ESTUDIOS
GENERALES
CIENCIAS

PONTIFICIA
UNIVERSIDAD
CATÓLICA
DEL PERÚ

FUNDAMENTOS DE PROGRAMACIÓN
LABORATORIO 7
PROPUESTAS DE SOLUCIÓN
SEMESTRE ACADÉMICO 2021-2

Horarios: Todos los horarios

Elaborado por Dra. Layla Hirsh y Mag. David Allasi

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. Cambio de base especial, de base n a base n^k

El método de cambio de base es una herramienta simple que permite convertir un mismo número en sus representaciones bajo diferentes sistemas númericos.

Por ejemplo, el número 15 en base 10 se puede escribir como el número 1111 en base 2, o como el número 33 en base 4.

Dentro de los métodos que existen para realizar un cambio de base, existe uno llamado cambio de base especial, el cual permite cambiar un número expresado en una base n directamente a un número expresado en una nueva base n^k.

Este método consiste en lo siguiente:

- Se debe determinar el exponente k, el cual debe ser un número entero k que debe cumplir lo siguiente: Nueva Base = n^k, donde n es la base original y k es el exponente que permite esta igualdad.
- Este exponente k es muy importante para realizar el cambio de base especial, ya que se deben procesar todas las cifras del número en grupos de k cifras cada uno, iniciando la agrupación desde el dígito menos significativo (derecha a izquierda) al más significativo.
- Cada grupo de k cifras representa a un número en la base n, por lo que ahora se debe realizar la descomposición polinómica del número en dicha base n para formar el nuevo dígito en la nueva base n^k.

- Cada nuevo dígito, obtenido en el punto anterior, formará parte del nuevo número en base n^k, para ello, el nuevo número colocará al nuevo dígito en su posición significativa dentro de los grupos. Es decir, el nuevo dígito obtenido por el primer grupo será el dígito menos significativo, el nuevo dígito obtenido por el segundo grupo será el segundo dígito menos significativo. Esto se repite por todos los grupos formados.
- El nuevo número en base n^k será el resultado de la unión de todos los nuevos dígitos, respetando su posición significativa en base a su grupo.

A continuación presentamos un ejemplo de este método:

- Se aplicará el método de cambio de base especial para el número 1210212 que está en base 3 a la base 9.
- Determinaremos el exponente k que permita lo siguiente: $9 = 3^k$, de aquí se obtiene que k=2.
- Se debe agrupar las cifras del número 1210212 en grupos de k cifras, que, en este caso, son 2 cifras, de derecha a izquierda (menos significativo a más significativo).
- Para este caso, los grupos formados de 2 cifras, de derecha a izquierda, con las cifras del número 1210212 son: 12, 02, 21 y 01.
- A cada grupo obtenido se le aplicará su descomposición polinómica en la base original, en este caso 3. El resultado de esto es:
  - Descomposición polinómica de $12 = 1 * 3^1 + 2 * 3^0 = 5$
  - Descomposición polinómica de $02 = 0 * 3^1 + 2 * 3^0 = 2$
  - Descomposición polinómica de $21 = 2 * 3^1 + 1 * 3^0 = 7$
  - Descomposición polinómica de $01 = 0 * 3^1 + 1 * 3^0 = 1$
- Con los resultados de la descomposición polinómica se debe generar el nuevo numero en base n^k, para ello, cada resultado obtenido se debe colocar en su posición significativa correspondiente, de acuerdo a los grupos del número original. En este caso el nuevo número sería 1725 en base 9.

Se le pide que elabore un programa en lenguaje C que permita ejecutar un cambio de base especial para un número expresado en una base n a un número en una base n^k. Para este cambio de base especial debe realizar los pasos indicados en el enunciado. Dentro del programa debe considerar las siguientes validaciones:

- La base n debe estar en el rango de 2 y 9, en caso no se cumpla se debe mostrar el mensaje `La base n ingresada no es correcta` y el programa debe terminar.
- El número ingresado debe ser mayor que 0, en caso no se cumpla se debe mostrar el mensaje `El numero debe ser mayor que 0` y el programa debe terminar.
- El número ingresado debe estar en la base n ingresada, en caso no se cumpla se debe mostrar el mensaje `El numero ingresado no corresponde a la base n ingresada` y el programa debe terminar. Recuerde que un número esta en una base n si todas las cifras del número son menores a la base n.
- La nueva base n^k debe ser mayor que la base original y menor que 10. En caso no se cumpla debe mostrar el mensaje `La nueva base no puede ser menor o igual que la base original` o `La nueva base no puede ser mayor que 10` según corresponda y el programa debe terminar.
- Si no se logra encontrar un exponente k válido que cumpla con el enunciado, se debe mostrar el mensaje `No se pudo determinar un exponente k entero de la forma` $base^k$ `= nuevaBase` y el programa debe terminar.

