

Claudia Martínez-Araneda

Libro digital

RESOLUCIÓN DE PROBLEMAS CON ADCP

Con implementación en C y Python



Agradecimientos

A mis hijos Lucas y Pedrito y a su padre Pedro, se merecen más que la mención en un libro de programación, por estar siempre junto a mí y soportar mi falta de paciencia.

Al Concurso Especial FAA Proyecto Ingeniería 2030 ING222010004 y al Proyecto InES Ciencia Abierta INCA210005 que financiaron la colaboración de estudiantes de Ingeniería Civil Informática de la UCSC en calidad de programadores, curadoras de datos e ilustrador.

A mi familia extendida y a la Chelita un beso al cielo.

La profe Claudia

Colaboraciones

Vicente Bastías, Ricardo Cartes, Ramón Cegarra, Nicolás Cereceda, Rodrigo Gómez, Brayan Nieto, Andrés Sánchez

Curadoras de datos

Monserrat Valderrama

Bárbara González

Ilustraciones

Daniel Puentes

Este libro digital se distribuye bajo la licencia

CC BY-NC-SA 4.0

Creative Commons Reconocimiento-NoComercial-CompartirIgual
4.0 Internacional



Citar como:

C. Martínez – Araneda, Resolución de problemas mediante estrategia ADCP – Con implementación en C y Python. Zenodo, 2024. doi: 10.5281/zenodo.14004719.

Contents

Introducción	6
ANÁLISIS-DISEÑO-CODIFICACIÓN y PRUEBAS – ADCP	7
Herramientas utilizadas y recursos recomendados	8
Parte A: Problemas desarrollados según ADCP.....	10
1 SumaQtesuma – dificultad: baja	10
Solución ADCP	10
2 Raíz i-ésima – dificultad: baja.....	15
Solución ADCP	15
3 Suma de Gauss – dificultad: baja	18
Solución ADCP	18
4 Adivina buen adivinador – dificultad: baja.....	22
Solución ADCP	22
5 Fibonacci – dificultad: baja.....	25
Solución ADCP	25
6 ¡Corredores listos fuera! - dificultad: baja.....	30
Solución ADCP	30
7 El hijo del medio - dificultad: media	33
Solución ADCP	33
8 Perrin – dificultad: media.....	36
Solución ADCP	36
9 Los cubos – dificultad: media.....	39
Solución ADCP	39
10 Sucesión la Bicicleta – dificultad: media	43
Solución ADCP	43
11 Euclides – dificultad: baja	46
Solución ADCP	46
12 Agua en botella – dificultad: baja.....	50
Solución ADCP	51
13 El medio <i>hot-dog</i> - dificultad: media.....	53
Solución ADCP	54
14 Los viajeros – dificultad: alta.....	58
Solución ADCP	58

15 Amigos – dificultad: alta.....	63
Solución ADCP	63
16 Problema clasificación de personajes de anime – dificultad: media	68
Solución ADCP	69
Parte B: Problemas propuestos	72
Problema fechas – dificultad: baja.....	72
Problema La AI en gestión de RR energéticos – dificultad: baja	73
Problema_SumaQTSuma++ – dificultad: baja	74
Problema Ayudando al abuelo Laíno – dificultad: baja.....	75
Problema Tropiconce – dificultad: baja.....	76
Problema Hottest Mountain – dificultad: baja	78
Problema del Panagrama – dificultad: media	78
Problema Batalla por la Galaxia – dificultad: media	79
Problema Emojis a la antigua – dificultad: alta	79
References.....	81
Anexo - Plantilla ADCP	82

Introducción

Bienvenido al interesante mundo de la resolución de problemas mediante la estrategia **ANÁLISIS-DISEÑO-CODIFICACIÓN-PRUEBAS** conocido por su acrónimo **ADCP** en el ámbito de la computación e informática. Este e-book está diseñado para aquellas personas, estudiantes o no, que desean adentrarse en la programación y desarrollar habilidades de pensamiento computacional, que incluye capacidad de análisis, razonamiento lógico, abstracción, capacidad de dividir para conquistar e identificación de patrones. No es un manual de Python ni C, pero se recomiendan unos tutoriales online muy completos.

En el mundo de la informática, la capacidad de resolver problemas de manera eficiente y estructurada es esencial. El método ADCP se convierte en una técnica fundamental para los estudiantes, ya que les permite reducir problemas complejos en otros más pequeños y manejables, facilitando así la creación de soluciones algorítmicas.

A lo largo de estas páginas, exploraremos la estrategia de **ADCP**, haciendo uso de dos lenguajes de programación fundamentales: C y Python. Estos lenguajes, aunque distintos en su sintaxis y orientaciones, comparten la capacidad de convertir ideas abstractas en soluciones concretas mediante la implementación de código.

Además, al abordar estos problemas, fomentaremos el desarrollo del pensamiento abstracto, matemático y lógico en el estudiante. La capacidad de plantear soluciones algorítmicas requiere habilidades analíticas y razonamiento lógico, las cuales serán cultivadas a lo largo de este e-book.

Prepárate para embarcarte en un viaje que te llevará desde la comprensión de un problema hasta la creación de soluciones algorítmicas implementadas en C y Python. A través de ejemplos, ejercicios y casos prácticos, te adentrarás en el emocionante mundo de la resolución de problemas utilizando la estrategia **ADCP**.

¡Que este libro sea tu guía para iniciarte en el fascinante universo de la programación y que aprendas cómo transformar problemas en soluciones algorítmicas!

ANÁLISIS-DISEÑO-CODIFICACIÓN Y PRUEBAS

La estrategia **ADCP** se basa en el enfoque tradicional usado en el Desarrollo de Software, conocido como el modelo en cascada o lineal. La estrategia fue concebida por Martinez & Muñoz (2017) con el propósito de guiar a sus estudiantes hacia un proceso sistemático para la resolución de problemas con enfoque centrado en el estudiante. En la Figura 1 se observa cada una de las etapas, qué se requiere como insumo y cuáles son sus salidas.

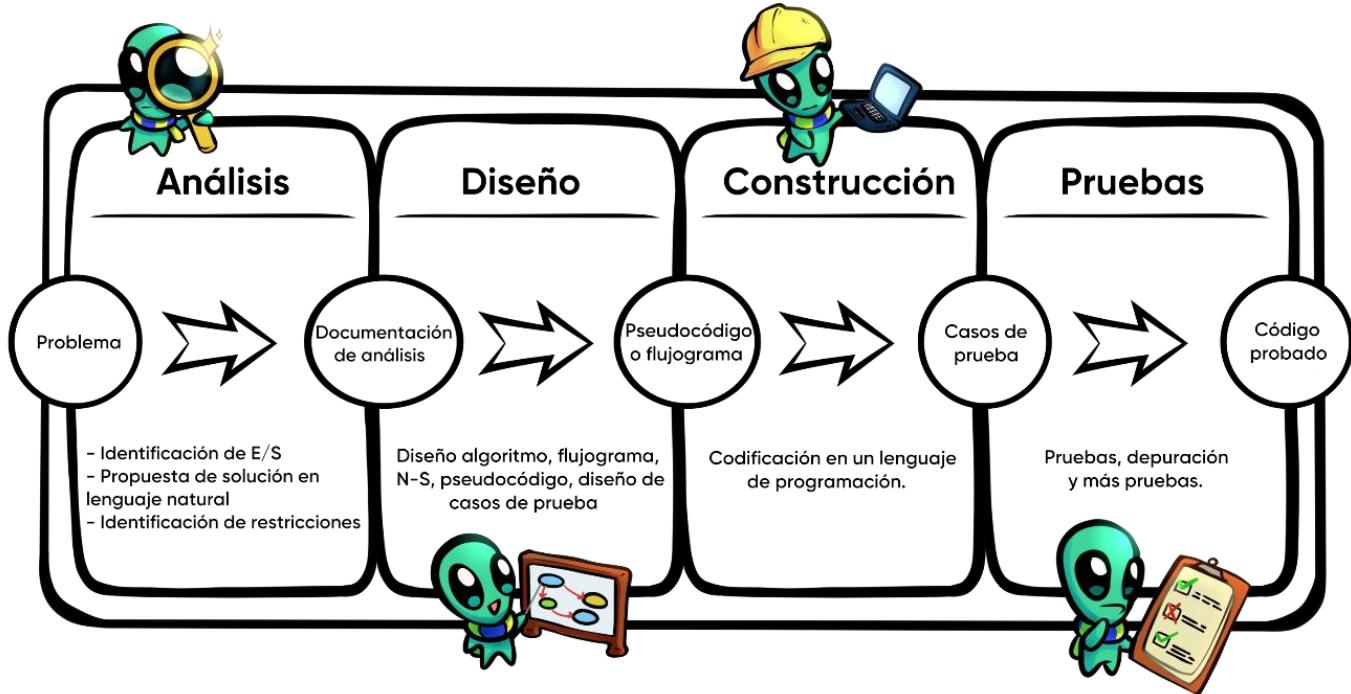


Figura 1. Estrategia **ADCP**.

