Pontificia Universidad Católica del Perú

Estudios Generales Ciencias

1INF01 - Fundamentos de Programación

Guía de laboratorio #7

Ciclos Iterativos Anidados

FUNDAMENTOS DE PROGRAMACIÓN
ESTUDIOS GENERALES CIENCIAS
PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

30 de octubre de 2021

Índice general

FUNDAMENTOS DE PROGRAMACIÓN
ESTUDIOS GENERALES CIENCIAS
PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FUNDAMENTOS DE PROGRAMACIÓN
ESTUDIOS GENERALES CIENCIAS
PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FUNDAMENTOS DE PROGRAMACIÓN
ESTUDIOS GENERALES CIENCIAS
PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Historial de Revisiones

Revisión	Fecha	Autor(es)	Descripción
1.0	11.05.2018	A. Melgar	Versión inicial.
2.0	18.05.2019	A. Melgar	Se incrementó la cantidad de problemas propuestos, se añadió color al código en lenguaje C y se completaron los casos de prueba de los problemas propuestos.
2.0	30.05.2019	L. Hirsh	Revisión de la versión 2.0.
2.1	27.08.2019	A. Melgar	Se modificaron algunos ejemplos en lenguaje C para incluir también la versión del programa usando funciones definidas por el usuario.
2.2	12.11.2020	A. Melgar	Se incluyen nuevos ejercicios, se hace revisión del material y reorganización.
2.3	07.06.2021	A. Melgar	Se incluyen nuevos ejercicios.
2.4	29.10.2021	J. Zárate y L. Hirsh	Se incluyen nuevos ejercicios de nivel avanzado y se revisa el material.

FUNDAMENTOS DE PROGRAMACIÓN
ESTUDIOS GENERALES CIENCIAS
PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Siglas

ASCII	American Standard Code for Information Interchange
ANSI	American National Standards Institute
EEGGCC	Estudios Generales Ciencias
IDE	Entorno de Desarrollo Integrado
OMS	Organización Mundial de la Salud
PUCP	Pontificia Universidad Católica del Perú
RAE	Real Academia Española

FUNDAMENTOS DE PROGRAMACIÓN
ESTUDIOS GENERALES CIENCIAS
PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Capítulo 1

Guía de Laboratorio #7

1.1. Introducción

Esta guía ha sido diseñada para que sirva como una herramienta de aprendizaje y práctica para el curso de Fundamentos de Programación de los Estudios Generales Ciencias (EEGGCC) en la Pontificia Universidad Católica del Perú (PUCP). En particular se focaliza en el tema "Ciclos iterativos anidados".

Se busca que el alumno resuelva paso a paso las indicaciones dadas en esta guía contribuyendo de esta manera a los objetivos de aprendizaje del curso, en particular con la comprensión de las estructuras algorítmicas iterativas anidadas y la instrucción `for`. Al finalizar el desarrollo de esta guía y complementando lo que se realizará en el correspondiente laboratorio, se espera que el alumno:

- Comprenda el funcionamiento de los ciclos iterativos anidados.
- Diseñe algoritmos que contengan ciclos iterativos anidados expresados en pseudocódigos y diagramas de flujo.
- Implemente algoritmos que contengan ciclos iterativos anidados usando un lenguaje de programación imperativo.
- Implemente ciclos iterativos usando instrucciones de ruptura de ciclos.

1.2. Materiales y métodos

Como herramienta para el diseño de pseudocódigos y diagramas de flujo se utilizará PSeInt[1]. El PSeInt deberá estar configurado usando el perfil PUCP definido por los profesores del curso. Como lenguaje de programación imperativo se utilizará el lenguaje C. Como Entorno de Desarrollo Integrado (IDE) para el lenguaje C se utilizará Dev C++[2]. No obstante, es posible utilizar otros IDEs como Netbeans y Eclipse.

1.3. Ciclos Iterativos Anidados

Como ya se vio anteriormente, las estructuras de control de flujo permiten que se modifique el flujo de ejecución de las instrucciones que forman parte de un programa. En particular, las estructuras algorítmicas iterativas permiten que los programas ejecuten un conjunto de instrucciones tantas veces como sea necesario. Esto es muy útil para resolver problemas que requieren realizar cálculos de forma repetitiva. Usando estructuras algorítmicas iterativas es posible, por ejemplo, recorrer un rango de valores $[a, b]$, generar los términos de una

[1] http://pseint.sourceforge.net/
[2] https://sourceforge.net/projects/orwelldevcpp/

FUNDAMENTOS DE PROGRAMACIÓN
ESTUDIOS GENERALES CIENCIAS
PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

sucesión, hallar el valor de sumatorias y productorias, ejecutar cálculos reiterativos sobre un conjunto de datos, entre otros.

Recordar que:

- Las estructuras algorítmicas iterativas se clasifican:
  - ciclo iterativo con entrada controlada.
  - ciclo iterativo con salida controlada.
- En el ciclo iterativo con entrada controlada, la evaluación de la condición del ciclo se realiza antes de la ejecución del conjunto de instrucciones. Dependiendo de la condición, puede que el conjunto de instrucciones no se ejecute.
- En el ciclo iterativo con salida controlada, la evaluación de la condición del ciclo se realiza después de la ejecución del conjunto de instrucciones. Se ejecuta por lo menos 1 vez el conjunto de instrucciones.

A menudo existen situaciones en donde es necesario realizar cálculos en cada ciclo[3] dentro de una estructura algorítmica iterativa. A menudo, para realizar estos cálculos, es necesario utilizar otra estructura algorítmica iterativa. Esta situación configura lo que se conoce como ciclo iterativo anidado. Un ciclo iterativo anidado no es más que una estructura algorítmica iterativa, que dentro del bloque de instrucciones que ejecuta en su *bucle*, tiene otra estructura algorítmica iterativa.

En caso de ser necesario anidar dos estructuras algorítmicas iterativas, se tendría las siguientes opciones:

- Un ciclo iterativo con entrada controlada dentro de otro ciclo iterativo con entrada controlada.
- Un ciclo iterativo con salida controlada dentro de otro ciclo iterativo con salida controlada.
- Un ciclo iterativo con entrada controlada dentro de un ciclo iterativo con salida controlada.
- Un ciclo iterativo con salida controlada dentro de un ciclo iterativo con entrada controlada.

Pero es posible anidar más de dos estructuras algorítmicas iterativas por lo que las opciones de combinación que se disponen para escribir algoritmos y programas es vasta. ¿De cuántas maneras puede anidar 3 estructuras algorítmicas iterativas?

Una de las principales desventajas que se produce al utilizar ciclos iterativos anidados es el decremento del rendimiento del programa. Mientras más ciclos iterativos anidados se tengan, más lenta será la ejecución del programa. Es recomendable evitar su uso siempre que esto sea posible. Pero claro, existen situaciones en donde el uso de ciclos anidados se hace inevitable.

En conclusión, un ciclo iterativo anidado no es en sí una estructura de control de flujo nueva, sino la utilización de estructuras algorítmicas iterativas dentro de otras estructuras algorítmicas iterativas. Esta guía se enfocará en la anidación de ciclos iterativo con entrada controlada.

1.3.1. Ciclos iterativos anidados en pseudocódigo

Tal como se vio anteriormente, es posible anidar más de un ciclo iterativo con entrada controlada por lo que no es posible presentar todas las posibles combinaciones. A modo de ejemplo se presentará la anidación de 2 ciclos iterativos con entrada controlada. En la figura 1.1 se puede apreciar un ciclo iterativo anidado en pseudocódigo.

[3] también llamado de *bucle*, *loop* o iteración.

FUNDAMENTOS DE PROGRAMACIÓN
ESTUDIOS GENERALES CIENCIAS
PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Mientras *condición1* **Hacer**
  conjunto de instrucciones 1a;
  Mientras *condición2* **Hacer**
    conjunto de instrucciones 2;
  FinMientras
  conjunto de instrucciones 1b;
FinMientras

Figura 1.1: Pseudocódigo: Ciclo iterativo anidado

Recordar que:

En un ciclo iterativo con entrada controlada se deben administrar 3 situaciones:

1. La inicialización de la(s) variable(s) de control que se utilizará(n) en la condición. Esto se realiza antes de iniciar el ciclo.
2. La definición de la condición, usando la(s) variable(s) de control.
3. El aseguramiento que la condición en algún momento llegue a finalizar. Normalmente se realiza en la última instrucción del ciclo.

En general algunas consideraciones se deben tener en cuenta al trabajar con ciclos anidados. Cada anidación agrega complejidad al algoritmo por lo que es importante manterner nombres adecuados de variables. Por ejemplo, cuando se usan ciclos anidados que son controlados por contadores, se suele utilizar i, j, k, l, como nombres de las variables de control. Por lo general i corresponde a la variable de control del ciclo más externo.

1.3.2. Ciclos iterativos anidados en diagrama de flujo

En la figura 1.2 se puede apreciar un ciclo iterativo con entrada controlada anidado representado usando el diagrama de flujo. En el se puede apreciar el ciclo interno formado por el bloque de instrucciones luego de la condición 2. El ciclo externo está formado por el bloque de instrucciones luego de la condición 1, incluyendo el ciclo iterativo interno.

1.3.3. Ciclos iterativos anidados en lenguaje C

En el programa 1.1 se puede apreciar un ciclo iterativo con una entrada controlada anidado implementado en lenguaje C. A pesar que para el compilador de lenguaje C le es indiferente la forma en que se escribe el código, es importante que este se encuentre identado[4] para que sea entendible al ojo humano. Con una correcta identación el ciclo más interno se distingue fácilmente.

Programa 1.1: Lenguaje C: Ciclo iterativo anidado

```
1
2       while (condición1){
3               conjunto de instrucciones1a;
4               while (condición2){
5                       conjunto de instrucciones2;
6               }
7               conjunto de instrucciones1b;
8       }
```

[4]Movimiento de un bloque de instrucciones a la derecha, muy similar al sangrado o sangría de los documentos de texto

FUNDAMENTOS DE PROGRAMACIÓN
ESTUDIOS GENERALES
PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

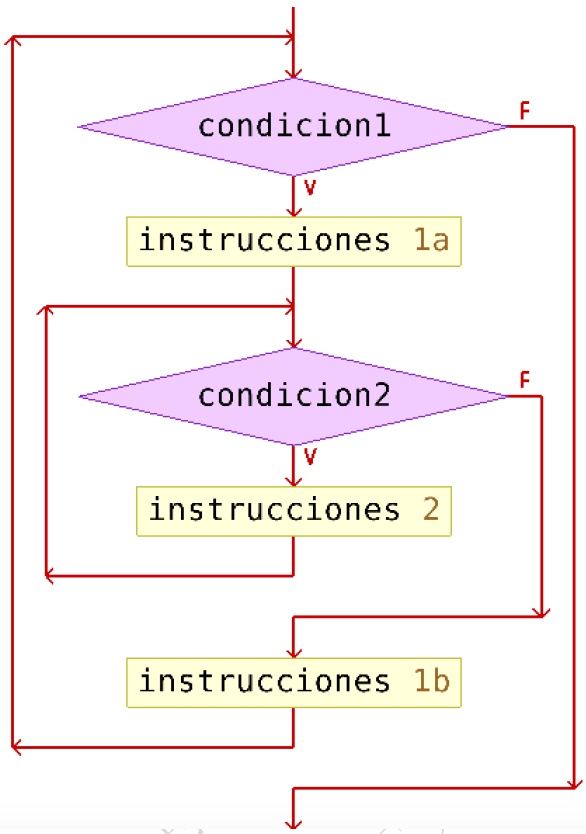


Figura 1.2: Diagrama de Flujo: Ciclo iterativo anidado

1.4. El triángulo de Pascal

Blaise Pascal fue un matemático francés que popularizó este triángulo de números. Está formado por una serie de números distribuidos en infinitas filas y alineados de una forma particular. En la figura 1.3 se puede apreciar las primeras 5 filas del triángulo de Pascal.

1

1 1

1 2 1

1 3 3 1

1 4 6 4 1

Figura 1.3: Triángulo de Pascal

El triángulo de Pascal se construyen de la siguiente forma:

- La primera fila posee solamente un 1.
- La segunda posee dos 1, a izquierda y derecha del uno anterior: 1 1.
- La tercera fila se construye de la siguiente manera: se suman los dos números de la fila anterior y se coloca el resultado (en este caso 2 que resulta de sumar $1 + 1$), debajo del espacio que dejan dichos números, y a izquierda y derecha se colocan dos 1. El resultado final es: 1 2 1 .

FUNDACIÓN DE ESTUDIOS GENERALES PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

- La filas posteriores, se construyen de forma muy similar a la tercera: debajo de cada espacio entre dos números de la fila anterior se escribe la suma de dichos números, y a izquierda y derecha se coloca dos 1. Por ejemplo, la fila 3 quedaría así: 1 3 3 1, la fila 4 así: 1 4 6 4 1.

Con el triángulo de Pascal es posible calcular las potencias de binomios $(a+b)^n$, donde a y b son variables cualquiera y n corresponde con el exponente que define la potencia. Los números de cada fila del triángulo de Pascal está relacionada con los coeficientes binomiales, de esta forma la primera fila (1) contiene el coeficiente para el binomio $(a+b)^0$, la segunda fila (1 1) contiene los coeficientes para el binomio $(a+b)^1$, la tercera fila (1 2 1) contiene los coeficientes para el binomio $(a+b)^2$, la cuarta fila (1 3 3 1) contiene los coeficientes para el binomio $(a+b)^3$ y así sucesivamente.

Recordar que:

$$(a+b)^0 = 1$$
$$(a+b)^1 = 1a + 1b$$
$$(a+b)^2 = 1a^2 + 2ab + 1b^2$$
$$(a+b)^3 = 1a^3 + 3a^2b + 3ab^2 + 1b^3$$
$$\ldots$$
$$(a+b)^n = k_1a^n + k_2a^{n-1}b + k_3a^{n-2}b^2 + \cdots + k_{n-1}ab^{n-1} + k_nb^n$$

¿De qué manera se puede automatizar la generación de las filas del triángulo de Pascal? Para poder responder a esta pregunta es necesario recordar el teorema del binomio. Este teorema estable que:

$$(x+y)^n = \sum_{k=0}^{n} \frac{n!}{k!(n-k)!} x^{n-k}y^k$$

La misma fórmula se puede expresar usando el combinatorio de la siguiente manera:

$$(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^{n-k}y^k$$

Recordar que:

Se define número combinatorio como el valor numérico de las combinaciones ordinarias (sin repetición) de un conjunto de p elementos tomados en grupos de r, siendo p y r dos números enteros y positivos tales que $p \geq r$. Matemáticamente, un número combinatorio se expresa como:

$$C_r^p = \binom{p}{r} = \frac{p!}{r! \times (p-r)!}$$

En realidad la parte de la sumatoria que interesa para la generación del triángulo de Pascal es la relacionada a los coeficientes, es decir $\sum_{k=0}^{n} \binom{n}{k}$. La única consideración que debemos tener es que en realidad no se realizará una suma en lugar de sumar se imprimirá cada término de la sumatoria. En la figura 1.4 se puede apreciar el triángulo de Pascal presentado anteriormente pero con los número expresados en función de los números combinatorios.

Sabiendo esto, ¿cómo imprimir un triángulo de Pascal dada una determinada cantidad n de filas? Se propondrá una alternativa de solución a este problema adoptando una estrategia *bottom-up*, es decir de "de abajo hacia arriba". Primero se calculará un número combinatorio, posteriormente se generará una determinada fila del triángulo de Pascal para finalmente presentar un algoritmo que imprima las n primeras filas del triángulo de Pascal. Se utilizarán estructuras algorítmicas iterativas anidadas. La solución se presentará tanto en diagrama de flujo como en pseudocódigo.

$$\binom{0}{0}$$

$$\binom{1}{0} \quad \binom{1}{1}$$

$$\binom{2}{0} \quad \binom{2}{1} \quad \binom{2}{2}$$

$$\binom{3}{0} \quad \binom{3}{1} \quad \binom{3}{2} \quad \binom{3}{3}$$

$$\binom{4}{0} \quad \binom{4}{1} \quad \binom{4}{2} \quad \binom{4}{3} \quad \binom{4}{4}$$

Figura 1.4: Coeficientes del triángulo de Pascal

1.4.1. Cálculo del número combinatorio

Vamos a "olvidarnos" por un instante del problema de la impresión de las filas del triángulo de Pascal para enfocarnos en el cálculo del número combinatorio. Como podrá recordar $C_r^p = \binom{p}{r} = \frac{p!}{r! \times (p-r)!}$. Entonces el problema se reduce al cálculo de 3 números factoriales dado 2 variables como entradas (en este caso p y r con $p \geq r$). Los números factoriales a calcular serían: $p!$, $r!$ y $(p-r)!$.

Anteriormente se ha visto cómo realizar el cálculo de el factorial de un número. Básicamente es la productoria $\prod_{i=1}^{n} i$. La implementación de esta productoria se realiza con un ciclo iterativo con entrada controlada usando un contador para recorrer desde i hasta n y una variable que almacene la productoria. Al finalizar el ciclo, la variable que almacena la productoria tendría el factorial requerido.

Pero, ¿cómo calcular los 3 números factoriales requeridos? Una primera idea podría ser la utilización de 3 ciclos iterativos secuenciales, uno detrás de otro. En el primero se recorre con un contador hasta p, luego en el segundo se recorre hasta r y finalmente en el tercer ciclo se recorre hasta $(p-r)$. Dentro de cada ciclo iterativo se realizan los cálculos respectivos de la productoria. A pesar que esta idea permitiría obtener los factoriales requeridos, no es una alternativa de solución eficiente pues se tendría 3 ciclos iterativos, lo cual es complicado de entender y sobre todo complica el mantenimiento del programa.

Entonces, ¿cómo realizar el cálculo en una sola iteración? Se sabe que $p \geq r$, por lo tanto se puede deducir que $p \geq (p-r)$. De la definición del factorial se puede afirmar que $p! \geq r!$ y $p! \geq (p-r)!$. En términos prácticos significa que recorriendo el número mayor, en este caso p, se pueden calcular los 3 factoriales. Esto es posible pues los rangos de los otros números son menores y son un subconjunto de $[1..p]$. En la propuesta se utilizará una variable denominada *contador* para recorrer el rango de $[1..p]$. Se utilizarán 3 variables para almacenar los factoriales $fact\_p$ que almacenará el factorial de p, $fact\_r$ que almacenará el factorial de r y $fact\_p\_menos\_r$ que almacenará el factorial de $(p-r)$. Todos estas variables se inicializarán en 1.

El cálculo de $fact\_p$ es similar a los casos vistos anteriormente. La diferencia está en el calculo de $fact\_r$ y $fact\_p\_menos\_r$. En el caso de $fact\_r$ solo se realizará la productoria si es que $contador \leq r$. En el caso de $fact\_p\_menos\_r$ solo se realizará la productoria si es que $contador \leq (p-r)$. Todo esto se realiza en el mismos ciclo iterativo.

Al final del ciclo iterativo se tendrán los valores requeridos de los factoriales en $fact\_p$, $fact\_r$ y $fact\_p\_menos\_r$ respectivamente. Usando estas variables se puede calcular fácilmente el $\binom{p}{r}$. En la figura 1.5 se puede apreciar un algoritmo representado en diagrama de flujo que lee dos números e imprime el combinatorio de ambos. No se ha realizado la validación de $p \geq r$, $p \geq 0$ y $r \geq 0$ pues a pesar que en este algoritmos son leídos, en el siguiente p y r serán tomados de datos de otro ciclo iterativo.

FACULTAD DE CIENCIAS E INGENIERÍA
DEPARTAMENTO ACADÉMICO DE CIENCIAS
PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

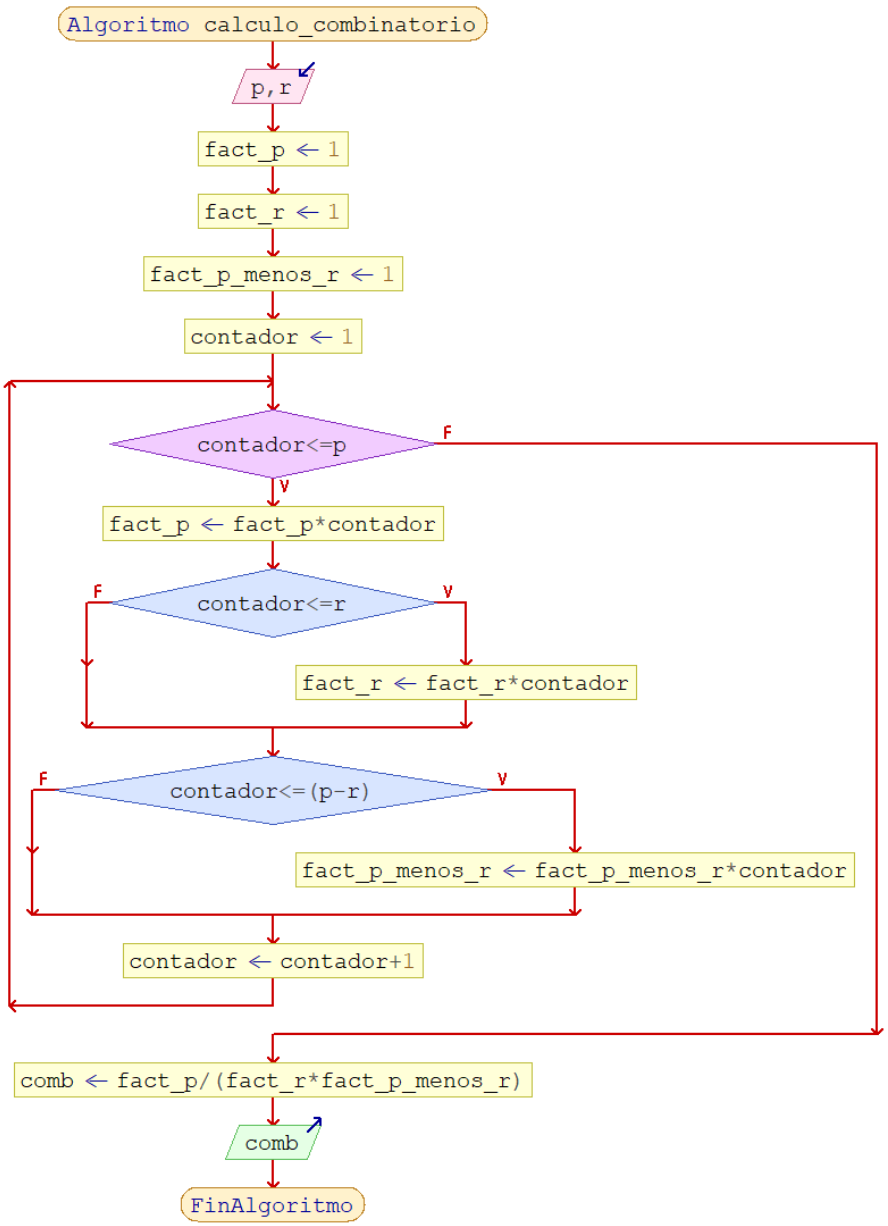


Figura 1.5: Diagrama de flujo: Cálculo del número combinatorio

Para poner en práctica

- ¿Qué cambios debería realizar en el diagrama de flujo si en lugar de calcular los factoriales en un solo ciclo iterativo utiliza tres ciclos iterativos secuenciales?
- ¿Qué cambios debería realizar en el diagrama de flujo si se requiere verificar que $p \geq r$, $p \geq 0$ y $r \geq 0$?
- Exprese este mismo algoritmo en pseudocódigo.
- Implemente este algoritmo usando el lenguaje C.

