ESTUDIOS
GENERALES
CIENCIAS

PONTIFICIA
UNIVERSIDAD
CATÓLICA
DEL PERÚ

FUNDAMENTOS DE PROGRAMACIÓN
LABORATORIO 8
PROPUESTAS DE SOLUCIÓN
SEMESTRE ACADÉMICO 2022-2

Horarios: Todos los horarios

Elaborado por Mag. Silvia Vargas

INDICACIONES:

- Debe utilizar variables descriptivas, comentarios y mensajes descriptivos.
- El orden y la eficiencia de su implementación serán considerados en la calificación.

RESULTADOS ESPERADOS:

- Al finalizar la sesión, el alumno comprenderá el funcionamiento de las estructuras algorítmicas iterativas anidadas.
- Al finalizar la sesión, el alumno construirá programas usando estructuras algorítmicas iterativas anidadas.

CONSIDERACIONES:

- La solución presentada para cada problema corresponde a una propuesta de solución por parte del autor.
- En programación pueden existir muchas soluciones para un mismo problema pero debe cumplir con todo lo solicitado, incluyendo las restricciones brindadas.

Desarrolle los siguientes problemas en lenguaje C:

1. Números primos buenos - Horarios: 0390

Existen diversos tipos de números, los números perfectos, amigos, abundantes, primos, etc.

Un número es un primo bueno si se cumple $(primo_n)^2 > (primo_{n-i})^2 * (primo_{n+i})^2$ para todo $1 \leq i < n$, n es la posición del número primo.

Los primeros números primos son: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, entre otros.

Por ejemplo:

- 17 que es el primo en la posición 7 es un primo bueno. Porque 17*17=289 y 289 es mayor que 13*19=247, 11*23=253, 7*29=203, 5*31=155, 3*37=111 y 2*41=82
- 7 que es el primo en la posición 4 no es un primo bueno. Porque 7*7=49 y 49 no es mayor que 5*11=55.

Se le pide implementar un programa en lenguaje C que solicite un rango de números enteros positivos, muestre los números primos que se encuentran en el rango, si uno de ellos es un primo bueno debe indicarlo y al final muestre la cantidad de números primos, números primos buenos y el porcentaje de números primos buenos del total de números primos. Antes de evaluar los números en el rango, debe validar que los números forman un rango válido, es decir que el primer número sea menor que el segundo número y mayor a 0.

Para desarrollar el problema debe utilizar el paradigma de programación modular, implementando como mínimo 4 módulos adicionales al principal. De los módulos que implemente (adicionales al main) por lo menos tres deben usar iterativas y el módulo principal también debe usar una iterativa. Debe implementar iterativas controladas por contador y por centinela.

Debe usar los mensajes que se muestran en los casos de prueba para el desarrollo del programa.

Casos de prueba

```
Ingresar el rango en el cual se evaluarán los números primos: 12 6
Rango incorrecto
```

```
Ingresar el rango en el cual se evaluarán los números primos: -4 6
Rango incorrecto
```

```
Ingresar el rango en el cual se evaluarán los números primos: 1 15
Lista de números primos en el rango:
2
3
5 es un número primo bueno
7
11 es un número primo bueno
13
En el rango [1,15] hay 6 números primos
En el rango [1,15] hay 2 números primos buenos
El porcentaje de números primos buenos en el rango es 33.333333
```

```
Ingresar el rango en el cual se evaluarán los números primos: 32 36
En el rango [32,36] no hay números primos
```

```
Ingresar el rango en el cual se evaluarán los números primos: 230 250
Lista de números primos en el rango:
233
239
241
En el rango [230,250] hay 3 números primos
En el rango [230,250] no hay números primos buenos
```

```
Ingresar el rango en el cual se evaluarán los números primos: 90 95
En el rango [90,95] no hay números primos
```