Comienza con la fase de **ANÁLISIS** del problema, donde se busca extraer lo fundamental para identificar Entradas, Proceso mental, Salida y Restricciones. Posteriormente, se procede al **DISEÑO** de la solución, que puede plasmarse en diagramas de flujo (DF), diagrama de Nassi-Schneiderman (N-S), pseudocódigo (Pseudo) o bien en lenguaje natural (LN). Luego, continúa con la etapa de **CODIFICACIÓN** en un lenguaje de programación específico y concluye con la ejecución de las **PRUEBAS** para validar y verificar el funcionamiento del programa. A continuación, se muestra el desarrollo paso a paso de un set de problemas bajo la estrategia ADCP, con comentarios y procesos mentales donde la profe Claudia y sus ayudantes que te permitirán entender la problemática, diseñar, programar y probar una solución. También se propondrán otros tantos para el desarrollo del propio estudiante. El código fuente estará disponible en un repositorio público para quien lo quiera revisar. Opcionalmente, se puede emplear una plantilla de documentación que se encuentra disponible en el Anexo. Los problemas no tienen un orden específico y usan una etiqueta estilo semáforo que denota su dificultad (baja, media, alta), esta etiqueta tiene que ver con el grado de conocimiento matemático o de algún requerimiento

Resolución de problemas con ADCP – Con implementación en C y Python especial y no necesariamente tiene que ver con la complejidad algorítmica, concepto que revisarán en un siguiente curso.



Herramientas utilizadas y recursos recomendados

Entre las herramientas utilizadas para la confección de este e-book se incluyó a **Pseint**¹ que se distribuye bajo la licencia GNU GENERAL PUBLIC LICENSE v2, corresponde a un intérprete de Pseudocódigo en español muy recomendado para quienes inician en la programación. Por otra parte, también se utilizó **VSCode**² que corresponde a un IDE multilenguaje y multiplataforma que permite agregar numerosas extensiones como el compilador de C (gcc) disponible en MinGW-w64 desde <https://sourceforge.net/projects/mingw-w64/> o el intérprete de Python disponible en <https://www.python.org/downloads/>. También se utilizó un generador de documentación para código fuente como **Doxygen**³ que se agregó como extensión de **VSCode**. Además, se incorporó un repositorio⁴ complementario y público donde están alojadas las soluciones a los problemas del libro digital disponible en https://github.com/cmartinezUCSC/Resolucion_problemas_con_ADCP. Para la fase de diseño se aprovechó una funcionalidad de **Pseint** y además la herramienta Zinjal⁵ que corresponde a un IDE open-source para C/C++ que permite hacer ingeniería inversa, esto es, convertir el código fuente a algún diagrama de representación algorítmica. Dentro de los recursos recomendados están los tutoriales, manuales y websites especializados donde encontrar más problemas para practicar.

+ Info sobre Python

<https://www.w3schools.com/python/>
<https://www.geeksforgeeks.org/python-programming-language-tutorial/>

+ Info sobre C

<https://www.w3schools.com/c/index.php>
<https://www.geeksforgeeks.org/c-programming-language/>

+ Sobre cómo instalar compilador de C e intérprete Python y usarlos desde VScode

<https://youtu.be/mmKSS38d058>
<https://procodeguide.com/python-tutorials/install-python-with-vs-code/>

¹ <https://pseint.sourceforge.net/>

² <https://code.visualstudio.com/download>

³ <https://www.doxygen.nl/>

⁴ <https://github.com/>

⁵ <https://zinjal.sourceforge.net/>

+Web sites para practicar programación

<https://www.codechef.com>

<https://www.codeforces.com>

<https://www.exercism.com>

<https://pythontutor.com/>

El código fuente de cada solución a los problemas de este libro digital los encuentras en el repositorio complementario y público https://github.com/cmartinezUCSC/Resolucion_problemas_con_ADCP. También se incluye allí, un apartado **Extra** con nuevos ejercicios para practicar, incluyendo algunas pistas para orientar tu avance.

Parte A: Problemas desarrollados según ADCP



1 SumaQtesuma – dificultad: baja

A un estudiante de 1º básico se le dio como desafío calcular el producto de 2 números no negativos (m, n) con $m > 0$ y $n > 0$, a pesar de que el estudiante sólo sabe hacer sumas simples. Ayúdale a entender la multiplicación y a resolver el problema a partir de lo que sabe.

Entradas	Salida
2 8	16
5 11	55
0 12	ERROR debe ser >0
144 0	ERROR debe ser >0
-1 5	ERROR debe ser >0

Solución ADCP

ANÁLISIS

Entrada: 2 números enteros m, n

Proceso mental: Recuerden que en este apartado ustedes pueden incluir una oda, una fórmula, un mapa conceptual, un esquema, un gráfico, una secuencia numérica, un diagrama de eventos...un dibujo. Todo aquello que se relacione con sus experiencias de aprendizajes previas y que le ayude a entender el problema y a proponer una solución. Como el problema menciona a un estudiante de primero básico, que por currículum sólo sabe realizar sumas simples, debemos explicarle la operación multiplicación a través de la siguiente analogía:



Consideremos los casos:

$$m = 1, n = 3, m * n = 1 * 3 = 1 + 1 + 1 \text{ 3 veces}$$

$$m = 2, n = 4, m * n = 2 * 4 = 2 + 2 + 2 + 2 \text{ 4 veces}$$

$$m = 5, n = 10, m * n = 5 * 10 = 5 + 5 + 5 + \dots + 5 \text{ 10 veces}$$

Generalizando: $m * n = m + m + \dots + m$ n veces

Salida: $m * n$

Restricciones: $m > 0, n > 0$ (ambos $\neq 0$)

DISEÑO

En esta fase debiese incluirse la representación algorítmica en diagrama de flujo, Pseudocódigo, diagrama N-S o incluso un bloque en lenguaje natural semiestructurado. Por tratarse del primer ejemplo te mostraremos todas las alternativas, desde el segundo problema sólo se mostrará uno. A continuación, se mostrarán las diferencias entre las diversas técnicas de representación algorítmica.

Lenguaje natural

1. Recordar la declaración e inicialización de variables n, m y $suma = 0$ (dato que $suma$ actúa como acumulador).
2. Se debe solicitar m y n por teclado, una vez ingresado y comprobar que ambos sean mayores a 0, caso contrario, desplegar el mensaje “Error debe ser >0” y volver a solicitarlo hasta que cumpla la regla de validación.
3. Una vez ingresado y validado el número, se utilizará un ciclo repetitivo que vaya de 1 hasta n , asignándole al acumulador el valor de m , esto es:

$$suma = suma + m$$

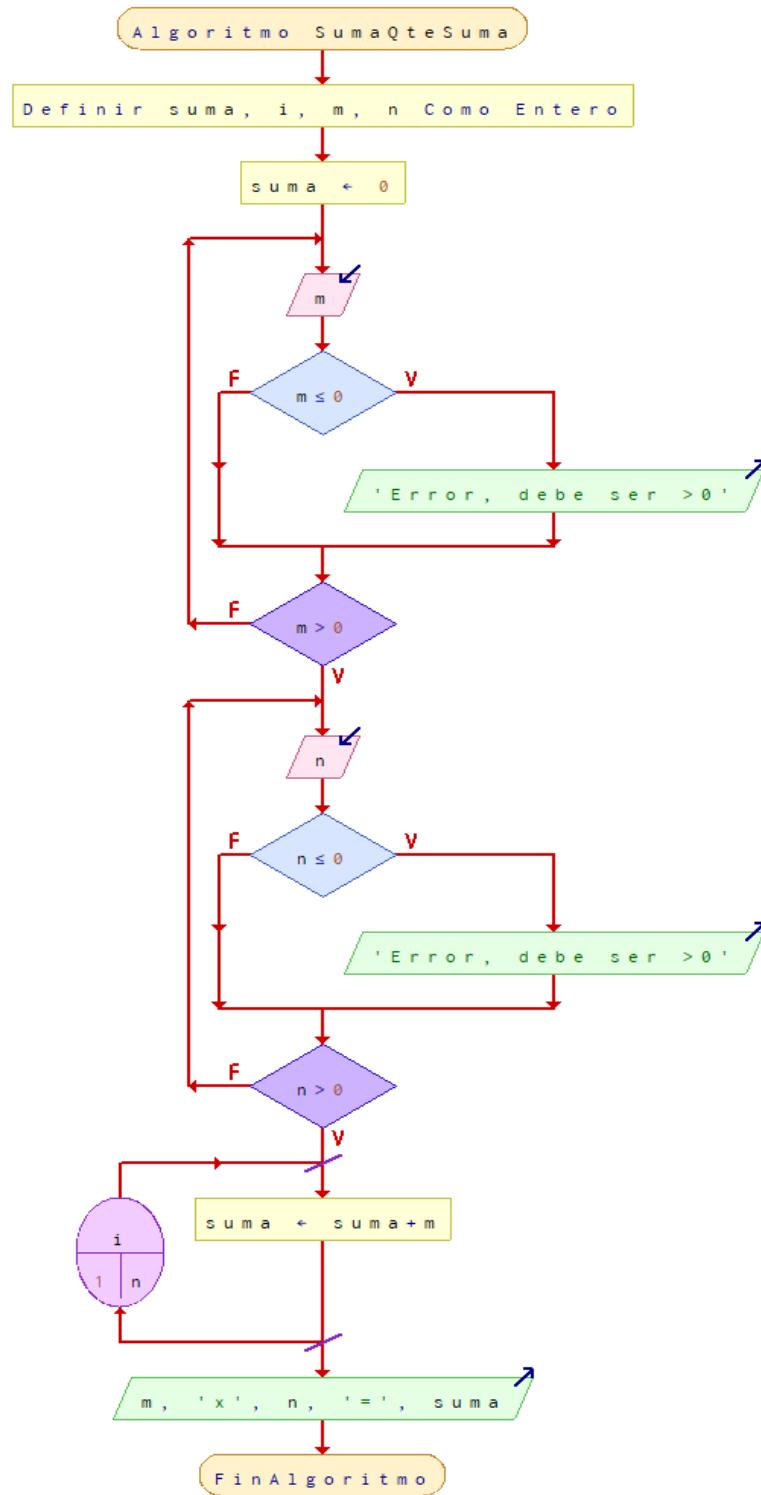
4. Finalmente, mostrar el resultado por pantalla: $m * n = suma$