1.4.2. Cálculo de una fila del triángulo de Pascal

Ahora que se sabe cómo calcular el combinatorio de dos números, se procederá a imprimir una determinada fila del triángulo de Pascal. Recuerde que se intenta resolver la sumatoria $\sum_{k=0}^{n} \binom{n}{k}$, en donde n representa el exponente del binomio así como el número de la fila del triángulo. Recordemos además que en lugar de acumular la suma, se estarán imprimiendo los valores de la sumatoria.

Dado un número de fila n del triángulo de Pascal esta fila tendrá n números que se calculan como sigue:

si $n = 1 \Rightarrow p = 0 \Rightarrow \binom{0}{0}$
si $n = 2 \Rightarrow p = 1 \Rightarrow \binom{1}{0}\ \binom{1}{1}$
si $n = 3 \Rightarrow p = 2 \Rightarrow \binom{2}{0}\ \binom{2}{1}\ \binom{2}{2}$
si $n = 4 \Rightarrow p = 3 \Rightarrow \binom{3}{0}\ \binom{3}{1}\ \binom{3}{2}\ \binom{3}{3}$
si $n = k \Rightarrow p = k - 1 \Rightarrow \binom{p}{0}\ \binom{p}{1}\ \binom{p}{2}\ \binom{p}{3} \cdots \binom{p}{p-1}\ \binom{p}{p}$

Como se puede apreciar dado un número de fila n se puede calcular fácilmente el número p del combinatorio $\binom{p}{r}$, $p = n - 1$. Además se puede apreciar que r varía en el rango $[0..p]$. Como ya se vio anteriormente, un rango se puede recorrer con una estructura iterativa con entrada controlada. Con p y r se puede calcular el combinatorio usando la propuesta de la sección anterior. Como la propuesta para el cálculo se basa en una iterativa con entrada controlada y como para recorrer el rango $[0..p]$ también se usará una iterativa con entrada controlada, estamos frente a un caso de estructuras iterativas anidadas.

En el ciclo más externo de la estructura anidada tiene por objetivo recorrer el rango $[0..p]$. Para esto se lee la variable n la cual representará el número de la fila. Para iterar se usará la variable $cont\_n$ que será inicializado con 0 e irá hasta $n - 1$ que en nuestro análisis representa el valor p del combinatorio. La condición de esta iteración es $cont\_n < n$, no toma n sino llega hasta $n - 1$. Se hubiera podido colocar como condición también $cont\_n \leq (n - 1)$.

En cada iteración se calcula el valor de p y r. A p se le asigna $n-1$ y a r el valor en curso del $cont\_n$. Recuerde que p permanece fijo[5] en la impresión de la fila y r varía. Con estos valores se calcula el número combinatorio $\binom{p}{r}$.

Para el cálculo del número combinatorio, antes de iniciar el ciclo interno, se inicializan las variables que contendrán el factorial ($fact\_p$, $fact\_r$, $fact\_p\_menos\_r$) con 1 así como el contador del ciclo interno ($cont\_fact$[6]). El ciclo interno es en esencia el mismo de la sección anterior.

En la figura 1.6 se puede apreciar un algoritmo representado en diagrama de flujo que lee un número de fila del triángulo de Pascal e imprime los números que conforman la fila. No se ha realizado la validación de $n \geq 0$ pues a pesar que en este algoritmos n es leído, en el siguiente algoritmo n será tomado como dato de otro ciclo iterativo.

Para poner en práctica

- Si en el ciclo iterativo más externo cambia la condición de $cont\_n < n$ por $cont\_n \leq (n-1)$, ¿el algoritmo retorna los mismos resultados?
- Si la instrucción $p \leftarrow n - 1$ es movida para fuera del ciclo iterativo más externo, ¿el algoritmo retorna los mismos resultados?
- Exprese este mismo algoritmo en pseudocódigo.
- Implemente este algoritmo usando el lenguaje C.

[5] Dado que p tiene siempre el mismo valor en la iteración, este podría ser inicializado fuera del ciclo iterativo. En esta alternativa de solución se optó por mantenerlo dentro de la iteración para no confundir en la solución final.

[6] En el algoritmo anterior está variable se llamaba *contador*, es este algoritmo se ha optado por cambiar de nombre pues se utilizan dos contadores en dos ciclos diferentes. De esta forma queda evidente que se refiere al contador para el cálculo del factorial.

FUNDAMENTOS DE PROGRAMACIÓN
ESTUDIOS GENERALES CIENCIAS
UNIVERSIDAD CATÓLICA DEL PERÚ

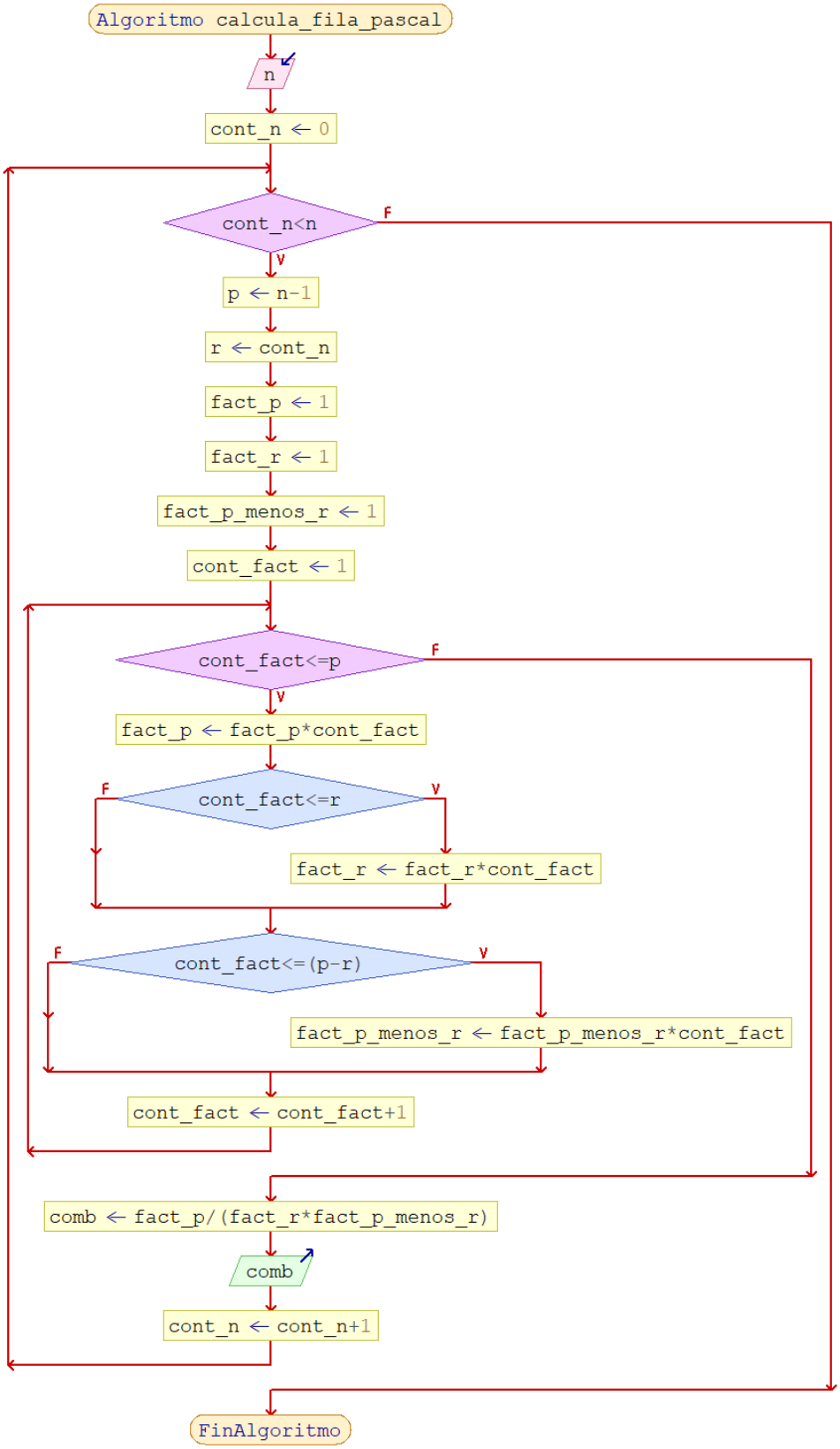


Figura 1.6: Diagrama de flujo: Cálculo de una fila del triángulo de Pascal

1.4.3. Impresión del triángulo de Pascal

Hasta el momento se ha podido imprimir los números de determinada fila n del triángulo de Pascal. ¿Cómo hacer para imprimir k filas? Si se desean imprimir k filas se tendrá que utilizar el algoritmo anterior k veces, usando un contador de filas para recorrer el rango $[1..k]$. Estamos nuevamente frente a una situación en donde se tiene un ciclo iterativo anidado.

Se utilizará la variable $total\_filas$ para solicitar al usuario el número de filas que se requiere imprimir. Se utilizará además la variable $cont\_filas$ para recorrer el rango $[1..total\_filas]$. Para esto se utilizará un ciclo iterativo con entrada controlada usando la condición $cont\_filas \leq total\_filas$. Lo que se hace en cada ciclo es invocar al algoritmo anterior haciendo que en cada iteración n sea igual a $cont\_filas$. Recuerde que en el algoritmo anterior n representaba al número de fila que es justamente lo que se está recorriendo en este algoritmo.

En la figura 1.7 se puede apreciar el pseudocódigo que permite imprimir las primeras k filas del triángulo de Pascal. En la 1.8 se puede apreciar la salida obtenida por este pseudocódigo cuando el usuario ingresa el valor de 4 a la variable $total\_filas$.

```
1  Algoritmo triangulo_de_pascal
2      Escribir 'Ingrese cantidad de filas: '
3      Leer total_filas
4      cont_filas<-1
5      Mientras  cont_filas<=total_filas Hacer
6          Escribir 'Fila: ', cont_filas
7          n<-cont_filas
8          cont_n <- 0
9          Mientras cont_n<n Hacer
10             p <- n-1
11             r <- cont_n
12             fact_p <- 1
13             fact_r <- 1
14             fact_p_menos_r <- 1
15             cont_fact <- 1
16             Mientras cont_fact<=p Hacer
17                 fact_p <- fact_p*cont_fact
18                 Si cont_fact<=r Entonces
19                     fact_r <- fact_r*cont_fact
20                 FinSi
21                 Si cont_fact<=(p-r) Entonces
22                     fact_p_menos_r <- fact_p_menos_r*cont_fact
23                 FinSi
24                 cont_fact <- cont_fact+1
25             FinMientras
26             comb <- fact_p/(fact_r*fact_p_menos_r)
27             Escribir comb
28             cont_n <- cont_n+1
29         FinMientras
30         cont_filas <- cont_filas+1
31     FinMientras
32 FinAlgoritmo
```

Figura 1.7: Pseudocódigo: Impresión del triángulo de Pascal

Para poner en práctica

- ¿Qué alteraciones debería realizar en el algoritmo si se desea imprimir solamente las filas pares del triángulo de Pascal?
- ¿Qué alteraciones debería realizar en el algoritmo si se desea sumar los números del triángulo de Pascal?

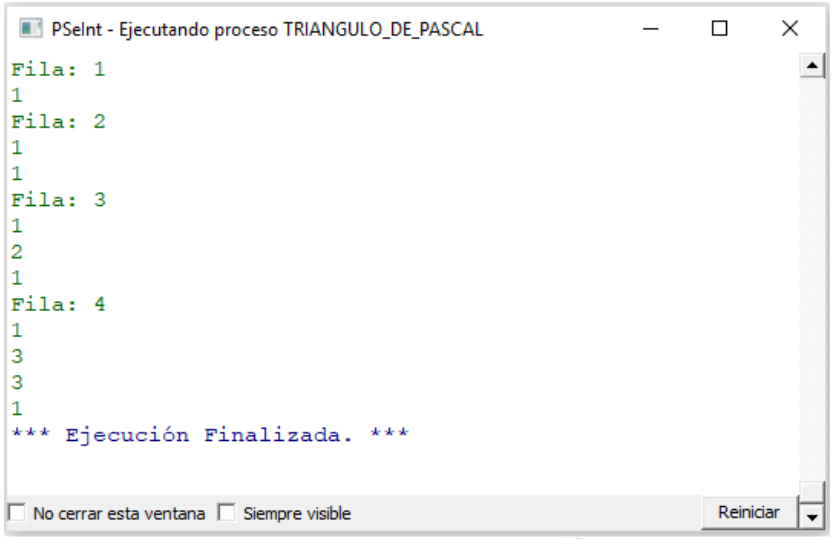


Figura 1.8: Salida: Impresión del triángulo de Pascal

En el programa 1.2 se puede apreciar la versión del triángulo de Pascal implementado en lenguaje C. En el programa 1.3 se apreciar la versión del programa 1.2 pero usando funciones definidas por el usuario. Como podrá notar la versión usando funciones es más larga, posee más líneas de código, pero, ¿qué versión es más entendible al ojo humano?

Programa 1.2: Triángulo de Pascal implementado en lenguaje C

```c
#include <stdio.h>

int main() {
    int total_filas, cont_filas;

    printf("Ingrese cantidad de filas: ");
    scanf("%d", &total_filas);

    cont_filas = 1;
    while (cont_filas <= total_filas) {
        printf("Fila: %d\n", cont_filas);

        int n = cont_filas;
        int cont_n = 0;
        while (cont_n < n) {
            int p = n - 1;
            int r = cont_n;
            int fact_p = 1, fact_r = 1, fact_p_menos_r = 1;
            int cont_fact = 1;
            while (cont_fact <= p) {
                fact_p *= cont_fact;
                if (cont_fact <= r)
                    fact_r *= cont_fact;
                if (cont_fact <= p - r)
                    fact_p_menos_r *= cont_fact;
                cont_fact++;
            }
            int comb= fact_p/(fact_r*fact_p_menos_r);
            printf("%d ", comb);
            cont_n++;
        }
        cont_filas++;
        printf("\n");
    }
    return 0;
}
```

FUNDAMENTOS DE PROGRAMA
ESTUDIOS GENERALES CIENCIAS
PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Programa 1.3: Triángulo de Pascal implementado en lenguaje C usando funciones definidas por el usuario

```c
#include <stdio.h>

void imprime_triangulo_de_pascal(int total_filas);
void imprime_fila(int cont_filas);
int combinatorio(int p, int r);

int main() {
    int total_filas;

    printf("Ingrese cantidad de filas: ");
    scanf("%d", &total_filas);
    imprime_triangulo_de_pascal(total_filas);

    return 0;
}

void imprime_triangulo_de_pascal(int total_filas) {
    int cont_filas = 1;
    while (cont_filas <= total_filas) {
        printf("Fila: %d\n", cont_filas);
        imprime_fila(cont_filas);
        cont_filas++;
        printf("\n");
    }
}

void imprime_fila(int cont_filas) {
    int n = cont_filas;
    int cont_n = 0;
    while (cont_n < n) {
        int p = n - 1;
        int r = cont_n;
        printf("%d ", combinatorio(p, r));
        cont_n++;
    }
}

int combinatorio(int p, int r) {
    int fact_p = 1, fact_r = 1, fact_p_menos_r = 1;
    int cont_fact = 1;
    while (cont_fact <= p) {
        fact_p *= cont_fact;
        if (cont_fact <= r)
            fact_r *= cont_fact;
        if (cont_fact <= p - r)
            fact_p_menos_r *= cont_fact;
        cont_fact++;
    }
    return fact_p / (fact_r * fact_p_menos_r);
}
```

FUNDAMENTOS DE PROGRAMACIÓN
ESTUDIOS GENERALES CIENCIAS
PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

1.5. La instrucción for

El ciclo iterativo con entrada controlada se puede implementar en lenguaje C mediante la instrucción `while`. Tal como se puede apreciar en los ejemplos realizados, siempre que se realiza un ciclo iterativo con entrada controlada debe de i) inicializarse la(s) variable(s) de control de flujo, luego ii) elaborar una condición para el control del flujo y finalmente iii) actualizar la(s) variable(s) de control para asegurarse que en algún momento la condición de control de flujo falle y se termine el *bucle*. La inicialización de la(s) variable(s) de control de flujo se suele realizar antes de iniciar el ciclo y la actualización se suele realizar en la última instrucción del ciclo, tal como se puede apreciar en el programa 1.4.

Programa 1.4: Lenguaje C: instrucción while

```
inicialización de variable(s) de control;
while (condición de control de flujo){
        conjunto de instrucciones;
        actualización de variable(s) de control
}
```

Debido a que estas 3 situaciones (inicialización, condición, actualización) están siempre presentes en un ciclo iterativo con entrada controlada, el lenguaje C permite que se pueda escribir, de forma agrupada en una sola instrucción. Esta instrucción es la instrucción `for`. En el programa 1.5 se puede apreciar la sintaxis de la instrucción `for`. En la práctica funciona de la misma manera que la instrucción `while`.

Programa 1.5: Lenguaje C: instrucción for

```
for(inicialización de variable(s) de control; condición de control de flujo; actualización de variable(s) de control)
        conjunto de instrucciones;
```

1.5.1. Implementación del triángulo de Pascal usando for

En el programa 1.6 se presenta la implementación del algoritmo trabajado previamente para el problema del triángulo de Pascal usando lenguaje C pero implementando los ciclos iterativos usando la instrucción `for`. Si usted ha realizado la implementación de este algoritmo usando la instrucción `while` podrá apreciar que al usar la instrucción `for` la cantidad de instrucciones se reducen, esto ocurre pues la inicialización y la actualización de la variable de control ocurren en la misma línea. Además, para algunos codificadores, se aprecia mejor el control de flujo del ciclo iterativo.

Programa 1.6: Triángulo de Pascal implementado en lenguaje C usando for

```
#include <stdio.h>

int main() {
    int total_filas, cont_filas;
    printf("Ingrese cantidad de filas: ");
    scanf("%d", &total_filas);
    for (cont_filas = 1; cont_filas <= total_filas; cont_filas++) {
        int n, cont_n;
        printf("Fila %d\n", cont_filas);
        for (n = cont_filas, cont_n = 0; cont_n < n; cont_n++) {
            int p, r, cont_fact, fact_p, fact_r, fact_p_menos_r, comb;
            p = n - 1;
            r = cont_n;
            fact_p = fact_r = fact_p_menos_r = 1;
            for (cont_fact = 1; cont_fact <= p; cont_fact++) {
                fact_p *= cont_fact;
                if (cont_fact <= r)
                    fact_r *= cont_fact;
                if (cont_fact <= (p - r))
                    fact_p_menos_r *= cont_fact;
            }
            comb = fact_p / (fact_r * fact_p_menos_r);
            printf("%d ", comb);
        }
        printf("\n");
    }
    return 0;
}
```

Hay algunos aspectos de la implementación del programa 1.6 que vale la pena comentar. En primer lugar en la línea 7 se puede observar la siguiente instrucción `for`.

```
for (cont_filas = 1; cont_filas <= total_filas; cont_filas++){
...
}
```

FUNDAMENTOS DE PROGRAMACIÓN
ESTUDIOS GENERALES CIENCIAS
PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Usando la instrucción `for` fácilmente se puede apreciar cómo se realiza el control del ciclo. El ciclo se controla con la variable $cont\_filas$ la cual se inicializa en 1 y se incremente en 1 al final de cada ciclo. Antes de ejecutar el *bucle*, se evalua la condición $cont\_fila \leq total\_filas$. El ciclo solo se ejecutará si esta condición se cumple, es decir si el valor de la expresión es 1 (se hace *verdadera*).

Esta instrucción `for` equivale a la siguiente instrucción `while`. La principal desventaja del uso de la instrucción `while` es que la inicialización y la actualización de la variable de control se encuentran en diferentes líneas, haciendo la comprensión del flujo una tarea un poco más compleja que cuando se utiliza la instrucción `for`. Esto se torna más evidente cuando se utilizan ciclos iterativos anidados.

```
cont_filas = 1;
while (cont_filas <= total_filas){
...
cont_filas++;
}
```

En la línea 10 se puede apreciar el siguiente ciclo `for`. Observe la expresión de inicialización de variables de control. ¿qué de diferente tiene con las demás instrucciones `for`?

```
...
for (n = cont_filas, cont_n = 0; cont_n < n; cont_n++) {
...
}
```

Como podrá observar, la diferencia radica que en este ciclo `for` se está realizando la inicialización de 2 variables de control (n y $cont\_n$). Esto es posible realizar tanto en la inicialización como en la actualización, basta separar las expresiones por una coma (`,`). Ambas inicializaciones se ejecutan en la misma instrucción.

Otro aspecto interesante para comentar en este programa aparece en la línea 14. Lo que sucede en esta expresión es lo que se conoce como asignación múltiple. El lenguaje C permite que se pueda asignar un valor a varias variables en una única instrucción. En la asignación múltiple, el operando que está más a la derecha puede ser un valor constante (como 1), una variable (como x) o una expresión (como $x + 5$), los demás operandos necesariamente deben ser variables pues se les asignará un valor.

```
fact_p = fact_r = fact_p_menos_r = 1;
```

En la línea 16 se tiene la siguiente instrucción. ¿Qué hace esta instrucción?

```
fact_p *= cont_fact;
```

Estamos frente a lo que en C se conoce como operadores de asignación. El operador `*=` realiza una multiplicación y asignación al mismo tiempo, de esta manera `fact_p *= cont_fact;` es equivalente a `fact_p = fact_p*cont_fact;`. Existen varios operadores de asignación que soporte el lenguaje C, entre ellos: `+=`, `-=`, `*=`, `/=` y `%=`.

En el programa 1.7 se puede apreciar la versión de la implementación del triángulo de Pascal usando la instrucción `for` y funciones definidas por el usuario. ¿De todas las versiones que se han presentado para este problema en lenguaje C, cuál le parece que es la que se entiende mejor?

Programa 1.7: Triángulo de Pascal implementado en lenguaje C usando for y funciones definidas por el usuario

```
1  #include <stdio.h>
2
3  void imprime_triangulo_de_pascal(int total_filas);
4  void imprime_fila(int cont_filas);
5  int combinatorio(int p, int r);
6
7  int main() {
8      int total_filas;
9      printf("Ingrese cantidad de filas: ");
10     scanf("%d", &total_filas);
11     imprime_triangulo_de_pascal(total_filas);
12     return 0;
13 }
14
15 void imprime_triangulo_de_pascal(int total_filas) {
```

FUNDAMENTOS DE PROGRAMACIÓN
ESTUDIOS GENERALES CIENCIAS
PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

```
16      int cont_filas;
17      for (cont_filas = 1; cont_filas <= total_filas; cont_filas++) {
18          imprime_fila(cont_filas);
19          printf("\n");
20      }
21  }
22
23  void imprime_fila(int cont_filas) {
24      int n, cont_n;
25      printf("Fila %d\n", cont_filas);
26      for (n = cont_filas, cont_n = 0; cont_n < n; cont_n++) {
27          int p = n - 1;
28          int r = cont_n;
29          printf(" %d ", combinatorio(p, r));
30      }
31  }
32
33  int combinatorio(int p, int r) {
34      int cont_fact, fact_p, fact_r, fact_p_menos_r;
35      fact_p = fact_r = fact_p_menos_r = 1;
36      for (cont_fact = 1; cont_fact <= p; cont_fact++) {
37          fact_p *= cont_fact;
38          if (cont_fact <= r)
39              fact_r *= cont_fact;
40          if (cont_fact <= (p - r))
41              fact_p_menos_r *= cont_fact;
42      }
43      return fact_p / (fact_r * fact_p_menos_r);
44  }
```

1.6. El cuadrado mágico

Un cuadrado mágico es una estructura de números organizados en filas y columnas de igual tamaño de tal forma que todas las filas, todas las columnas y las diagonales sumen el mismo número. Este número se denomina la constante mágica.