Para su solución debe utilizar programación modular (al menos 4 módulos incluyendo el main), considerando que por lo menos uno de los módulos, que no sea el main, debe tener una iterativa anidada dentro de su implementación. Además, debe considerar que para calcular el exponente k no podrá utilizar la función sqrt. Si su solución no sigue el método de cambio de base especial explicado y realiza la conversión de otra manera, se calificará con la nota de 0.

A continuación se presentan algunos ejemplos de ejecución de este programa que podrá usar para validar su solución.

```
Ingrese la base: 12
La base n ingresada no es correcta.
```

```
Ingrese la base: 3
Ingrese el número en la base ingresada: 4122
El número ingresado no corresponde a la base ingresada.
```

```
Ingrese la base: 3
Ingrese el número en la base ingresada: 4122
El número ingresado no corresponde a la base ingresada.
```

```
Ingrese la base: 3
Ingrese el número en la base ingresada: 1210212
Ingrese la nueva base a donde desea pasar el número: 2
La nueva base no puede ser menor o igual que la base original.
```

```
Ingrese la base: 3
Ingrese el número en la base ingresada: 1210212
Ingrese la nueva base a donde desea pasar el número: 10
La nueva base no puede ser mayor o igual que 10
```

```
Ingrese la base: 3
Ingrese el número en la base ingresada: 1210212
Ingrese la nueva base a donde desea pasar el número: 8
No se pudo determinar un exponente k entero de la forma baseˆk = nuevaBase.
```

```
Ingrese la base: 3
Ingrese el número en la base ingresada: 1210212
Ingrese la nueva base a donde desea pasar el número: 9
El número 1210212 en base 3 es igual al número 1725 en base 9.
```

Programa 1: Propuesta de solución - Cambio de base especial

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

int validarNumero(int numero,int base);
int calcularExponenteK(int base,int nuevaBase);
int cambiarBase(int numero,int exponenteK,int base);

int main(){
        int base, numero, esValido, nuevaBase, exponenteK, nuevoNum;
```

```
10          printf("Ingrese la base: ");
11          scanf(" %d",&base);
12          if (base>1 && base<10){
13                  printf("Ingrese el número en la base ingresada: ");
14                  scanf(" %d",&numero);
15                  if (numero>0){
16                          esValido = validarNumero(numero,base);
17                          if (esValido){
18                                  printf("Ingrese la nueva base a donde desea pasar el número: ");
19                                  scanf(" %d",&nuevaBase);
20                                  if (nuevaBase>base && nuevaBase<10){
21                                          exponenteK = calcularExponenteK(base,nuevaBase);
22                                          if (exponenteK==0){
23                                                  printf("No se pudo determinar un exponente k entero de la forma baseˆk =
                                                        nuevaBase");
24                                          }
25                                          else{
26                                                  nuevoNum = cambiarBase(numero,exponenteK,base);
27                                                  printf("El número %d en base %d es igual al número %d en base %d",numero,
                                                        base,nuevoNum,nuevaBase);
28                                          }
29                                  }
30                                  else {
31                                          if (nuevaBase<=base){
32                                                  printf("La nueva base no puede ser menor o igual que la base original");
33                                          }
34                                          else{
35                                                  printf("La nueva base no puede ser mayor o igual que 10");
36                                          }
37                                  }
38                          }
39                          else {
40                                  printf("El número ingresado no corresponde a la base ingresada");
41                          }
42                  }
43                  else{
44                          printf("El número debe ser mayor que 0");
45                  }
46          }
47          else{
48                  printf("La base n ingresada no es correcta");
49          }
50          return 0;
51  }
52
53  int validarNumero(int numero,int base){
54          int es_valido=1, digito;
55          while (numero!=0){
56                  digito = numero %10;
57                  numero = numero/10;
58                  if (digito>=base){
59                          es_valido=0;
60                          break;
61                  }
62          }
63          return es_valido;
64  }
65
66  int calcularExponenteK(int base,int nuevaBase){
67          int exponente, producto;
68          exponente = 0;
69          producto = 1;
70          while (producto<nuevaBase){
71                  producto = producto*base;
72                  exponente++;
73          }
74          if (producto!=nuevaBase){
```