Pseudocódigo (Pseint)

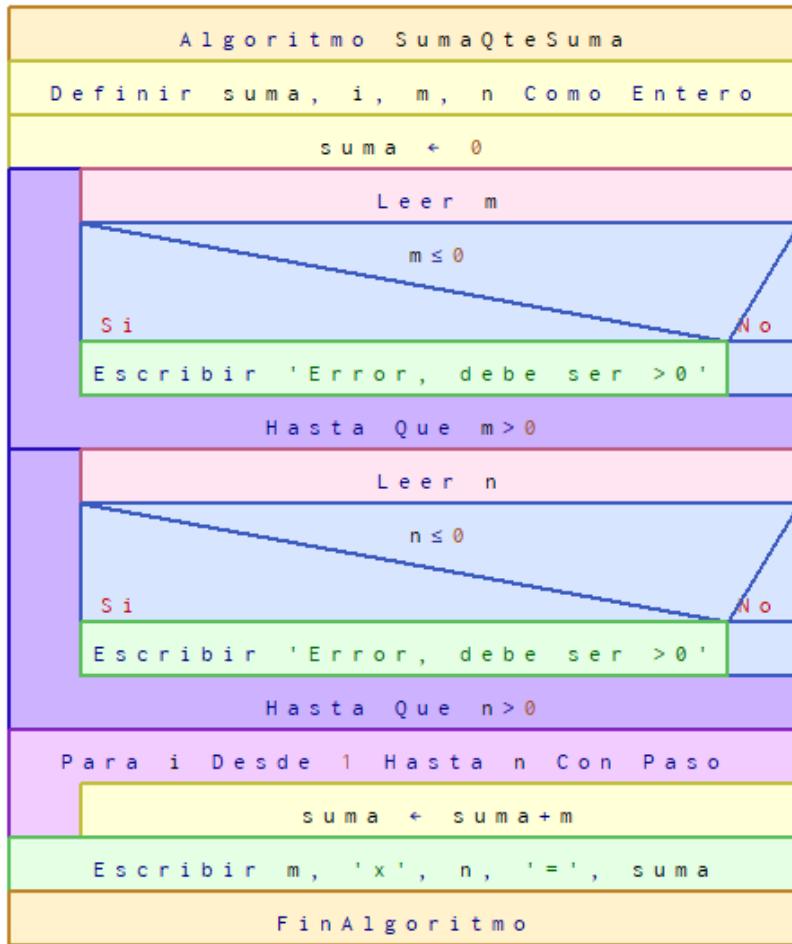
```

1 Algoritmo SumaQteSuma
2   definir suma, i, m, n como enteros
3   suma=0
4   Repetir
5     Leer m
6     si m≤0 entonces
7       escribir "Error, debe ser >0"
8       FinSi
9     Hasta que m>0
10    Repetir
11      Leer n
12      si n≤0 entonces
13        escribir "Error, debe ser >0"
14        FinSi
15      Hasta que n>0
16
17      Para i←1 hasta n Hacer
18        suma←suma+m
19      FinPara
20
21    Escribir m, "x", n, "=", suma
22 FinAlgoritmo

```

Diagrama de Flujo (Pseint)

Notación N-S (Pseint)



Para efectos de este ejemplo se han mostrado algunos de los diagramas utilizados para representar un algoritmo, sin embargo, en la práctica es necesario utilizar sólo uno. Se observa en el diagrama N-S es mucho más compacto que las otras 2 representaciones, aun cuando todas son equivalentes.

Para cada caso se inicia con un nombre del algoritmo y en términos generales debiera tener al menos una secuencia de Entrada(s)–Proceso–Salida(s).

CODIFICACIÓN



```

1 #include <stdio.h>
2 #include <stdlib.h>
3 int main()
4 {
5     int suma, i, n,m;
6     suma = 0;//inicialización del acumulador
7     do|
8     {
9         printf("Ingrese un numero mayor a 0\n");
10        scanf("%i", &m);//lectura entrada m
11        if (m <= 0)
12            printf("Error, debe ser>0\n");
13    } while (m <= 0); //validación para m>0
14    do
15    {
16        printf("Ingrese un numero mayor a 0\n");
17        scanf("%i", &n); //lectura entrada n
18        if (n <= 0)
19            printf("Error, debe ser>0\n");
20    } while (n <= 0); //validación para n>0
21    //ciclo de 1 a n para acumular m (m*n =m+m+m.. n veces
22    for (i = 1; i <= n; i++)
23        suma = suma + m;
24    //salida
25    printf("%i x %i = %i\n", m, n, suma);
26    return EXIT_SUCCESS;
27 }
```



```

1 def main():
2     suma = 0 # inicialización del acumulador
3
4     # Validación para m>0
5     while True:
6         m = int(input("Ingrese un numero mayor a 0\n"))
7         if m > 0:
8             break
9         else:
10            print("Error, debe ser>0")
11
12     # Validación para n>0
13     while True:
14         n = int(input("Ingrese un numero mayor a 0\n"))
15         if n > 0:
16             break
17         else:
18            print("Error, debe ser>0")
19
20     # Ciclo de 1 a n para acumular m (m * n = m + m + m... n veces)
21     for i in range(1, n + 1):
22         suma = suma + m
23
24     # Salida
25     print(f"{m} x {n} = {suma}")
26
27 if __name__ == "__main__":
28     main()
```

PRUEBAS

Entrada	Salida	Estado ⁶
2 8	16	😊
5 11	55	😊
0 12	ERROR debe ser >0	😊
144 0	ERROR debe ser >0	😊
-1 5	ERROR debe ser >0	😊

2 Raíz i -ésima – dificultad: baja

Se desea determinar la raíz i -ésima, $i > 0$ de un valor n ingresado. Para ello, el usuario ingresará 2 números, donde el primer término es el número al que se le calculará la raíz y el segundo el valor de i . El resultado de la operación debe generarse con 3 cifras decimales.

Entrada	Salida
1427 3	11.258
52538 2	229.212
144 0	ERROR, debe ser > 0
-27 3	Error, debe ser >= 0
0 3	0.000

Solución ADCP

ANÁLISIS

Entrada: Un número representando al radicando n y otro representando el recíproco del exponente i o sea $1/i$

Proceso mental: Se sabe que la raíz i -ésima de un número n se puede trabajar como potencia con exponente fraccionario. Por ejemplo:

$$\sqrt[2]{4} = 4^{1/2}$$

$$\sqrt[3]{8} = 8^{1/3}$$

$$\sqrt[4]{16} = 16^{1/4}$$

Generalizando,

$$\sqrt[i]{n} = n^{1/i} \quad \forall i > 0$$



Salida: La raíz i -ésima del número n

⁶ Smiley face 😊 representa que el código se ejecutó con éxito para el caso prueba, de lo contrario use disappointed face 😞.

Restricciones: n debe ser mayor que 0, para evitar raíces imaginarias e i estrictamente positivo para evitar la división por 0, $n \geq 0$ e $i > 0$

DISEÑO

En lenguaje natural

1. Definir 3 variables $n, i, raiz$.
2. Leer por teclado las variables de entrada n, i (no olvidarse de las restricciones).
3. Una vez leídos ambos valores, se hará el cálculo y se almacenará en $raiz$.
4. Finalmente, mostrar el resultado $raiz$ en pantalla.