Dado que un cuadrado mágico tiene el mismo número n de filas y de columnas, se suelen colocar números entre 1 y n^2 en el cuadrado. De esta forma la constante mágica se calcula como $C_n = \frac{n(n^2+1)}{2}$.

Tabla 1.1: Cuadrado mágico

a	b	c
d	e	f
g	h	i

Si el cuadrado que se encuentra en la tabla 1.1 fuese mágico, se deberían cumplir ciertas condiciones:

- Todas las filas suman lo mismo y la suma es igual a la constante mágica. Es decir $a+b+c = d+e+f = g+h+i = C_n$.
- Todas las columnas suman lo mismo y la suma es igual a la constante mágica. Es decir $a+d+g = b+e+h = c+f+i = C_n$.
- Las diagonales suman lo mismo y la suma es igual a la constante mágica. Es decir $a+e+i = c+e+g = C_n$.

Si se tuviera un cuadrado mágico de orden 3, entonces la constante mágica $C_3 = \frac{3(3^2+1)}{2} = \frac{3(9+1)}{2} = 15$. Una posible solución a un cuadrado mágico se puede apreciar en la tabla 1.2.

¿De qué manera se pueden generar cuadrados mágicos de orden 3 usando el lenguaje C? Se pueden generar de diversas maneras, en esta guía nos basaremos en la técnica de la fuerza bruta. La técnica de la fuerza bruta

FUNDAMENTOS DE PROGRAMACIÓN
ESTUDIOS GENERALES CIENCIAS
PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Tabla 1.2: Cuadrado mágico de orden 3

8	1	6
3	5	7
4	9	2

consiste en un método directo para resolver un problema, por lo general basada directamente en el enunciado del problema y en las definiciones de los conceptos involucrados [1]. Fuerza bruta implica la capacidad de un computador para realizar cálculos, no suelen ser algoritmos que utilicen inteligencia en su diseño.

Usando la técnica de la fuerza bruta lo que se realizará es la generación de todas las posibles combinaciones de cuadrados de orden 3 y para cada uno verificar las condiciones de problema. Se imprimirán los cuadrados que cumplan la condición.

Pero, ¿cómo se generan todas las posibles combinaciones? Para esto usaremos el ciclo iterativo y la anidación de estas iteraciones tantas veces como sea necesario. Procederemos a analizar el problema e ir proponiendo alternativas de solución. Iniciaremos generando filas, luego filas con números diferentes, posteriormente generaremos cuadrados y aplicaremos a estos la condición del cuadrado mágico.

1.6.1. Imprimiendo filas

Analicemos la tabla 1.1. La primera fila está formada por los números a, b y c. Se sabe que tanto a, b como c varía en el rango [1..9]. Entonces lo que se requiere hacer es iterar para cada número del 1 al 9. Pero para generar todas las posibles combinaciones, estas iteraciones deben estar anidadas. En el programa 1.8 se puede apreciar una propuesta para generar todas las posibles filas.

Programa 1.8: Imprimiendo filas

```c
#include <stdio.h>

int main() {
    int a,b,c;

    for(a=1;a<=9;a++)
        for(b=1;b<=9;b++)
            for(c=1;c<=9;c++)
                printf("%d %d %d\n", a,b,c);
    return 0;
}
```

La salida de este programa generaría lo siguiente:

```
1 1 1
1 1 2
1 1 3
1 1 4
.....
.....
.....
9 9 7
9 9 8
9 9 9
```

FUNDAMENTOS DE PROGRAMACIÓN
ESTUDIOS GENERALES CIENCIAS
PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Para poner en práctica

- Implemente el algoritmo usando la instrucción `while`.
- Como podrá observar la impresión inicia con la fila 1, 1, 1 y finaliza con la fila 9, 9, 9. ¿Qué cambios deberá realizar en el programa para que la impresión inicie con la fila 9, 9, 9 y finalice con la fila 1, 1, 1?

1.6.2. Imprimiendo filas con números diferentes

Si observa detalladamente la salida del programa, verá que existen filas para las cuales los números que las componen se repiten. Por ejemplo la segunda fila es 1, 1, 2, se tienen dos veces 1. Para el caso del cuadrado mágico, estas filas no formarían parte de la solución. El programa debe de filtrarlas, es decir no las debe imprimir.

¿Cómo se puede realizar el filtro? Para esto se utilizará una instrucción selectiva. Solo se imprimirá si se cumple la condición de que todas los números sean diferentes.

¿Cómo establecemos que todos los números sean diferentes? En el programa la variable a representa al número a, la variable b al número b y la variable c al número c de la fila. Debe cumplirse que $a \neq b$, $b \neq c$ y $a \neq c$. Esta condición la podemos expresar en C de la siguiente manera $a! = b$ && $b! = c$ && $a! = c$. Recuerde que en C `!=` es el operador de comparación y retorna 1 si los operandos son diferentes. `&&` es el operador de conjunción y retorna 1 si los operandos son diferentes de 0.

En el programa 1.9 se puede apreciar un programa que imprime las filas que cumplen la condición previamente establecida.

Programa 1.9: Imprimiendo filas con números diferentes

```
1  #include <stdio.h>
2
3  int main() {
4      int a,b,c;
5
6      for(a=1;a<=9;a++)
7          for(b=1;b<=9;b++)
8              for(c=1;c<=9;c++)
9                  if (a!=b && b!=c && a!=c)
10                     printf("%d %d %d\n", a,b,c);
11     return 0;
12 }
```

La salida de este programa generaría lo siguiente:

```
1 2 3
1 2 4
1 2 5
1 2 6
1 2 7
1 2 8
1 2 9
1 3 2
.....
.....
.....
```

1.6.3. Imprimiendo "cuadrados"

Ya se ha conseguido imprimir una fila de números, ¿cómo se podrían generar los números para el cuadrado completo? Hasta el momento se han utilizado 3 ciclos iterativos para generar una fila, si se desean generar

FUNDAMENTOS DE PROGRAMACIÓN
ESTUDIOS GENERALES CIENCIAS
PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

3 filas se deberá utilizar 9 ciclos iterativos anidados. En el programa 1.10 se puede apreciar un programa que imprime las 3 filas de un posible cuadrado mágico. Las variables a, b, c representan a la primera fila, las variables d, e, f representan a la segunda fila y las variables g, h, i a la tercera fila. Se ha impreso unos asteriscos antes de cada cuadrado para poder diferenciarlos en la salida.

Programa 1.10: Imprimiendo "cuadrados"

```
1  #include <stdio.h>
2
3  int main() {
4      int a, b, c, d, e, f, g, h, i;
5
6      for (a = 1; a <= 9; a++)
7          for (b = 1; b <= 9; b++)
8              for (c = 1; c <= 9; c++)
9                  for (d = 1; d <= 9; d++)
10                     for (e = 1; e <= 9; e++)
11                         for (f = 1; f <= 9; f++)
12                             for (g = 1; g <= 9; g++)
13                                 for (h = 1; h <= 9; h++)
14                                     for (i = 1; i <= 9; i++) {
15                                         printf("*****\n");
16                                         printf(" %d  %d  %d\n", a, b, c);
17                                         printf(" %d  %d  %d\n", d, e, f);
18                                         printf(" %d  %d  %d\n", g, h, i);
19                                     }
20     return 0;
21 }
```

La salida de este programa generaría lo siguiente:

```
.....
.....
*****
1  1  1
1  1  4
7  9  2
*****
1  1  1
1  1  4
7  9  3
.....
.....
```

Para poner en práctica

- Implemente el algoritmo usando la instrucción `while`. Para este problema en particular, ¿conviene utilizar la instrucción `while` o `for`?

1.6.4. Imprimiendo "cuadrados" con números diferentes

Si observa la salida del programa anterior, verá que dentro de un mismo cuadrado, existen números que se repiten. Se debe aplicar la misma lógica que se aplicó cuando se filtraron las filas que tenían números repetidos. La principal diferencia ahora es que no se tienen 3 variables sino 9 y se debe garantizar que todos sean diferentes entre todos. En el programa 1.11 se apreciar entre la línea 15 y la 24 la condición para filtrar los cuadrados con números diferentes. Lo extensa de la condición se debe a que se deben colocar la expresión lógica que garantice que todos las variables sean diferentes entre sí.

Programa 1.11: Imprimiendo "cuadrados" con números diferentes

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

FUNDAMENTOS DE PROGRAMACIÓN
ESTUDIOS GENERALES CIENCIAS
PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

```
2
3  int main() {
4      int a, b, c, d, e, f, g, h, i;
5
6      for (a = 1; a <= 9; a++)
7        for (b = 1; b <= 9; b++)
8          for (c = 1; c <= 9; c++)
9            for (d = 1; d <= 9; d++)
10             for (e = 1; e <= 9; e++)
11               for (f = 1; f <= 9; f++)
12                 for (g = 1; g <= 9; g++)
13                   for (h = 1; h <= 9; h++)
14                     for (i = 1; i <= 9; i++)
15                       if (a != b && a != c && a != d && a != e && a != f && a != g &&
16                           a != h && a != i &&
17                           b != c && b != d && b != e && b != f && b != g && b != h &&
18                           b != i &&
19                           c != d && c != e && c != f && c != g && c != h && c != i &&
20                           d != e && d != f && d != g && d != h && d != i &&
21                           e != f && e != g && e != h && e != i &&
22                           f != g && f != h && f != i &&
23                           g != h && g != i &&
24                           h != i) {
25                         printf("*****\n");
26                         printf("%d %d %d\n", a, b, c);
27                         printf("%d %d %d\n", d, e, f);
28                         printf("%d %d %d\n", g, h, i);
29                       }
30     return 0;
31 }
```

La salida de este programa generaría lo siguiente:

```
.....
.....
*****
1 3 7
8 9 6
5 4 2
*****
1 3 7
9 2 4
5 6 8
.....
.....
```

1.6.5. Imprimiendo “cuadrados” mágicos

Ahora que se ha generada todas las posibles combinaciones de cuadrados que contienen números diferentes hay que agregarle la condición del problema. Un cuadrado mágico es aquel en el que la suma de las todas las filas, columnas y diagonales es igual a la constante mágica. En este caso 15. En el programa 1.12 se aprecia la condición del cuadrado mágico entre las líneas 25 y 32.

Programa 1.12: Imprimiendo “cuadrados” mágicos

```
1  #include <stdio.h>
2
3  int main() {
4      int a, b, c, d, e, f, g, h, i;
5
6      for (a = 1; a <= 9; a++)
7        for (b = 1; b <= 9; b++)
8          for (c = 1; c <= 9; c++)
9            for (d = 1; d <= 9; d++)
```

FUNDAMENTOS DE PROGRAMACIÓN
ESTUDIOS GENERALES CIENCIAS
PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

```
10              for (e = 1; e <= 9; e++)
11                for (f = 1; f <= 9; f++)
12                  for (g = 1; g <= 9; g++)
13                    for (h = 1; h <= 9; h++)
14                      for (i = 1; i <= 9; i++) {
15                        if (a != b && a != c && a != d && a != e && a != f && a != g &&
16                            a != h && a != i &&
17                            b != c && b != d && b != e && b != f && b != g && b != h &&
18                            b != i &&
19                            c != d && c != e && c != f && c != g && c != h && c != i &&
20                            d != e && d != f && d != g && d != h && d != i &&
21                            e != f && e != g && e != h && e != i &&
22                            f != g && f != h && f != i &&
23                            g != h && g != i &&
24                            h != i) {
25                            if ((a + b + c) == 15 &&
26                                  (d + e + f) == 15 &&
27                                  (g + h + i) == 15 &&
28                                  (a + d + g) == 15 &&
29                                  (b + e + h) == 15 &&
30                                  (c + f + i) == 15 &&
31                                  (a + e + i) == 15 &&
32                                  (g + e + c) == 15) {
33                                printf("*****\n");
34                                printf("%d %d %d\n", a, b, c);
35                                printf("%d %d %d\n", d, e, f);
36                                printf("%d %d %d\n", g, h, i);
37                              }
38                            }
39                        }
40      return 0;
41  }
```

La salida de este programa generaría lo siguiente:

```
*****
2 7 6
9 5 1
4 3 8
*****
2 9 4
7 5 3
6 1 8
.....
.....
```

Para poner en práctica

- ¿Qué cambios debería realizar en el programa anterior si en lugar de imprimir los cuadrados mágicos le solicitan que cuente los cuadrados mágicos de orden 3?
- Implemente un programa que permita imprimir y contar los cuadrados mágicos de orden 4.

Para pensar

- En el programa anterior existe una estructura selectiva anidada (vea la línea 15 y la línea 25), ¿es necesario que esta estructura se encuentre anidada?, ¿se podría juntar las dos condiciones en una sola expresión?, ¿es esto conveniente?
- Para la implementación de la generación de cuadrados mágicos de orden 4, conviene utilizar la instrucción `while` o la instrucción `for`.

FUNDAMENTOS DE PROGRAMACIÓN
ESTUDIOS GENERALES CIENCIAS
UNIVERSIDAD CATÓLICA DEL PERÚ

1.7. Los números Armstrong

Un número Armstrong, también llamado número narcisista, es todo aquel número que es igual a la suma de cada uno de sus dígitos elevado al número total de dígitos.

A continuación siguen algunos ejemplos de números Armstrong.

- $371 = 3^3 + 7^3 + 1^3$. Total de dígitos 3.
- $8208 = 8^4 + 2^4 + 0^4 + 8^4$. Total de dígitos 4.
- $4210818 = 4^7 + 2^7 + 1^7 + 0^7 + 8^7 + 1^7 + 8^7$. Total de dígitos 7.

Se desea elaborar un programa en lenguaje C que permita simular un juego con el usuario para adivinar números Armstrong. El programa solicitará un número entero al usuario. Si el número ingresado por el usuario es un número Armstrong, el programa termina, caso contrario seguirá solicitando un número al usuario.

Para resolver este problema se debe primero contar los dígitos de un número, luego sumar los dígitos del número elevando cada dígito al total de dígitos para finalmente hacer el control de flujo. Se introducirá además dos nuevas instrucciones que permiten interrumpir el control de flujo.

1.7.1. Contando los dígitos de un número

Lo primero que se debe hacer es realizar el conteo de dígitos del número ingresado por el usuario. En el programa 1.13 se solicita el ingreso de un número y se almacena en la variable *numero*. El control del flujo se realiza con la instrucción `while` usando la variable *encontrado* para el control. El ciclo iterativo se ejecutará mientras *encontrado* sea diferente de 0. Por este motivo la variable *encontrado* se inicializa con 0 para que la primera vez se ejecute. La idea es que cuando el usuario ingrese un número que sea Armstrong, la variable *encontrado* se actualice con el valor de 1 y en ese momento la condición del `while` falla. Si *encontrado* tiene el valor de 1, !*encontrado* tendría el valor de 0 haciendo que la condición del `while` sea *falsa*.

Para encontrar la cantidad de dígitos de *numero* lo que se realiza son divisiones sucesivas entre 10 hasta que se llegue a 0. La cantidad de dígitos se almacena en la variable *total_digitos*. Se utiliza una copia de *numero* para hacer las divisiones sucesivas. En este programa dicha variable se denomina *numero_original*. Se usa esta variable pues como se realizarán divisiones sucesivas, el número original se perdería y no se podría hacer la comparación con la suma de dígitos. Recuerde que `numero_original /= 10` equivale a `numero_original = numero_original / 10`.

Programa 1.13: Contando los dígitos de un número

```
1  #include <stdio.h>
2
3  int main() {
4      int encontrado = 0, numero;
5      while (!encontrado){
6          int numero_original, total_digitos;
7          printf("Ingrese un número Armstrong: ");
8          scanf(" %d", &numero);
9
10         numero_original = numero;
11         total_digitos = 0;
12         while (numero_original != 0){
13           ++total_digitos;
14           numero_original /= 10;
15         }
16     }
17     return 0;
18 }
```

Si el usuario ingresa los números 1, 12, 123, 1234 y 12345, la salida de este programa generaría lo siguiente:

```
Ingrese un número Armstrong: 1
Total de dígitos = 1
Ingrese un número Armstrong: 12
Total de dígitos = 2
Ingrese un número Armstrong: 123
Total de dígitos = 3
Ingrese un número Armstrong: 1234
Total de dígitos = 4
Ingrese un número Armstrong: 12345
Total de dígitos = 5
```

Para poner en práctica

- Implemente el programa utilizando la instrucción `for` en lugar del `while`.
- Existe otra manera de obtener la cantidad de dígitos de un número usando el logaritmo en base 10 de un número. Recuerde que: $\log(10) = 1$, $\log(100) = 2$, $\log(1000) = 3$. La cantidad de cifras de un número cualquiera se obtiene sumando 1 a la parte entera de su logaritmo decimal. En lenguaje C puede obtener el logaritmo en base 10 de un número usando la función `log10` cuyo prototipo se encuentra en el archivo de cabecera `math.h`. Cambie el programa para usar el logaritmo en el cálculo de los dígitos de un número.

1.7.2. Sumando los dígitos del número

Una vez hallada la cantidad de dígitos del número, se procede a calcular la sumatoria de los dígitos del número elevando cada uno de estos dígitos a la cantidad de dígitos calculado anteriormente. Para calcular los dígitos se utiliza la misma técnica de divisiones sucesivas entre 10. El dígito se obtiene en cada iteración usando el módulo del número dividido entre 10. Al final de la iteración se realiza la actualización de la variable de control de flujo, si la *suma* == *numero* entonces a la variable *encontrado* se le asigna el valor de 1. Haciendo que el ciclo finalice. Este tipo de control se conoce como ciclo iterativos controlados por centinela. Se utilizan cuando no se sabe cuántas veces se realizará el ciclo. En el programa 1.14 se puede apreciar el centinela *encontrado*.

Programa 1.14: Sumando los dígitos del número

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

int main() {
    int encontrado = 0, numero;
    while (!encontrado){
        int numero_original, total_digitos, suma, digito, i;
        printf("Ingrese un número Armstrong: ");
        scanf("%d", &numero);

        numero_original = numero;
        total_digitos = 0;
        while (numero_original != 0){
          ++total_digitos;
          numero_original /= 10;
        }

        i=1;
        suma = 0;
        numero_original = numero;
        while (numero_original != 0){
            digito = numero_original % 10;
            suma += (int)pow(digito, total_digitos);
            numero_original /= 10;
        }
        if (suma==numero){
            encontrado=1;
            printf("El número %d es Armstrong\n", numero);
```

FUNDAMENTOS DE PROGRAMACIÓN
ESTUDIOS GENERALES CIENCIAS
PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

```
        }
    }
    return 0;
}
```

1.7.3. Filtrando los números negativos

El programa anterior no realiza ningún control en caso el número fuese negativo o 0. Si deseamos filtrar las situaciones en donde no se ingresa un número que cumple la condición deseada, se puede utilizar una instrucción selectiva. En el programa 1.15 se puede apreciar el programa con el filtro mencionado en la línea 11.

Programa 1.15: Filtrando los números negativos

```
1  #include <stdio.h>
2  #include <math.h>
3
4  int main() {
5      int encontrado = 0, numero;
6      while (!encontrado) {
7          int numero_original, total_digitos, suma, digito, i;
8          printf("Ingrese un número Armstrong: ");
9          scanf("%d", &numero);
10
11         if (numero > 0) {
12             numero_original = numero;
13             total_digitos = 0;
14             while (numero_original != 0) {
15                 ++total_digitos;
16                 numero_original /= 10;
17             }
18
19             i = 1;
20             suma = 0;
21             numero_original = numero;
22             while (numero_original != 0) {
23                 digito = numero_original % 10;
24                 suma += (int) pow(digito, total_digitos);
25                 numero_original /= 10;
26             }
27             if (suma == numero) {
28                 encontrado = 1;
29                 printf("El número %d es Armstrong\n", numero);
30             }
31         }
32     }
33     return 0;
34 }
```

Si el usuario ingresa los números −4, −6, −7, 0, 0 y 153, la salida de este programa generaría lo siguiente:

```
Ingrese un número Armstrong: -4
Ingrese un número Armstrong: -6
Ingrese un número Armstrong: -7
Ingrese un número Armstrong: 0
Ingrese un número Armstrong: 0
Ingrese un número Armstrong: 153
El número 153 es Armstrong
```

1.7.4. Uso de la instrucción continue

El usar una instrucción selectiva efectivamente hace que se consiga el objetivo de filtrado de números pero incremente la complejidad del programa. Si deseamos que cuando el usuario ingreso un número que sea menor o igual a 0 no se haga nada y se vuelva a solicitar otro número se puede utilizar la instrucción `continue`.

FUNDAMENTOS DE PROGRAMACIÓN
ESTUDIOS GENERALES CIENCIAS
PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

La instrucción `continue` es una de las instrucciones de ruptura de flujo. Hace que el control de flujo se dirija hacia la siguiente iteración. En términos prácticos, si se ejecuta un `continue`, el control del programa se dirige hacia el inicio del ciclo iterativo. Lo que resta del *loop* no se ejecuta.

En el programa 1.16 puede verse una instrucción `continue` en la línea 12. Esto significa que cuando un número es menor o igual a 0 esta instrucción se ejecuta. Esto hace que las líneas 14 hasta la 32 no se ejecuten enviando el control hacia la línea 6 iniciando nuevamente una iteración.

Programa 1.16: Uso de la instrucción continue

```
1  #include <stdio.h>
2  #include <math.h>
3
4  int main() {
5      int encontrado = 0, numero;
6      while (!encontrado) {
7          int numero_original, total_digitos, suma, digito, i;
8          printf("Ingrese un número Armstrong: ");
9          scanf("%d", &numero);
10
11         if (numero <= 0)
12             continue;
13
14         numero_original = numero;
15         total_digitos = 0;
16         while (numero_original != 0) {
17             ++total_digitos;
18             numero_original /= 10;
19         }
20
21         i = 1;
22         suma = 0;
23         numero_original = numero;
24         while (numero_original != 0) {
25             digito = numero_original % 10;
26             suma += (int) pow(digito, total_digitos);
27             numero_original /= 10;
28         }
29         if (suma == numero) {
30             encontrado = 1;
31             printf("El número %d es Armstrong\n", numero);
32         }
33     }
34     return 0;
35 }
```

FUNDAMENTOS DE PROGRAMACIÓN
ESTUDIOS GENERALES CIENCIAS
PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

1.7.5. Uso de la instrucción break

La otra instrucción de ruptura de flujo es la intrucción `break`. La instrucción `break` hace que un ciclo iterativo termine su ejecución completa. Luego de un `break` no se ejecutará ningún *bucle* independientemente si la condición de control de flujo se cumple o no. En el programa 1.17 se puede apreciar el programa anterior modificado para que el control de salida de la iteración se realice mediante un `break`. Ya no se controla el ciclo por el centinela *encontrado*, en lugar de eso la condición del `while` es 1, eso significa un ciclo que nunca termina. Pero en la línea 31 se puede observar un uso del `break` que hará que ciclo termine cuando se encuentre un número Armstrong.

Programa 1.17: Uso de la instrucción break

```
1  #include <stdio.h>
2  #include <math.h>
3
4  int main() {
5      int numero;
6      while (1) {
7          int numero_original, total_digitos, suma, digito, i;
8          printf("Ingrese un número Armstrong: ");
```

```
        scanf(" %d", &numero);

        if (numero <= 0)
            continue;

        numero_original = numero;
        total_digitos = 0;
        while (numero_original != 0) {
            ++total_digitos;
            numero_original /= 10;
        }

        i = 1;
        suma = 0;
        numero_original = numero;
        while (numero_original != 0) {
            digito = numero_original % 10;
            suma += (int) pow(digito, total_digitos);
            numero_original /= 10;
        }
        if (suma == numero) {
            printf("El número %d es Armstrong\n", numero);
            break;
        }
    }
    return 0;
}
```

En el programa 1.18 se puede apreciar la versión implementada con funciones definidas por el usuario. Como se puede apreciar al usar funciones, la complejidad del problema se reduce.

Programa 1.18: Implementación de los números Armstrong usando funciones definidas por el usuario

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

int es_Armstrong(int numero);
int suma_digitos(int numero);
int calcular_total_digitos(int numero);

int main() {
    int numero;
    while (1) {
        printf("Ingrese un número Armstrong: ");
        scanf(" %d", &numero);

        if (numero <= 0)
            continue;

        if (es_Armstrong(numero)) {
            printf("El número %d es Armstrong\n", numero);
            break;
        }
    }
    return 0;
}

int es_Armstrong(int numero) {
    int suma = suma_digitos(numero);
    return suma == numero;
}

int suma_digitos(int numero) {
    int suma = 0;
    int total_digitos = calcular_total_digitos(numero);
    while (numero != 0) {
        int digito = numero % 10;
        suma += (int) pow(digito, total_digitos);
        numero /= 10;
    }
```

FUNDAMENTOS DE PROGRAMACIÓN
ESTUDIOS GENERALES CIENCIAS
PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

```
38      return suma;
39  }
40
41  int calcular_total_digitos(int numero) {
42      int total_digitos = 0;
43      while (numero != 0) {
44          ++total_digitos;
45          numero /= 10;
46      }
47      return total_digitos;
48  }
```

FUNDAMENTOS DE PROGRAMACIÓN
ESTUDIOS GENERALES CIENCIAS
PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Capítulo 2

Ejercicios Propuestos

Para cada uno de los ejercicios propuestos se solicita que elabore el correspondiente algoritmo representado tanto en diagrama de flujo como en pseudocódigo así como la implementación de un programa en lenguaje C conforme a los temas revisados en las guías #1, #2, #3, #4, #5 y #6 del curso de Fundamentos de Programación. Se recomienda que las soluciones en lenguaje C incluyan funciones definidas por el usuario.