Programa 1: Propuesta de solución - Números primos buenos

```c
#include <stdio.h>
#include <math.h>

int validarRango(int inicio, int fin);
int validarPrimoBueno(int num);
int calcularPrimoPos(int primo,int tipo);
int validarPrimo(int num);

int main(){
	int inicio,fin,cantPrimos=0,cantPrimosBuenos=0,i;
	printf("Ingresar el rango en el cual se evaluarán los números primos: ");
	scanf(" %d  %d",&inicio,&fin);
	if (validarRango(inicio,fin)){
		for (i=inicio;i<=fin;i++){
			if (validarPrimo(i)){
```

```
16                                      if (cantPrimos==0)
17                                              printf("Lista de números primos en el rango:\n");
18                                      cantPrimos++;
19                                      printf(" %d",i);
20                                      //valida si es primo bueno si previamente es primo
21                                      if (i>2 && validarPrimoBueno(i)){
22                                              printf(" es un número primo bueno\n");
23                                              cantPrimosBuenos++;
24                                      }
25                                      else
26                                              printf("\n");
27                              }
28                      }
29                      if (cantPrimos>0){
30                              printf("En el rango [ %d, %d] hay  %d números primos\n",inicio,fin,cantPrimos);
31                              if (cantPrimosBuenos>0){
32                                      printf("En el rango [ %d, %d] hay  %d números primos buenos\n",inicio,fin,cantPrimosBuenos);
33                                      printf("El porcentaje de números primos buenos en el rango es  %lf\n",((double)cantPrimosBuenos/
                                                cantPrimos)*100);
34                              }
35                              else
36                                      printf("En el rango [ %d, %d] no hay números primos buenos\n",inicio,fin);
37                      }
38                      else
39                              printf("En el rango [ %d, %d] no hay números primos\n",inicio,fin);
40              }
41              else
42                      printf("Rango incorrecto\n");
43              return 0;
44      }
45
46      int validarRango(int inicio, int fin){
47              return inicio>0 && inicio<fin;
48      }
49
50      int validarPrimoBueno(int num){
51              int numOri=num,centinela=1,primoBueno=1;
52              int sigPrimo=num;
53              int primoAnt=num;
54              while (centinela){
55                      //encuentra el primo anterior i-1 y siguiente i+1
56                      sigPrimo=calcularPrimoPos(sigPrimo,1);
57                      primoAnt=calcularPrimoPos(primoAnt,0);
58                      if (primoAnt==-1)
59                              //termina la iteración cuando no existe el primo anterior
60                              centinela=0;
61                      if (primoAnt*sigPrimo>(int)pow(numOri,2)){
62                              //termina la iteración si no se cumplela relación de primo bueno
63                              centinela=0;
64                              primoBueno=0;
65                      }
66              }
67              return primoBueno;
68      }
69
70      int calcularPrimoPos(int primo,int tipo){
71              int distancia,encontrado=0,numInicial;
72              //tipo 1 busca el siguiente primo
73              //tipo 0 busca el primo anterior
74              if (tipo==1){
75                      distancia=1;
76                      numInicial=primo+1;
77              }
78              else{
79                      distancia=-1;
80                      numInicial=primo-1;
81              }
```

```
        while(!encontrado){
                //busca el primo anterior o siguiente, cuando lo encuentra termina
                //la iteración
                if (validarPrimo(numInicial))
                        encontrado=1;
                else{
                        numInicial+=distancia;
                        if (numInicial<=1)
                                break;
                }
        }
        if (encontrado)
                return numInicial;
        else
                return -1;
}

int validarPrimo(int num){
        int cantDiv=0,i=1;
        while(i<=num){
                if (num %i==0)
                        cantDiv++;
                i++;
        }
        return cantDiv==2;
}
```

2. Números primos débiles - Horarios: B301, 0384, 0385 y 0391

Existen diversos tipos de números, los números perfectos, amigos, abundantes, primos, etc.

Un número primo es débil si no se puede convertir en otro primo modificando un dígito. Los primeros números débiles son: 294001, 505447, 584141, 604171, 971767, 1062599, 1282529, 1524181, 2017963, 2474431, 2690201, 3085553, 3326489, 4393139, 5152507, entre otros.

Por ejemplo:

- 17 no es un primo débil, porque cambiando el 7 por 3 se convierte al número 13 que es primo.
- 294001 es un primo débil porque si se modifica cada dígito del número por otro, el número al que se convierte no es primo. Si cambiamos el último dígito: por 0, 294000 no es primo; por 2, 294002 no es primo; por 3, 294003 no es primo; por 4, 294004 no es primo; por 5, 294005 no es primo; por 6, 294006 no es primo; por 7, 294007 no es primo; por 8, 294008 no es primo y por 9, 294009 no es primo. Si repetimos estas operaciones cambiando los otros dígitos del número no se encontrará un número primo, por lo cual el número 294001 es un primo débil.

