# Assignment 4
# Bit Vectors and Primes

Prof. Max Dunne
CSE 13S

## 1 Introduction

We are going to answer some simple but fundamental questions about the sequence of natural numbers $\mathbb{N} = 1, 2, 3, \ldots$ We are going to look at primality and some other curious types of primes. Primes numbers are fundamental in mathematics and Computer Science. You should also be concerned with the execution time of your programs.

### 1.1 Prime Numbers

Prime numbers are among the most interesting of the natural numbers. A number $p \in \mathbb{N}$ is *prime* if it is evenly divisible by only 1 and $p$. That means that $(\forall m \in \mathbb{N}, 1 < m < p)\, m \nmid p$ or alternatively, $(\forall m \in \mathbb{N}, 1 < m < p)\, p \bmod m \neq 0$. The first prime number is 2, all primes except 2 must be odd, since all even numbers are divisible by 2. There are an infinite number of primes, which was proved by Euclid about 300 B.C. There is no formula for finding the primes, but the *prime number theorem* tells us that the probability of a given $m \in \mathbb{N}, 1 < m \leq N$ being prime is very close to $\frac{1}{\ln N}$, since the number of primes less than $N$, denoted $\pi(N)$, is

$$\frac{N}{\ln N - (1-\epsilon)} < \pi(N) < \frac{N}{\ln N - (1+\epsilon)}.$$

Determining whether a number $n$ is prime and if it evenly divides by $n$ is conceptually simple, all that must be done is to try every $k \in \{2, \ldots, n-1\}$. The execution time for this method is $O(n)$, which for large $n$ is prohibitive. You are invited to try this for a relatively small number $2^{64} - 1$. A little thought shows that we do not need to check so many, and that evaluating $k \mid n$ for $k \in \{2, \ldots, \lceil \sqrt{n} \rceil\}$ is sufficient. The resulting execution time of $O(\sqrt{n})$ is a great improvement. Is that the best that we can hope for? No, but a lower bound is *unknown*.

All natural numbers that are not prime are called *composite*. The *fundamental theorem of arithmetic,* also called the *unique factorization theorem*, states that every integer $m > 1$ is either prime or a unique product of primes $(p_0, \ldots, p_k)$. That is,

$$m = p_0^{\alpha_0} \times p_1^{\alpha_1} \times \ldots \times p_k^{\alpha_k} = \prod_{i=0}^{k} p_i^{\alpha_k}.$$

For example, $83736 = 2^3 \times 3^2 \times 1163$. In 1801, the fundamental theorem of arithmetic was proved by Carl Friedrich Gauss in his book *Disquisitiones Arithmeticæ*.

 

## 1.2 Positional Number Systems

A positional number system is one that has its roots in the Babylonian numeral system (base 60) dating back to around 2000 B.C. – which is why we have 60 seconds in a minute, 60 minutes in an hour, 360 degrees in a circle ($1° = 60'$). The modern number system which is most familiar to us is decimal (base 10), also known as the Hindu-Arabic system. This is the most commonly used base for counting, and stems from the fact that most humans have ten fingers. In the positional number system, a number is represented asc an ordered set of digits, where the contribution of each digit depends on its position. Each digit is multiplied by a value which determines its contribution to the number. A any natural number

$$N = a_k b^k + a_{k-1} b^{k-1} + \ldots + a_1 b^1 + a_0 b^0$$

has $b$ as its base for each digit (such as 2 (*binary*) or 16 (*hexadecimal*)), and values on the range of $a_i \in \{0, 1, \ldots, b-1\}$. $k$ represents the highest-ordered digit position in the number $N$. For example, the number 2020 can be expressed in base 10 as,

$$2020 = 2 \times 10^3 + 0 \times 10^2 + 10 \times 10^1 + 0 \times 10^0$$

or alternatively, in base 2 and base 16,

$$2020 = 11111100100_2 = 7E4_{16}.$$

### 1.2.1 Base Change

A single number found in a positional number system can be represented in any number of different bases. A *base change* occurs when a number is converted from one numeral base to another, such as going from base 10 to base 16. For example, the decimal number $7562_{10}$ can be converted to hexadecimal, noting that each decimal remainder is converted to its hexadecimal value.

| base 16 | quotient | remainder (decimal) | remainder (hexadecimal) |
|---|---|---|---|
| 7562/16 | 472 | 10 | A |
| 472/16 | 29 | 8 | 8 |
| 29/16 | 1 | 13 | D |
| 1/16 | 0 | 1 | 1 |

The result which is written as $1D8A_{16}$. Additionally, the decimal number $100_{10}$ can be converted to base 3.

| base 3 | quotient | remainder (decimal) |
|---|---|---|
| 100/3 | 33 | 1 |
| 33/3 | 11 | 0 |
| 11/3 | 3 | 2 |
| 3/3 | 1 | 0 |
| 1/3 | 0 | 1 |

The result which is written as $10201_3$.