2.1. Nivel Básico

2.1.1. Tabla de multiplicar

Se le pide que genere las tablas de multiplicar del 1 al 9. Para cada tabla deberá presentar el producto desde el número 1 hasta el número 12. A continuación se presentan un ejemplo de ejecución:

```
TABLA 01 TABLA 02 TABLA 03 TABLA 04 TABLA 05 TABLA 06 TABLA 07 TABLA 08 TABLA 09
01*01=01 01*02=02 01*03=03 01*04=04 01*05=05 01*06=06 01*07=07 01*08=08 01*09=09
02*01=02 02*02=04 02*03=06 02*04=08 02*05=10 02*06=12 02*07=14 02*08=16 02*09=18
03*01=03 03*02=06 03*03=09 03*04=12 03*05=15 03*06=18 03*07=21 03*08=24 03*09=27
04*01=04 04*02=08 04*03=12 04*04=16 04*05=20 04*06=24 04*07=28 04*08=32 04*09=36
05*01=05 05*02=10 05*03=15 05*04=20 05*05=25 05*06=30 05*07=35 05*08=40 05*09=45
06*01=06 06*02=12 06*03=18 06*04=24 06*05=30 06*06=36 06*07=42 06*08=48 06*09=54
07*01=07 07*02=14 07*03=21 07*04=28 07*05=35 07*06=42 07*07=49 07*08=56 07*09=63
08*01=08 08*02=16 08*03=24 08*04=32 08*05=40 08*06=48 08*07=56 08*08=64 08*09=72
09*01=09 09*02=18 09*03=27 09*04=36 09*05=45 09*06=54 09*07=63 09*08=72 09*09=81
10*01=10 10*02=20 10*03=30 10*04=40 10*05=50 10*06=60 10*07=70 10*08=80 10*09=90
11*01=11 11*02=22 11*03=33 11*04=44 11*05=55 11*06=66 11*07=77 11*08=88 11*09=99
12*01=12 12*02=24 12*03=36 12*04=48 12*05=60 12*06=72 12*07=84 12*08=96 12*09=108
```

FUNDAMENTOS DE PROGRAMACIÓN
FACULTAD DE CIENCIAS E INGENIERÍA
PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Variación al problema #1

Dado un número entero n positivo y mayor que cero, se le pide que genere las tablas de multiplicar para cada dígito contenido en el número n. Las tablas deberán aparecer en orden ascendente. A continuación se presenta un ejemplo de ejecución:

```
Ingrese n: 54221

TABLA 01 TABLA 02 TABLA 04 TABLA 05
01*01=01 01*02=02 01*04=04 01*05=05
02*01=02 02*02=04 02*04=08 02*05=10
03*01=03 03*02=06 03*04=12 03*05=15
04*01=04 04*02=08 04*04=16 04*05=20
05*01=05 05*02=10 05*04=20 05*05=25
06*01=06 06*02=12 06*04=24 06*05=30
07*01=07 07*02=14 07*04=28 07*05=35
08*01=08 08*02=16 08*04=32 08*05=40
09*01=09 09*02=18 09*04=36 09*05=45
10*01=10 10*02=20 10*04=40 10*05=50
11*01=11 11*02=22 11*04=44 11*05=55
12*01=12 12*02=24 12*04=48 12*05=60
```

Variación al problema #2

Dado un número entero n positivo y mayor que cero, se le pide que genere las tablas de multiplicar para cada dígito contenido en el número n. Las tablas deberán aparecer en el orden en que aparecen en el número n. A continuación se presenta un ejemplo de ejecución:

```
Ingrese n: 52421

TABLA 05 TABLA 02 TABLA 04 TABLA 01
01*05=05 01*02=02 01*04=04 01*01=01
02*05=10 02*02=04 02*04=08 02*01=02
03*05=15 03*02=06 03*04=12 03*01=03
04*05=20 04*02=08 04*04=16 04*01=04
05*05=25 05*02=10 05*04=20 05*01=05
06*05=30 06*02=12 06*04=24 06*01=06
07*05=35 07*02=14 07*04=28 07*01=07
08*05=40 08*02=16 08*04=32 08*01=08
09*05=45 09*02=18 09*04=36 09*01=09
10*05=50 10*02=20 10*04=40 10*01=10
11*05=55 11*02=22 11*04=44 11*01=11
12*05=60 12*02=24 12*04=48 12*01=12
```

Sugerencia para lenguaje C

La función `printf` permite formatear la salida de datos. En particular si se desea rellenar con ceros al inicio de un número entero, basta que se modifique la cadena de formato en el argumento de la función `printf`. Como ya se sabe, para imprimir un número entero `int` se utiliza la cadena de formato "`%d`", si se quiere rellenar con ceros, basta con anteponer el símbolo `0` seguido de la cantidad de ceros a rellenar antes de la letra `d`. Por ejemplo, si se quisiera rellenar con ceros números de hasta 2 dígitos se puede colocar como formato "`%02d`", de esta manera, el número `3` se imprimirá como `03` y el número `11` seguirá imprimiéndose como `11`.

2.1.2. Tabla Pitagórica

Las tablas de multiplicar clásicas requieren de mucho espacio para su impresión, además a simple vista no permiten apreciar ciertas propiedades como por ejemplo la propiedad conmutativa de la multiplicación (*El orden de los factores no altera el producto*). La tabla pitagórica es otra propuesta de representación de tablas de multiplicar que se motiva en las limitaciones de las tablas de multiplicar clásicas. En esta tabla, la primera fila y la primera columna contienen los números que se van a multiplicar mientras que en cada celda de la tabla se encuentra el producto de la fila primera fila y la primera columna que le corresponden.

Asumiendo que la primera columna es la que está más a la izquierda y que la primera fila es la que está más abajo, se le pide que lea un número entero $n \in [1..10]$ e imprima la tabla pitagórica para los n números ingresados. A continuación se presenta un ejemplo de ejecución:

```
Ingrese n: 10

10 | 00 10 20 30 40 50 60 70 80 90 100
09 | 00 09 18 27 36 45 54 63 72 81 90
08 | 00 08 16 24 32 40 48 56 64 72 80
07 | 00 07 14 21 28 35 42 49 56 63 70
06 | 00 06 12 18 24 30 36 42 48 54 60
05 | 00 05 10 15 20 25 30 35 40 45 50
04 | 00 04 08 12 16 20 24 28 32 36 40
03 | 00 03 06 09 12 15 18 21 24 27 30
02 | 00 02 04 06 08 10 12 14 16 18 20
01 | 00 01 02 03 04 05 06 07 08 09 10
00 | 00 00 00 00 00 00 00 00 00 00 00
    ---------------------------------
 X   00 01 02 03 04 05 06 07 08 09 10
```

Variación al problema #1

Realice las adecuaciones necesarias para que se imprima la tabla pitagórica como sigue:

```
Ingrese n: 5

00 | 00 00 00 00 00 00
01 | 00 01 02 03 04 05
02 | 00 02 04 06 08 10
03 | 00 03 06 09 12 15
04 | 00 04 08 12 16 20
05 | 00 05 10 15 20 25
     ------------------
 X   00 01 02 03 04 05
```

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ FACULTAD DE CIENCIAS E INGENIERÍA CURSOS DE PROGRAMACIÓN

Variación al problema #2

Realice las adecuaciones necesarias para que se imprima la tabla pitagórica como sigue:

```
Ingrese n: 5

00 | 00 00 00 00 00 00
01 | 05 04 03 02 01 00
02 | 10 08 06 04 02 00
03 | 15 12 09 06 03 00
04 | 20 16 12 08 04 00
05 | 25 20 15 10 05 00
     ------------------
 X   05 04 03 02 01 00
```

2.1.3. SEND+MORE=MONEY

El problema $SEND + MORE = MONEY$ consisten en encontrar valores distintos para los dígitos D, E, M, N, O, R, S, Y de forma tal que que S y M sean diferentes de 0 y se satisfaga la ecuación $SEND + MORE = MONEY$. Elabore un programa en lenguaje C que permite encontrar los valores para los dígitos del problema $SEND + MORE = MONEY$.

A continuación se presenta un ejemplo de ejecución:

```
 9567 +
 1085
=====
10652
```

Sugerencia

- Utilice la técnica de la fuerza bruta para generar todas las posibles combinaciones para las letras D, E, M, N, O, R, S, Y.
- Forme el número $SEND$, el número $MORE$ y el número $MONEY$. Si D, E, M, N, O, R, S, Y son variables que contienen dígitos, el número $SEND$ se puede obtener de la siguiente manera: $SEND = S * 1000 + E * 100 + N * 10 + D$.
- Utilice una estructura selectiva para verificar si es que se satisface la ecuación.

2.1.4. Calendario

Dado un número de mes y un año ambos válidos, se le pide que imprima el calendario de dicho mes. Para esto deberá:

- Calcular el día de la semana con el que inicia el mes (e.g., sábado, domingo, lunes, martes, miércoles, jueves, viernes). Para este fin deberá utilizar la Congruencia de Zeller.
- Calcular la cantidad de días que posee el mes.
- Imprimir el calendario iniciando en el día lunes.

FUNDAMENTOS DE PROGRAMACIÓN
PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA

Congruencia de Zeller

Julius Christian Johannes Zeller es un matemático alemán que dentro de sus trabajos publicados destaca un algoritmo para determinar el día de la semana de cualquier fecha del calendario gregoriano o del calendario juliano. Este algoritmo se conoce como la congruencia de Zeller. A continuación se presenta la fórmula para calcular, en un computador, el día de la semana de una determinada fecha expresada como dd/mm/aaaa en el calendario gregoriano.

$$dds = \left(dd + \left\lfloor \frac{13(mm+1)}{5} \right\rfloor + aaaa + \left\lfloor \frac{aaaa}{4} \right\rfloor - \left\lfloor \frac{aaaa}{100} \right\rfloor + \left\lfloor \frac{aaaa}{400} \right\rfloor \right) mod\ 7$$

Donde:

- $\lfloor x \rfloor$ corresponde con la función piso.
- *dds* corresponde con el día de la semana. El día de la semana estará codificado como sigue: 0=sábado, 1=domingo, 2=lunes, 3=martes, 4=miércoles, 5=jueves y 6=viernes.
- *dd* corresponde con el día del mes para el cual se quiere hallar el día de la semana.
- *mm* corresponde con el mes para el cual se quiere hallar el día de la semana. El mes se codifica como sigue: 3=marzo, 4=abril, 5=mayo, 6=junio, 7=julio, 8=agosto, 9=septiembre, 10=octubre, 11=noviembre, 12=diciembre, 13=enero, 14=febrero.
- *aaaa* corresponde con el año para el cual se quiere hallar el día de la semana. Cuando el mes sea enero o febrero, se deberá decrementar en una unidad el valor de *aaaa*.

Por ejemplo si la fecha para la cual se desea determinar el día de la semana fuera $dd = 1$, $mes = 5$, $año = 2019$, entonces:

$$dds = \left(1 + \left\lfloor \frac{13(5+1)}{5} \right\rfloor + 2019 + \left\lfloor \frac{2019}{4} \right\rfloor - \left\lfloor \frac{2019}{100} \right\rfloor + \left\lfloor \frac{2019}{400} \right\rfloor \right) mod\ 7$$

$$dds = \left(1 + \left\lfloor 15.6 \right\rfloor + 2019 + \left\lfloor 504.75 \right\rfloor - \left\lfloor 20.19 \right\rfloor + \left\lfloor 5.0475 \right\rfloor \right) mod\ 7$$

$$dds = (1 + 15 + 2019 + 504 - 20 + 5)\ mod\ 7 = (2524)\ mod\ 7 = 4$$

Como $dds = 4$, entonces la fecha $dd = 1$, $mes = 5$, $año = 2019$ cae el día miércoles.

A continuación se presentan un ejemplo de ejecución:

```
Ingrese mes: 5
Ingrese año: 2019

  L  M  X  J  V  S  D
        1  2  3  4  5
  6  7  8  9 10 11 12
 13 14 15 16 16 18 19
 20 21 22 23 24 25 26
 27 28 29 30 31
```

Variación al problema #1

Se le pide que hagas las alteraciones correspondientes para que el calendario se imprima a iniciando el día de la semana con el sábado.

FUNDAMENTOS DE PROGRAMACIÓN
ESTUDIOS GENERALES CIENCIAS
PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Variación al problema #2

Se le pide que hagas las alteraciones correspondientes para que el calendario se imprima a iniciando el día de la semana con el domingo.

Recordar que:

Los meses poseen la siguientes cantidades de días.

- Enero, Marzo, Mayo, Julio, Agosto, Octubre, Diciembre poseen 31 días.
- Abril, Junio, Septiembre, Noviembre poseen 30 días.
- Febrero posee 28 días. Salvo los años bisiestos en donde posee 29 días.

Recordar que:

Un año es bisiesto si el número que lo representa es divisible entre 4, salvo que sea año secular –último de cada siglo, terminado en 00–, en cuyo caso también ha de ser divisible entre 400.

Dadas las siguientes preposiciones:

- p: El número que representa al año es divisible entre 4.
- q: El número que representa al año es divisible entre 100.
- r: El número que representa al año es divisible entre 400.

La expresión lógica que permite determinar si un año es es $p \wedge (\neg q \vee r)$.

2.1.5. Listado de números en un determinado rango

Se le pide que lea dos números enteros a y b que representan los valores mínimo y máximo del rango $[a..b]$. Luego verifique que $a \leq b$ y si se cumple esta condición, imprima todos los números primos existentes en dicho rango. Si no existiera ningún número primo en ese rango, deberá imprimirse el mensaje `No existen números primos en el rango indicado`.

A continuación se presentan unos ejemplos de ejecución:

```
Ingrese rango [a, b]: 2004 2010

No existen números primos en el rango indicado
```

```
Ingrese rango [a, b]: 4440 4530

En el rango [4440, 4530] se tienen los siguientes números primos:
4441, 4447, 4451, 4457, 4463, 4481, 4483, 4493, 4507, 4513, 4517, 4519, 4523
```

```
Ingrese rango [a, b]: 9750 9780

En el rango [9750..9780] se tienen los siguientes números primos:
9767, 9769
```

Recordar que:

Un número primo es un número natural mayor que 1 que posee únicamente 2 divisores: la unidad y el mismo número. Los primeros números primos son 2, 3, 5, 7, 11, 19 y 23

Variación al problema #1

Realice las adecuaciones necesarias para que en lugar de imprimirse los números primos, se impriman los números Armstrong.

Recordar que:

Un número Armstrong, también llamado número narcisista, es todo aquel número que es igual a la suma de cada uno de sus dígitos elevado al número total de dígitos.
A continuación siguen algunos ejemplos de números Armstrong:

- $371 = 3^3 + 7^3 + 1^3$. Total de dígitos 3.
- $8208 = 8^4 + 2^4 + 0^4 + 8^4$. Total de dígitos 4.
- $4210818 = 4^7 + 2^7 + 1^7 + 0^7 + 8^7 + 1^7 + 8^7$. Total de dígitos 7.

A continuación se presenta uno ejemplo de ejecución:

```
Ingrese rango [a, b]: 100 100000

En el rango [100..100000] se tienen los siguientes números Armstrong:
153, 370, 371, 407, 1634, 8208, 9474, 54748, 92727, 93084
```

Variación al problema #2

Realice las adecuaciones necesarias para que en lugar de imprimirse los números primos, se impriman los números perfectos.

Recordar que:

Un número perfecto es un número natural que es igual a la suma de sus divisores propios positivos. Un divisor propio positivo de un número n es un número entero que divide a n pero diferente de n. De esta forma por ejemplo el número 6 es un número perfecto dado que es igual a la suma de $1 + 2 + 3$.

A continuación se presenta uno ejemplo de ejecución:

```
Ingrese rango [a, b]: 1 10000

En el rango [1..10000] se tienen los siguientes números perfectos:
6, 28, 496, 8128
```

2.1.6. Números primos gemelos

Los primos gemelos son pares de números primos que se encuentran separados únicamente por un número par. De esta manera por ejemplo, los números 17 y 19 son números primos gemelos dado que entre ellos solamente

se encuentra el número 18 el cual es un número par. Se le pida que lea el valor de dos números enteros a y b que representan los valores mínimo y máximo del rango $[a..b]$ y luego de hacer todas las verificaciones necesarias, presenta los pares de números primos gemelos que se encuentran en dicho rango.

```
Ingrese rango [a, b]: 1 100

En el rango [1..100] se tienen los siguientes números primos gemelos:
3, 5
5, 7
11, 13
17, 19
29, 31
41, 43
59, 61
71, 73
```

Sugerencia

Guarde el último número primo generado para que pueda ser comparado en la siguiente iteración.

Variación al problema

Realice las adecuaciones necesarias para que en lugar de solicitarse el rango, se solicite la cantidad de pares de números primos gemelos a imprimir.

2.1.7. Conteo de frecuencias por rangos

Se le pide que lea dos números enteros a y b que representan los valores mínimo y máximo del rango $[a..b]$, un valor s que representa al tamaño del sub-rango. Luego de verificar que $a \leq b$ y que $(b-a+1) >= s$, se le pida que calcule y presente la cantidad de números primos que existen en cada sub-rango.

A continuación se presenta uno ejemplo de ejecución:

```
Ingrese rango [a, b]: 1 1000
Ingrese tamaño de sub-rango: 100

Subrangos #Primos
[001-100] 25
[101-200] 21
[201-300] 16
[301-400] 16
[401-500] 17
[501-600] 14
[601-700] 16
[701-800] 14
[801-900] 15
[901-1000] 14
```

Variación al problema

Realice las adecuaciones necesarias para que en lugar de realizarse el conteo de los números primos, se haga el conteo de los números Armstrong.

A continuación se presenta uno ejemplo de ejecución:

```
Ingrese rango [a, b]: 1 100000
Ingrese tamaño de sub-rango: 50000

Subrangos #Armstrongs
[00001-50000] 16
[50001-100000] 3
```

2.1.8. Los Números de Mersenne

Marin Mersenne fue un matemático y filósofo francés que hoy en día es recordado por los números que llevan su nombre, los números de Mersenne. Los números de Mersenne son números que en magnitud son una unidad menos que las potencias de 2 y los números de Mersenne que son primos, guardan una estrecha relación con los números perfectos (es posible generar números perfectos a través de números de Mersenne primos).

Los números de Mersenne poseen la siguiente fórmula de generación $M_n = 2^n - 1$ por lo que generarlos es bastante simple. Se le pide que construya una tabla con los n primeros números de Mersenne. En dicha tabla debe incluir las siguientes columnas:

- Generador. Es el número que genera el número de Mersenne. Este número debe iniciar en 2 e incrementarse en 1 en cada fila.
- #Mersenne. Es el número de Mersenne que ha sido generado.
- ¿Número primo?. En esta columna escribirá `Es primo` en caso el número Merssene generado sea primo. En caso contrario se escribirá `No es primo`.

A continuación se presenta uno ejemplo de ejecución:

```
Ingrese número de filas: 10

Generador #Mersenne ¿Número primo?
2         3         Es primo
3         7         Es primo
4         15        No es primo
5         31        Es primo
6         63        No es primo
7         127       Es primo
8         255       No es primo
9         511       No es primo
10        1023      No es primo
11        2047      No es primo
```

FUNDAMENTOS DE PROGRAMACIÓN
ESTUDIOS GENERALES CIENCIAS
PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

2.2. Nivel Intermedio

2.2.1. Calculadora trigonométrica

Se pide que elabore una calculadora que permita computar el valor de las funciones trigonométricas seno, coseno y tangente a través de un menú de opciones. El menú de opciones deberá de aparecer repetidas veces hasta que el usuario seleccione la opción de `saliR`. A continuación se presenta la descripción de cada una de las opciones que debe presentar el menú.

`térmiNos` Esta opción le permitirá al usuario ingresar la cantidad de términos que deberá usar para calcular el valor de la función trigonométrica. Deberá de verificar que este valor sea $\geq$ que 1 y $\leq$ que 11. En caso no se cumpla esta condición, deberá de asignársele el valor de 10 por defecto. Esta opción se deberá ejecutar si es que el usuario ingresa la letra `N` o `n`.

`Error` Esta opción le permitirá al usuario ingresar el valor del error que deberá usar para calcular el valor de la función trigonométrica. Deberá de verificar que este valor sea $\geq$ que 0 y $\leq$ que 1. En caso no se cumpla esta condición, deberá de asignársele el valor de 0.0001 por defecto. Esta opción se deberá ejecutar si es que el usuario ingresa la letra `E` o `e`.

`Ángulo` Esta opción le permitirá al usuario ingresar el valor del ángulo en grados sexagesimales. Esta opción se deberá ejecutar si es que el usuario ingresa la letra `A` o `a`.

`Seno` Esta opción le permitirá al usuario calcular el valor del seno del ángulo ingresado por el usuario. Para este cálculo deberá usar la serie de Taylor. Si el usuario ha seleccionado la opción `térmiNos` deberá usar en la serie la cantidad de términos indicados por el usuario. Si el usuario ha seleccionado la opción `Error` deberá incluir en la serie los términos hasta encontrar uno que sea, en valor absoluto, menor que el error indicado por el usuario. Si el usuario ha seleccionado la opción `térmiNos` y la opción `Error`, deberá usar para el cálculo la última selección. Si el usuario no ha seleccionado el valor del ángulo, deberá usar el valor de 0 como valor por defecto. Esta opción se deberá ejecutar si es que el usuario ingresa la letra `S` o `s`.