Se le pide implementar un programa en lenguaje C que solicite una lista de números, para cada número ingresado determine si el número es primo y si es primo identifique si es un primo débil. Antes de evaluar el número ingresado, debe validar que tenga como mínimo 2 dígitos y como máximo 9 dígitos. Si se ingresa un número que no cumple estas características se termina la solicitud de números. Luego de terminar la solicitud y evaluación de los números se debe mostrar un resumen que indique la cantidad de números ingresados, la cantidad de números primos, la cantidad de números primos débiles y el menor número primo débil ingresado.

Para desarrollar el problema debe utilizar el paradigma de programación modular, implementando como mínimo 4 módulos adicionales al principal. De los módulos que implemente todos deben usar iterativas (incluído el principal). Debe implementar iterativas controladas por contador y por lo menos una controlada por centinela.

Debe usar los mensajes que se muestran en los casos de prueba para el desarrollo del programa.

Casos de prueba

```
Ingrese una lista de números para determinar si son primos débiles
Ingrese un número: -2
--------------Resumen---------------
Se ingresaron 0 números
De los cuales 0 son números primos
De los cuales 0 son números primos débiles
```

```
Ingrese una lista de números para determinar si son primos débiles
Ingrese un número: 6
--------------Resumen---------------
Se ingresaron 0 números
De los cuales 0 son números primos
De los cuales 0 son números primos débiles
```

```
Ingrese una lista de números para determinar si son primos débiles
Ingrese un número: 12
El número ingresado no es primo
Ingrese un número: 6236179
El número ingresado es primo y también es un primo débil
Ingrese un número: 13
El número ingresado es primo  pero no es un primo débil
Ingrese un número: 971767
El número ingresado es primo y también es un primo débil
Ingrese un número:  505447
El número ingresado es primo y también es un primo débil
Ingrese un número: 17
El número ingresado es primo  pero no es un primo débil
Ingrese un número: 1524181
El número ingresado es primo y también es un primo débil
Ingrese un número: -2
--------------Resumen---------------
Se ingresaron 7 números
De los cuales 6 son números primos
De los cuales 4 son números primos débiles
El menor número primo débil es 505447
```

```
Ingrese una lista de números para determinar si son primos débiles
Ingrese un número: 15
El número ingresado no es primo
Ingrese un número: 16
El número ingresado no es primo
Ingrese un número: 14
El número ingresado no es primo
Ingrese un número: -5
--------------Resumen---------------
Se ingresaron 3 números
De los cuales 0 son números primos
De los cuales 0 son números primos débiles
```

