Nicolas Ayala
nmayala@ucsc.edu

Pre-Lab 1)

```
//assuming all our numbers are uint32_t, 32 bit natural numbers (including 0)
N = number of primes to test
	Types_of_primes{
		bitvector(N) Fibonacci_till_N;
		bitvector(N) Lucas_till_N;


		int prev_term = 0;
		int term = 1;
		int sum = 1;

		set_bit(prev_term, Fibonacci_till_N);
		set_bit(term, Fibonacci_till_N);

		while(prev_term + term<N){   //filling Fibonacci bit vector
			sum = prev_term + term;
			prev_term = term;
			term = sum;
			set_bit(term, Fibonacci_till_N);
		}

		int prev_term = 2;
		int term = 1;

		set_bit(prev_term, Lucas_till_N);
		set_bit(term, Lucas_till_N);

		while(prev_term + term<N){   //filling Lucas bit vector
			sum = prev_term + term;
			prev_term = term;
			term = sum;
			set_bit(term, Lucas_till_N);
		}

		For every prime p up to N
			successor = p+1;
			bits_in_successor = 0;
			for(int i = 0;i<32;i++){  //counting bits in successor
				bit_i = 1 << i;      //a number where only bit i is 1
```

```
                if(bit_i & successor){
                    bits_in_successor++;
                }
            }
            if(bits_in_successor == 1)
                p is a mersenne prime;
            if(is_bit_set(N,Fibbonacci_till_N)
                p is a Fibonacci prime;
            if(is_bit_set(N,Lucas_till_N)
                p is a Lucas prime;
    }
```

2) Fibonacci and Lucas implementation is the same, Mersenne is different

```
        For every prime p up to N{
            successor = p +1;
            while(successor>1){
                if(successor%2 == 1){
                    break;
                }
                sucessor /= 2;
            }
            if(successor == 1)
                p is a mersenne prime
        }
```

Pre_Lab 2)

1)

```
BitVector *bv_create ( uint32_t bit_len ){
    BitVector *fresh = (BitVector* ) malloc(sizeof(struct BitVector));
    if(!fresh){
        return NULL;
    }
    fresh->length = bit_len;
    uint32_t bytes = bit_len / 8;
    if(bit_len%8 != 0){
        bytes++;
    }
    fresh->vector = (uint8_t *)malloc(bytes);
    if(!fresh->vector){
        return NULL;
    }
    return fresh;
```

```
}

void bv_delete ( BitVector *v){
	if(v){
		if(v->vector){
			free(v->vector);
		}
		free(v);
	}
}

uint32_t bv_get_len ( BitVector *v){
	if(v){
		return v->length;
	}
	return 0;
}

void bv_set_bit ( BitVector *v, uint32_t i){
	if(v && v->vector && i < v->length){
		v->vector[i/8] |= 1 << (i%8);
	}
}

void bv_clr_bit ( BitVector *v, uint32_t i){
	if(v && v->vector && i < v->length){
		v->vector[i/8] &= ~(1 << (i%8));
	}
}

uint8_t bv_get_bit ( BitVector *v, uint32_t i){
	if(v && v->vector && i < v->length){
		uint8_t bit = v->vector[i/8] >> (i%8);
		return bit%2;
	}
	return 2;
}

void bv_set_all_bits ( BitVector *v){
	if(v && v->vector && v->length > 0){
		uint32_t bytes = ((v->length - 1)>>3) + 1;
		for(int i = 0;i<bytes;i++){
			v->vector[i] = 0xFF;
```

```
        }
    }
}
```

2) avoid memory leaks by deleting every created bitvector

3) Before the for loop, one can clear all even numbers larger that two with a different for loop. Then, in the main loop, replace k++ with k+=2. This is because (odd*odd + odd) is even, so we are doing twice as many bit clears as we should be.

Design of program

The program will have a Bit Vector struct implementation, the functions and their
    implementations can be found in Pre-Lab 2.1

The program will have a prime sieve up to a specified number N, This will be implemented as a sieve of eratosthenes

```
    mark all numbers up to N
    unmark 0,1
    for every integer 2 to N, L
        if L is marked{
            unmark L*L +k*L where k = {0,1,2,3.. and L*L+k*L < k
        }
    By the end of this sieve all the marked numbers will be prime
```

The main function of this program will receive three flags through getopt()

    -s
    -p
    -n <value>

-s means the program will print out all the primes up to N and if they are a fibonacci, Lucas, or Mersenne prime it will state so.

-p means the program will print the primes that are palindromic in base 2, 10, 14, 11

-n <value> make N = <value>, if this flag is absent then N=100;

In the main function, A bit vector is created with a length of N

The bit vector is passed to the sieve, which sets only the prime indexed elements

For each of the prime numbers left by the sieve, it is printed.

if -s is present, the primes are tested to see if they are Fibonacci, Lucas, or Mersenne primes, the pseudo code for this is found in Pre_Lab 1.1

if -p is present, the primes are tested to see if they are palindromes in base 2, 10, 14, 11, and then printed

The pseudo code to determine if a number is a palindrome in arbitrary bases is as follows

```
boolean Palindrome(int n, int base){
	int digits[ciel(ln(n)/ln(base)];
	for(int i = 0; i< digits.length; i++){ // fills array with digits
		digits[i] = n%base;
		n/=base;
	}
	for(int i = 0;i < digits.length/2; i++){ //tests if it is a palindrome
		if(digits[i] != digits[n-i-1])
			return false;
	}
	return true;
}
```

Design changes

After implementation, the only change to the design was to make a seperate function that fills a bit vector with fibonacci numbers, one that fills a bit vector with lucas numbers, and a function that tests if a number is a predecessor to a power of two.