`Coseno` Esta opción le permitirá al usuario calcular el valor del coseno del ángulo ingresado por el usuario. De forma análoga para la opción `Seno` deberá usar la serie de Taylor. Los controles a usar para el cálculo de la serie de Taylor así como el ángulo serán los mismos que en el caso del seno. Esta opción se deberá ejecutar si es que el usuario ingresa la letra `C` o `c`.

`Tangente` Esta opción le permitirá al usuario calcular el valor de la tangente del ángulo ingresado por el usuario. Este valor se obtendrá de dividir el valor del seno y del coseno. Los controles a usar para el cálculo de la serie de Taylor así como el ángulo serán los mismos que en el caso del seno y coseno. Esta opción se deberá ejecutar si es que el usuario ingresa la letra `T` o `t`.

`saliR` Esta opción le permitirá al usuario finalizar el menú de opciones. Esta opción se deberá ejecutar si es que el usuario ingresa la letra `R` o `r`.

A continuación se presenta un ejemplo de ejecución de esta calculadora:

```
Menú de opciones:
        Térmi[n]os
        [E]rror
        [A]ngulo
        [S]eno
        [C]oseno
        [T]angente
        Sali[r]
Ingrese una opción: n

Ingrese número de término: 5
```

LOS DE PROGR... GENERALES CIE... VERSIDAD...

```
Menú de opciones:
        Térmi[n]os
        [E]rror
        [A]ngulo
        [S]eno
        [C]oseno
        [T]angente
        Sali[r]
Ingrese una opción: A

Ingrese ángulo: -30

Menú de opciones:
        Térmi[n]os
        [E]rror
        [A]ngulo
        [S]eno
        [C]oseno
        [T]angente
        Sali[r]
Ingrese una opción: S

El valor del seno de -30 usando 5 términos es -0.5000000000

Menú de opciones:
        Térmi[n]os
        [E]rror
        [A]ngulo
        [S]eno
        [C]oseno
        [T]angente
        Sali[r]
Ingrese una opción: R

La calculadora de funciones trigonométricas ha finalizado.
```

Restricciones

- En su solución solamente podrá usar dos estructuras algoritmos iterativas. Una para controlar la ejecución del menú y otra para el cálculo de las funciones trigonométricas.

FUNDAMENTOS DE PROGRAMACIÓN
ESTUDIOS GENERALES CIENCIAS
PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Casos de prueba para verificación de solución

Use los siguiente casos para verificar si su solución está correcta.

x	# **términos**	$sen(x)$	$cos(x)$	$tan(x)$
-30	5	≈ -0.5000000000	≈ 0.8660254042	≈ -0.5773502689
45	7	≈ 0.7071067812	≈ 0.7071067812	≈ 1.0000000000
60	10	≈ 0.8660254038	≈ 0.5000000000	≈ 1.7320508076

x	**error**	$sen(x)$	$cos(x)$	$tan(x)$
-53	0.1	≈ -0.7987491393	≈ 0.6026719888	≈ -0.7987491393
90	0.05	≈ 0.9998431014	≈ -0.0008945230	≈ -1117.7388430550
105	0.008	≈ 0.9659450252	≈ -0.2587042516	≈ -3.7337810234

Recordar que:

Si x representa al valor de un ángulo en radianes, entonces:

$$sen(x) = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!}$$

$$cos(x) = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n}}{(2n)!}$$

$$tan(x) = \frac{sen(x)}{cos(x)}$$

$$360° = 2\pi$$

2.2.2. El último teorema de Fermat

El último teorema de Fermat, conjeturado por Pierre de Fermat en 1637, establece que: si n es un número entero mayor que 2, entonces no existen los números enteros positivos x, y y z, tales que se cumpla la igualdad: $x^n + y^n = z^n$.

Se pide que verifique si el último teorema de Fermat se cumple para el conjunto de valores x, y y z tales que $x < 100$, $y < 100$, $z < 100$, $n \in [3..5]$

2.2.3. Conjetura débil de Goldbach

Harald Andrés Helfgott Seier es un matemático peruano que en el año 2015 encontró la solución a un problema que permanecía irresuelto desde el año 1742[1], se trata de la conjetura débil de Goldbach. La conjetura débil de Goldbach indica que *"todo número impar mayor que 5 puede expresarse como suma de tres números primos"*. Se le pide dado un número natural impar n mayor que 5, verifique si la conjetura débil de Goldbach se cumple. Deberá presentar la justificación de la verificación. En caso existan varias justificaciones para el número, deberá presentar solamente una.

A continuación se presenta un ejemplo de ejecución:

```
Ingrese número: 25
```

[1] http://archivo.elcomercio.pe/tecnologia/actualidad/peruano-demostro-solucion-problema-matematico-hace-tres-siglos-noticia-1579725

```
Se cumple la conjetura débil de Goldbach pues:
        25 = 3 + 3 + 19 y
        3 es un número primo
        3 es un número primo
        19 es un número primo
```

Sugerencia

Para verificar si se cumple la conjetura deberá busca 3 números dentro de todas las posibles opciones. Para esto se le recomienda que utilice iterativas anidadas. Pero, ¿cómo hacer para que no se impriman varias justificaciones? Para que solamente se imprima la primera justificación use una bandera para controlar la impresión. "Levante la bandera" la primera vez que imprima la justificación, cuando la bandera ya está levantada, deje de imprimir las demás justificaciones.

Sugerencia para lenguaje C

Si ya se encontró la primera justificación no tiene mucho sentido continuar buscando otras pues solo nos piden una. ¿Cómo parar la iteraciones anidadas entonces? Para esto puede usar la instrucción `break` que permite terminar una iteración. La instrucción `break` solo termina la iteración del ciclo en donde se encuentra, es decir si existe otra iteración anidada en un nivel superior, esta no terminará.

2.2.4. Suma de fracciones

Se pide que dada una lista de fracciones, calcule e imprima la suma de ellas. El flujo de lectura de datos finalizará cuando se ingrese una fracción que tenga como numerador 0 o un número negativo.

Ejemplo de ejecución del programa:

```
Ingrese fracción (numerador denominador): 1 2
Ingrese fracción (numerador denominador): 3 4
Ingrese fracción (numerador denominador): 1 5
Ingrese fracción (numerador denominador): 0 0

La suma es (numerador denominador): 29 20
```

Sugerencia

- Para hallar la suma de fracciones se sugiere que sume las dos primeras fracciones, el resultado de esto lo sume con la tercera fracción, el resultado de esta con la cuarta fracción y así sucesivamente.
- Recuerde que para sumar dos fracciones $\frac{a}{b}+\frac{b}{c}=\frac{(m.c.m.(b,c)/b)\times a+(m.c.m.(b,c)/c)\times b}{m.c.m.(b,c)}$
- Recuerde que $m.c.m.(a,b)=\frac{a\times b}{m.c.d.(a,b)}$.
- Utilice un ciclo iterativo controlado por un centinela. El centinela podría ser una variable como *encontrado* que cuando se ingresa un número 0 se le asigna el valor de 1. Esta variable la podría inicializar en 0.

2.2.5. Cambio de base

Se le pide que elabore un algoritmo y luego un programa en lenguaje C que permita realizar el cambio de base de una lista de números ingresados por el usuario. La base destino será ingresada al inicio del programa

y deberá ser un número en el rango [2..9]. El flujo de lectura terminará cuando se ingrese el número 0 o un número negativo. Se asumirá que los números ingresados se encuentran en base 10.

Ejemplo de ejecución del programa:

```
Ingrese base: 3
Ingrese número: 15
El número 15 en base 3 es = 120

Ingrese número: 26
El número 27 en base 3 es = 222

Ingrese número: 0
```

Sugerencia

- Para realizar el cambio de base, divida el número por la base de forma sucesiva hasta llegar a 0. En el resto de la división encontrará el dígito. Con el dígito encontrado puede ir armando el número en la otra base.
- Utilice un ciclo iterativo controlado por un centinela. El centinela podría ser una variable como *encontrado* que cuando se ingresa un número 0 se le asigna el valor de 1. Esta variable la podría inicializar en 0.

2.2.6. El algoritmo de Euclides

El algoritmo de Euclides es un método que permite calcular el máximo común divisor de dos números. Dados dos enteros a y b cuyo máximo común divisor se desea hallar, y asumiendo que $a > 0$, $b > 0$ y $a \geq b$, se realiza lo siguiente:

- Se calcula r como el resto de la división entre a y b.
- Si r es igual a 0 el algoritmo termina y se dice que b es el máximo común divisor de a y b.
- Caso contrario máximo común divisor de a y b será igual al máximo común divisor de b y r, por lo que se repite el procedimiento nuevamente ahora con b y r.

Se pide que lea una lista de n números todos ellos mayores que 0 e imprima el máximo común divisor. El programa terminará cuando se ingrese el número 0 o un número negativo. Este último número, no se deberá considerar en el cálculo del máximo común divisor.

Ejemplo de ejecución del programa:

```
Ingrese número: 224
Ingrese número: 693
Ingrese número: 504
Ingrese número: 0

El máximo común divisor es: 7
```

FUNDAMENTOS DE PROGRAMACIÓN
ESTUDIOS GENERALES CIENCIAS
PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

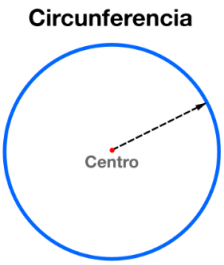
Sugerencia

- Para hallar el máximo común divisor de varios números puede aplicar el algoritmo de Euclides con los dos primeros números. Luego use este número (máximo común divisor) para calcular el máximo común divisor con el tercer número, el resultado lo puede usar para calcular el máximo común divisor con el cuarto número y así sucesivamente.
- Tenga presente que para que el algoritmo funcione, a debe ser mayor o igual que b. Los números ingresados por el usuario no necesariamente siguen un orden establecido por lo que será necesario calcular el mayor de 2 números.
- Al igual que en el caso anterior, utilice un ciclo iterativo controlado por un centinela. El centinela podría ser una variable como *encontrado* que cuando se ingresa un número 0 se le asigna el valor de 1. Esta variable la podría inicializar en 0.

2.2.7. La Circunferencia

En geometría plana, una circunferencia es el lugar geométrico de todos los puntos que cumplen con la propiedad de equidistar de otro punto fijo denominado centro.

Figura 2.1: Gráfica de una circunferencia



Al punto centro se le denomina como centro de la circunferencia y la distancia que existe desde el centro hacia cualquier punto de la circunferencia se le llama radio y siempre es constante.

Se le pide que, elabore un programa en lenguaje C que permita, evaluar un número determinado de circunferencias. Para la evaluación, por cada circunferencia debe solicitar que ingrese el punto x e y del centro de la circunferencia y el radio de la misma. Además, debe solicitar una lista de puntos (x,y) sobre los cuales verficará si estos puntos pertenecen a la circunferencia en evaluación. Para ello, deberá ingresar las coordenadas x e y de cada punto que desea verificar si pertenece a la circunferencia ingresada y debe calcular la distancia que existe desde el punto ingresado hacia el centro de la circunferencia. Si la distancia calculada es igual al radio entonces el punto pertenece a la circunferencia y se continua con la evaluación del siguiente punto.

Si durante la evaluación de un punto se verifica que este no pertenece a la circunferencia, se detiene la verificación de la misma y se determina que la circunferencia es inválida colocando como área el valor de 0. En caso todos los puntos ingresados pertenezcan a la circunferencia, la misma se considera como válida y se calcula el área. Para terminar el ingreso de puntos a evaluar se debe colocar los valores de 0 para las coordenadas x e y.

El programa debe realizar esta evaluación por cada circunferencia ingresada, además debe indicar cuál fue el mayor área calculada entre todas las circunferencias que fueron válidas.

Para la solución debe implementar y utilizar solo los siguientes módulos:

- Un módulo llamado **validarPuntosycalcularArea** que permita realizar la evaluación de todos los puntos ingresados para una circunferencia y el cálculo del área de la misma. Para ello debe recibir como parámetros el punto x e y del centro de la circunferencia, el radio y debe devolver si la circunferencia es válida y el área calculada.

FUNDAMENTOS DE PROGRAMACIÓN
INSTITUTO DE CIENCIAS
PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

- Un módulo llamado **calcularDistancia** que permita calcular la distancia entre dos puntos. Para ello debe recibir como parámetros las coordenadas x e y de los dos puntos y debe devolver la distancia entre ellos.

Para el cálculo de la distancia de dos puntos puede utilizar la siguiente fórmula:

$$distancia = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Nota.- Para la comparación de la distancia calculada con el radio, al ser datos reales, debe calcular la diferencia entre ellos y si esta diferencia es menor a 0.0001 debe considerar que son iguales. Debe utilizar el valor de 3.1415 para PI e imprimr el valor del área con 2 decimales.

Casos de prueba para verificación de solución

Use los siguiente casos para verificar si su solución está correcta.

Caso 1
Ingrese la cantidad de circunferencias a evaluar: 2
Ingrese el punto x e y del punto central de la circunferencia a evaluar: 3 3
Ingrese el radio de la circunferencia: 2
Análisis de puntos
Ingrese las coordenadas x e y del punto a evaluar si pertenece a la circunferencia: 5 3
Ingrese las coordenadas x e y del punto a evaluar si pertenece a la circunferencia: 3 5
Ingrese las coordenadas x e y del punto a evaluar si pertenece a la circunferencia: 0 0
Todos los puntos pertenecen a la circunferencia y tiene un área de 12.57
Ingrese el punto x e y del punto central de la circunferencia a evaluar: 4 6
Ingrese el radio de la circunferencia: 1.5
Análisis de puntos
Ingrese las coordenadas x e y del punto a evaluar si pertenece a la circunferencia: 5.5 6
Ingrese las coordenadas x e y del punto a evaluar si pertenece a la circunferencia: 12.5 7.6
Se encontró un punto que no pertenece a la circunferencia.
La circunferencia válida con mayor área es: 12.57

Caso 2
Ingrese la cantidad de circunferencias a evaluar: 1
Ingrese el punto x e y del punto central de la circunferencia a evaluar: 6 3
Ingrese el radio de la circunferencia: 10
Análisis de puntos
Ingrese las coordenadas x e y del punto a evaluar si pertenece a la circunferencia: 1.5 3.82
Se encontró un punto que no pertenece a la circunferencia.
Ninguna de las circunferencias pasó la evaluación.

2.2.8. El juego de los números capicúas

El juego de los números capicúas consiste en leer un número, invertirlo, luego sumar este número invertido al número original, luego otra vez lo invierte y lo suma. Este proceso se repite hasta que la suma sea capicúa o se ejecuten 10 sumas.

Por ejemplo, leemos el número 14, lo invertimos y obtenemos 41, luego sumamos este número invertido al número original obteniendo 55 que es capicúa por lo que el proceso termina e imprimimos 55. En caso haya repetido el proceso 10 veces, es decir 10 sumas como indica el enunciado, y no ha logrado tener como respuesta un numero capicúa, entonces deberá parar el proceso e imprimir “Es imposible lograr un número capicúa”.

Se le pide que, elabore un programa en lenguaje C que lea la cantidad de números a evaluar y luego ingresar número a número. Por cada número debe evaluar si consigue o no ser un número capicúa siguiendo la lógica indicada. Al final debe mostrar el mayor número capicúa encontrado.

Para la solución debe implementar y utilizar solo los siguientes módulos:

- Un módulo llamado **evaluarCapicua** que permita si se consigue un número capicua siguiendo la lógica descrita para un número ingresado. En caso se consiga, se debe devolver un valor indicando que se consiguió, así como el número capicúa encontrado. En caso no se consiga, debe devolver un valor indicando que no se consiguió y como número capicúa el valor de 0. Para ello debe recibir como parámetros el número y devolver si se encontró un número capicúa o no y el número capicúa encontrado, siguiendo exactamente lo descrito.

- Un módulo llamado **invertirNumero** que permita invertir un número. Para ello debe recibir como parámetros el número a invertir y devolver el número invertido.

Nota.- Los módulos indicados son adicionales al main.

Casos de prueba para verificación de solución

Use los siguiente casos para verificar si su solución está correcta.

Caso 1
Ingrese la cantidad de números a evaluar: 3
Ingrese el número: 14
El número 14 logró un número capicúa que es 55
Ingrese el número: 23
El número 23 logró un número capicúa que es 55
Ingrese el número: 58
El número 58 logró un número capicúa que es 484
El mayor número capicúa encontrado fue 484

Caso 2
Ingrese la cantidad de números a evaluar: 4
Ingrese el número: 105
El número 105 logró un número capicúa que es 606
Ingrese el número: 856
El número 856 logró un número capicúa que es 5665
Ingrese el número: 87
El número 87 logró un número capicúa que es 4884
Ingrese el número: 1254
El número 1254 logró un número capicúa que es 5775
El mayor número capicúa encontrado fue 5775

2.2.9. Números sociables

Una secuencia de números sociables esta formada por una cantidad finita de números de forma que la suma de divisores (sin contar el propio número) del primer número es igual al segundo número, la suma de divisores (sin contar el propio número) del segundo número es igual al tercero, y así hasta el último número, cuya suma de divisores (sin contar el propio número) es igual al primer número de la secuencia ingresada. Este concepto de secuencia de números sociables se obtuvo en 1918 gracias al matemático belga Paul Paulet.

Por ejemplo, la siguiente secuencia de números son números sociables:

12496
14288
15472
14536
14264

Lo anterior se cumple debido a que la suma de divisores (sin contar el propio número) de 12496 es igual al segundo número ingresado en la secuencia que es 14288, además la suma de divisores (sin contar el propio número) de 14288 es igual al tercer número ingresado en la secuencia que es 15472, así sucesivamente hasta

que la suma de divisores (sin contar el propio número) del último número ingresado 14264 es igual 12496 que es el primer número de la secuencia ingresada.

Se le pide que, elabore un programa en lenguaje C que permita evaluar una secuencia de números y determinar si los números de dicha secuencia son sociables. Debe tener en cuenta las siguientes consideraciones:

- Para finalizar la secuencia de números ingresará el número 0.
- Los números a ingresar deben ser mayores que 0, en caso no se cumpla ello, debe mostrar un mensaje de error y debe volver ingresar el número de la secuencia. - En caso encuentre que un número de la secuencia no es sociable, debe detener el ingreso de los números y mostrar el mensaje correspondiente.

Para la solución debe implementar y utilizar solo los siguientes módulos:

* Un módulo llamado **calcularSumaDivisores** que permita realizar el cálculo de la suma de divisores del número, sin contar como divisor al propio número. Para ello debe recibir como parámetro el número y debe devolver la suma de divisores indicada.

* Un módulo llamado **imprimirMensaje** que permita mostrar el mensaje final de la evaluación de la secuencia de números. Para ello debe recibir como parámetro un identificador que le indique cual mensaje mostrar. Considere como mensajes válidos los últimos mensajes de cada caso de prueba.

Nota.- Los módulos indicados son adicionales al main.

Casos de prueba para verificación de solución

Use los siguiente casos para verificar si su solución está correcta.

Caso 1
Ingrese el número 1: 12496
Ingrese el número 2: 14288
Ingrese el número 3: 15472
Ingrese el número 4: 14536
Ingrese el número 5: 14264
Ingrese el número 6: 0
Los números ingresados son sociables.

Caso 2
Ingrese el número 1: 0
No se ingreso ningún número válido en la secuencia.

Caso 3
Ingrese el número 1: 12496
Ingrese el número 2: -54
El número ingresado no es correcto, debe ser mayor que cero.
Ingrese el número 2: 14288
Ingrese el número 3: 15472
Ingrese el número 4: 5472
Los números ingresados no son sociables.

2.2.10. Números poderosos

Un número natural n se llama **poderoso** si cumple que, si un número primo p es un divisor suyo entonces p^2 también lo es. Por ejemplo, el número 36 es un número poderoso ya que los únicos primos que son divisores suyos son 2 y 3 y se cumple que 4 y 9 también son divisores de 36.

Se le pide que, elabore un programa en lenguaje C que permita determinar qué números son poderosos dentro de un rango de números. Para ello debe ingresar primero el rango de números a evaluar, si el número inicial

es mayor al número final, debe mostrar un mensaje de error y terminará el programa. En caso de ser correcto, debe determinar por cada número que se encuentra en dicho rango si es poderoso o no y en caso de serlo, debe mostrar un mensaje indicando que es un número poderoso. Al finalizar la evaluación de todo el rango de números, debe mostrar el mayor número poderoso encontrado.

Para la solución debe implementar y utilizar solo los módulos que se indican:

- Un módulo llamado **evaluarSiEsPoderoso** que permita determinar si un número es poderoso o no. Para ello debe recibir como parámetros el número y debe devolver el indicador de si el número es poderoso o no, siguiendo lo descrito.

- Un módulo llamado **evaluarSiEsPrimo** que permita determinar si un número es primo o no. Para ello debe recibir como parámetros el número y debe devolver el indicador de si el número es primo o no.

Nota.- Los módulos indicados son adicionales al main.

Casos de prueba para verificación de solución

Use los siguiente casos para verificar si su solución está correcta.

Caso 1
Ingrese el rango de números a evaluar: 30 50
El número 32 es poderoso
El número 36 es poderoso
El número 49 es poderoso
El mayor número poderoso encontrado fue 49

Caso 2
Ingrese el rango de números a evaluar: 40 45
Ningún número procesado fue poderoso

Caso 3
Ingrese el rango de números a evaluar: 40 30
El rango de números ingresados es incorrecto

2.2.11. Números intocables

Un número natural n se llama **intocable** si **no** es la suma de los divisores propios de ningún número m (los divisores propios son los números que dividen a m, excepto el mismo m).

Para entender mejor el concepto de número intocable veamos los siguientes ejemplos:

El número 3 no es intocable debido a que es igual a la suma de los divisores propios del número 4, al ser, $1 + 2 = 3$.

El número 5 es intocable, debido a que $5 = 1 + 4$ y es la única manera de escribir 5 como suma de enteros positivos diferentes incluyendo al 1, considerando que el 1 es divisor de todo número, pero dicha combinación de 1 y 4 no corresponde a la suma de los divisores propios de ningún entero

Un ejemplo más:

El número 95 no es intocable debido a que es igual a la suma de los divisores propios del número 445, los cuales son: $1 + 5 + 89 = 95$.

Se le pide que, elabore un programa en lenguaje C que permita determinar qué números son intocables dentro de un rango de números. Para ello debe ingresar primero el rango de números a evaluar, si el número inicial es mayor al número final, debe mostrar un mensaje de error y terminará el programa. En caso de ser correcto, debe determinar por cada número que se encuentra en dicho rango si es intocable o no y en caso de no serlo,

debe indicar cuál es el número cuya suma de divisores propios no lo vuelve intocable. Al finalizar la evaluación de todo el rango de números, debe mostrar el mayor número cuya suma de divisores produjo un número que no es intocable.

Para la solución debe implementar y utilizar solo los siguientes módulos:

- Un módulo llamado **evaluarIntocable** que permita determinar si un número es intocable y en caso de no serlo debe devolver el valor del número cuya suma de divisores propios es igual al número que se esta evaluando, en caso de ser intocable debe devolver el valor de 0 como valor del número que no lo hace intocable. Para ello debe recibir como parámetros el número y debe devolver el indicador de si es un número intocable o no así como el valor del número que hace que no sea intocable, siguiendo lo descrito.