Programa 2: Propuesta de solución - Números primos débiles

```c
#include <stdio.h>
#include <math.h>

int calcularCantDig(int num);
int validarPrimoDebil(int ,int );
int validarPrimoDebilPos(int ,int );
int validarPrimo(int );

int main(){
        int num,cantDig,cantPrimos=0,cantPrimosDeb=0,cantNum=0,menorDebil;
```

```
11          printf("Ingrese una lista de números para determinar si son primos débiles\n");
12          while(1){
13                  printf("Ingrese un número: ");
14                  scanf(" %d",&num);
15                  if (num<10)
16                          break;
17                  cantDig=calcularCantDig(num);
18                  if (cantDig>9)
19                          break;
20                  cantNum++;
21                  if (validarPrimo(num)){
22                          cantPrimos++;
23                          printf("El número ingresado es primo ");
24                          //evalúa si es primo débil solo si previamente es primo
25                          if (validarPrimoDebil(num,cantDig)){
26                                  cantPrimosDeb++;
27                                  printf("y también es un primo débil\n");
28                                  //identifica al menor primo débil
29                                  if (cantPrimosDeb==1)
30                                          menorDebil=num;
31                                  else
32                                          if (num<menorDebil)
33                                                  menorDebil=num;
34                          }
35                          else
36                                  printf(" pero no es un primo débil\n");
37                  }
38                  else
39                          printf("El número ingresado no es primo\n");
40          }
41          printf("----------Resumen----------\n");
42          printf("Se ingresaron  %d números\n",cantNum);
43          printf("De los cuales  %d son números primos\n",cantPrimos);
44          printf("De los cuales  %d son números primos débiles\n",cantPrimosDeb);
45          if (cantPrimosDeb>0)
46                  printf("El menor número primo débil es  %d\n",menorDebil);
47          return 0;
48  }
49
50  int calcularCantDig(int num){
51          int cantDig=0;
52          while(num>0){
53                  cantDig++;
54                  num/=10;
55          }
56          return cantDig;
57  }
58
59  int validarPrimoDebil(int num,int cantDig){
60          int pos=1,debil=1;
61          while(pos<=cantDig){
62                  //verifica si el primo no es débil en alguna posición
63                  if (!validarPrimoDebilPos(num,pos)){
64                          debil=0;
65                          break;
66                  }
67                  pos++;
68          }
69          return debil;
70  }
71
72  int validarPrimoDebilPos(int num,int pos){
73          int parDer,parIzq,digito,debil=1,nuevoNum,numEvaluar,dig;
74          parDer=num %(int)pow(10,pos-1);
75          parIzq=num/(int)pow(10,pos);
76          digito=(num-parIzq*(int)pow(10,pos))/(int)pow(10,pos-1);
77          nuevoNum=parIzq*(int)pow(10,pos)+parDer;
```

```
78          //cambia los dígitos de una posición del número
79          for (dig=0;dig<=9;dig++){
80                  if (digito!=dig){
81                          numEvaluar=nuevoNum+dig*(int)pow(10,pos-1);
82                          if (validarPrimo(numEvaluar)){
83                                  debil=0;
84                                  break;
85                          }
86                  }
87          }
88          return debil;
89  }
90
91  int validarPrimo(int num){
92          int cantDiv=0,i=1;
93          while(i<=num){
94                  if (num %i==0)
95                          cantDiv++;
96                  i++;
97          }
98          return cantDiv==2;
99  }
```

3. Números Woodall, primos de Woodall y primos Honaker - Horarios: 0380, 0381, 0386 y 0387

Existen diversos tipos de números, los números perfectos, amigos, abundantes, primos, etc.

Un número Woodall es aquel que cumple con $(Woodall_n)$ = n * 2^n - 1.

Por ejemplo:

- 23 es un número Woodall porque 3 * 2^3 - 1
- 895 es un número Woodall porque 7 * 2^7 - 1

Los primeros números Woodall son: 1, 7, 23, 63, 159, 383, 895, 2047, 4607, 10239, 22527, 49151, entre otros.

Un número primo de Woodall es aquel que cumple con la característica anterior y también es un número primo. Los primeros números primos de Woodall son: 7, 23, 383, 32212254719, entre otros.

Un número primo Honaker es aquel en el cual se cumple que la suma de los dígitos de la posición del primo es igual a la suma de los dígitos del número. Los primeros números primos Honaker son: 131, 263, 457, 1039, 1049, 1091, 1301, 1361, 1433, 1571, 1913, 1933, 2141, 2221, 2273, entre otros.

Por ejemplo:

- 131 es un primo Honaker porque su posicion como primo es 32, la suma de estos dígitos 3 + 2 = 5; y 5 = 1 + 3 + 1
- 2221 es un primo Honaker porque su posicion como primo es 331, la suma de estos dígitos 3 + 3 + 1 = 7; y 7 = 2 + 2 + 2 + 1

Los primeros números primos son: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, entre otros.

Se le pide implementar un programa en lenguaje C que solicite una lista de números, para cada número ingresado determine si el número es Woodall, primo, primo Honaker o primo Woodall. Si el número es Woodall debe mostrar el factor n que permite que se cumple la fórmula indicada. Si el número es primo debe indicar su posición. Antes

de evaluar el número ingresado, debe validar que sea positivo y que tenga como máximo 8 dígitos, si se ingresa un número que no cumple estas características se termina la solicitud de números. Luego de terminar la solicitud y evaluación de los números se debe mostrar un resumen que indique la cantidad de números ingresados, la cantidad de números Woodall que no son primos, la cantidad de números primos, la cantidad de números primos de Woodall y la cantidad de números primos Honaker.