CODIFICACIÓN



```

1 #include <stdio.h> //librería básica de C (entrada, salida, etc)
2 #include <math.h> //librería matemática que permite realizar operaciones
3 #include <stdlib.h>
4 int main()
5 {
6     //se define las 3 variables como flotante,
7     //reflexiona en lo adecuado de ésto en relación a línea 20
8     | float numero,i,raiz;
9     //vea las ventajas de la lectura separada dada la validación de c/u
10    do{
11        scanf("%f",&numero);
12        if(numero<0)printf("ERROR, debe ser >=0");
13        }while (numero<0);
14
15    do{
16        scanf("%f",&i);
17        if(i<=0)printf("ERROR, debe ser >0");
18        }while (i<=0);
19
20    //pow permite calcular potencias (base, exponente), en este caso numero elevado 1/i
21    raiz=pow(numero,(1/i));
22    //se muestra el resultado en pantalla con 3 decimales
23    printf("%.3f",raiz);
24    return EXIT_SUCCESS;
25 }
```



– versión 1

```

1 import math
2
3 def main():
4     # Se define las 3 variables como flotante,
5     # reflexiona en lo adecuado de esto en relación a la línea 20
6     numero, i, raiz = 0.0, 0.0, 0.0
7
8     # Vea las ventajas de la lectura separada dada la validación de cada uno
9     # y el manejo de excepciones de Python try-except)
10    while True:
11        try:
12            numero = float(input())
13            if numero < 0:
14                print("ERROR, debe ser >= 0")
15            else:
16                break
17        except ValueError:
18            print("ERROR, entrada no válida. Ingrese un número.")
19
20    while True:
21        try:
22            i = float(input())
23            if i <= 0:
24                print("ERROR, debe ser > 0")
25            else:
26                break
27        except ValueError:
28            print("ERROR, entrada no válida. Ingrese un número.")
29
30    # pow permite calcular potencias (base, exponente),
31    # en este caso numero elevado 1/i
32    raiz = numero ** (1 / i)
33
34    # se muestra el resultado en pantalla con 3 decimales
35    print("{:.3f}".format(raiz))
36
37 if __name__ == "__main__":
38     main()
39

```



– versión 2

```

1 import math
2 numero = float(input())
3 # Vea las ventajas de la lectura separada dada la validación de cada uno
4 while numero <0:
5     print("ERROR, debe ser >= 0")
6     numero = float(input())
7
8 i = float(input())
9 while i<=0:
10    print("ERROR, debe ser > 0")
11    i = float(input())
12 # pow permite calcular potencias (base, exponente),
13 # en este caso numero elevado 1/i
14 raiz = numero ** (1 / i)
15
16 # se muestra el resultado en pantalla con 3 decimales
17 print(f"{raiz:.3f}")
18

```

PRUEBAS

Entrada	Salida	Estado
1427 3	11.258	😊
52538 2	229.212	😊
144 0	ERROR, i debe ser > 0	😊
-27 3	ERROR, n debe ser ≥ 0	😊
0 3	0.000	😊

3 Suma de Gauss – dificultad: baja

Pedrito, un estudiante de 1º básico de un colegio local aprendió hace muy poco a sumar números enteros, sin embargo, se le complica cuando se le acaban los dedos de las manos para acumularlos y requiere la ayuda de un programador experto que acumule todos los enteros comprendidos entre 1 y n (ambos incluidos), con $n \geq 1$.

Entrada	Salida
3	6
2	3
5	15
-1	Error, entrada debe ser ≥ 1
0	0



Gauss Alien

Solución ADCP**ANÁLISIS**

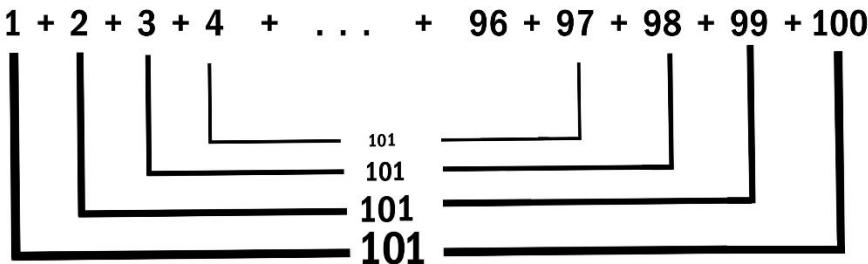
Entrada: cota superior (n)

Proceso mental: ¿Se deben sumar todos los números comprendidos entre 1 y n ? ¿Cómo se llama esta sumatoria? Mira el razonamiento que hizo Carl Friedrich Gauss, su suma es posible calcularla mediante una fórmula que equivale a sumar los primeros números naturales.



1. Observó que al sumar el primer número con el último, $1 + 100$, se obtiene 101.
2. También se obtiene 101 al sumar el segundo con el penúltimo, $2 + 99$, y así sucesivamente.
3. Multiplicó los 50 pares de números por 101, obteniendo el resultado de 5050 en menos de un minuto.

Observa como se representa en la figura:



Gauss Alien

$$\begin{aligned} &= 101 \times (100/2) \text{ parejas de números} \\ &= 5050 \end{aligned}$$

Así surge la fórmula $= n \cdot (n + 1) / 2$.

A continuación, se implementará la solución iterativa, esto es, sumando de 1...al enésimo valor, sin uso de la fórmula.

Casos particulares:

Para $n = 1$, se debería acumular 1

Para $n = 2$, se debería acumular: $1 + 2 = 3$

Para $n = 5$, se debería acumular: $1 + 2 + 3 + 4 + 5 = 15$

Generalizando,

Para $n = k$, se debería acumular: $1 + 2 + 3 + \dots + k$

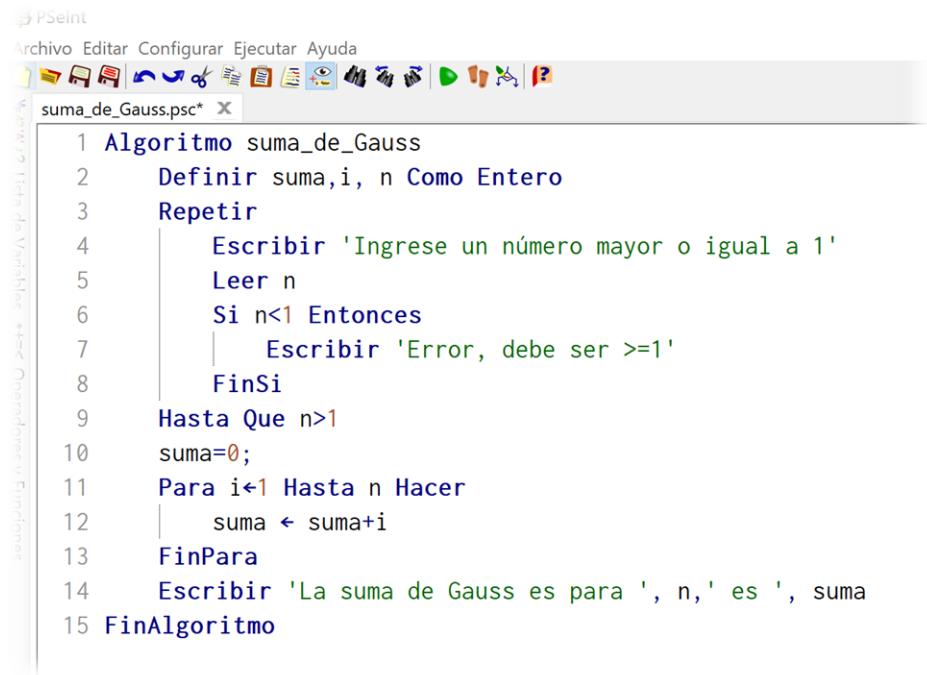
Salida: $1 + 2 + 3 + \dots + n$

Restricciones: $n \geq 1$

DISEÑO

1. Definir 3 variables (n , i , $sumar$ esultado).
2. Verificar que n sea mayor a 0.
3. Repetir n veces (dado que se conoce el número de iteraciones) incluyendo la instrucción
 $sumar$ esultado = $sumar$ esultado + i ; $\forall i \in [1 \dots n]$ actúa como iterador. La variable $sumar$ esultado actúa como acumulador, observe el efecto de estar a ambos lados de la asignación.
4. Finalmente, imprimir $sumar$ esultado

Y en pseudocódigo quedaría como la siguiente figura si usa *Pseint* (observe la diferencia con el lenguaje natural)



```

1 Algoritmo suma_de_Gauss
2   Definir suma,i, n Como Entero
3   Repetir
4     Escribir 'Ingrese un número mayor o igual a 1'
5     Leer n
6     Si n<1 Entonces
7       Escribir 'Error, debe ser >=1'
8     FinSi
9   Hasta Que n>1
10  suma=0;
11  Para i<1 Hasta n Hacer
12    suma ← suma+i
13  FinPara
14  Escribir 'La suma de Gauss es para ', n, ' es ', suma
15 FinAlgoritmo