- Un módulo llamado **calcularSumaDivisoresPropios** que permita determinar la suma de divisores propios de un número. Para ello debe recibir como parámetros el número y debe devolver la suma de divisores propios del número.

Nota.- Los módulos indicados son adicionales al main. Además, se le recomienda considerar que el mayor número donde podría encontrar que la suma de divisores propios es igual a el será el cuadrado del mismo. Por ejemplo, si evalúo el número 4, debo verificar si la suma de divisores propios entre el número 3 hasta el número 16 es igual al número 4.

Casos de prueba para verificación de solución

Use los siguiente casos para verificar si su solución está correcta.

Caso 1
Ingrese el rango de números a evaluar: 115 125
El número 115 no es intocable por ser igual a la suma de divisores propios de 545.
El número 116 no es intocable por ser igual a la suma de divisores propios de 226.
El número 117 no es intocable por ser igual a la suma de divisores propios de 100.
El número 118 no es intocable por ser igual a la suma de divisores propios de 148.
El número 119 no es intocable por ser igual a la suma de divisores propios de 565.
El número 120 es intocable.
El número 121 no es intocable por ser igual a la suma de divisores propios de 243.
El número 122 no es intocable por ser igual a la suma de divisores propios de 130.
El número 123 no es intocable por ser igual a la suma de divisores propios de 72.
El número 124 es intocable.
El número 125 no es intocable por ser igual a la suma de divisores propios de 1243.
El número mayor cuya suma de divisores propios produjo un número que no es intocable fue el 1243.

Caso 2
Ingrese el rango de números a evaluar: 125 115
El rango ingresado no es correcto

2.2.12. Número capicúa en otra base

Decimos que un número es capicúa si es el mismo cuando se lee de izquierda a derecha o de derecha a izquierda. Por ejemplo, el número 75457 es capicúa. Por supuesto, la propiedad depende de la base en la que se representa el número.

El número 17 no es capicúa en la base 10, pero su representación en la base 2 (10001) es capicúa.

Se le pide que, elabore un programa en lenguaje C, que permita determinar para un rango de números, si sus representaciones en las bases del 4 al 9 son capicúas. Para ello debe ingresar el número inicial y final del rango, validar que este sea correcto, es decir que el número final sea mayor que el número inicial y que el rango este entre 0 y 500 como máximo. En caso de no serlo, debe mostrar un mensaje de error y debe terminar el

programa. En caso el rango sea correcto, por cada número debe hallar su representación en las bases del 4 al 9. Por cada representación debe evaluar si se trata de un número capicúa, en caso de ser así, debe escribir que el número es capicúa en dicha base.

Para la solución debe implementar y utilizar solo los siguientes módulos:

- Un módulo llamado **calcularCantidadcapicuasxBase** que permita determinar la cantidad de bases en que el número es capicúa, además de escribir el mensaje indicando que el número es capicúa en dicha base, conforme las vaya encontrando, en caso no sea capicúa en ninguna base, debe devolver el valor de 0. Para ello debe recibir como parámetros el número y debe devolver la cantidad de bases en que el número es capicúa. Debe seguir las indicaciones descritas en el enunciado del problema.

- Un módulo llamado **convertirBase** que permita determinar el número en una base determinada. Para ello debe recibir como parámetros el número, la base y debe devolver número en dicha base.

- Un módulo llamado **validarSiEscapicua** que permita determinar si un número es capicúa. Para ello debe recibir como parámetros el número y debe devolver si el número es capicúa o no.

Nota.- Los módulos indicados son adicionales al main.

Casos de prueba para verificación de solución

Use los siguiente casos para verificar si su solución está correcta.

Caso 1
Ingrese el rango: 150 160
El número 150 es capicúa en base 4.
El número 150 es capicúa en base 7.
El número 151 no es capicúa en ninguna base del 4 al 9.
El número 152 no es capicúa en ninguna base del 4 al 9.
El número 153 no es capicúa en ninguna base del 4 al 9.
El número 154 es capicúa en base 6.
El número 154 es capicúa en base 8.
El número 154 es capicúa en base 9.
El número 155 no es capicúa en ninguna base del 4 al 9.
El número 156 es capicúa en base 5.
El número 157 es capicúa en base 7.
El número 158 no es capicúa en ninguna base del 4 al 9.
El número 159 no es capicúa en ninguna base del 4 al 9.
El número 160 es capicúa en base 6.

Caso 2
Ingrese el rango: 160 150
El rango ingresado no es correcto.

2.3. Nivel Avanzado

2.3.1. Representación de Conjuntos

Una manera de representar conjuntos de dígitos es mediante números enteros de 9 dígitos sin incluir el 0. Cada dígito del número entero representa un elemento del conjunto. Por ejemplo si N_1=1246 y N_2=1238, se puede decir que el número N_1 representa al conjunto formado por los dígitos {1, 2, 4, 6} y número N_2 representa al conjunto formado por los dígitos {1, 2, 3, 8}. El número 0 representaría un conjunto vacío.

Dados dos números que representan a dos conjuntos como los descritos anteriormente, se le solicita que determine si el conjunto formado por el número N_1 es subconjunto del conjunto formado por el número N_2. A continuación se presenta uno ejemplo de ejecución:

```
Ingrese conjunto 1: 47
Ingrese conjunto 2: 67541

El conjunto N1={4,7} es subconjunto del N2={6,7,5,4,1}
```

Variación al problema #1

¿Qué cambios habría que realizar en la solución anterior para que ahora determine si el conjunto formado por el número N_2 es subconjunto del conjunto formado por el número N_1?

A continuación se presenta uno ejemplo de ejecución:

```
Ingrese conjunto 1: 47
Ingrese conjunto 2: 67541

El conjunto N2={6,7,5,4,1} no es subconjunto del conjunto N1={4,7}
```

Variación al problema #2

¿Qué cambios habría que realizar en la solución anterior para que ahora calcule e imprima el conjunto resultante de la unión del conjunto formado por el número N_1 con el conjunto formado por el número N_2?

A continuación se presenta uno ejemplo de ejecución:

```
Ingrese conjunto 1: 473
Ingrese conjunto 2: 67541

La unión del conjunto N1={4,7,3} y del conjunto N2={6,7,5,4,1}
        resulta en N3={4,7,3,6,5,1}
```

Variación al problema #3

¿Qué cambios habría que realizar en la solución anterior para que ahora calcule e imprima el conjunto resultante de la intersección del conjunto formado por el número N_1 con el conjunto formado por el número N_2?

A continuación se presenta uno ejemplo de ejecución:

```
Ingrese conjunto 1: 473
Ingrese conjunto 2: 67541

La intersección del conjunto N1={4,7,3} y del conjunto N2={6,7,5,4,1}
        resulta en N3={4,7}
```

Variación al problema #4

¿Qué cambios habría que realizar en la solución anterior para que ahora calcule e imprima el conjunto resultante de la diferencia del conjunto formado por el número N_1 con el conjunto formado por el número N_2?

A continuación se presenta uno ejemplo de ejecución:

```
Ingrese conjunto 1: 473
Ingrese conjunto 2: 67541

La diferencia del conjunto N1={4,7,3} y del conjunto N2={6,7,5,4,1} resulta en N3={3}
```

Variación al problema #5

¿Qué cambios habría que realizar en la solución anterior para que ahora calcule e imprima el complemento del conjunto formado por el número N_1?

A continuación se presenta uno ejemplo de ejecución:

```
Ingrese conjunto 1: 473

El complemento del conjunto N1={4,7,3} resulta en N3={1,2,5,6,8,9}
```

Variación al problema #6

¿Qué cambios habría que realizar en la solución anterior para que ahora calcule e imprima el conjunto resultante de la diferencia simétrica del conjunto formado por el número N_1 con el conjunto formado por el número N_2?

A continuación se presenta uno ejemplo de ejecución:

```
Ingrese conjunto 1: 473
Ingrese conjunto 2: 67541

La diferencia del conjunto N1={4,7,3} y del conjunto N2={6,7,5,4,1}
        resulta en N3={3,6,5,1}
```

Variación al problema #7

¿Qué cambios habría que realizar en la solución anterior para que ahora calcule e imprima el producto cartesiano del conjunto formado por el número N_1 con el conjunto formado por el número N_2?

A continuación se presenta uno ejemplo de ejecución:

```
Ingrese conjunto 1: 473
Ingrese conjunto 2: 213

El producto cartesiano es:
        N1xN2={(4,2), (4,1), (4,3), (7,2), (7,1), (7,3), (3,2), (3,1), (3,3)}
```

2.3.2. Más representaciones de Conjuntos

Dados tres números que representan a tres conjuntos como los descritos en la pregunta anterior, se le solicita que determine si es que se cumplen las leyes asociativas de conjuntos.

Leyes asociativas de conjuntos:

$$A \cup (B \cup C) = (A \cup B) \cup C$$

$$A \cap (B \cap C) = (A \cap B) \cap C$$

A continuación se presenta uno ejemplo de ejecución:

```
Ingrese conjunto 1: 13
Ingrese conjunto 2: 579
Ingrese conjunto 3: 245

Se cumple la Ley Asociativa (caso unión).
Justificación:
Por un lado
        N2 U N3 = {5,7,9,2,4}
        N1 U (N2 U N3) = {1,3,5,7,9,2,4}
De otro lado
        N1 U N2 = {1,3,5,7,9}
        (N1 U N2) U N3 = {1,3,5,7,9, 2,4}
Por lo tanto
        como {1,3,5,7,9,2,4} = {1,3,5,7,9,2,4}
Se concluye que
         N1 U (N2 U N3) = (N1 U N2) U N3
```

Variación al problema #1

De forma análoga verifique si se cumple las leyes distributivas.

Leyes distributivas de conjuntos:

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$$

$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$$

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ FACULTAD DE CIENCIAS E INGENIERÍA DEPARTAMENTO ACADÉMICO DE CIENCIAS

Variación al problema #2

De forma análoga verifique si se cumple las leyes de Morgan.

Leyes de Morgan:

$$\overline{A \cup B} = \overline{A} \cap \overline{B}$$

$$\overline{A \cap B} = \overline{A} \cup \overline{B}$$

2.3.3. Ordenamiento por el método de Selección Lineal

Dado un número entero n positivo mayor que cero, se pide que obtenga el menor número posible que se puede formar usando todos los dígitos de n. Para resolver este problema deberá de ordenar los dígitos del número n de forma ascendente. Existen varios métodos de ordenación, en este problema deberá emplear el método de selección lineal.

Método de Selección Lineal

El método de Selección Lineal es un método clásico de ordenación para conjunto de datos. Se utiliza frecuentemente con arreglos pero puede ser adaptado a cualquier estructura. Este método ejecuta tantas iteraciones como datos existan en el conjunto. En cada iteración i se obtiene el menor elemento del conjunto de datos, se coloca el menor elemento en la i-ésima posición en otro conjunto de datos y se retira el menor elemento del conjunto de datos original. Al final de todas las iteraciones, en el nuevo conjunto de datos se encontrarán los datos ordenados de forma ascendente. A continuación se presenta una adaptación de este método para ordenar los dígitos de un número.

Entrada: $n \in \mathbb{N}, n > 0$
Salida : $n\_ordenado$

```
Leer n;
total_de_dígitos_de_n ← contar dígitos de n;
n_ordenado ← 0;
i ← 1;
Mientras i ≤ total_de_dígitos_de_n Hacer
    menor_dígito_de_n ← obtener menor dígito de n;
    n ← retirar menor dígito de n;
    n_ordenado ← n_ordenado×10 + menor_dígito_de_n;
    i ← i + 1;
Fin Mientras
```

Algoritmo 1: Selección Lineal

A continuación se presentan unos ejemplos de ejecución:

```
Ingrese n: 467312

Cantidad de dígitos: 6
Iteración 1: Menor dígito: 1, n sin menor dígito: 46732, nuevo número formado: 1
Iteración 2: Menor dígito: 2, n sin menor dígito: 4673, nuevo número formado: 12
Iteración 3: Menor dígito: 3, n sin menor dígito: 467, nuevo número formado: 123
Iteración 4: Menor dígito: 4, n sin menor dígito: 67, nuevo número formado: 1234
Iteración 5: Menor dígito: 6, n sin menor dígito: 7, nuevo número formado: 12346
Iteración 6: Menor dígito: 7, n sin menor dígito: 0, nuevo número formado: 123467
```

```
El número obtenido es: 123467
```

```
Ingrese n: 3943

Cantidad de dígitos: 4
Iteración 1: Menor dígito: 3, n sin menor dígito: 394, nuevo número formado: 3
Iteración 2: Menor dígito: 3, n sin menor dígito: 94, nuevo número formado: 33
Iteración 3: Menor dígito: 4, n sin menor dígito: 9, nuevo número formado: 334
Iteración 4: Menor dígito: 9, n sin menor dígito: 0, nuevo número formado: 3349

El número obtenido es: 3349
```

```
Ingrese n: 320

Cantidad de dígitos: 3
Iteración 1: Menor dígito: 0, n sin menor dígito: 32, nuevo número formado: 0
Iteración 2: Menor dígito: 2, n sin menor dígito: 3, nuevo número formado: 2
Iteración 3: Menor dígito: 3, n sin menor dígito: 0, nuevo número formado: 23

El número obtenido es: 23
```

Variación al problema

Realice las adecuaciones necesarias para que en lugar de ordenar los dígitos del número de forma ascendente, lo realice de manera descendente.

2.3.4. Perímetro de Polígonos Irregulares

El perímetro se define en la geometría como la suma de las longitudes de cada uno de los lados de una figura geométrica plana. De forma general se puede decir que el perímetro P se puede calcular como sigue: $P = \sum_{i=1}^{n} l_i$, donde, l_i representa el i-ésimo lado de una figura geométrica y n es la cantidad de lados del polígono. En la figura 2.2 puede ver un ejemplo de un polígono irregular de 4 lados en donde el perímetro $P = l_1 + l_2 + l_3 + l_4$.

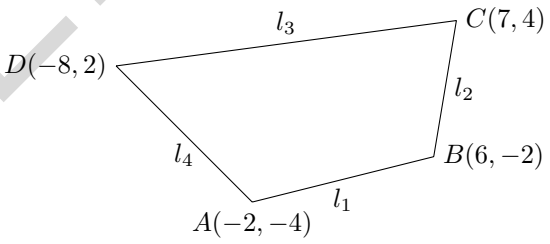


Figura 2.2: Polígono irregular.

Se pide que dado un número n entero positivo que representa una cantidad de polígonos, solicite para cada polígono, la cantidad de vértices que conforman dicho polígono, lea los puntos de todos sus vértices y muestre el perímetro de cada polígono. A continuación se presenta un ejemplo de ejecución:

```
Ingrese cantidad de polígonos: 3
```

```
Polígono 1
Ingrese cantidad de vértices: 5
Ingrese punto 1 (x,y): 3 4
Ingrese punto 2 (x,y): 5 11
Ingrese punto 3 (x,y): 12 8
Ingrese punto 4 (x,y): 9 5
Ingrese punto 5 (x,y): 5 6

El perímetro del polígono es: 26.090

Polígono 2
Ingrese cantidad de vértices: 4
Ingrese punto 1 (x,y): -2 -4
Ingrese punto 2 (x,y): 6 -2
Ingrese punto 3 (x,y): 7 4
Ingrese punto 4 (x,y): -8 2

El perímetro del polígono es: 37.947

Polígono 3
Ingrese cantidad de vértices: 3
Ingrese punto 1 (x,y): 2 4
Ingrese punto 2 (x,y): 3 -8
Ingrese punto 3 (x,y): 1 2

El perímetro del polígono es: 24.423
```

FUNDAMENTOS DE PROGRAMACIÓN
ESTUDIOS GENERALES CIENCIAS
PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Variación al problema

Se le pide que actualice la solución al problema anterior para que en lugar de solicitar la cantidad de polígonos a leer, solicite directamente la cantidad de vértices del polígono. El flujo de lectura de polígonos terminará cuando se introduzca una cantidad de vértices igual a 0.
A continuación se presenta un ejemplo de ejecución:

```
Polígono 1
Ingrese cantidad de vértices: 5
Ingrese punto 1 (x,y): 3 4
Ingrese punto 2 (x,y): 5 11
Ingrese punto 3 (x,y): 12 8
Ingrese punto 4 (x,y): 9 5
Ingrese punto 5 (x,y): 5 6

El perímetro del polígono es: 26.090

Polígono 2
Ingrese cantidad de vértices: 4
Ingrese punto 1 (x,y): -2 -4
Ingrese punto 2 (x,y): 6 -2
Ingrese punto 3 (x,y): 7 4
Ingrese punto 4 (x,y): -8 2

El perímetro del polígono es: 37.947

Ingrese cantidad de vértices: 0
Ha finalizado el flujo de lectura de polígonos.
```

2.3.5. Búsqueda de Patrones

Dado un número natural de n dígitos llamado `número` y un número natural de m dígitos llamado `patrón`, donde $m \leq n$, se pide determinar si es que el `patrón` se encuentra contenido dentro del `número`.

A continuación siguen unos ejemplos de una posible ejecución:

```
Ingrese patrón: 21
Ingrese número: 321421321

El patrón 21 se encuentra en el número 321421321
```

```
Ingrese patrón: 44
Ingrese número: 64443

El patrón 44 se encuentra en el número 64443
```

```
Ingrese patrón: 45
Ingrese número: 5473

El patrón 45 no se encuentra en el número 5473
```

Variación al problema

Dado un número entero de n dígitos llamado `número` y un número entero de m dígitos llamado `patrón`, donde $m \leq n$, se pide encontrar la cantidad de veces que se encuentra el `patrón` dentro del `número`. A continuación siguen unos ejemplos de una posible ejecución:

```
Ingrese patrón: 21
Ingrese número: 321421321

El patrón 21 se encuentra 3 veces en el número 321421321
```

```
Ingrese patrón: 44
Ingrese número: 64443

El patrón 44 se encuentra 2 veces en el número 64443
```

2.3.6. Números pseudo-aleatorios

Una forma de generar números pseudo-aleatorios es mediante el algoritmo de cuadrados medios. Este método fue propuesto en los años 40 por los matemáticos John von Neumann y Nicholas Metropolis. El método comienza con un valor entero x_0 de $2n$ cifras denominado semilla que al elevar al cuadrado resulta en un número de hasta $4n$ cifras. Sea x_1 el número resultante de seleccionar las $2n$ cifras centrales de x_0^2. El número pseudo-aleatorio se obtiene poniéndole un punto decimal delante de las $2n$ cifras de x_1. Para generar el próximo número pseudo-aleatorio se cambia el valor de la semilla por el valor de x_1.

Uno de los problemas de este algoritmo es incapaz de generar una secuencia de números pseudo-aleatorios con periodo de vida grande. En ocasiones sólo es capaz de generar un número. En otras ocasionas sí permite generar una lista grande de números.

Se le pide que dada un número semilla, genere una lista de números pseudo-aleatorios siguiendo el algoritmo de cuadrados medios. La lista deberá contener a lo más 1000 números aleatorios. El algoritmo deberá parar en caso no sea posible generar más números pseudo-aleatorios, es decir cuando el x_1 sea cero. A continuación sigue un ejemplo de ejecución de este algoritmo usando como semilla el número 5735.

intento	x	y	dígitos_medio	número pseudoaleatorio
1	5735	32890225	8902	0.890200
2	8902	79245604	2456	0.245600
3	2456	6031936	319	0.031900
4	319	101761	176	0.017600
5	176	30976	3097	0.309700
6	3097	9591409	5914	0.591400
7	5914	34975396	9753	0.975300
8	9753	95121009	1210	0.121000
9	1210	1464100	4641	0.464100
10	4641	21538881	5388	0.538800
11	5388	29030544	305	0.030500
12	305	93025	9302	0.930200
13	9302	86527204	5272	0.527200
14	5272	27793984	7939	0.793900
15	7939	63027721	277	0.027700
16	277	76729	7672	0.767200
17	7672	58859584	8595	0.859500
18	8595	73874025	8740	0.874000
19	8740	76387600	3876	0.387600
20	3876	15023376	233	0.023300

```
21              233            54289            5428      0.542800
22             5428         29463184            4631      0.463100
23             4631         21446161            4461      0.446100
24             4461         19900521            9005      0.900500
25             9005         81090025             900      0.090000
26              900           810000            1000      0.100000

Se han conseguido 26 números aleatorios.
```

2.3.7. Números vampiros

Un número vampiro es un número natural, con un número par de dígitos, igual al producto de dos números naturales, cada uno con la mitad de dígitos que el original, que contienen, entre ambos, todos los dígitos del número inicial y no acaban los dos en cero.

Por ejemplo, el número 1260 es número vampiro porque 1260 = 21 x 60

En esta ocasión realizaremos una adaptación de la definición de números vampiros y lo llamaremos, el número vampiro adaptado.

Entonces diremos que un **número vampiro adaptado** es un número natural que es igual al producto de dos números naturales, sin considerar al 1 y al mismo número, cuyos dígitos pertenecen a los dígitos del número inicial, sin ser estos necesariamente todos los dígitos del número original.

Bajo esta nueva definición, por ejemplo el número 1395 es un número vampiro adaptado porque 1395 = 9 x 155 y tanto los dígitos 9, 1 y 5 pertenecen a los dígitos del númeor original.

Se le pide que, elabore un programa en lenguaje C que permita evaluar un número e indicar si se trata de un **número vampiro adaptado**.

Para la solución debe implementar y utilizar solo los siguientes módulos:

- Un módulo llamado **evaluarVampiro** que permita validar si un número es vampiro adaptado, en caso lo sea debe devolver los dos números cuyo producto cumple con la definición brindada. Para ello debe recibir como parámetros el número y debe devolver si es un número vampiro o no y los dos números que cumplen con la definición. En caso el número no sea vampiro, debe devolver el valor de 0 en ambos números.

- Un módulo llamado **validarNumero** que permita validar si los digitos de un número pertenecen a los dígitos de otro número. Para ello debe recibir como parámetros el número cuyo digitos queremos validar, el número donde queremos realizar la validación y debe devolver el resultado de la validación indicada.

- Un módulo llamado **validarDigito** que permita validar si un digito pertenece a los dígitos de un número. Para ello debe recibir como parámetros el digito a validar, el número donde queremos realizar la validación y debe devolver el resultado de la validación indicada.

Nota.- Para determinar una posible combinación de números cuyo producto sea igual al número a evaluar, puede hallar los divisores del número y de esta manera hallar las posibles combinaciones.

FUNDAMENTOS DE PROGRAMACIÓN I
ESTUDIOS GENERALES CIENCIAS
PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Casos de prueba para verificación de solución

Use los siguiente casos para verificar si su solución está correcta.

Caso 1
Ingrese el número a evaluar: 1395
El número 1395 es un número vampiro adaptado debido a que 9 x 155 = 1395

Caso 2
Ingrese el número a evaluar: 1256
El número 1256 no es un número vampiro adaptado.

Caso 3
Ingrese el número a evaluar: -5
El número debe ser mayor que cero.

2.3.8. La recta

Desde la geometría analítica, decimos que una recta es la representación gráfica de una expresión algebraica que tiene la forma de una función o ecuación lineal de primer grado. La ecuación general de una recta viene dada por la siguiente ecuación:

$$Ax + By + C = 0$$

La pendiente permite obtener el grado de inclinación que tiene una recta y se denota con la letra m. La pendiente de una recta se puede calcular de la siguiente manera si conocemos la ecuación genera de la recta:

$$m = A/B$$

Además, si conocemos las coordenadas de dos puntos P(x1,y1) y Q(x2,y2) que pertenecen a una recta, también podríamos calcular la pendiente utilizando la siguie fórmula:

$$m = (y2 - y1)/(x2 - x1)$$

Debemos recordar que dos rectas se denominan paralelas si sus pendientes son iguales.