Para desarrollar el problema debe utilizar el paradigma de programación modular, implementando como mínimo 5 módulos adicionales al principal. De los módulos que implemente por lo menos 4 deben usar iterativas (incluído el principal). Debe implementar iterativas controladas por contador y por lo menos una controlada por centinela. Uno de los módulos que implemente debe simular el uso de parámetros por referencia modificando por lo menos dos parámetros y no debe usarse para leer los datos.

Debe usar los mensajes que se muestran en los casos de prueba para el desarrollo del programa.

Casos de prueba

```
Ingrese una lista de números para evaluarlos
Ingrese un número: -2
--------------Resumen---------------
Se ingresaron 0 números
De los cuales 0 son números Woodall no primos
De los cuales 0 son números primos
De los cuales 0 son números primos Woodall
De los cuales 0 son números primos Honaker
```

```
Ingrese una lista de números para evaluarlos
Ingrese un número: 123456789
--------------Resumen---------------
Se ingresaron 0 números
De los cuales 0 son números Woodall no primos
De los cuales 0 son números primos
De los cuales 0 son números primos Woodall
De los cuales 0 son números primos Honaker
```

```
Ingrese una lista de números para evaluarlos
Ingrese un número: 63
El número ingresado no es un número primo
Es un número Woodall, 63 = 4 x 2 ^ 4 - 1
Ingrese un número: 128
El número ingresado no es un número primo
Ingrese un número: 73
Es un número primo y ocupa la posición 21
Ingrese un número: 457
Es un número primo y ocupa la posición 88
Es un primo Honaker
Ingrese un número: 23
Es un número primo y ocupa la posición 9
Es un primo Woodall 23 = 3 x 2 ^ 3 - 1
Ingrese un número: -1
--------------Resumen---------------
Se ingresaron 5 números
De los cuales 1 son números Woodall no primos
De los cuales 3 son números primos
De los cuales 1 son números primos Woodall
De los cuales 1 son números primos Honaker
```

```
Ingrese un número: 66
El número ingresado no es un número primo
Ingrese un número: 2273
Es un número primo y ocupa la posición 338
Es un primo Honaker
```

```
Ingrese un número: 7
Es un número primo y ocupa la posición 4
Es un primo Woodall 7 = 2 x 2 ^ 2 - 1
Ingrese un número: 2221
Es un número primo y ocupa la posición 331
Es un primo Honaker
Ingrese un número: 11
Es un número primo y ocupa la posición 5
Ingrese un número: 0
Se ingresaron 7 números
De los cuales 0 son números Woodall no primos
De los cuales 4 son números primos
De los cuales 1 son números primos Woodall
De los cuales 2 son números primos Honaker
```