```

CODIFICACIÓN



```

1 #include <stdio.h>
2 #include <stdlib.h>
3 int main()
4 {
5     int n, i, resultado;
6     resultado = 0; // inicialización
7     scanf("%d", &n); // lectura
8     // validación básica
9     if (n < 1)
10        printf("Error, debe ser >= 1");
11     else
12     { // ciclo repetitivo desde 1 hasta n
13         for (i = 1; i <= n; i++)
14         {
15             // acumulando i por cada iteración
16             resultado += i;
17         }
18         printf("%d", resultado); // muestra resultado
19     }
20     return EXIT_SUCCESS;
21 }

```



```

1 def main():
2     # Lectura
3     n = int(input("Ingrese un número: "))
4     resultado = 0 # Inicializar el resultado en 0
5
6     # Validación (n>=1)
7     if n < 1:
8         print("Error, debe ser >= 1")
9     else:
10        # Ciclo for para sumar los números desde 1 hasta n
11        for i in range(1, n + 1):
12            resultado += i # Acumular la suma en la variable resultado
13
14    # Imprimir el resultado final
15    print("La suma de los números desde 1 hasta {} es: {}".format(n, resultado))
16
17 if __name__ == "__main__":
18     main()

```

PRUEBAS

Entrada	Salida	Estado
-3	Error, debe ser >=1	😊
0	Error, debe ser >=1	😊
5	15	😊
10	55	😊

¡Consejo!

Después de completar este ejercicio recomiendo ver un extracto de cine matemático, la película *Midiendo el Mundo* (Buck, 2012) en <https://youtu.be/LpNHKkFSQII> para conocer la historia de Carl Frederic Gauss.

4 Adivina buen adivinador – dificultad: baja

El Jugador-virtual (**P**) piensa en un número comprendido en entre 1 y n . El jugador A trata de adivinarlo por tanteos sucesivos hasta llegar a él. Cada intento debe orientar al jugador A. Recuerde la existencia de una función `rand(n)` o `azar(n)` para generar números aleatorios entre 0 y $n - 1$.



Solución ADCP

ANÁLISIS

Entrada: Número n del jugador A por cada intento de adivinar.

Proceso mental: ¿Qué es un número aleatorio? ¿Cómo se genera? Un número aleatorio es un valor numérico que es seleccionado al azar o de manera impredecible a partir de un rango o conjunto de valores posibles. La generación de números aleatorios es fundamental en diversas áreas, como la estadística, la programación, la simulación, los juegos, entre otros campos. Estos números son utilizados para diversas aplicaciones, como la generación de datos para pruebas, criptografía, juegos, entre otros.

Es esencial que los números aleatorios sean impredecibles y equitativos, es decir, que tengan la misma probabilidad de ser seleccionados dentro de un rango específico. La aleatoriedad se logra utilizando algoritmos o procesos que generan secuencias de números que aparentemente no siguen un patrón discernible.

En la práctica, los números aleatorios se generan utilizando algoritmos de generación de números pseudoaleatorios en la mayoría de los computadores. Estos algoritmos producen secuencias de números que parecen ser aleatorios, pero en realidad son deterministas: si se inicia con la misma semilla (un valor inicial), se generará la misma secuencia de números.



Los lenguajes de programación suelen ofrecer funciones o librerías para generar números aleatorios (función `random` o `rand` o equivalente según el lenguaje de programación).

El jugador A debe intentar adivinar el número (`azar`) que se generó previamente, y orientar al usuario que tan cerca está de ese número hasta que lo encuentre o bien hasta que se le acaben las oportunidades, definir número de oportunidades.

Salida: Mensaje de éxito si adivina, mensaje de orientación si no, al agotar las oportunidades mensaje de fracaso.

Restricciones: El número n debe estar en el rango de generación de `random`. En lenguaje C, por ejemplo: `numero = rand(100) + 1`, secreto debiera estar entre 1 y 100.

DISEÑO

En lenguaje natural

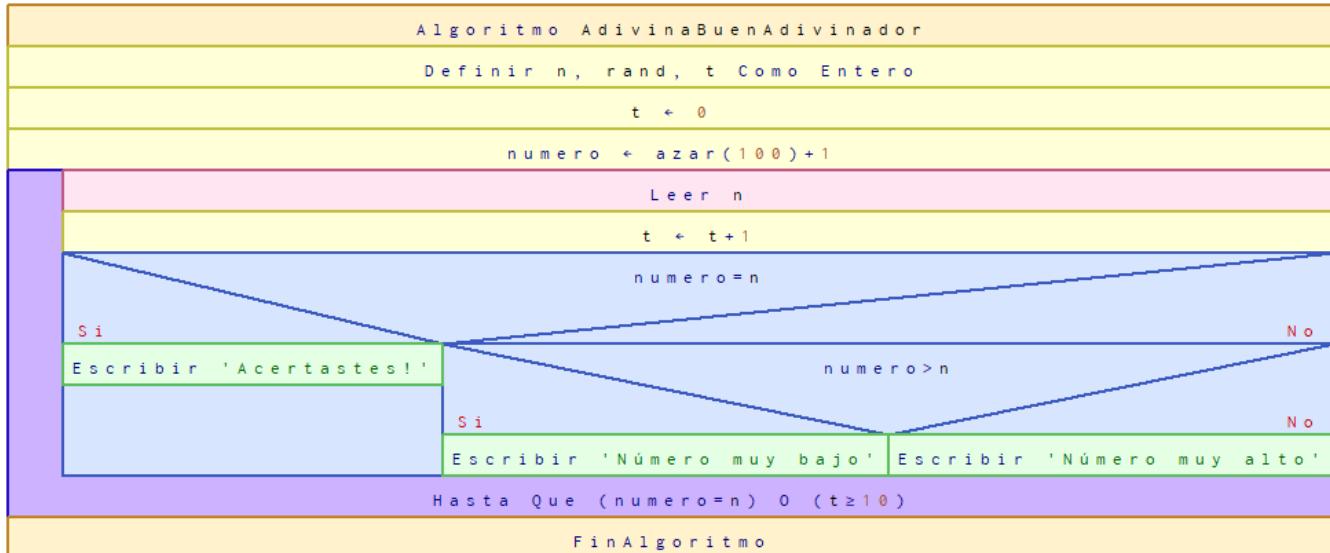
1. Se deben declarar variables (secreto, número).
2. La variable secreta se debe obtener mediante `numero = rand(100) + 1` o el equivalente según el lenguaje de programación.

Repetir

1. Solicitar al participante un número.
2. Si adivina se le reporta con el mensaje “Felicidades, ¡has adivinado!” y termina, de lo contrario comprobar si este número leído es mayor o menor que el secreto y orientar al usuario, al final se debe reportar “Fallo” en caso de no acertar.

10 intentos o hasta que acierte

En Diagrama N-S



CODIFICACIÓN



- versión 1

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 int main()
4 {
5     int numero,leido;
6     srand(time(NULL)); //genera una nueva semilla de números aleatorios
7     numero=rand()%100+1; //números entre 1 y 100
8     do
9     {
10         printf("Ingresa un número");
11         //se lee un número por teclado
12         scanf("%d",&leido);
13         //avisar si el numero ingresado es mayor que el generado al azar
14         if(numero<leido)printf("El número a adivinar menor que el ingresaste\n");
15         //avisar si el numero ingresado es menor que el generado al azar
16         if(numero>leido)printf("El número a adivinar es mayor que el que ingresaste\n");
17         /*vuelvo a repetir la lectura hasta que el número ingresado sea igual
18         al generado*/
19     }while(leido!=numero);
20     //se muestra mensaje de felicitaciones
21     printf("Felicidades, has adivinado!! ");
22     return EXIT_SUCCESS;
23 }
    
```



– versión 2

```

1 #include <stdio.h>
2 #include <time.h>
3 #include <stdlib.h>
4 int main()
5 {
6     int numero,leido,intentos,acerto;
7     srand(time(NULL)); //genera una nueva semilla de números aleatorios
8     numero=rand()%100 + 1; //números entre 1 y 100
9     intentos=10;
10    acerto=0;
11    do
12    {
13        do
14        {
15            printf("Ingresa un numero\n");
16            scanf("%d",&leido); //leo un número por teclado y válido
17            if (leido > 100 || leido<1) printf ("El número no está en el rango del juego");
18            }while (leido > 100 || leido<1);
19
20            //avisar si el número ingresado es mayor que el generado al azar
21            if(numero<leido)printf("El número a adivinar menor que el ingresaste\n");
22            //avisar si el numero ingresado es menor que el generado al azar
23            if(numero>leido)printf("EL número a adivinar es mayor que el que ingresaste\n");
24            if(leido==numero) acerto=1;
25            intentos--;
26        //vuelvo a repetir la lectura hasta que el número ingresado sea igual al generado
27    }while(acerto==0 && intentos>0); system("cls");
28
29    //muestro mensaje de felicitaciones cuando adivino el numero
30    if(acerto==1)printf("Felicidades, has adivinado en (%d) intentos!. El número es el (%d)",10-intentos,numero);
31    else printf("Lo siento, has fallado. El número generado al azar es el (%d)", numero);
32    system ("pause");
33    return EXIT_SUCCESS;
34 }
```