## 1.3 Interesting Primes

A Fibonacci number is defined by a linear recurrence equation:

$$F_n = F_{n-1} + F_{n-2}$$

It is conventional to define $F_0 = 0$. The above equation defines the next Fibonacci number as the sum of the two numbers at the $(n-1)^{\text{st}}$ and the $(n-2)^{\text{nd}}$ position. The first 10 Fibonacci numbers are 0, 1, 1, 2, 3, 5, 8, 13, 21 and 34.

A ***Fibonacci prime*** is a Fibonacci number that is prime. For example out of the above list of Fibonacci numbers, the numbers 2, 3, 5 and 13 are Fibonacci prime numbers. One of your tasks for this assignment will be to find out if a given prime number is a Fibonacci prime.

Francois Edouard Anatole Lucas was a French mathematician who devised methods like the Lucas Lehmer primality test for testing primality of numbers. He is also the person who gave the Fibonacci numbers their name, after Leonardo Pisano detto il Fibonacci. A Lucas number, named after the same mathematician, is defined by a linear recurrence equation:

$$L_n = L_{n-1} + L_{n-2}$$

It is conventional to define $L_0 = 2$. Similar to Fibonacci numbers, the above equation defines the next Lucas number as the sum of the previous two numbers at the $(n-1)^{\text{st}}$ and the $(n-2)^{\text{nd}}$ position. The first 10 Lucas numbers are 2, 1, 3, 4, 7, 11, 18, 29, 47 and 76.

A ***Lucas prime*** is a Lucas number that is prime. For example out of the above list of Lucas numbers, the numbers 2, 3, 7, 11, 29 and 47 are Lucas prime numbers. One of your tasks for this assignment will be to find out if a given prime number is a Lucas prime.

A ***Mersenne Prime***, named after Marin Mersenne, a $17^{\text{th}}$ century French polymath. He also developed Mersenne's laws, which describe the harmonics of a vibrating string (such as may be found on guitars and pianos), and his seminal work on music theory, *Harmonie universelle*. He may be best known for a list of primes with exponents up to 257, is a prime number that can be represented as one less than an $n^{\text{th}}$ power of two. For an integer $n$, a Mersenne prime is represented in the form:

$$M_n = 2^n - 1.$$

The first four Mersenne primes are $M_2 = 3$, $M_3 = 7$, $M_5 = 31$ and $M_7 = 127$, and can be represented in the above format as:

$$\begin{aligned} M_2 &= 2^2 - 1 = 3 \\ M_3 &= 2^3 - 1 = 7 \\ M_5 &= 2^5 - 1 = 31 \\ M_7 &= 2^7 - 1 = 127. \end{aligned}$$

## 1.4 Palindromic Primes

A palindrome is a word, verse, or sentence or a number that reads the same backward or forward. Examples of palindromes are: "Able was I ere I saw Elba" and the year 1881. The pseudocode to check if a string is Palindrome is as follows:

```
def isPalindrome(s):
    f = True
    for i in range(len(s) / 2):
        if s[i] != s[-(i + 1)]:
            f = False
    return f

w = raw_input("word = ")
if isPalindrome(w):
    print w, "is a palindrome"
else:
    print w, "is not a palindrome"
```

Whether a number is a palindrome or not depends on the *radix* or *base* of the number system being used. Let us look at the two examples.

**Example 1**: For the decimal number 3855, the binary number is 111100001111, and the hexadecimal number is F0F.

**Example 2**: For the decimal number 195, the binary number is 11000011, and the hexadecimal number is C3.

In Example 1, the decimal number 3855 is not a palindrome but the hexadecimal and binary counterparts of the number are palindromes, while in example 2, only the binary number is a palindrome.

A ***palindromic prime*** is a prime number, whose corresponding string at a certain base is a palindrome. Few examples of palindromic primes are $(373)_{10}$, the binary version of the number 7 which is 111, the binary version of the number 5 which is 101.

## 2 Your Task

For this assignment you have two tasks:

- **Task 1:** Use a sieve to generate a list of prime numbers up to a value $n$, inclusive. Go through each prime number and determine whether it is a *Fibonacci Prime* (`F`), *Lucas Prime* (`L`) or a *Mersenne Prime* (`M`).
- **Task 2:** For each prime number check if its a palindromic prime for the following bases:
  - Base 2
  - Base 9
  - Base 10
  - Base of the first letter of your last name + 10. For example, Professor Dunne would compute his base as 4 + 10 = 14, since the letter D is the $4^{\text{th}}$ letter of the alphabet.

  For each base only print the primes that are palindromic primes.

We will test your program by comparing its output with the output of a known correct program. Refer to the example output given at the end of the assignment. Your program should produce an output in the exact same format as shown in the example output. The values in the output will change based on your $n$ value and the bases for palindromic prime.

