Práctica No. 3

Estructuras de control

Competencia:
Elaborar programas donde se utilicen estructuras de repetición para la solución de problemas reales en el área de ingeniería, con disposición para trabajar en forma individual y responsabilidad en el uso del equipo cómputo del laboratorio.

Descripción:
Implementación de programas utilizando estructuras de repetición disponibles.

Antecedentes:

ESTRUCTURAS DE CONTROL REPETITIVAS

Las estructuras de control repetitivas realizan 'n' veces una operación ó bloque de sentencias, conociendo con anticipación el número de iteraciones a realizar.
La estructura de control repetitiva for realiza 'n' veces una instrucción de manera fija como un contador.

```
for(variable_control=valor_inicial;expresión_condicional;expresión_incremento) {
	bloque de sentencias;
}
```

La estructura de control repetitiva while el número de iteraciones no se conoce por anticipado y las sentencias se repiten MIENTRAS se cumple determinada condición.

```
while (expresión_condicional) {
	bloque de sentencias;
}
```

La estructura de control repetitiva do_while es una variación de la estructura while, el operador while evalúa la condición al principio de cada iteración. El operador do efectúa un bloque de sentencias hasta que la condición se haga verdadera, y por lo menos lo realizará una vez. Su función se caracteriza por un REPITE-HASTA.

```
do {
	bloque de sentencias;
} while (expresión_condicional);
```

Actividad:
Realizar un programa que reciba parámetros por consola, dependiendo del primer parámetro (que llamaremos **opción**) será la tarea a realizar. El número de parámetros esperados depende de la tarea a realizar.
Cada una de las tareas deberá estar en una **biblioteca** implementada por el alumno (basarse

en la práctica 2). El archivo que contenga el método main (ej. prac3.c) solo deberá analizar los parámetros recibidos desde la consola e imprimir mensajes.
Si el usuario no ingresa los parámetros de forma correcta se deberá mostrar un mensaje indicando al usuario las opciones disponibles y ejemplos de cómo podría ejecutar el programa. Las tareas que debe realizar el programa son las siguientes:

1. Si la opción es "**reversa**" el programa deberá recibir **un número** entero para escribirlo en reversa.

2. Si la opción es "**digitos**" el programa deberá recibir **un número** entero para imprimir el número de dígitos de éste.

3. Si la opción es "**suma**" el programa deberá recibir **un número** entero para imprimir la suma de sus dígitos.

4. Si la opción es "**perfecto**" el programa deberá recibir **un número** entero para imprimir si es un número perfecto o no. Un número perfecto es un entero que es igual a la suma de los divisores propios menores que él mismo. Así, 6 es un número perfecto, porque sus divisores propios son 1, 2 y 3; y 6 = 1 + 2 + 3.

5. Si la opción es "**amigos**" el programa deberá recibir **dos números** enteros para imprimir si son amigos. Dos números amigos son dos enteros positivos tales que la suma de los divisores propios de uno de ellos es igual al otro (la unidad se considera divisor propio, pero no lo es el mismo número).

Ejemplos de ejecución:

```
>prac3.exe reversa 172
>El número 172 al revés es: 271

>prac3.exe amigos 8 7
>Los números 8 y 7 son amigos porque la suma de los divisores propios de 8 es igual a 7
```

Restricciones

1. El código fuente deberá contener comentarios que indique el objetivo del programa y descripción de instrucciones más relevantes.
2. Evitar nombres de variables y funciones que no reflejen su propósito claramente.
3. Seguir la estrategia divide y vencerás, desglosar el programa en pequeñas funciones que tengan un solo propósito y una sola acción. Esto se debería reflejar en una función principal (`main`) con pocas instrucciones.
4. Se debe evitar repetir código en medida de lo posible, se deben realizar las funciones de tal forma que puedan ser reutilizables por otras funciones.

Conclusiones y Comentarios.