Programa 3: Propuesta de solución - Números Woodall

```c
#include <stdio.h>
#include <math.h>

int validarPrimo(int );
int calcularPosicionPrimo(int );
int calcularSigPrimo(int );
void validarNumeroWoodall(int ,int *,int *);
int validarPrimoWoodall(int );
int sumarDigitos(int );

int main(){
	int num,posicion,cantNumeros=0,cantPrimos=0,cantPrimosHon=0;
	int cantNumWoo=0,cantPrimosWoo=0,validoWoo,factor;
	printf("Ingrese una lista de números para evaluarlos\n");
	while(1){
		printf("Ingrese un número: ");
		scanf(" %d",&num);
		if (num<1 || num>99999999) //la iteración termina cuando se ingresa un número inválido
			break;
		cantNumeros++;
		validarNumeroWoodall(num,&validoWoo,&factor);
		if (validarPrimo(num)){
			cantPrimos++;
			posicion=calcularPosicionPrimo(num);
			printf("Es un número primo y ocupa la posición %d\n",posicion);
			if (validarPrimoHonaker(posicion,num)){
				cantPrimosHon++;
				printf("Es un primo Honaker\n");
			}
			if (validoWoo){
				cantPrimosWoo++;
				printf("Es un primo Woodall %d = %d x 2 ^ %d - 1\n",num,factor,factor);
			}
		}
		else{
			printf("El número ingresado no es un número primo\n");
			if (validoWoo){
				cantNumWoo++;
				printf("Es un número Woodall, %d = %d x 2 ^ %d - 1\n",num,factor,factor);
			}
		}
	}
	printf("----------Resumen----------\n");
	printf("Se ingresaron %d números\n",cantNumeros);
	printf("De los cuales %d son números Woodall no primos\n",cantNumWoo);
	printf("De los cuales %d son números primos\n",cantPrimos);
	printf("De los cuales %d son números primos Woodall\n",cantPrimosWoo);
	printf("De los cuales %d son números primos Honaker\n",cantPrimosHon);
	return 0;
```

```
}

void validarNumeroWoodall(int num,int *valido,int *factor){
	int i=1,calc;
	while(1){
		//para identificar el i que cumple con la condición de Woodall
		calc=i*(int)pow(2,i)-1;
		//si existe un i que cumple, termina la iterativa indicando el factor
		if (calc==num){
			*valido=1;
			*factor=i;
			break;
		}
		//si el valor de la fórmula es mayor que el número entonces no
		//existe un i que cumpla con la condición de Woodall
		if (calc>num){
			*valido=0;
			break;
		}
		i++;
	}
}

int validarPrimo(int num){
	int divisor=1,contDivisores=0;
	while(divisor<=num){
		if (num %divisor==0)
			contDivisores++;
		divisor++;
	}
	return contDivisores==2;
}

int calcularPosicionPrimo(int num){
	int pos=1,primo=2;
	while(primo<num){
		//busca el siguiente primo hasta llegar a num y devuelve la posición del primo correspondiente
		primo=calcularSigPrimo(primo);
		pos++;
	}
	return pos;
}

int calcularSigPrimo(int num){
	while (1){
		num++;
		if (validarPrimo(num))
			return num;
	}
}

int validarPrimoHonaker(int posicion,int primo){
	//verifica si se cumple con la condición de primo Honaker
	int sumaPrimo,sumaPos;
	sumaPrimo=sumarDigitos(primo);
	sumaPos=sumarDigitos(posicion);
	return sumaPrimo==sumaPos;
}

int sumarDigitos(int num){
	int suma=0,digito;
	while (num>0){
		digito=num %10;
		num/=10;
		suma+=digito;
	}
	return suma;
```

```
117 }
```

4. Números extravagantes - Horarios: 0382, 0383, 0388 y 0389

Existen diversos tipos de números, los números perfectos, amigos, abundantes, primos, etc.

Un número es un número extravagante si tiene menos dígitos que la cantidad de dígitos en su descomposición de factores primos incluídos los respectivos exponentes (si son mayores a 1).

Por ejemplo:

- 38 es un número extravagante, ya que 38 = $2^1 * 19^1$, la cantidad de dígitos de factores primos es igual 3 (1 dígito del 2 y 2 dígitos del 19) y cantidad de dígitos de los exponentes mayores a 1 es igual a 0 (porque los exponentes de 2 y 19 son 1), 3 + 0 es igual a 3 y 3 es mayor que 2 que es la cantidad de dígitos de 38.
- 125 no es un número extravagante, ya que 125 = 5^3, la cantidad de dígitos de factores primos es igual a 1 (1 dígito del 5) y cantidad de dígitos del exponente mayor a 1 es igual 1 (1 dígito del exponente 3), 1 + 1 es igual a 2 y 2 es menor que 3 que es la cantidad de dígitos de 125.

Los primeros números extravagantes son: 4, 6, 8, 9, 12, 18, 20, 22, 24, 26, 28, 30, 33, 34, 36, 38, 39, 40, entre otros.

Se le pide implementar un programa en lenguaje C que solicite un rango de números enteros positivos, muestre los números extravagantes que se encuentran en el rango y al final muestre la cantidad de números extravagantes que se encontraron en el rango. Antes de evaluar los números en el rango, debe validar que se forme un rango válido, es decir que el primer valor sea menor que el segundo valor y mayor a 0.

Para desarrollar el problema debe utilizar el paradigma de programación modular, implementando como mínimo 5 módulos adicionales al principal. De los módulos que implemente (adicionales al main) por lo menos cuatro deben usar iterativas y el módulo principal también debe usar una iterativa. Uno de los módulos debe simular el uso de parámetros por referencia modificando por lo menos dos parámetros y no debe utilizare para la lectura de datos. Debe implementar iterativas controladas por contador y por centinela.

