Cole Marquard
cdmarqua@ucsc.edu
11/8/2020

CSE13s Fall 2020
Assignment : Bit Vectors and Primes
Design Document

This project uses a vector of bits to find primes, these primes are then further checked to see if they are a fibonacci prime, lucas prime, mersenne prime. This project also implements a palindrome check for different bases and prints palindromic primes

Inputs

The maximum value you want all your primes under achieved by using the command line argument of “-n x” where x is the integer you want the primes printed to be less than

Boolean if you want the primes and further categorizations of fibonacci prime, lucas prime, mersenne prime by utilizing the “-s” flag as a command line argument

Boolean if you want the palindromes of the prime numbers printed in the bases 2, 9,10, and 23

Included below is pseudo code for all of my functions and the main() function goes over how they are all utilized in conjunction with one another.

Prelab part1:

Fibonacci Check(primeToCheck):

```
Num1 = 1
Num2 = 1
IsFib = False
```

Iterates through fib numbers until either it equals the prime given and therefore would be a fib number or until the fib number is larger than the prime being checked which would mean that it is not a fib number as the check would have caught it prior

```
while(Num1 < primeToCheck):
    If (Num1 == primeToCheck):
        isFib = True
        break
    Temp = Num2
    Num2 = Num1 + Num2 //gets the next fib number and moves down the sequence
    Num1 = Temp

Return isFib
```

Lucas Check(primeToCheck):

```
Num1 = 2
Num2 = 1
IsLucas  = False
```

Iterates through Lucas numbers until either it equals the prime given and therefore would be a Lucas number or until the Lucas number is larger than the prime being checked which would mean that it is not a Lucas number as the check would have caught it prior

```
while(Num1 < primeToCheck):
    If (Num1 == primeToCheck):
        isLucas  = True
        break
    Temp = Num2
    Num2 = Num1 + Num2//gets the next lucas number and moves the sequence
    Num1 = Temp

Return isLucas
```

```
Mersenne Check(primeToCheck):
        Power = 2 //value that will just multiply itself continuously and be checked

        IsMers  = False
        Iterates through Mersenne numbers until either it equals the prime given and therefore
would be a Mersenne number or until the Mersenne number is larger than the prime being
checked which would mean that it is not a Mersenne number as the check would have caught it
prior

        while(power-1 < primeToCheck):
                If (power-1 == primeToCheck):
                        IsMers  = True
                        break
                power = power*2

        Return IsMers

Palindrome Check(PossiblePal):
        Array =  [] //empty array to add the digits of PossiblePal to
        Top = -1 //Will be used to track the length of the array and where the final number is
//                      Added
        isPalindrome = False
        While (PossiblePal > 0): //The 10 below would be replaced by the base of the number
                Array.append(PossiblePal%10) // adds the last digit to the array in base 10
                PossiblePal = int(PossiblePal/10) //a truncated division of the palindrome to get-
                Top+=1                                  //-the number w/o the digit previously removed
                //the number minus the one appended to the array
        Bot = 0 //bottom of the array
        While True:
                If(Array[top] == Array[bot]): //continue if they match else return false
//this means every value has been checked already as top & bot would be switching places or
the same in the case of an odd amount of digits
                        If(bot+1 == top or bot+1 == top-1): //base case, if met then it is true
                                Return True
                        top-=1
                        bot+=1
                Else:
                        Return False
```

PreLab pt 2

1functions implemented

2 free the vector array variables with allocated memory to release the pointers and free up space in the heap

3 I would make k=i*i to start from the first possible factor of the number and would then loop while k< bv_get_len(v) and finally the loop would instead increment by i, this would all work together to decrease the amount of multiplication the program would have to do and would instead have to only add which would improve the runtime (hopefully)

```
main(argc,**argv)
        For x in range(argc):
                If argv[x] == “-s”
                        Print_prime = 1 //sets boolean to print primes
                If argv[x] == “-p”
                        Print_palindrome = 1 //sets boolean to print palindrome
                If argv[x] == “-n”
                        Largest_number = argv{x+1] //gets the next argument after “-n”

        BitVec = bv_create(size = largest_number) //initialize a bitvector
        sieve(BitVec) //finding prime numbers using a sieve
        If print_prime: //if they want the primes printed

                //loop goes through all the primes and checks to see what types of prime they are
                For x in range(len(BitVec)): //goes through all the bits and if they are a 1 they are a prime else ignore
                        if(bv_get_bit(BitVec,x)){
                                print(x,”prime”)
                        Else{
                                Continue //if x is not a prime go to the next number
                        }
                        if  merscheck(x): //prints if they are also mersenne lucas and fibonacci
                                print(“mersenne”)
                        if  lucascheck(x):
                                print(“lucas”)
                        if  Fibcheck(x):
                                print(“fibonacci”)
        If print_pal:
                printP(BitVec,2)
                printP(BitVec,9)
                printP(BitVec,10) //prints the palindromes of different bases
                printP(BitVec,23)
        bv_delete(BitVec) //frees up memory
```

```
printP(bitvector,base): //function for printing palindromes
        For x in range(len(BitVec))
                if(bv_get_bit(BitVec,x)){ //go through all the primes and print the palindromes
                        If PalCheck(x,base){
                                Char convertedNumArr[1000] //initalize array to hold 1000 digits
                                convert(x,base,convertedNumArr)//convert num x to given base
                                print(x, “=” ,convertedNumArr)//prints decimal and converted Pal...
                        }
                Else{
                        Continue //if x is not a prime go to the next number
                }
```

//can be used on any decimal number to convert to any base(below 23 but can be expanded upon very easily by adding more characters to the ArrChar array)

```
convert(number,base_to,converted_array): //must pass array as you cannot return arrays
        Index = 0;
 char ArrChar[23]  = "0123456789abcdefghijklm"; //array used to convert upto base 23

        While number !=0:
                D = number / base_to
                R = decimal % base_to
                Converted_array[index] = ArrChar[r]
                Index++
                Number = d
        Return
```

Design Process

Over the course of this lab I learned how to utilize bit vectors to save space in memory which could help in many situations where a boolean array is used. The hardest part was printing the numbers when converted as I had to find a way to convert to base 23 but after a few nights of thinking it was trivally easy.

Overall it was pretty easy and the functions are very easy to implement and it helped me learn some specific c tools I can use in future projects such as passing an array to a function to change it instead of returning an array from a function.

I would change the amount of checks as they were all very easy to implement but seemed to mostly be a waste of time from my perspective as they did the same thing essentially.