```

1 import random
2
3 def main():
4     numero = random.randint(1, 100) # Números entre 1 y 100
5     leido = 0
6
7     while leido != numero:
8         print("Ingresa un número:")
9         leido = int(input())
10
11         if numero < leido:
12             print("El número a adivinar es menor que el que ingresaste.")
13         elif numero > leido:
14             print("El número a adivinar es mayor que el que ingresaste.")
15
16         print("¡Felicidades, has adivinado!")
17
18     if __name__ == "__main__":
19         main()
```

PRUEBAS

Entrada (Número) ⁷	Salida	Estado
-1	El número no está en el rango de juego.	😊
54	El número por adivinar es mayor que el que ingresaste.	😊
100	El número por adivinar es menor que el que ingresaste.	😊
55	¡¡Felicidades, has adivinado!!	😊

5 Fibonacci – dificultad: baja

La sucesión de *Fibonacci* es la sucesión de números:

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, \dots$$

Cada número se calcula sumando los dos anteriores a él. Ejemplo:



El 2 se calcula sumando (1+1)

análogamente, el 3 es sólo (1+2),

y el 5 es (2+3),

¡y así!

Genere la sucesión de Fibonacci hasta el n -ésimo término de Fibonacci.

Solución ADCP

ANÁLISIS

Entrada: n

Proceso mental: Para explicar la secuencia se podría aplicar la siguiente función de recurrencia:

$$\begin{aligned} t_0 &= 0, t_1 = 1 \\ t_2 &= (t_0 + t_1) = 0 + 1 = 1 \\ t_3 &= (t_1 + t_2) = 1 + 1 = 2 \\ t_4 &= (t_2 + t_3) = 1 + 2 = 3 \end{aligned}$$

Por lo tanto, para obtener el n -ésimo término de Fibonacci, se tiene

$$t_n = t_{n-1} + t_{n-2}$$



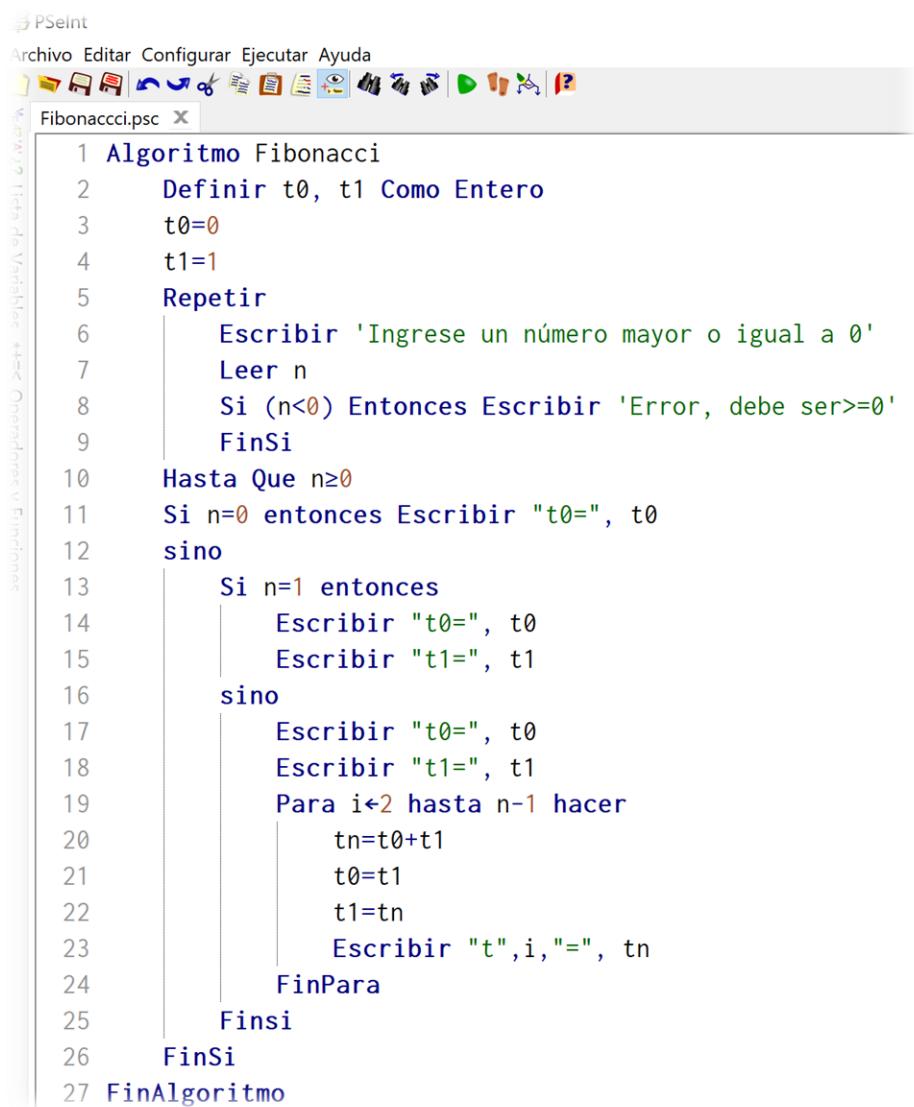
Salida: t_n

Restricción: $n \geq 0$

⁷Considerando que $numero = 55$

DISEÑO**En lenguaje natural**

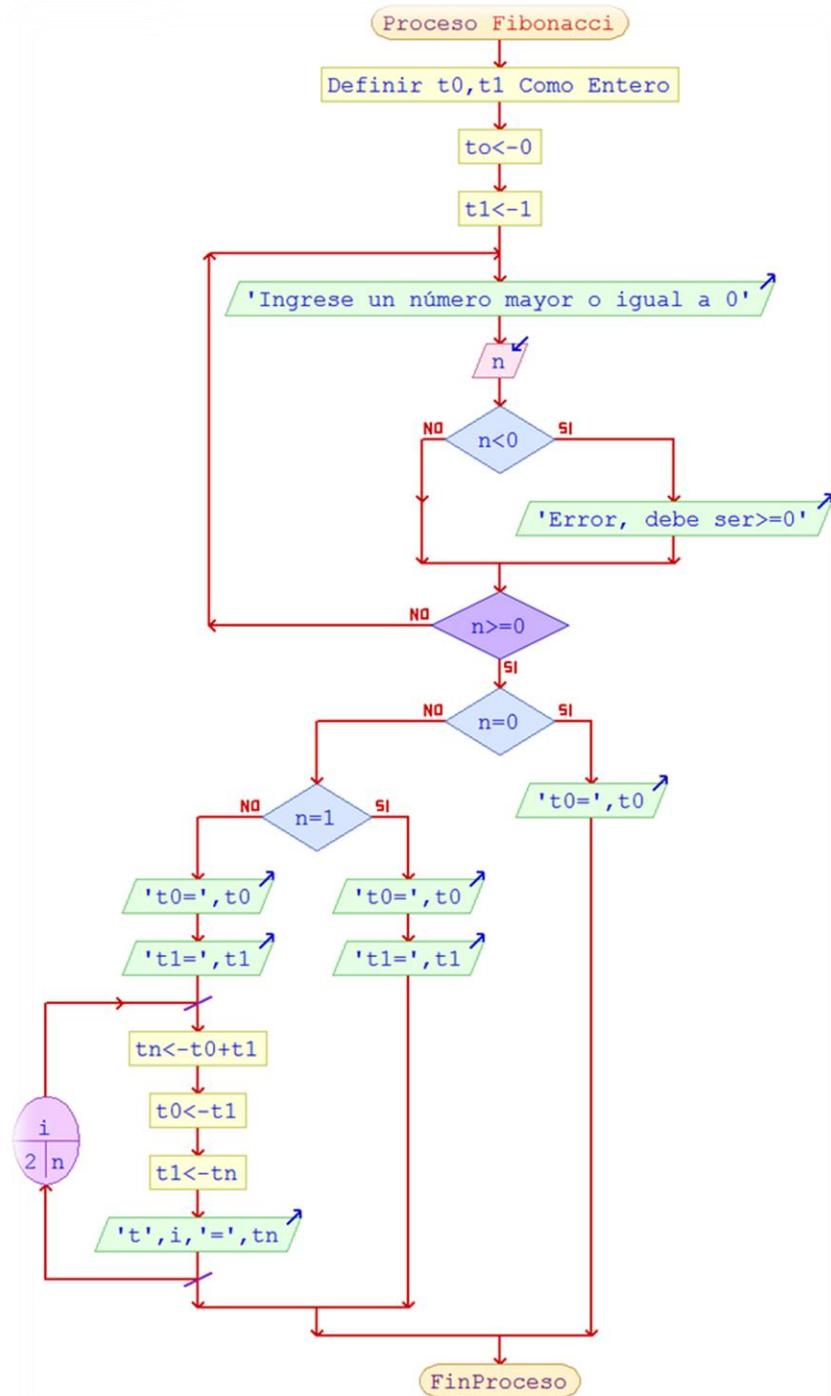
1. Como se sabe que los casos iniciales $t_0 = 0$ y $t_1 = 1$, debiéramos definir estas dos variables, en 0 y 1 respectivamente, también se debe definir un acumulador t_n (initializado en 0, $t_n = 0$).
2. Solicitar por teclado el valor de n y comprobar que sea mayor o igual a cero. Acá se pueden dar 3 casos:
3. Si $n = 0$ entonces simplemente mostrar la salida $t_0 = 0$
4. Si $n = 1$ entonces mostrar la salida $t_1 = 1$
5. Si $n > 1$ entonces se debe repetir (desde 1 hasta $n-1$), y por cada iteración darle al acumulador la suma de t_0 y t_1 , una vez realizada la suma, intercambiar t_0 por t_1 y t_1 por el acumulador t_n . Finalmente, mostrar el resultado en pantalla.

Pseudocódigo (Pseint)