```
                exponente = 0;
        }
        return exponente;
}

int cambiarBase(int numero,int exponenteK,int base){
        int nuevoNum, posicion, digitos, nuevoDigito, exponente, cifra;
        nuevoNum = 0;
        posicion = 0;
        while (numero>0){
                digitos = numero  % (int)pow(10,exponenteK);
                numero = numero/((int)pow(10,exponenteK));
                nuevoDigito = 0;
                exponente = 0;
                while (digitos>0){
                        cifra = digitos %10;
                        digitos = digitos/10;
                        nuevoDigito = nuevoDigito + cifra*(int)pow(base,exponente);
                        exponente++;
                }
                nuevoNum = nuevoNum + nuevoDigito*(int)pow(10,posicion);
                posicion++;
        }
        return nuevoNum;
}
```

2. Cambio de base especial, de base n^k a base n

El método de cambio de base es una herramienta simple que permite convertir un mismo número en sus representaciones bajo diferentes sistemas númericos.

Por ejemplo, el número 15 en base 10 se puede escribir como el número 1111 en base 2, o como el número 33 en base 4.

Dentro de los métodos que existen para realizar un cambio de base, existe uno llamado cambio de base especial, el cual permite cambiar un número expresado en una base n^k directamente a un número expresado en una nueva base n.

Este método consiste en lo siguiente:

- Se debe determinar el exponente k, el cual debe ser un número entero k que debe cumplir lo siguiente: n^k es la base original y n es la nueva base.
- Se debe procesar cada cifra del número, de izquierda a derecha realizando lo siguiente:
  - A cada cifra se debe aplicar divisiones sucesivas entre la nueva base hasta que llegue a 0. En cada división debe ir formando un nuevo número con los residuos de las divisiones.
  - Es importante considerar que el nuevo número debe tener k cifras. En caso no se cumpla ello, debe completar las cifras con 0 a la izquierda hasta completar las k cifras.
- Cada nuevo número de k cifras formado por cada cifra del número se debe unir con el siguiente.
- El nuevo número en base n será el resultado de la unión de todos los nuevos números, respetando su posición significativa en base a su grupo (más significativo a menos significativo).

A continuación presentamos un ejemplo de este método:

- Se aplicará el método de cambio de base especial para el número 272 que está en base 8 a la base 2.

- Determinaremos el exponente k que permita lo siguiente: $8 = 2^k$, de aquí se obtiene que k=3.
- Se debe procesar cada una de las cifras del número 272 de izquierda a derecha.
- Para cada cifra se aplicará la técnica de divisiones sucesivas entre la nueva base y se formarán los nuevos números (ver figura 1). Noten que para el caso de las cifras con valor 2, se esta completando el nuevo número con una cifra 0 a la izquierda porque la cantidad de cifras es menor a 3 (valor de k).

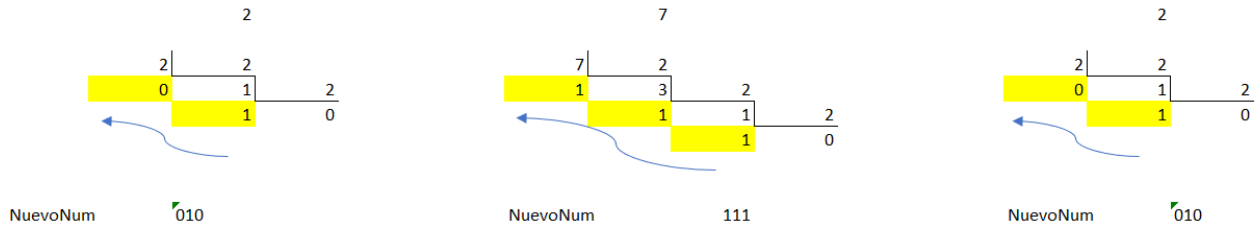

Figura 1: Divisiones sucesivas por cada cifra del número

- Con los resultados de la divisiones sucesivas se debe generar el nuevo numero en base n, para ello, cada resultado obtenido se debe colocar en su posición significativa correspondiente, de acuerdo a las cifras del número original. En este caso el nuevo número sería 10111010 en base 2.

Se le pide que elabore un programa en lenguaje C que permita ejecutar un cambio de base especial para un número expresado en una base n^k a un número en una base n. Para este cambio de base especial debe realizar los pasos indicados en el enunciado. Dentro del programa debe considerar las siguientes validaciones:

- La base original n^k debe estar en el rango de 2 y 9, en caso no se cumpla se debe mostrar el mensaje `La base` n^k `ingresada no es correcta` y el programa debe terminar.
- El número ingresado debe ser mayor que 0, en caso no se cumpla se debe mostrar el mensaje `El numero debe ser mayor que 0` y el programa debe terminar.
- El número ingresado debe estar en la base n ingresada, en caso no se cumpla se debe mostrar el mensaje `El numero ingresado no corresponde a la base n ingresada` y el programa debe terminar. Recuerde que un número está en una base n si todas las cifras del número son menores a la base n.
- La nueva base n debe ser menor que la base original y mayor que 1. En caso no se cumpla debe mostrar el mensaje `La nueva base no puede ser mayor o igual que la base original` o `La nueva base no puede ser menor o igual que 1` según corresponda y el programa debe terminar.
- Si no se logra encontrar un exponente k válido que cumpla con el enunciado, se debe mostrar el mensaje `No se pudo determinar un exponente k entero de la forma` $base^k$ `= nuevaBase` y el programa debe terminar.

Para su solución debe utilizar programación modular (al menos 5 módulos incluyendo el main), considerando que por lo menos uno de los módulos, que no sea el main, debe tener una iterativa anidada dentro de su implementación. Además, debe considerar que para calcular el exponente k no podrá utilizar la función sqrt. También debe considerar que un módulo, que no sea el main, debe efectuar una llamada a otro módulo. Si su solución no sigue el método de cambio de base especial explicado y realiza la conversión de otra manera, se calificará con la nota de 0.

A continuación se presentan algunos ejemplos de ejecución de este programa que podrá usar para validar su solución.