Se le pide que, elabore un programa en lenguaje C que permita, evaluar un número determinado de rectas. Para la evaluación, por cada recta debe solicitar que se ingrese los coeficientes A, B y C que corresponde a la ecuación general de la recta. Además, debe solicitar que ingrese un conjunto de rectas que verificaremos si son paralelas a la ingresada en la ecuación general, para ello debe ingresar las coordenadas x e y de dos puntos P(x1,y1), Q(x2,y2) que pertecen a la recta paralela que queremos verificar.

Para esta verificación debe calcular la pendiente de la recta formada por los 2 puntos ingresados y compararlo con la pendiente de la recta ingresada mediante su ecuación general. Si las pendientes son iguales entonces la recta formada por los 2 puntos es paralela a la recta ingresada por la ecuación, entonces debe calcular la distancia que existe entre estos dos puntos y se continua con la evaluación de la siguiente recta paralela, solicitando el ingreso de dos nuevos puntos P(x1,y1) y Q(x2,y2).

Si durante la verificación de la recta paralela se verifica que esta recta no es paralela a la recta de la ecuación general, se detiene la verificación y se determina que la evaluación es inválida colocando como distancia mayor el valor de 0. En caso todos los puntos ingresados formen rectas que si son paralelas a la recta de la ecuación general, la verificación se considera como válida y se determina la mayor distancia de las rectas paralelas

FUNDAMENTOS DE PROGRAMACIÓN
ESTUDIOS GENERALES CIENCIAS
PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

verificadas. Para terminar el ingreso de los puntos P(x1,y1), Q(x2,y2) a evaluar se debe colocar los valores de 0 para las coordenadas x e y de ambos puntos.

El programa debe realizar esta evaluación por cada ecuación general de la recta ingresada, además debe indicar cuantas ecuaciones generales de recta que fueron válidas.

Para la solución debe implementar y utilizar solo los siguientes módulos:

* Un módulo llamado **validarRectasParalelasyDistanciaMayor** que permita realizar la evaluación de todas las rectas paralelas ingresadas, por medio de dos puntos, para una recta de ecuación general y determinar la mayor longitud de dichas rectas paralelas. Para ello debe recibir como parámetros la pendiente de la recta ingresada por la ecuación general y debe devolver si todas las rectas paralelas verificadas fueron válidas y la mayor longitud de ellas.

* Un módulo llamado **calcularDistancia** que permita calcular la distancia entre dos puntos. Para ello debe recibir como parámetros las coordenadas x e y de los dos puntos y debe devolver la distancia entre ellos.

Para el cálculo de la distancia de dos puntos puede utilizar la siguiente fórmula:

$$distancia = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Nota.- Para la comparación de las pendientes, al ser datos reales, debe calcular la diferencia entre ellos y si esta diferencia es menor a 0.0001 debe considerar que son iguales. La impresión del valor de la distancia debe ser con 2 decimales.

FUNDAMENTOS DE PROGRAMACIÓN
ESTUDIOS GENERALES CIENCIAS
PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Casos de prueba para verificación de solución

Use los siguiente casos para verificar si su solución está correcta.

Caso 1
Ingrese la cantidad de ecuaciones de recta a evaluar: 2
Ingrese los coeficientes A, B y C de la recta a evaluar: 2 1 5
Análisis de puntos
Ingrese las coordenadas x e y de los 2 puntos que pertenecen a la recta paralela a evaluar:
Coordenadas x e y del punto P: 4 6
Coordenadas x e y del punto Q: 5 8
Ingrese las coordenadas x e y de los 2 puntos que pertenecen a la recta paralela a evaluar:
Coordenadas x e y del punto P: 6.5 9
Coordenadas x e y del punto Q: 7.5 11
Ingrese las coordenadas x e y de los 2 puntos que pertenecen a la recta paralela a evaluar:
Coordenadas x e y del punto P: 3 15
Coordenadas x e y del punto Q: 5 19
Ingrese las coordenadas x e y de los 2 puntos que pertenecen a la recta paralela a evaluar:
Coordenadas x e y del punto P: 0 0
Coordenadas x e y del punto Q: 0 0
Todas las rectas verificadas son paralelas a la recta 2x + 1y + 5 = 0
La mayor longitud de las rectas paralelas verificadas fue de 4.47
Ingrese los coeficientes A, B y C de la recta a evaluar: 5 8 3
Análisis de puntos
Ingrese las coordenadas x e y de los 2 puntos que pertenecen a la recta paralela a evaluar:
Coordenadas x e y del punto P: 1.5 7
Coordenadas x e y del punto Q: 3.2 9
Se encontró una recta que no es paralela a la recta 5x + 8y + 3 = 0
La cantidad de ecuaciones de rectas que pasaron la validación fue de: 1

Caso 2
Ingrese la cantidad de ecuaciones de recta a evaluar: 1
Ingrese los coeficientes A, B y C de la recta a evaluar: 5 8 3
Análisis de puntos
Ingrese las coordenadas x e y de los 2 puntos que pertenecen a la recta paralela a evaluar:
Coordenadas x e y del punto P: 1.5 7
Coordenadas x e y del punto Q: 3.2 9
Se encontró una recta que no es paralela a la recta 5x + 8y + 3 = 0
La cantidad de ecuaciones de rectas que pasaron la validación fue de: 0

2.3.9. Operaciones en distintas bases

La siguiente operación 6 x 9 = 42 no se cumple para la base 10, pero si se cumple para la base 13. Es decir:

$$42_{(13)} = 6_{(13)} x 9_{(13)}$$

esto debido a que 42 en base 13 = 54 en base 10, 6 en base 13 = 6 en base 10 y 9 en base 13 = 9 en base 10, por lo tanto 6 x 9 = 54, al pasar de la base 13 a la base 10.

Para convertir el número 42 de base 13 a base 10 se realiza lo siguiente:

$$42_{(10)} = 4x13^1 + 2x13^0 = 54$$

Ojo, debe tener en cuenta que la menor base desde la cual debe comenzar a validar si la multiplicación se

cumple es la base 10 en adelante (para este ejemplo), esto debido a que el mayor digito de los 3 números es 9. Recuerde que cuando un número se encuentra en una base, los dígitos deben ser menores que dicha base.

Se le pide que, elabore un programa en lenguaje C que permita ingresar una cantidad de terna de números a procesar. Por cada terna debe leer 3 números, los cúales son: numero 1, numero 2 y resultado. Con estos tres números debe determinar la menor base para la cual se cumple que numero 1 x numero 2 = resultado, en dicha base, siguiendo lo descrito en el ejemplo. Considere que la base máxima será 16.

Para la solución debe implementar y utilizar solo los módulos que se indican:

- Un módulo llamado **determinarMenorBase** que permita determinar la menor base en que se cumpla la igualdad indicada en el enunciado, en caso no exista ninguna base que cumpla ello debe devolver el valor de 0. Para ello debe recibir como parámetros el numero 1, numero 2 y resultado de una terna y debe devolver la la menor base correspondiente. Debe seguir las indicaciones descritas en el enunciado del problema y en la descripción de este módulo.

- Un módulo llamado **determinarMayorDigito** que permita determinar el mayor dígito de numero. Para ello debe recibir como parámetros el número y debe devolver el mayor dígito de dicho número.

- Un módulo llamado **convertirBase10** que permita convertir un numero de una base a base 10. Para ello debe recibir como parámetros el número, la base y debe devolver el número en base 10.

Nota.- Los módulos indicados son adicionales al main.

Casos de prueba para verificación de solución

Use los siguiente casos para verificar si su solución está correcta.

Caso 1
Ingrese la cantidad de ternas de números a evaluar: 3
Ingrese la terna 1: 6 9 42
La menor base en que se cumple la terna es: 13
Ingrese la terna 2: 11 11 121
La menor base en que se cumple la terna es: 3
Ingrese la terna 3: 2 2 2
La terna no se cumple en ninguna base

Caso 2
Ingrese la cantidad de ternas de números a evaluar: 0
La cantidad de ternas debe ser mayor que 0.

2.3.10. Números Romanos

Los **números romanos** son un grupo de símbolos formado por letras mayúsculas que inventaron los antiguos romanos para poder representar valores y cantidades.

Este sistema de numeración romano se utilizó durante muchos siglos por todo el Imperio romano, así que debido al gran número de lugares donde los romanos tuvieron presencia, después de la caída del Imperio, dejaron huella en todas las regiones donde se asentaron.

En el sistema de números romanos, I es el símbolo para 1, V para 5, X para 10, L para 50, C para 100, D para 500 y M para 1000.

Los símbolos con un valor grande usualmente aparecen antes que los símbolos de menor valor. El valor de un número romano es, en general, la suma de los valores de los símbolos. Por ejemplo, II es 2, VIII es 8. Sin embargo, si un símbolo de menor valor aparece antes de un símbolo de mayor valor, el valor de los dos símbolos es la diferencia de los dos valores. Por ejemplo, IV es 4, IX es 9, y LIX es 59.

Note que no hay cuatro símbolos consecutivos iguales. Por ejemplo, IV, pero no IIII, es el número 4. Los números romanos construidos de esta forma pueden no ser únicos.

Se le pide que, elabore un programa en lenguaje C que permita convertir una cantidad de números naturales a números romanos. Para ello debe ingresar primero la cantidad de números a convertir, luego debe leer cada número que va a convertir, debe validar que el número debe ser mayor que 0 y menor que 500. En caso el número no se encuentre en ese rango, debe mostrar un mensaje de error y debe volver a solicitar el ingreso del número. Por cada número válido debe imprimir su representación romana y al terminar todas las conversiones, debe mostrar la mayor cantidad de caracteres que se utilizó en las representaciones romanas.

Para la solución debe implementar y utilizar solo el siguiente módulo:

- Un módulo llamado **convertirNumeroRomano** que permita realizar la conversión de un número a su representación romana e imprimirlo. Además, debe devolver la cantidad de caracteres (letras) que utilizó en su representación. Para ello debe recibir como parámetros el número y debe devolver la cantidad de caracteres que utilizó en su representación romana, así como realizar la impresión de la misma.

Nota.- El módulo indicado es adicional al main.

Casos de prueba para verificación de solución

Use los siguiente casos para verificar si su solución está correcta.

Caso 1
Ingrese la cantidad de números a procesar: 3
Ingrese el número 1 a convertir: 316
El número en romano es: CCCXVI
Ingrese el número 2 a convertir: 108
El número en romano es: CVIII
Ingrese el número 3 a convertir: 205
El número en romano es: CCV
La mayor cantidad de caracteres utilizada en un número romano fue de 6 caracteres.

Caso 2
Ingrese la cantidad de números a procesar: 4
Ingrese el número 1 a convertir: 115
El número en romano es: CXV
Ingrese el número 2 a convertir: -5
El número debe ser mayor que 0 y menor que 500.
Ingrese el número 2 a convertir: 237
El número en romano es: CCXXXVII
Ingrese el número 3 a convertir: 500
El número debe ser mayor que 0 y menor que 500.
Ingrese el número 3 a convertir: 103
El número en romano es: CIII
Ingrese el número 4 a convertir: 204
El número en romano es: CCIV
La mayor cantidad de caracteres utilizada en un número romano fue de 8 caracteres.

2.3.11. Números afortunados

Un **número afortunado** (nombre atribuido por Reo Franklin Fortune) es un número primo que puede resultar de la siguiente expresión:

$$Q = q - (P_n)$$

Reo Fortune conjeturó que si q es el primo más cercano y mayor que P+1, entonces q-P es primo.

Por ejemplo, si n es 3, entonces P es 2x3x5=30, q=37, y q-P es primo: 7.

El valor de P es 2x3x5 = 30 porque los números 2, 3 y 5 son los 3 primeros números primos (n=3). El valor de q es 37 porque es el siguiente número primo mayor a 31 (P+1 = 31, porque P=30). Entonces q-P es 37 - 30 = 7, el cual es número primo.

Veamos otro ejemplo para n = 4.

El valor de P es 2x3x5x7 = 210 porque los números 2, 3, 5 y 7 son los 4 primeros números primos (n=4). El valor de q es 223 porque es el siguiente número primo mayor a 211 (P+1 = 211, porque P=210). Entonces q-P es 223 - 210 = 13, el cual es número primo.

Se le pide que, elabore un programa en lenguaje C que permita determinar los primeros n números afortunados. Para ello debe ingresar el número n, debe considerar que el valor de n debe ser mayor a 0 y menor que 10, en caso no cumpla esta condición debe mostrar un mensaje de error y terminará el programa. En caso el valor de n sea correcto, debe calcular lo solicitado de acuerdo a lo descrito en el enunciado.

Para la solución debe implementar y utilizar solo los siguientes módulos:

- Un módulo llamado **calcularNumeroAfortunado** que permita determinar el número afortunado para un número n siguiendo lo descrito en el caso. Para ello debe recibir como parámetros el número y debe devolver número afortunado.

- Un módulo llamado **hallarProductoPrimerosPrimos** que permita determinar el producto de los primeros n números primos. Para ello debe recibir como parámetros el número n y debe devolver el producto solicitado.

- Un módulo llamado **hallarSiguientePrimo** que permita determinar el siguiente número primo que es mayor a un número primo dado. Para ello debe recibir como parámetros el número y debe devolver el siguiente número primo mayor al número pasado como parámetro. Debe seguir las indicaciones descritas en el enunciado del problema.

- Un módulo llamado **evaluarPrimo** que permita determinar si un número es primo o no. Para ello debe recibir como parámetros el número a evaluar y debe devolver si el número es primo o no.

Nota.- Los módulos indicados son adicionales al main.

Casos de prueba para verificación de solución

Use los siguiente casos para verificar si su solución está correcta.

Caso 1
Ingrese el valor de n: 6
El número afortunado 1 es 3.
El número afortunado 2 es 5.
El número afortunado 3 es 7.
El número afortunado 4 es 13.
El número afortunado 5 es 23.
El número afortunado 6 es 17.

Caso 2
Ingrese el valor de n: 10
El número ingresado debe ser mayor que 0.

2.3.12. Raíz digital prima

La **raíz digital** de un número se encuentra definida por un dígito único que resulta de sumar los dígitos de dicho número, y sumando los dígitos del número resultante hasta que se obtenga un solo dígito.

Por ejemplo, la raíz digital del número 129 se puede encontrar de la siguiente manera: 1+ 2 + 9 = 12, ya que el número 12 tiene más de un dígito, lo hacemos de nuevo: 1 + 2 = 3, entonces la raíz digital de 129 es 3.

Esta vez realizaremos una adaptación de la raíz digital y la llamaremos la **raíz digital prima**, la cual consiste en realizar lo siguiente:

1.- Se evalúa si el número es primo, en caso lo sea, dicho número será la raíz digital prima.
2.- Si el número no es primo, se debe realizar la suma de sus dígitos y volver al paso 1.

Como puede observar, el proceso se detiene cuando se encuentra un número primo, sin embargo, puede darse el caso que al sumar los dígitos se obtenga finalmente un número de un dígito, en ese momento se realizará la validación de si dicho número es primo o no y culminará la repetición de los pasos porque ya no se podrá realizar la suma de dígitos.

Aquí se muestra un ejemplo:

Se tiene el número 369, al no ser un número primo se realiza la suma de sus dígitos 3 + 6 + 9 = 18, al no ser primo el número 18, se realiza la suma de sus dígitos 1 + 8 = 9, al no ser primo el número 9 y ser un número de un dígito el proceso termina y se concluye que el número 369 no tiene raíz digital prima.

Aquí se muestra otro ejemplo:

Se tiene el número 373, al no ser un número primo se realiza la suma de sus dígitos 3 + 7 + 3 = 13, al ser primo el número 13, el proceso termina y se concluye que el número 373 tiene raíz digital prima y es el número 13.

Se le pide que, elabore un programa en lenguaje C que permita determinar qué números poseen una raíz digital prima dentro de un rango de números. Para ello debe ingresar primero el rango de números a evaluar, si el número inicial es mayor al número final, debe mostrar un mensaje de error y terminará el programa. En caso de ser correcto, debe determinar por cada número que se encuentra en dicho rango si tiene tiene raíz digital prima y en caso de tenerla debe indicar cuál es su valor.

Para la solución debe implementar y utilizar solo los siguientes módulos:

- Un módulo llamado **evaluarRaizDigital** que permita determinar si un número tiene raíz digital prima y en caso de tenerla debe devolver cual es su valor, en caso de no tenerla debe devolver el valor de 0. Para ello debe recibir como parámetros el número y debe devolver el indicador de si tiene ráiz digital prima o no así como el valor de la raíz digital prima, siguiendo lo descrito.

- Un módulo llamado **evaluarPrimo** que permita determinar si un número es primo. Para ello debe recibir como parámetros el número y debe devolver si el número es primo o no.

- Un módulo llamado **calcularSumaDigitos** que permita realizar la suma de los dígitos de un número. Para ello debe recibir como parámetros el número y debe devolver la suma de los dígitos del número.

Nota.- Los módulos indicados son adicionales al main.

FUNDAMENTOS DE PROGRAMACIÓN
ESTUDIOS GENERALES CIENCIAS
PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Casos de prueba para verificación de solución

Use los siguiente casos para verificar si su solución está correcta.

Caso 1
Ingrese el rango de números a evaluar: 360 365
El número 360 no tiene raíz digital prima.
El número 361 no tiene raíz digital prima.
El número 362 tiene raíz digital prima y su valor es 11.
El número 363 tiene raíz digital prima y su valor es 3.
El número 364 tiene raíz digital prima y su valor es 13.
El número 365 tiene raíz digital prima y su valor es 5.

Caso 2
Ingrese el rango de números a evaluar: 365 360
El rango ingresado no es correcto

2.3.13. Número semiperfecto (adaptado del laboratorio 7 2020-2)

Un número semiperfecto es un número natural que es igual a la suma de algunos de sus divisores propios (por lo menos dos divisores).

Por ejemplo, el número 12 es un número semiperfecto porque sus divisores son: 1, 2, 3, 4, 6, 12 y se cumple que la suma de algunos de sus divisores como: 2, 4 y 6 es igual a 12.

Como se puede observar, los divisores del número, cuya suma tiene que ser igual al número, pueden omitir algunos divisores en orden de menor a mayor, por lo que en esta ocasión realizaremos una adaptación de la definición de números semiperfectos y lo llamaremos, el número semiperfecto adaptado.

Entonces diremos que un **número semiperfecto adaptado** es igual a la suma de algunos de sus divisores propios (por lo menos dos divisores), considerando que estos divisores deben estar en orden de menor a mayor, sin obviar alguno de los mismos.

Bajo esta nueva definición, por ejemplo el número 12 no sería semiperfecto adaptado, porque sus divisores en orden de menor a mayor son: 1, 2, 3, 4, 6, 12 y la combinación de divisores cuya suma es igual a 12 sería 2, 4 y 6. Sin embargo, entre el 2 y el 4 falta el divisor 3 por lo que no cumple con la condición de que los divisores deben estar en orden de menor a mayor sin obviar alguno de los mismos, ya que en este caso se esta obviando al 3.

Un número semiperfecto adaptado sería el número 6, porque sus divisores son: 1, 2, 3 y 6. La suma de los divisores 1, 2 y 3 da como resultado 6 que es el mismo número y entre dichos divisores no se está obviando ningún divisor.

Otro ejemplo de número semiperfecto adaptado sería el número 18, porque sus divisores son: 1, 2, 3, 6, 9 y 18. La suma de los divisores 3, 6 y 9 da como resultado 18 que es el mismo número y entre dichos divisores no se está obviando ningún divisor.

Se le pide que, elabore un programa en lenguaje C que permita evaluar un número e indicar si se trata de un **número semiPerfecto adaptado**.

Para la solución debe implementar y utilizar solo los siguientes módulos de forma adicional al módulo main:

- Un módulo llamado **validarSemiPerfecto** que permita validar si un número es semiperfecto adaptado. Para ello debe recibir como parámetro el número y debe devolver si es un número semiperfecto adaptado o no.

- Un módulo llamado **validarDatos** que permita validar si el número es natural. Para ello debe recibir como parámetro el número y debe devolver el resultado de la validación indicada.

Nota.- Un número es natural cuando es mayor que cero.

FUNDAMENTOS DE PROGRAMACIÓN
ESTUDIOS GENERALES CIENCIAS
PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Casos de prueba para verificación de solución

Use los siguiente casos para verificar si su solución está correcta.

Caso 1
Ingrese el número a evaluar: 12
El número 12 no es semiPerfecto adaptado.

Caso 2
Ingrese el número a evaluar: 18
El número 18 es semiPerfecto adaptado.

Caso 3
Ingrese el número a evaluar: -5
El número no es natural.

Caso 4
Ingrese el número a evaluar: 24
El número 24 es semiPerfecto adaptado.

2.3.14. Número de Smith (adaptado del laboratorio 7 2020-2)

Un número natural n se llama **de Smith** si cumple que la suma de sus dígitos es igual a la suma de los dígitos de sus divisores primos.

Para entender mejor el concepto de número de Smith veamos los siguientes ejemplos:

El número 166 es de Smith debido a que la suma de sus dígitos es igual a la suma de los dígitos de sus divisores primos.

Suma de dígitos del número 166 = 1 + 6 + 6 = 13.

Los divisores del número 166 son 1, 2, 83 y 166. Los divisores primos son 2 y 83, por lo tanto al sumar los dígitos de dichos divisores sale: 2 + 8 + 3 = 13.

En este caso la suma de los dígitos del número es 13 y la suma de los dígitos de los divisores primos del número 166 también es 13, por lo tanto el número 166 es de Smith.

A continuación veamos otro ejemplo:

El número 165 no es de Smith, debido a que la suma de sus dígitos no es igual a la suma de los dígitos de sus divisores primos.

Suma de dígitos del número 165 = 1 + 6 + 5 = 12.

Los divisores del número 165 son 1, 3, 5, 11, 15, 33, 55 y 165. Los divisores primos son 3, 5 y 11, por lo tanto al sumar los dígitos de dichos divisores primos sale: 3 + 5 + 1 + 1 = 10.

En este caso la suma de los dígitos del número es 12 y la suma de los dígitos de los divisores primos del número 165 también es 10, por lo tanto el número 165 no es de Smith.

Se le pide que, elabore un programa en lenguaje C que permita evaluar un número e indicar si se trata de un **número de Smith**.

Para la solución debe implementar y utilizar solo los siguientes módulos adicionales al main:

FUNDAMENTOS DE PROGRAMACIÓN
ESTUDIOS GENERALES CIENCIAS
PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

- Un módulo llamado **validarDatos** que permita validar si el número es natural. Para ello debe recibir como parámetro el número y debe devolver el resultado de la validación indicada.

- Un módulo llamado **validarSmith** que permita determinar si un número es de Smith siguiendo lo explicado en el enunciado. Dentro de su implementación debe utilizar los módulos **calcular_SumaDigitos** y **evaluar_Primo** que se explican en los siguientes puntos. Para ello, debe recibir como parámetros el número y debe devolver el indicador de si es un número de Smith o no.

- Un módulo llamado **evaluar_Primo** que permita validar si un número es primo. Para ello debe recibir como parámetro el número y debe devolver el resultado de la validación indicada.

