \cdot 10^2 + 3 \cdot 10^1 + 4 \cdot 10^0$$

It is, however, possible to use any number as a base. For example, assuming we are in a base 5 number system, the notation 2301 would generate the decimal number 326:

$$2 \cdot 5^3 + 3 \cdot 5^2 + 0 \cdot 5^1 + 1 \cdot 5^0 = 326 \text{ (this equation is in decimal)}$$

Note that when using base 10 we have precisely 10 unique digits for each position, that is 0,1,2,3,4,5,6,7,8,9. Similarly, base 5 only allows for 5 digits 0,1,2,3,4 (To represent the "normal" (i.e. decimal) value 5 in base 5 we would write 10; 6 would be represented as 11, 7 would be 12 etc. To see this, consider 12 (in base 5), which means we compute  $1 \cdot 5^1 + 2 \cdot 5^0$  to get 7).

Task: Create the function

```
// pre: 1<bs<10 and num>0 a valid integer in base bs
```

```
int base2nat(int bs, int num)
```

which returns a positive integer representing the decimal value of `num` (`num` is in base `bs`).

Assumptions: You may assume that the values entered are valid integers such that the magnitude of `num` is at most  $10^9$ .

## III)

```
// pre: 1<bs<10 and num>0
```

```
int nat2base(int bs, int num);
```

It takes a base (`bs`) and a non-negative integer (`num`) in decimals and returns the value in base `bs`.

The following code will help you with testing

```
1. #include <stdio.h>
2. #include <assert.h>
3. #include "fun.h "
4.
5. int main(void) {
6.     assert(isSophieGermainPrime(11));
7.     assert(isSophieGermainPrime(41));
8.     assert(base2nat(5,23114)==1659);
9.     assert(base2nat(7,1)==1);
10.    assert(base2nat(3,1211012)==1328);
11.    assert(base2nat(8,715)==461);
12.    assert(nat2base(5,1659)==23114);
13.    assert(nat2base(9,1331)==1738);
14. }
```