```
Ingrese la base: 12
La base ingresada no es correcta.
```

```
Ingrese la base: 8
Ingrese el número en la base ingresada: 972
El número ingresado no corresponde a la base ingresada.
```

```
Ingrese la base: 8
Ingrese el número en la base ingresada: 1210212
Ingrese la nueva base a donde desea pasar el número: -1
La nueva base no puede ser menor o igual que 1
```

```
Ingrese la base: 8
Ingrese el número en la base ingresada: 1210212
Ingrese la nueva base a donde desea pasar el número: 3
No se pudo determinar un exponente k entero de la forma baseˆk = nuevaBase.
```

```
Ingrese la base: 8
Ingrese el número en la base ingresada: 272
Ingrese la nueva base a donde desea pasar el número: 2
El número 272 en base 8 es igual al número 10111010 en base 2.
```

Programa 2: Propuesta de solución - Cambio de base especial

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

int validarNumero(int numero,int base);
int calcularExponenteK(int base,int nuevaBase);
int cambiarBase(int numero,int exponenteK,int base);
int calcularCantidadDigitos(int numero);

int main(){
        int base, numero, esValido, nuevaBase, exponenteK, nuevoNum;
        printf("Ingrese la base: ");
        scanf(" %d",&base);
        if (base>1 && base<10){
                printf("Ingrese el número en la base ingresada: ");
                scanf(" %d",&numero);
                if (numero>0){
                        esValido = validarNumero(numero,base);
                        if (esValido){
                                printf("Ingrese la nueva base a donde desea pasar el número: ");
                                scanf(" %d",&nuevaBase);
                                if (nuevaBase<base && nuevaBase>1){
                                        exponenteK = calcularExponenteK(nuevaBase,base);
                                        if (exponenteK==0){
                                                printf("No se pudo determinar un exponente k entero de la forma baseˆk =
                                                        nuevaBase");
                                        }
                                        else{
                                                nuevoNum = cambiarBase(numero,exponenteK,nuevaBase);
                                                printf("El número %d en base %d es igual al número %d en base %d",numero,
                                                        base,nuevoNum,nuevaBase);
                                        }
                                }
```

```
                                        else {
                                                if (nuevaBase>=base){
                                                        printf("La nueva base no puede ser mayor o igual que la base original");
                                                }
                                                else{
                                                        printf("La nueva base no puede ser menor o igual que 0");
                                                }
                                        }
                                }
                                else {
                                        printf("El número ingresado no corresponde a la base ingresada");
                                }
                        }
                        else{
                                printf("El número debe ser mayor que 0");
                        }
                }
                else{
                        printf("La base ingresada no es correcta");
                }
                return 0;
}

int validarNumero(int numero,int base){
                int es_valido=1, digito;
                while (numero!=0){
                        digito = numero %10;
                        numero = numero/10;
                        if (digito>=base){
                                es_valido=0;
                                break;
                        }
                }
                return es_valido;
}

int calcularExponenteK(int base,int nuevaBase){
                int exponente, producto;
                exponente = 0;
                producto = 1;
                while (producto<nuevaBase){
                        producto = producto*base;
                        exponente++;
                }
                if (producto!=nuevaBase){
                        exponente = 0;
                }
                return exponente;
}

int cambiarBase(int numero,int exponenteK,int base){
                int cantDigitos, nuevoNum, digito, nuevo, exponente, cifra;
                cantDigitos = calcularCantidadDigitos(numero);
                nuevoNum = 0;
                while (numero>0){
                        digito = numero/(int)pow(10,cantDigitos-1);
                        numero = numero %(int)pow(10,cantDigitos-1);
                        cantDigitos--;
                        nuevo = 0;
                        exponente = 0;
                        while (digito>0){
                                cifra = digito %base;
                                digito = digito/base;
                                nuevo = nuevo + cifra*(int)pow(10,exponente);
                                exponente++;
                        }
                        nuevoNum = nuevoNum*(int)pow(10,exponenteK) + nuevo;
```

```
98          }
99          return nuevoNum;
100 }
101
102 int calcularCantidadDigitos(int numero){
103         int cantCifras;
104         cantCifras = 0;
105         while (numero>0){
106                 numero = numero/10;
107                 cantCifras++;
108         }
109         return cantCifras;
110 }
```