```

PSeint
Archivo Editar Configurar Ejecutar Ayuda
Fibonacci.psc x
1 Algoritmo Fibonacci
2   Definir t0, t1 Como Entero
3   t0=0
4   t1=1
5   Repetir
6     Escribir 'Ingrese un número mayor o igual a 0'
7     Leer n
8     Si (n<0) Entonces Escribir 'Error, debe ser>=0'
9     FinSi
10    Hasta Que n≥0
11    Si n=0 entonces Escribir "t0=", t0
12    sino
13      Si n=1 entonces
14        Escribir "t0=", t0
15        Escribir "t1=", t1
16      sino
17        Escribir "t0=", t0
18        Escribir "t1=", t1
19        Para i←2 hasta n-1 hacer
20          tn=t0+t1
21          t0=t1
22          t1=tn
23          Escribir "t",i,"=", tn
24        FinPara
25      Finsi
26    FinSi
27 FinAlgoritmo

```

Diagrama de Flujo (Pseint)

Recuerde que la herramienta *Pseint* mencionada en las secciones previas puede generar el diagrama de flujo y el diagrama N-S a partir del pseudocódigo que se le entrega o viceversa.

CODIFICACIÓN



```
C Problema_Fibonacci.c > ...
1  #include<stdio.h>
2  #include<stdlib.h>
3  int main() {
4      int i,n, t0=0;
5      int t1=1, tn;
6      //lectura y validación del n (desde 0)
7      do {
8          printf("Ingrese n:");
9          scanf("%d",&n);
10         if (n<0) {
11             printf("Error, debe ser>=0\n");
12         }
13     } while (n<0);
14
15     if (n==0) {
16         printf("t0=%d\n",t0);//término 0
17     } else {
18         printf("t0=%d\n",t0);//término 0
19         printf("t1=%d\n",t1);//término 1
20         for (i=2;i<=n;i+=1) {//hasta término n
21             tn = t0+t1;//nuevo término
22             t0 = t1;//reasignación
23             t1 = tn;
24             printf("t%d=%d\n",i,tn);
25         }
26     }
27     return EXIT_SUCCESS;
28 }
```



```

ProblemaFibo.py > ...
1  def main():
2      n = -1
3
4      while n < 0:
5          try:
6              n = float(input("Ingrese un número mayor o igual a 0: "))
7              if n < 0:
8                  print("Error, debe ser >= 0")
9              except ValueError:
10                  print("Error, ingrese un número válido.")
11
12      t0 = 0
13      t1 = 1
14
15      if n == 0:
16          print(f"t0={t0}")
17      else:
18          print(f"t0={t0}")
19          print(f"t1={t1}")
20
21      for i in range(2, int(n) +1):
22          tn = t0 + t1
23          t0, t1 = t1, tn
24          print(f"t{i}={tn}")
25
26  if __name__ == "__main__":
27      main()

```

PRUEBAS

Entrada	Salida	Estado
0	0	😊
1	0 1	😊
2	0 1 1	😊
9	0 1 1 2 3 5 8 13 21 34	😊
-1	Error debe ser >=0	😊

¡Consejo!

Después de resolver el problema de generación de los números de *Fibonacci* revisa el video <https://youtu.be/8bCYiUIIF2k> te sorprenderás.

6 ¡Corredores listos fuera! - dificultad: baja



Los corredores del *Ironman* de Pucón, una bella ciudad al sur de Chile, requieren determinar la velocidad promedio (m/s) por competidor alcanzada en una de las rutas de 1500 metros. El registro incluye parejas de números (minutos: segundos) por cada corredor correspondiente al tiempo exacto que les tomó la ruta. El término de la lista de corredores está determinado porque la dupla minutos: segundos es 0:0.

Solución ADCP

ANÁLISIS



Entrada: Tiempo (minutos, segundos) que le toma a cada corredor completar la ruta.

Proceso mental: ¿Debería hacer la transformación de minutos a segundos? ¿Cómo lo hago?

¿Cómo obtener la velocidad? ¿Debería saber cuántos corredores participaron en la ruta?

Imaginemos un ejemplo en las entradas, minutos: segundos corresponde al tiempo que se demoraron en completar los 1500 metros un grupo de 4 participantes.

Entradas (minutos:segundos)	Salidas (m/s)
2:30	$1500(\text{m})/150(\text{s})=10 \text{ m/s}$
3:00	$1500(\text{m})/180(\text{s})=8.33 \text{ m/s}$
1:58	$1500(\text{m})/118(\text{s})=12.71 \text{ m/s}$
3:01	$1500(\text{m})/181(\text{s})=8.28 \text{ m/s}$

DISEÑO

Lenguaje natural

1. Definir las variables *minuto*, *segundo*, *promedio* y *velocidad*.
2. Leer variables *minuto* y *segundo* hasta ingresar un par 0 0.
3. Se valida que los tiempos sean válidos (*minuto*, *segundo* ≥ 0).
4. Si ingreso una pareja válida, se debe obtener su equivalente en segundos

$$\text{tiempo} = \text{segundos} + 60 * \text{minutos}$$

5. Obtenido el tiempo en *segundos*, calcular *velocidad* (longitud/tiempo) y mostrar en pantalla.

CODIFICACIÓN



```

C Corredores_listos.c ×
C Corredores_listos.c > ...
● 1 #include <stdio.h>
  2 #include <stdlib.h>
  3 int main()
  4 {
  5     int minutos,segundos,tiempo;
  6     float velocidad;
  7
  8     //leer datos mientras minutos y segundos sea distinto a 0
  9     while(1)
 10    {
 11        //Ler minutos, segundos hasta que estén un rango válido
 12        do
 13        {
 14            printf("Ingrese tiempos en mm ss: ");
 15            scanf("%d %d",&minutos,&segundos);
 16            if(minutos<0 || segundos<0)
 17                printf("\nError! entradas deben ser >=0. Reintente!\n ");
 18            }while(minutos<0 || segundos<0);
 19        //Evitar que al ingresar un 0 0 se haga el cálculo
 20        if(segundos==0 && minutos==0) break;
 21        else
 22        {
 23            // transformación de tiempo a segundos
 24            tiempo=segundos+(minutos*60);
 25            //calcular la velocidad
 26            velocidad=1500/(float)tiempo;
 27            printf("La velocidad promedio es de: %.2f m/s\n",velocidad);
 28        }
 29    }
 30    return EXIT_SUCCESS;
 31 }
 32

```



```

Corredores_listos.py ●

C: > Users > hp > Downloads > Corredores_listos.py > ...