**Pre-lab Part 1**

1. Assuming you have a list of primes to consult, write pseudo-code to determine if a number is a Fibonacci prime, a Lucas prime, and/or a Mersenne prime.
2. Assuming you have a list of primes to consult, write pseudo-code to determine if a number in base 10 is a palindrome. Note that the technique is the same in any base.

## 3 Specifics

Writing the program using the naïve approach of finding primes up to the value of a given number into order to find its prime factorization is, frankly, silly. You can and will do much better, and you will do so by using a *sieve*. Your sieve will take as its sole argument a *BitVector*.

```
#ifndef __SIEVE_H__
#define __SIEVE_H__

#include "bv.h"

//
// The Sieve of Eratosthenes.
// Set bits in a BitVector represent prime numbers.
// Composite numbers are sieved out by clearing bits.
//
// v:  The BitVector to sieve.
//
void sieve(BitVector *v);

#endif
```

sieve.h

## 4 Bit Vectors

A Bit Vector is an ADT that represents an array of bits, the bits in which are used to denote if something is true or false (1 or 0). This is an efficient ADT since, in order to represent the truth or falsity of an array of $n$ items, we can use $n/8+1$ `uint8_t`'s instead of $n$, and being able to access 8 indices with a single integer access is extremely cost efficient. You are expected to implement the following ADT interface for Bit Vectors.

```
// bv.h - Contains the function declarations for the BitVector ADT.

#ifndef __BV_H__
#define __BV_H__

```

```c
#include <inttypes.h>
#include <stdbool.h>

//
// Struct definition for a BitVector.
//
// vector : The vector of bytes to hold bits.
// length : The length in bits of the BitVector.
//
typedef struct BitVector {
  uint8_t *vector;
  uint32_t length;
} BitVector;

//
// Creates a new BitVector of specified length.
//
// bit_len : The length in bits of the BitVector.
//
BitVector *bv_create(uint32_t bit_len);

//
// Frees memory allocated for a BitVector.
//
// v : The BitVector.
//
void bv_delete(BitVector *v);

//
// Returns the length in bits of the BitVector.
//
// v : The BitVector.
//
uint32_t bv_get_len(BitVector *v);

//
// Sets the bit at index in the BitVector.
//
// v : The BitVector.
// i : Index of the bit to set.
//
void bv_set_bit(BitVector *v, uint32_t i);

//
// Clears the bit at index in the BitVector.
```

```
51 //
52 // v : The BitVector.
53 // i : Index of the bit to clear.
54 //
55 void bv_clr_bit(BitVector *v, uint32_t i);
56
57 //
58 // Gets a bit from a BitVector.
59 //
60 // v : The BitVector.
61 // i : Index of the bit to get.
62 //
63 uint8_t bv_get_bit(BitVector *v, uint32_t i);
64
65 //
66 // Sets all bits in a BitVector.
67 //
68 // v : The BitVector.
69 //
70 void bv_set_all_bits(BitVector *v);
71
72 #endif
```

bv.h

The header file `bv.h` defines the BitVector ADT and its associated operations. Even though **C** will not prevent you from directly accessing `struct` fields, you must avoid that temptation and only use the functions defined in `bv.h` — no exceptions!

You must implement each of the functions specified in the header file. Most of them are just a line of two of **C** code, but their implementation can be subtle. You are warned *again* against using code that you may find on the Internet.

```
1  //
2  // The Sieve of Erastothenes
3  // Sets bits in a BitVector representing prime numbers.
4  // Composite numbers are sieved out by clearing bits.
5  //
6  // v: The BitVector to sieve.
7  //
8  void sieve(BitVector *v) {
9    bv_set_all_bits(v);
10   bv_clr_bit(v, 0);
11   bv_clr_bit(v, 1);
12   bv_set_bit(v, 2);
13   for (uint32_t i = 2; i < sqrtl(bv_get_len(v)); i += 1) {
14     // Prime means bit is set
15     if (bv_get_bit(v, i)) {
```

```
16          for (uint32_t k = 0; (k + i) * i <= bv_get_len(v); k += 1) {
17            bv_clr_bit(v, (k + i) * i);
18          }
19        }
20      }
21      return;
22  }
```

Since you have been given the code for the Sieve of Eratosthenes, you *must* cite it and give proper credit if you use it. If, for example, you were to implement the Sieve of Sundaram, or the more modern Sieve of Atkin, you would not need to cite beyond the source of the algorithm and any pseudocode that you followed.

Hint: You are not penalized for using this code. Choose another *only* if you are highly motivated.

**Pre-lab Part 2**

1. Implement each BitVector ADT function.
2. Explain how you avoid memory leaks when you free allocated memory for your BitVector ADT.
3. While the algorithm in sieve() is correct, it has room for improvement. What change would you make to the code in sieve() to improve the runtime?