3. Número primo de Honaker

Un primo p_n es un primo de Honaker si su índice n y el propio p_n tienen la misma suma de dígitos.

Por ejemplo, $p_{32} = 131$ es un número primo de Honaker porque $3 + 2 = 1 + 3 + 1$. El índice se refiere al í-esimo primo encontrado, por ejemplo: 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 es el primo en la posición 32.

Otros dos ejemplos válidos son: $(p_{88} = 457, p_{457} = 3229)$.

Se le pide que elabore un programa en lenguaje C que permita identificar si un número es de honaker o si el primo en una posición es de Honaker. Para esto debe realizar los siguientes pasos:

- Mostrar al usuario un menú con 3 opciones: A) Verificar si el número es Honaker, B) Hallar el número primo en la posición y verificar si es Honaker C) Terminar las consultas.
- Validar los datos ingresados, recuerde que las opciones válidas son A, B y C (en mayúsculas) y que los números ingresados (ya sea la posición o el número debe ser mayor a 0) y en el caso de la opción A, el número ingresado debe ser primo.
- Si el usuario selecciona la opción A, deberá hallar la posición en la que se encuentra, calcular la suma de los dígitos de la posición y del número y verificar si cumple la condición para ser Honaker. Adicionalmente, debe mostrar el mensaje `El número primo X en la posición Y es Honaker` o `El número primo X en la posición Y no es Honaker`, según corresponda.
- Si el usuario selecciona la opción B, deberá hallar el número que se ubica en la posición dada, calcular la suma de los dígitos de la posición y del número y verificar si cumple la condición para ser Honaker. Adicionalmente, debe mostrar el mensaje `El número primo en la posición Y es X y es Honaker` o `El número primo en la posición Y es X y no es Honaker`, según corresponda.
- Si el usuario selecciona la opción C se deberá dejar que consultar al usuario (termina el programa).
- Si el usuario ingresa una opción que no es válida, debe mostrar el siguiente mensaje `Ha ingresado una opción inválida` y debe volver a mostrar el menú de opciones.

Para su solución debe utilizar programación modular (al menos 5 módulos incluyendo el main), considerando que por lo menos uno de los módulos, que no sea el main, debe tener una estructura iterativa anidada dentro de su implementación.