Debe usar los mensajes que se muestran en los casos de prueba para el desarrollo del programa.

Casos de prueba

```
Ingresar el rango en el cual se evaluarán los números extravagantes: 5 1
Rango incorrecto
```

```
Ingresar el rango en el cual se evaluarán los números extravagantes: -2 10
Rango incorrecto
```

```
Ingresar el rango en el cual se evaluarán los números extravagantes: 100 110
Lista de números extravagantes en el rango:
100
102
104
108
110
En el rango [100,110] hay 5 números extravagantes
```

```
Ingresar el rango en el cual se evaluarán los números extravagantes: 420 430
Lista de números extravagantes en el rango:
420
```

```
422
423
424
425
426
428
429
430
En el rango [420,430] hay 9 números extravagantes
```

```
Ingresar el rango en el cual se evaluarán los números extravagantes: 13 16
En el rango [13,16] no hay números extravagantes
```

Programa 4: Propuesta de solución - Números extravagantes

```c
#include <stdio.h>

int validarRango(int inicio, int fin);
int evaluarNumeroExtravagante(int );
void calcularExponenteFactorizacion(int ,int ,int *, int *);
int calcularCantDigitos(int );
int calcularSigPrimo(int );
int validarPrimo(int );

int main(){
	int inicio,fin,cantExtravagantes=0,i;
	printf("Ingresar el rango en el cual se evaluarán los números extravagantes: ");
	scanf(" %d  %d",&inicio,&fin);
	if (validarRango(inicio,fin)){
		for (i=inicio;i<=fin;i++){
			if (evaluarNumeroExtravagante(i)){
				cantExtravagantes++;
				if (cantExtravagantes==1)
					printf("Lista de números extravagantes en el rango:\n");
				printf(" %d\n",i);
			}
		}
		if (cantExtravagantes>0)
			printf("En el rango [ %d, %d] hay  %d números extravagantes\n",inicio,fin,cantExtravagantes);
		else
			printf("En el rango [ %d, %d] no hay números extravagantes\n",inicio,fin);
	}
	else
		printf("Rango incorrecto\n");
	return 0;
}

int validarRango(int inicio, int fin){
	return inicio>0 && inicio<fin;
}

int evaluarNumeroExtravagante(int num){
	int divisor=2,exponente,cantDigNum,restoNum,cantDigFact=0,cantDigExp=0;
	cantDigNum=calcularCantDigitos(num);
	while(1){
		if (num %divisor==0){
			//si se identifica un divisor se busca el exponente que corresponde
			//y se actualiza el número para la siguiente iteración
			calcularExponenteFactorizacion(num,divisor,&exponente,&restoNum);
			//acumula la cantidad de dígitos del divisor
			cantDigFact+=calcularCantDigitos(divisor);
			if (exponente>1)
				//acumula la cantidad de dígitos del exponente si es mayor a 1
				cantDigExp+=calcularCantDigitos(exponente);
			num=restoNum;
```

```
51                      }
52                      //busca el siguiente primo para evaluarlo como divisor
53                      divisor=calcularSigPrimo(divisor);
54                      if (num<divisor) break;
55              }
56              return cantDigNum<(cantDigFact+cantDigExp);
57      }
58
59      void calcularExponenteFactorizacion(int num,int divisor,int *exponente, int *restoNum){
60              *exponente=0;
61              //divide num entre divisor mientras el resto sea 0 y va acumulando exponente
62              while(1){
63                      if (num %divisor==0){
64                              (*exponente)++;
65                              num/=divisor;
66                      }
67                      else
68                              break;
69              }
70              *restoNum=num;
71      }
72
73      int calcularCantDigitos(int num){
74              int cantDig=0;
75              while(num>0){
76                      num/=10;
77                      cantDig++;
78              }
79              return cantDig;
80      }
81
82      int calcularSigPrimo(int num){
83              while (1){
84                      num++;
85                      if (validarPrimo(num))
86                              return num;
87              }
88      }
89
90      int validarPrimo(int num){
91              int cantDiv=0,i=1;
92              while(i<=num){
93                      if (num %i==0)
94                              cantDiv++;
95                      i++;
96              }
97              return cantDiv==2;
98      }
```

Puede usar cualquier estructura selectiva y cualquier estructura iterativa.