## 5 Deliverables

You will need to turn the following files into the `assignment4` directory:

1. Your program *must* have the source and header files:
   - `bv.h` to specify the bit vector operations and abstract data type `bitV`.
   - `bv.c` to implement the functionality.
   - `sieve.h` specifies the interface to the sieve.
   - `sieve.c` to implement the sieve algorithm of your choice.
   - `sequence.c` contains `main()` and *may* contain the other functions necessary to complete the assignment.

   You may have other source and header files, but *do not try to be overly clever*.
2. `Makefile`: This is a file that will allow the grader to type `make` to compile your program. At this point you will have learned about `make` and can create your own `Makefile`.
   - `CFLAGS=-Wall -Wextra -Werror -Wpedantic` must be included.
   - `CC=clang` must be specified.
   - `make clean` must remove all files that are compiler generated.

- `make` should build your program, as should `make all`.
- Your program executable must be named `sequence`.

3. `README.md`: This *must* be in *markdown*. This must describe how to use your program and Makefile.
4. `DESIGN.pdf`: This *must* be a PDF. The design document should contain answers to the pre-lab questions at the beginning and describe your design for your program with enough detail that a sufficiently knowledgeable programmer would be able to replicate your implementation. This does not mean copying your entire program in verbatim. You should instead describe how your program works with supporting pseudocode.

   You *must* push the `DESIGN.pdf` before you push *any* code.

Your program must also:

- Support the following `getopt()` flags:
  1. `-s` : Print out all primes and identify whether or not they are interesting (Lucas, Mersenne, Fibonacci).
  2. `-p` : Print out palindromic primes in bases 2, 9, 10, and first letter of your last name + 10. Your personal base must be last and match the formatting of the rest.
  3. `-n <value>` : Specifies the largest value to consider, inclusively, for your prime sieve. By default your program runs up through 1000.
- Dynamically allocate memory.
- Not have *any* memory leaks.
- Cleanly pass infer — fix any errors or explain any complaints that infer has in your `README`.
- *Not* have any compiler warnings.

## 6 Submission

To submit your assignment, refer back to `assignment0` for the steps on how to submit your assignment through `git`. Remember: *add, commit,* and *push*!

Your assignment is turned in *only* after you have pushed *and* submitted the commit ID on Canvas. If you forget to push, you have not turned in your assignment and you will get a *zero*. "I forgot to push" is not a valid excuse. It is *highly* recommended to commit and push your changes *often*.

## 7 Supplemental Readings

- *The C Programming Language* by Kernighan & Ritchie
  - Chapter 2 §2.9
  - Chapter 4 §4.5–4.11
  - Chapter 5 §5.1–5.5
  - Chapter 6 §6.1–6.2, 6.7

## 8 Examples

Consider the two examples shown below. This shows the *required* format of the output when running the two commands `./sequence -s -n 100` and `./sequence -p -n 50`, respectively. You will notice that the output for the second command does not match what is specified for your program; it is only used to showcase program output format using other bases. The order when both the s and p flags are passed is not defined.

```
2: prime, fibonacci
3: prime, mersenne, lucas, fibonacci
5: prime, fibonacci
7: prime, mersenne, lucas
11: prime, lucas
13: prime, fibonacci
17: prime
19: prime
23: prime
29: prime, lucas
31: prime, mersenne
37: prime
41: prime
43: prime
47: prime, lucas
53: prime
59: prime
61: prime
67: prime
71: prime
73: prime
79: prime
83: prime
89: prime, fibonacci
97: prime
```

./sequence -s -n 100

```
Base  3
----  --
2 = 2
13 = 111
23 = 212

Base  4
----  --
2 = 2
3 = 3
5 = 11
```

```
17 = 101
29 = 131

Base  5
---- --
2 = 2
3 = 3
31 = 111
41 = 131

Base  6
---- --
2 = 2
3 = 3
5 = 5
7 = 11
37 = 101
43 = 111

Base  7
---- --
2 = 2
3 = 3
5 = 5

Base  8
---- --
2 = 2
3 = 3
5 = 5
7 = 7

Base  9
---- --
2 = 2
3 = 3
5 = 5
7 = 7

Base 11
---- --
2 = 2
3 = 3
5 = 5
7 = 7
```

```
57
58 Base 12
59 ---- --
60 2 = 2
61 3 = 3
62 5 = 5
63 7 = 7
64 11 = b
65 13 = 11
66
67 Base 13
68 ---- --
69 2 = 2
70 3 = 3
71 5 = 5
72 7 = 7
73 11 = b
74
75 Base 15
76 ---- --
77 2 = 2
78 3 = 3
79 5 = 5
80 7 = 7
81 11 = b
82 13 = d
```

./sequence -p -n 50