A continuacion se presentan algunos ejemplos de ejecución de este programa que podrá usar para validar su solución.

```
Opciones:
A) Verificar si el número es Honaker,
```

```
B) Hallar el número primo en la posición y verificar si es Honaker
C) Salir
A
Ingrese el número : 3229
El número primo 3229 en la posicion 457 es Honaker
Opciones:
A) Verificar si el número es Honaker,
B) Hallar el número primo en la posición y verificar si es Honaker
C) Salir
A
Ingrese el número : 3228
El número no es primo
Opciones:
A) Verificar si el número es Honaker,
B) Hallar el número primo en la posición y verificar si es Honaker
C) Salir
C
```

```
Opciones:
A) Verificar si el número es Honaker,
B) Hallar el número primo en la posición y verificar si es Honaker
C) Salir
B
Ingrese la posición : 32
El número primo en la posición 32 es 131 y es Honaker
Opciones:
A) Verificar si el número es Honaker,
B) Hallar el número primo en la posición y verificar si es Honaker
C) Salir
B
Ingrese la posición : 33
El número primo en la posición 33 es 137 y no es Honaker
Opciones:
A) Verificar si el número es Honaker,
B) Hallar el número primo en la posición y verificar si es Honaker
C) Salir
B
Ingrese la posición : -54
La posición debe ser un valor mayor a 0
Opciones:
A) Verificar si el número es Honaker,
B) Hallar el número primo en la posición y verificar si es Honaker
C) Salir
A
Ingrese el número : -54
El número debe ser tener valor mayor a 0
Opciones:
A) Verificar si el número es Honaker,
B) Hallar el número primo en la posición y verificar si es Honaker
C) Salir
F
Ha ingresado una opción inválida
Opciones:
A) Verificar si el número es Honaker,
B) Hallar el número primo en la posición y verificar si es Honaker
C) Salir
C
```

Programa 3: Propuesta de solución - Número primo de Honaker

```
1 #include<stdio.h>
2 int evaluarSiEsPrimo(int numero);
3 int IdentificarPosicion(int num);
4 int HallarElPrimoEnPosicion(int pos);
5 int sumaDigitos(int numero);
```

```
int main(){
	char opcion;
	int esprimo,pos,sumaDig,sumaPos,primo,numero;
	while(1){
		printf("Opciones:\n");
		printf("A) Verificar si el número es Honaker\n");
		printf("B) Hallar el número primo en la posición y verificar si es Honaker\n");
		printf("C) Salir\n");
		scanf("\n %c",&opcion);
		if(opcion=='C'){
			break;
		}
		else{
			if(opcion=='A'){
				printf("Ingrese el número : ");
				scanf(" %d",&numero);
				if(numero>0){
					esprimo=evaluarSiEsPrimo(numero);
					if(esprimo){
						pos=IdentificarPosicion(numero);
						sumaPos=sumaDigitos(pos);
						sumaDig=sumaDigitos(numero);
						if(sumaPos==sumaDig){
							printf("El número primo  %d en la posicion  %d es Honaker\n",numero,
pos);
						}
						else{
							printf("El número primo  %d en la posicion  %d no es Honaker\n",numero
,pos);
						}
					}
					else{
						printf("El número no es primo\n");
					}
				}
				else{
					printf("El número debe ser tener valor mayor a 0\n");
				}
			}
			else{
				if(opcion=='B'){
					printf("Ingrese la posición : ");
					scanf(" %d",&pos);
					if(pos>0){
						primo=HallarElPrimoEnPosicion(pos);
						sumaPos=sumaDigitos(pos);
						sumaDig=sumaDigitos(primo);
						if(sumaPos==sumaDig){
							printf("El número primo en la posición  %d es  %d y es Honaker\n",pos,
primo);
						}
						else{
							printf("El número primo en la posición  %d es  %d y no es Honaker\n",pos
,primo);
						}
					}
					else{
						printf("La posición debe ser un valor mayor a 0\n");
					}
				}
				else{
					printf("Ha ingresado una opción inválida\n");
				}
			}

		}
	}
```

```
69          return 0;
70  }
71  int evaluarSiEsPrimo(int numero){
72          int cont=0,i=1;
73          while(i<=numero){
74                  if(numero %i==0){
75                          cont++;
76                  }
77                  i++;
78          }
79          if(cont==2){
80                  return 1;
81          }
82          else{
83                  return 0;
84          }
85  }
86  int sumaDigitos(int numero){
87          int suma=0,dig;
88          while(numero>0){
89                  dig=numero %10;
90                  suma=suma+dig;
91                  numero=numero/10;
92          }
93          return suma;
94  }
95  int IdentificarPosicion(int num){
96          int i=2,cont=0,esprimo;
97          while(1){
98                  esprimo=evaluarSiEsPrimo(i);
99                  if(esprimo){
100                         cont++;
101                 }
102                 if(i==num){
103                         break;
104                 }
105                 i++;
106         }
107         return cont;
108 }
109 int HallarElPrimoEnPosicion(int pos){
110         int i=2,cont=0,esprimo;
111         while(1){
112                 esprimo=evaluarSiEsPrimo(i);
113                 if(esprimo){
114                         cont++;
115                 }
116                 if(cont==pos){
117                         break;
118                 }
119                 i++;
120         }
121         return i;
122 }
```