- Un módulo llamado **calcular_SumaDigitos** que permita determinar la suma de los dígitos de un número. Para ello debe recibir como parámetros el número y debe devolver la suma de dígitos del número.

Nota.- Un número es natural cuando es mayor que cero.

Casos de prueba para verificación de solución

Use los siguiente casos para verificar si su solución está correcta.

Caso 1
Ingrese el número a evaluar: 166
El número 66 es de Smith.

Caso 2
Ingrese el número a evaluar: 265
El número 265 es de Smith.

Caso 3
Ingrese el número a evaluar: 165
El número 165 no es de Smith.

Caso 4
Ingrese el número a evaluar: -15
El número no es natural

2.3.15. Sucesión de Fareyh (adaptado del laboratorio 7 2020-2)

La Sucesión de Farey es una de esas curiosidades matemáticas a la vez llenas de belleza y fáciles de entender que casualmente descubrió un no-matemático, John Farey, en 1928.

Es una sucesión matemática de fracciones irreductibles entre 0 y 1 que tienen un denominador menor o igual a n en orden creciente, donde n es un número entero positivo. Los términos de la sucesión de Farey comienzan con el número 0, denotado por la fracción: $\frac{0}{1}$, y termina con el número 1, denotado por la fracción:$\frac{1}{1}$, sin embargo, algunos autores suelen omitir estos términos.

Una forma algorítmica sencilla de construir la sucesión de Farey para un número n (por ejemplo, en este caso n=4) es la siguiente:

- Se construyen fracciones con todas las combinaciones posibles de los números del 1 al 4.

  $\frac{1}{1}, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{2}{1}, \frac{2}{2}, \frac{2}{3}, \frac{2}{4}, \frac{3}{1}, \frac{3}{2}, \frac{3}{3}, \frac{3}{4}, \frac{4}{1}, \frac{4}{2}, \frac{4}{3}, \frac{4}{4}$

- Se eliminan aquellas fracciones iguales o superiores a 1, o dicho de otra manera, en las que el numerador sea mayor o igual al denominador.

  $\frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{2}{3}, \frac{2}{4}, \frac{3}{4}$

- Las fracciones que se puedan simplificar se descartan dentro de la serie porque estarían repetidas.

  $\frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{2}{3}, \frac{3}{4}$

- Se agrega el número 0 al inicio y el número 1 al final.

  $0, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{2}{3}, \frac{3}{4}, 1$

Se le pide que, elabore un programa en lenguaje C que permita mostrar la sucesión de Farey para un número n que se encuentre dentro de un rango de números. Para ello, primero debe ingresar el rango de números a evaluar, si el número inicial es mayor al número final o alguno de ellos no es un número natural, debe mostrar un mensaje de error y terminará el programa. En caso de ser correcto, debe mostrar para cada número su sucesión de Farey.

Para la solución debe implementar y utilizar solo los siguientes módulos:

- Un módulo llamado **validarRango** que permita determinar si el número inicial y número final del rango son válidos. Para ello debe recibir como parámetros el número inicial y número final y debe devolver el indicador de si los datos son válidos o no.

- Un módulo llamado **mostrarSucesionFarey** que permita mostrar los términos de la sucesión de Farey para un número. Para ello debe recibir como parámetros el número y debe mostrar los términos de la sucesión de Farey para ese número. Dentro de su implementación debe utilizar el módulo **calcularMCD**.

- Un módulo llamado **calcularMCD** que permita determinar el máximo común divisor de dos números. Para ello debe recibir como parámetro los dos números y debe devolver el máximo común divisor de los dos.

Nota.- Los módulos indicados son adicionales al main.

Casos de prueba para verificación de solución

Use los siguiente casos para verificar si su solución está correcta.

Caso 1
Ingrese el rango a y b: 2 5
La serie de Farey para n= 2 es: 0, 1/2, 1
La serie de Farey para n= 3 es: 0, 1/2, 1/3, 2/3, 1
La serie de Farey para n= 4 es: 0, 1/2, 1/3, 1/4, 2/3, 3/4, 1
La serie de Farey para n= 5 es: 0, 1/2, 1/3, 1/4, 1/5, 2/3, 2/5, 3/4, 3/5, 4/5, 1

Caso 2
Ingrese el número a evaluar: 6 7
La serie de Farey para n= 6 es: 0, 1/2, 1/3, 1/4, 1/5, 1/6, 2/3, 2/5, 3/4, 3/5, 4/5, 5/6, 1
La serie de Farey para n= 7 es: 0, 1/2, 1/3, 1/4, 1/5, 1/6, 1/7, 2/3, 2/5, 2/7, 3/4, 3/5, 3/7, 4/5, 4/7, 5/6, 5/7, 6/7, 1

Caso 3
Ingrese el número a evaluar: 10 6
El rango ingresado es incorrecto.

PROGRAMACIÓN DE PROGRAMAS GENERALES CIENCIAS DAD CATÓLICA DEL PERÚ

2.3.16. Número automorfo (adaptado del laboratorio 7 2020-2)

Un número natural n se llama **automorfo** si cumple que los últimos dígitos del cuadrado del número es igual al mismo número. Por ejemplo, el número 25 es automorfo dado que el cuadrado del número 25 es 625, entonces los últimos 2 dígitos del número 625 es 25 el cuál es igual al mismo número 25.

Otro ejemplo de número automorfo sería el número 376 dado que el cuadrado del número 376 es 141376, entonces los últimos 3 dígitos del número 141376 es 376 el cuál es igual al mismo número 376

Se le pide que, elabore un programa en lenguaje C que permita determinar qué números son automorfos dentro de un rango de números. Para ello debe ingresar primero el rango de números a evaluar, si el número inicial es mayor al número final o alguno de ellos no es un número natural, debe mostrar un mensaje de error y terminará el programa. En caso de ser correcto, debe determinar por cada número que se encuentra en dicho rango si es automorfo o no y en caso de serlo, debe mostrar un mensaje indicando que es un número automorfo. Al finalizar la evaluación de todo el rango de números, debe mostrar la cantidad de números automorfos encontrados así como el menor número automorfo encontrado.

Para la solución debe implementar y utilizar solo los módulos que se indican:

- Un módulo llamado **validar_Datos** que permita determinar si el número inicial y número final son válidos. Para ello debe recibir como parámetros el número inicial y número final y debe devolver el indicador de si los datos son válidos o no.

- Un módulo llamado **validar_Automorfo** que permita determinar si un número es automorfo o no. Para ello debe recibir como parámetros el número y debe devolver el indicador de si el número es automorfo o no. Debe tener encuenta que en la implementación de este módulo debe utilizar a los módulos evaluarCantidadDigitos y obtenerUltimosDigitos que se describen en los siguientes puntos.

- Un módulo llamado **evaluarCantidadDigitos** que permita determinar la cantidad de dígitos que posee un número. Para ello debe recibir como parámetro el número y debe devolver la cantidad de dígitos que posee.

- Un módulo llamado **obtenerUltimosDigitos** que permita obtener el número formado por los últimos dígitos de un número. Para ello debe recibir como parámetros el número de donde obtendrá los últimos dígitos y la cantidad de últimos dígitos que desea obtener del mismo y debe devolver dicho número extraído. Para dar un ejemplo, si este módulo recibe como parámetros el número 1234 y como cantidad de dígitos el valor de 3, entonces debe devolver el valor de 234 que representan a los 3 últimos dígitos del número 1234.

Nota.- Los módulos indicados son adicionales al main. Un número es natural cuando es mayor que 0.

FUNDAMENTOS DE PROGRAMACIÓN
ESTUDIOS GENERALES CIENCIAS
PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Casos de prueba para verificación de solución

Use los siguiente casos para verificar si su solución está correcta.

Caso 1
Ingrese el rango de números a evaluar: 20 30
El número 20 no es automorfo
El número 21 no es automorfo
El número 22 no es automorfo
El número 23 no es automorfo
El número 24 no es automorfo
El número 25 es automorfo
El número 26 no es automorfo
El número 27 no es automorfo
El número 28 no es automorfo
El número 29 no es automorfo
El número 30 no es automorfo

Estadísticas

Se encontraron 1 números automorfos en el rango de 20 y 30
El menor número automorfo encontrado fue 25.

Caso 2
Ingrese el rango de números a evaluar: 31 34
El número 31 no es automorfo
El número 32 no es automorfo
El número 33 no es automorfo
El número 34 no es automorfo

Estadísticas

Existen 0 números automorfos en el rango de 31 y 34

Caso 3
Ingrese el rango de números a evaluar: 40 30
El rango de números ingresados es incorrecto

2.3.17. Número oblongo (adaptado del laboratorio 7 2020-2)

Un número natural n se llama **oblongo** si cumple que es el producto de dos números naturales consecutivos. Por ejemplo, el número 30 es oblongo porque es el resultado de multiplicar los números 5 y 6. Otro ejemplo es el número 42 que es el resultado de multiplicar los números 6 y 7.

Se le pide que, elabore un programa en lenguaje C que permita determinar qué números son oblongos dentro de un rango de números. Para ello debe ingresar primero el rango de números a evaluar, si el número inicial es mayor al número final, debe mostrar un mensaje de error y terminará el programa. En caso de ser correcto, debe determinar por cada número que se encuentra en dicho rango si es oblongo o no y en caso de serlo, debe mostrar un mensaje indicando que es un número oblongo así como los números cuyo producto originan que sea oblongo. Al finalizar la evaluación de todo el rango de números, debe mostrar la cantidad de números oblongos encontrados así como el mayor número oblongo encontrado.

Para la solución debe implementar y utilizar solo los módulos que se indican:

- Un módulo llamado **validar_Datos** que permita determinar si el número inicial y número final son válidos. Para ello debe recibir como parámetros el número inicial y número final y debe devolver el indicador de si los datos son válidos o no.

- Un módulo llamado **evaluar_Oblongo** que permita determinar si un número es oblongo o no. Para ello debe recibir como parámetros el número y debe devolver el indicador de si el número es oblongo o no, además debe devolver los dos números consecutivos cuyo producto permite que el número evaluado sea oblongo. En caso el número no sea oblongo, debe devolver en los dos números consecutivos el valor de 0.

Nota.- Los módulos indicados son adicionales al main.

Casos de prueba para verificación de solución

Use los siguiente casos para verificar si su solución está correcta.

Caso 1
Ingrese el rango de números a evaluar: 28 45
El número 28 no es oblongo
El número 29 no es oblongo
El número 30 es oblongo porque es el resultado de la multiplicación de 5 y 6.
El número 31 no es oblongo
El número 32 no es oblongo
El número 33 no es oblongo
El número 34 no es oblongo
El número 35 no es oblongo
El número 36 no es oblongo
El número 37 no es oblongo
El número 38 no es oblongo
El número 39 no es oblongo
El número 40 no es oblongo
El número 41 no es oblongo
El número 42 es oblongo porque es el resultado de la multiplicación de 6 y 7.
El número 43 no es oblongo
El número 44 no es oblongo
El número 45 no es oblongo

Estadísticas

Existen 2 números oblongos en el rango de 28 y 45
El mayor número oblongo fue 42.

Caso 2
Ingrese el rango de números a evaluar: 31 34
El número 31 no es oblongo
El número 32 no es oblongo
El número 33 no es oblongo
El número 34 no es oblongo

Estadísticas

Existen 0 números oblongos en el rango de 31 y 34

Caso 3
Ingrese el rango de números a evaluar: 40 30
El rango de números ingresados es incorrecto

2.4. Taylor Expansión 1 (adaptado del laboratorio 7 2021-1)

Una de las expansiones de la serie de Taylor permite calcular el valor de $\frac{1}{1-x}$, de la siguiente manera:

$$\frac{1}{1-x} = 1 + x + x^2 + x^3 + x^4 + \cdots = \sum_{i=0}^{\infty} x^i \tag{2.1}$$

Se le pide que elabore un programa en lenguaje C que permita ejecutar un serie de experimentos que intentarán determinar con cual de ellos se logra una mejor precisión. Para ello deberá:

- Leer el valor de x así como del valor de la cantidad de experimentos que desea realizar.
- Con el valor de x leído, deberá presentar el valor esperado.
- Para cada experimento deberá leer la cantidad de iteraciones con la que va a calcular el valor de la serie y posteriormente deberá hacer el calculo de dicha serie.
- Al finalizar la ejecución de los experimentos deberá imprimir la cantidad de iteraciones con que se consiguió la mejor precisión. En caso de que se consigan dos precisiones iguales, deberá mostrar la cantidad de iteraciones de la primera de ellas.

Para su solución debe utilizar programación modular (al menos 2 módulos incluyendo el main).

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

```
Ingrese el valor de x: 0.5
Ingrese cantidad de experimentos: 5
El valor esperado es: 2.000000
Ingrese cantidad de iteraciones para experimento 1: 30
El valor obtenido es: 2.000000
Ingrese cantidad de iteraciones para experimento 2: 10
El valor obtenido es: 1.998047
Ingrese cantidad de iteraciones para experimento 3: 6
El valor obtenido es: 1.968750
Ingrese cantidad de iteraciones para experimento 4: 15
El valor obtenido es: 1.999939
Ingrese cantidad de iteraciones para experimento 5: 16
El valor obtenido es: 1.999969

La mejor precisión se logra con 30 iteraciones
```

```
Ingrese el valor de x: 0.9
Ingrese cantidad de experimentos: 3
El valor esperado es: 10.000000
Ingrese cantidad de iteraciones para experimento 1: 5
El valor obtenido es: 4.095100
Ingrese cantidad de iteraciones para experimento 2: 15
El valor obtenido es: 7.941089
Ingrese cantidad de iteraciones para experimento 3: 200
El valor obtenido es: 10.000000

La mejor precisión se logra con 200 iteraciones
```

```
Ingrese el valor de x: 0.1
Ingrese cantidad de experimentos: 6
El valor esperado es: 1.111111
Ingrese cantidad de iteraciones para experimento 1: 10
El valor obtenido es: 1.111111
Ingrese cantidad de iteraciones para experimento 2: 9
El valor obtenido es: 1.111111
Ingrese cantidad de iteraciones para experimento 3: 8
El valor obtenido es: 1.111111
Ingrese cantidad de iteraciones para experimento 4: 7
El valor obtenido es: 1.111111
Ingrese cantidad de iteraciones para experimento 5: 6
El valor obtenido es: 1.111110
Ingrese cantidad de iteraciones para experimento 6: 5
El valor obtenido es: 1.111100

La mejor precisión se logra con 10 iteraciones
```

2.5. Taylor Expansión 2 (adaptado del laboratorio 7 2021-1)

Una de las expansiones de la serie de Taylor permite calcular el valor de $\frac{1}{1-cx}$, de la siguiente manera:

$$\frac{1}{1-cx} = 1 + cx + c^2x^2 + c^3x^3 + c^4x^4 + \cdots = \sum_{i=0}^{\infty} c^i x^i \tag{2.2}$$

Se le pide que elabore un programa en lenguaje C que permita ejecutar un serie de experimentos que intentarán determinar con cual de ellos se logra la peor precisión. Para ello deberá:

- Leer el valor de x, el valor de c así como del valor de la cantidad de experimentos que desea realizar.
- Con los valor de x y c leídos, deberá presentar el valor esperado.
- Para cada experimento deberá leer la cantidad de iteraciones con la que va a calcular el valor de la serie y posteriormente deberá hacer el calculo de dicha serie.
- Al finalizar la ejecución de los experimentos deberá imprimir la cantidad de iteraciones con que se consiguió la peor precisión. En caso de que se consigan dos precisiones iguales, deberá mostrar la cantidad de iteraciones de la primera de ellas.

Para su solución debe utilizar programación modular (al menos 2 módulos incluyendo el main).

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

```
Ingrese el valor de x: 0.5
Ingrese el valor de c: 0.1
Ingrese cantidad de experimentos: 3
El valor esperado es: 1.052632
Ingrese cantidad de iteraciones para experimento 1: 10
El valor obtenido es: 1.052632
Ingrese cantidad de iteraciones para experimento 2: 5
El valor obtenido es: 1.052631
Ingrese cantidad de iteraciones para experimento 3: 3
```

```
El valor obtenido es: 1.052500

La peor precisión se logra con 3 iteraciones
```

```
Ingrese el valor de x: 0.5
Ingrese el valor de c: 0.1
Ingrese cantidad de experimentos: 5
El valor esperado es: 1.052632
Ingrese cantidad de iteraciones para experimento 1: 10000
El valor obtenido es: 1.052632
Ingrese cantidad de iteraciones para experimento 2: 20000
El valor obtenido es: 1.052632
Ingrese cantidad de iteraciones para experimento 3: 15000
El valor obtenido es: 1.052632
Ingrese cantidad de iteraciones para experimento 4: 100
El valor obtenido es: 1.052632
Ingrese cantidad de iteraciones para experimento 5: 500
El valor obtenido es: 1.052632

La peor precisión se logra con 10000 iteraciones
```

```
Ingrese el valor de x: 0.9
Ingrese el valor de c: 0.5
Ingrese cantidad de experimentos: 2
El valor esperado es: 1.818182
Ingrese cantidad de iteraciones para experimento 1: 100
El valor obtenido es: 1.818182
Ingrese cantidad de iteraciones para experimento 2: 10
El valor obtenido es: 1.817563

La peor precisión se logra con 10 iteraciones
```

2.6. Taylor Expansión 3 (adaptado del laboratorio 7 2021-1)

Una de las expansiones de la serie de Taylor permite calcular el valor de $\frac{1}{1-x^n}$, de la siguiente manera:

$$\frac{1}{1-x^n} = 1 + x^n + x^{2n} + x^{3n} + x^{4n} + \cdots = \sum_{i=0}^{\infty} x^{ni} \tag{2.3}$$

Se le pide que elabore un programa en lenguaje C que permita ejecutar un serie de experimentos que intentarán determinar con cual de ellos se logra una mejor precisión. Para ello deberá:

- Leer el valor de x, el valor de n así como del valor de la cantidad de experimentos que desea realizar.
- Con los valores de x y n leídos, deberá presentar el valor esperado.
- Para cada experimento deberá leer la cantidad de iteraciones con la que va a calcular el valor de la serie y posteriormente deberá hacer el calculo de dicha serie.
- Al finalizar la ejecución de los experimentos deberá imprimir la cantidad de iteraciones con que se consiguió la mejor precisión. En caso de que se consigan dos o más precisiones iguales, deberá mostrar la cantidad de iteraciones de la última de ellas.

FUNDAMENTOS DE PROGRAMACIÓN
ESTUDIOS GENERALES CIENCIAS
PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Para su solución debe utilizar programación modular (al menos 2 módulos incluyendo el main).

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

```
Ingrese el valor de x: 0.1
Ingrese cantidad de experimentos: 6
El valor esperado es: 1.111111
Ingrese cantidad de iteraciones para experimento 1: 10
El valor obtenido es: 1.111111
Ingrese cantidad de iteraciones para experimento 2: 9
El valor obtenido es: 1.111111
Ingrese cantidad de iteraciones para experimento 3: 8
El valor obtenido es: 1.111111
Ingrese cantidad de iteraciones para experimento 4: 7
El valor obtenido es: 1.111111
Ingrese cantidad de iteraciones para experimento 5: 6
El valor obtenido es: 1.111110
Ingrese cantidad de iteraciones para experimento 6: 5
El valor obtenido es: 1.111100

La mejor precisión se logra con 10 iteraciones
```

```
Ingrese el valor de x: 0.5
Ingrese el valor de n: 0.5
Ingrese cantidad de experimentos: 3
El valor esperado es: 3.414214
Ingrese cantidad de iteraciones para experimento 1: 10000
El valor obtenido es: 3.414214
Ingrese cantidad de iteraciones para experimento 2: 20000
El valor obtenido es: 3.414214
Ingrese cantidad de iteraciones para experimento 3: 30000
El valor obtenido es: 3.414214

La mejor precisión se logra con 30000 iteraciones
```

```
Ingrese el valor de x: 0.9
Ingrese el valor de n: 3
Ingrese cantidad de experimentos: 3
El valor esperado es: 3.690037
Ingrese cantidad de iteraciones para experimento 1: 10000
El valor obtenido es: 3.690037
Ingrese cantidad de iteraciones para experimento 2: 20000
El valor obtenido es: 3.690037
Ingrese cantidad de iteraciones para experimento 3: 30000
El valor obtenido es: 3.690037

La mejor precisión se logra con 30000 iteraciones
```

FUNDAMENTOS DE PROGRAMACIÓN
ESTUDIOS GENERALES CIENCIAS
PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

2.7. Taylor Expansión 4 (adaptado del laboratorio 7 2021-1)

Una de las expansiones de la serie de Taylor permite calcular el valor de $\frac{x}{(1-x)^2}$, de la siguiente manera:

$$\frac{x}{(1-x)^2} = x + 2x^2 + 3x^3 + 4x^4 + \cdots = \sum_{i=0}^{\infty} ix^i \tag{2.4}$$

Se le pide que elabore un programa en lenguaje C que permita ejecutar un serie de experimentos que intentarán determinar con cual de ellos se logra la peor precisión. Para ello deberá:

- Leer el valor de x así como del valor de la cantidad de experimentos que desea realizar.
- Con el valor de x leído, deberá presentar el valor esperado.
- Para cada experimento deberá leer la cantidad de iteraciones con la que va a calcular el valor de la serie y posteriormente deberá hacer el calculo de dicha serie.
- Al finalizar la ejecución de los experimentos deberá imprimir la cantidad de iteraciones con que se consiguió la peor precisión. En caso de que se consigan dos o más precisiones iguales, deberá mostrar la cantidad de iteraciones de la última de ellas.

Para su solución debe utilizar programación modular (al menos 2 módulos incluyendo el main).

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

```
Ingrese el valor de x: 0.5
Ingrese cantidad de experimentos: 5
El valor esperado es: 2.000000
Ingrese cantidad de iteraciones para experimento 1: 1000
El valor obtenido es: 2.000000
Ingrese cantidad de iteraciones para experimento 2: 700
El valor obtenido es: 2.000000
Ingrese cantidad de iteraciones para experimento 3: 500
El valor obtenido es: 2.000000
Ingrese cantidad de iteraciones para experimento 4: 400
El valor obtenido es: 2.000000
Ingrese cantidad de iteraciones para experimento 5: 300
El valor obtenido es: 2.000000

La peor precisión se logra con 300 iteraciones
```

```
Ingrese el valor de x: 0.9
Ingrese cantidad de experimentos: 3
El valor esperado es: 90.000000
Ingrese cantidad de iteraciones para experimento 1: 100
El valor obtenido es: 89.971048
Ingrese cantidad de iteraciones para experimento 2: 200
El valor obtenido es: 89.999999
Ingrese cantidad de iteraciones para experimento 3: 300
El valor obtenido es: 90.000000

La peor precisión se logra con 100 iteraciones
```

```
Ingrese el valor de x: 0.6
Ingrese cantidad de experimentos: 3
El valor esperado es: 3.750000
Ingrese cantidad de iteraciones para experimento 1: 100
```

```
El valor obtenido es: 3.750000
Ingrese cantidad de iteraciones para experimento 2: 200
El valor obtenido es: 3.750000
Ingrese cantidad de iteraciones para experimento 3: 300
El valor obtenido es: 3.750000

La peor precisión se logra con 300 iteraciones
```

FUNDAMENTOS DE PROGRAMACIÓN
ESTUDIOS GENERALES CIENCIAS
PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Referencias

[1] Anany Levitin. *Introduction to the design & analysis of algorithms.* Boston: Pearson,, 2012.

FUNDAMENTOS DE PROGRAMACIÓN
ESTUDIOS GENERALES CIENCIAS
PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