4. Subsecuencias Capicúas

Un número es capicúa cuando se lee igual de izquierda a derecha que de derecha a izquierda. Por ejemplo, los números 22, 111111, 343, 5665 y 17371 son capicúas.

Los números pueden ser escritos en un sistema de numeración particular llamada base, como por ejemplo: el número 159 está en base 10, el número 542 está en base 6 o el número 1001 está en base 2.

Se dice que un número está escrito en una base dada cuando todas sus cifras son menores a la base indicada.

En está ocasión, vamos a trabajar con un número escrito en alguna base dada y vamos a detectar cuantas subsecuencias de números cacpicúas se pueden formar tomando en cuenta el orden de sus cifras.

Para entender mejor esto veamos los siguientes ejemplos:

- El número 1221 está escrito en base 3 y tomando en cuenta el orden de sus cifras, sin alterarlas, podemos encontrar 2 subsecuencias que forman números capicúas, las cuales son:

  - El número 22, que se obtiene al tomar la subsecuencia formada por las cifras de las centenas y decenas del número.
  - El número 1221, que se obtiene al tomar la subsecuencia formada por todas las cifras del número.

  Dentro del análisis, hemos descartado casos, como por ejemplo el número 21 obtenido por la subsecuencia formada por las últimas 2 cifras y 221 obtenido por la subsecuencia formada por las últimas 3 cifras, y así para las demás subsecuencias que se pueden obtener y que no son capicúas.

- El número 34343 está escrito en base 6 y tomando en cuenta el orden de sus cifras, sin alterarlas, podemos encontrar 4 subsecuencias que forman números capicúas, las cuales son:

  - El número 343, que se obtiene al tomar la subsecuencia formada por las cifras de las centenas, decenas y unidades del número.
  - El número 434, que se obtiene al tomar la subsecuencia formada por las cifras de las unidades de millar, centenas y decenas del número.
  - El número 343, que se obtiene al tomar la subsecuencia formada por las cifras de las decenas de millar, unidades de millar y centenas del número.
  - El número 34343, que se obtiene al tomar la subsecuencia formada por todas las cifras del número.

  Dentro del análisis, hemos descartado casos, como por ejemplo el número 43 obtenido por la subsecuencia formada por las últimas 2 cifras, el número 34 obtenido por la subsecuencia formada por las cifras de las centenas y decenas, el número 3434 obtenido por la subsecuencia formada por las cifras de las decenas de millar, unidades de millar, centenas y decenas del número y así para las demás subsecuencias que se pueden obtener y que no son capicúas.

Se le pide que elabore un programa en lenguaje C que permita determinar la cantidad de subsecuencias capicúas que se pueden obtener con las cifras de un número que pertenece a una base ingresada, sin alterar el orden de sus cifras.

Dentro del programa debe considerar las siguientes validaciones:

- La base debe estar en el rango de 2 y 10, en caso no se cumpla se debe mostrar el mensaje `La base ingresada no es correcta` y el programa debe terminar.
- El número ingresado debe ser mayor que 0, en caso no se cumpla se debe mostrar el mensaje `El número debe ser mayor que 0` y el programa debe terminar.
- El número ingresado debe estar en la base ingresada, en caso no se cumpla se debe mostrar el mensaje `El número ingresado no corresponde a la base ingresada` y el programa debe terminar. Recuerde que un número esta en una base si todas sus cifras son menores a la base.
- El número ingresado debe tener dos o más cifras, en caso no se cumpla se debe mostrar el mensaje `El número no se puede evaluar porque tiene menos de 2 cifras` y el programa debe terminar.

Para su solución debe utilizar programación modular (al menos 5 módulos incluyendo el main), considerando que por lo menos uno de los módulos, que no sea el main, debe tener una estructura iterativa anidada dentro de su implementación.

Si por algún motivo desea calcular la cantidad de cifras que posee el número, deberá realizar este cálculo a través de una estructura iterativa.

A continuación se presentan algunos ejemplos de ejecución de este programa que podrá usar para validar su solución.

```
Ingrese la base: 12
La base ingresada no es correcta.
```

```
Ingrese la base: 3
Ingrese el número en la base ingresada: 0
El número debe ser mayor que 0.
```

```
Ingrese la base: 3
Ingrese el número en la base ingresada: 4122
El número ingresado no corresponde a la base ingresada.
```

```
Ingrese la base: 6
Ingrese el número en la base ingresada: 5
El número no se puede evaluar porque tiene menos de 2 cifras.
```

```
Ingrese la base: 3
Ingrese el número en la base ingresada: 1221
El número tiene 2 subsecuencias capicúas.
```

```
Ingrese la base: 6
Ingrese el número en la base ingresada: 34343
El número tiene 4 subsecuencias capicúas.
```

```
Ingrese la base: 8
Ingrese el número en la base ingresada: 74542
El número tiene 1 subsecuencias capicúas.
```

```
Ingrese la base: 10
Ingrese el número en la base ingresada: 123
El número tiene 0 subsecuencias capicúas.
```

Programa 4: Propuesta de solución - Subsecuencias Capicúas

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

int validarNumeroEnBase(int numero, int base);
int calcularCantDigitos(int numero);
int calcularCantidadSecuenciasCapicuas(int numero,int cantDigitos);
int verificarCapicua(int numero);
```

```
8
9  int main(){
10         int numero, esValido, cantDigitos, cantSecuenciasCapicuas, base;
11         printf("Ingrese la base: ");
12         scanf(" %d",&base);
13         if (base>=2 && base<=10){
14                 printf("Ingrese el número en la base indicada: ");
15                 scanf(" %d",&numero);
16                 if (numero>0){
17                         esValido = validarNumeroEnBase(numero,base);
18                         if (esValido){
19                                 cantDigitos = calcularCantDigitos(numero);
20                                 if (cantDigitos>=2){
21                                         cantSecuenciasCapicuas = calcularCantidadSecuenciasCapicuas(numero,cantDigitos);
22                                         printf("El número  %d tiene  %d subsecuencias capicúas",numero,cantSecuenciasCapicuas);
23                                 }
24                                 else {
25                                         printf("El número no se puede evaluar porque tiene menos de 2 cifras\n");
26                                 }
27                         }
28                         else {
29                                 printf("El número ingresado no corresponde a la base ingresada");
30                         }
31                 }
32                 else {
33                         printf("El número debe ser mayor que 0");
34                 }
35         }
36         else {
37                 printf("La base ingresada no es correcta");
38         }
39         return 0;
40 }
41
42 int calcularCantDigitos(int numero){
43         int cantDigitos;
44         cantDigitos = 0;
45         while (numero>0){
46                 numero = numero/10;
47                 cantDigitos++;
48         }
49         return cantDigitos;
50 }
51
52 int calcularCantidadSecuenciasCapicuas(int numero,int cantDigitos){
53         int total, i, j, subSecuencia, esCapicua, cantidad;
54         cantidad = 0;
55         total = cantDigitos;
56         for (i=1; i<cantDigitos; i++){
57                 for (j=2; j<=total; j++){
58                         subSecuencia = numero %(int)pow(10,j);
59                         esCapicua = verificarCapicua(subSecuencia);
60                         if (esCapicua){
61                                 printf(" %d\n",subSecuencia);
62                                 cantidad++;
63                         }
64                 }
65                 total--;
66                 numero = numero/10;
67         }
68         return cantidad;
69 }
70
71 int validarNumeroEnBase(int numero, int base){
72         int esCorrecto = 1, digito;
73         while (numero>0){
74                 digito = numero %10;
```

```
75                      numero = numero/10;
76                      if (digito>=base){
77                              esCorrecto=0;
78                              break;
79                      }
80              }
81              return esCorrecto;
82      }
83
84      int verificarCapicua(int numero){
85              int invertido, copiaNumero, digito, esCapicua;
86              invertido = 0;
87              copiaNumero = numero;
88              while (numero>0){
89                      digito = numero %10;
90                      numero = numero/10;
91                      invertido = invertido*10 + digito;
92              }
93              if (invertido==copiaNumero){
94                      esCapicua = 1;
95              }
96              else {
97                      esCapicua = 0;
98              }
99              return esCapicua;
100     }
```

Puede usar cualquier estructura selectiva y cualquier estructura iterativa