1  def main():
2      while True:
3          try:
4              # Leer minutos y segundos mientras estén en un rango válido
5              minutos, segundos = map(int, input("Tiempo en mm ss (0 0 para salir):"))
6              if minutos < 0 or segundos < 0:
7                  print("\nError! Entradas deben ser >= 0. Reintente!\n")
8                  continue
9
10             # Salir si el usuario ingresa 0 0
11             if minutos == 0 and segundos == 0:
12                 print("Programa finalizado.")
13                 break
14
15             # Transformación de tiempo a segundos
16             tiempo = segundos + (minutos * 60)
17
18             # Calcular la velocidad
19             velocidad = 1500 / tiempo
20             print(f"La velocidad promedio es de: {velocidad:.2f} m/s\n")
21
22         except ValueError:
23             print("\nError! Ingrese 2 números separados por espacio.\n")
24
25     if __name__ == "__main__":
26         main()

```

PRUEBAS

Entrada (minutos: segundos)	Salida	Estado
2:30	La velocidad promedio es 10 m/s	😊
4:00	La velocidad promedio es 6.2 m/s	😊
-3:00	Entrada errónea. ¡Reintente!	😊
0:0	Cierre el programa	😊

7 El hijo del medio - dificultad: media

Se tienen tres números diferentes correspondientes a las edades de 3 hermanos de una familia chilena. Se debe determinar cuál es el hijo del medio.



Solución ADCP

ANÁLISIS

Entradas: 3 números (a, b, c) distintos

Proceso mental: Esto se puede ver como un problema de intervalos: Sólo basta con saber que entre tres números a, b, c , con $a \neq b$ y $a \neq c$, el del medio, corresponde al que esté entre dos números.

Escenarios posibles:

escenario 1: $b < a < c$

escenario 2: $c < a < b$

escenario 3: $a < b < c$

escenario 4: $c < b < a$

escenario 5: $a < c < b$

escenario 6: $b < c < a$

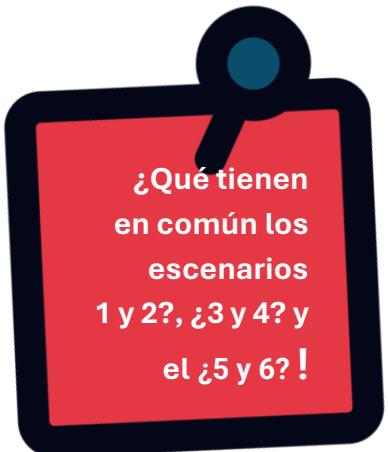
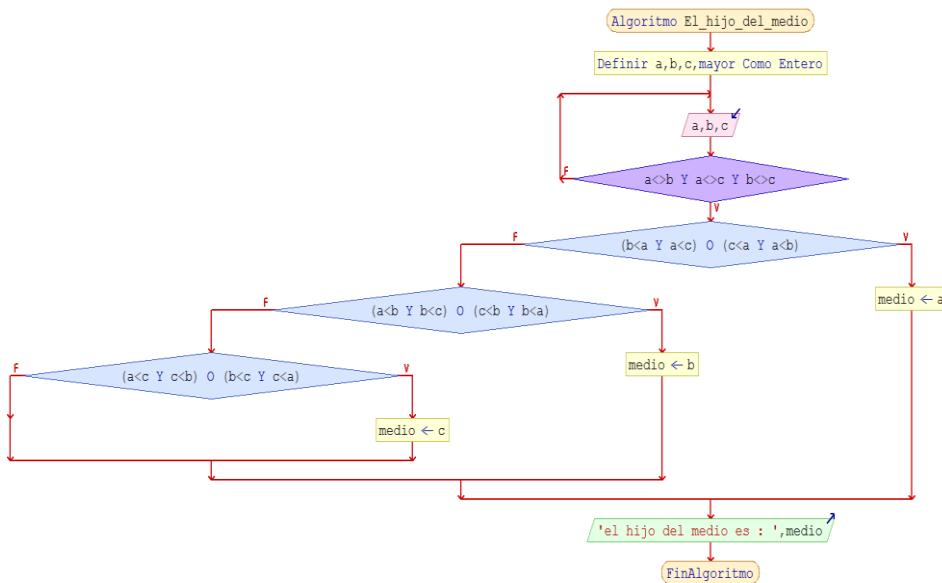
Salida: Valor del hijo del medio

Restricciones: edades distintas

DISEÑO (Lenguaje natural)

1. Definir 3 variables a, b y c
2. Leer cada una por teclado.
3. Validar que sean distintas (si no lo son, volver a leer)
4. Una vez leídas comprobar si cada variable se encuentra comprendida entre las dos restantes.

DISEÑO



CODIFICACIÓN



```

C Problema_HijodelMedio.c > ...
1 #include <stdio.h>
2 #include <stdlib.h>
3 int main()
4 {
5     int a,b,c, medio;
6     //Entrada y validación (todos deben ser no negativos distintos de 0)
7     do
8     {
9         printf ("Edades de los 3 hermanos? (cada una separada por un espacio)\n");
10        scanf("%d %d %d",&a,&b,&c);
11    }while(a<=0 || b<=0 || c<=0);
12
13    //Condicionando para establecer el del medio
14    //b<a<c OR c<a<b
15    if ((b<a && a<c) || (c<a && a<b)) {
16        medio = a;
17    } else {
18        //a<b<c OR c<b<a
19        if ((a<b && b<c) || (c<b && b<a)) {
20            medio = b;
21        } else {
22            //a<c<b OR b<c<a
23            if ((a<c && c<b) || (b<c && c<a)) {
24                medio = c;
25            }
26        }
27    }
28    //Salida
29    printf("el hijo del medio es : %i\n",medio);
30
31    return EXIT_SUCCESS;
32 }
```



(1/2)

```

1 def main():
2     a, b, c = 0, 0, 0
3     # Entrada y validación (todos deben ser no negativos distintos de 0)
4     while True:
5         try:
6             a, b, c = map(int, input("Edades de los 3 hermanos? (separados por un espacio): ").split())
7
8             if a > 0 and b > 0 and c > 0:
9                 break
10            else:
11                print("ERROR, todas las edades deben ser no negativas y distintas de 0.")
12        except ValueError:
13            print("ERROR, entrada no válida. Ingrese números enteros.")
14 
```



(2/2)

```

15 # Condicionando para establecer el del medio
16 # b < a < c OR c < a < b:
17 if (b < a < c) or (c < a < b):
18     medio = a
19 else:
20     # a < b < c OR c < b < a
21     if (a < b < c) or (c < b < a):
22         medio = b
23     else:
24         # a < c < b OR b < c < a
25         if (a < c < b) or (b < c < a):
26             medio = c
27
28 # Salida
29 print("El hijo del medio es: {}".format(medio))
30
31 if __name__ == "__main__":
32     main()

```

PRUEBAS

Entradas			Salida	Estado
a	b	c		
10	5	15	el hijo del medio es 10	😊
10	15	5	el hijo del medio es 10	😊
5	10	15	el hijo del medio es 10	😊
15	10	5	el hijo del medio es 10	😊
5	20	10	el hijo del medio es 10	😊
20	5	10	el hijo del medio es 10	😊

Observe en este caso se cambió la posición del hijo del medio para abordar los distintos casos de prueba.

8 Perrin – dificultad: media

En matemáticas, los **números de Perrín** están definidos por la relación de recurrencia

$$P_0 = 3, P_1 = 0, P_2 = 2 \dots \quad P_n = P_{n-2} + P_{n-3}, n > 2$$

La serie sería: 3, 0, 2, 3, 2, 5, 5, 7, 10, 12, 17, 22, 29, 39...



Determine el n -ésimo número de Perrín ($n \geq 0$).

Entrada	Salida
3	3
6	5
8	10
0	3
1	0

Solución ADPC

ANÁLISIS



Entrada: n

Proceso mental: Es una sucesión infinita de números naturales introducida por François Olivier Raoul Perrin en 1899. Tiene su origen en un trabajo de 1876 de Édouard Lucas. Para explicar la regla de generación de los números, primero se debe inicializar los valores de $p_0 = 3, p_1 = 0$ y $p_2 = 2$

$$p_n = p_{n-2} + p_{n-3}, \quad n > 2$$

Ejemplo

$$p_0 = 3, p_1 = 0, p_2 = 2$$

$$p_3 = (p_1 + p_0) = 0 + 3 = 3$$

$$p_4 = (p_2 + p_1) = 2 + 0 = 2$$

$$p_5 = (p_3 + p_2) = 3 + 2 = 5$$

$$p_6 = (p_4 + p_3) = 2 + 3 = 5$$

$$p_7 = (p_5 + p_4) = 5 + 2 = 7$$

Por lo tanto, para obtener el n -ésimo término de la serie se utilizará la expresión:

$$p_n = p(n-2) + p(n-3)$$

Salida: n -ésimo término p_n

Restricción: $n \geq 0$, (con ello se podrá obtener p_0, p_1 y p_2)

DISEÑO

1. Primero, declaramos e inicializamos algunas variables: n (el número en la secuencia que queremos encontrar), i (variable de control para el ciclo) y las variables para los tres valores iniciales p_0, p_1 y p_2 .

2. Luego, le pedimos al usuario que ingrese el valor de n .
3. A continuación, comprobamos si $n = 0, n = 1$ o bien $n = 2$, y se despliega el valor correspondiente de la secuencia ($p0, p1$ ó $p2$).
4. Después, con un ciclo repetitivo (*for*) que comienza desde $i = 3$ hasta $i = n$. En cada iteración, se determina el siguiente valor de la secuencia de Perrin, que es la suma de $p0$ y $p1$, y se almacena en la variable ***perrin***. Luego, actualizamos los valores de $p0, p1$ y $p2$ para la siguiente iteración.
5. Finalmente, se despliega la variable ***perrin*** en pantalla, que es el número de la secuencia de Perrin correspondiente al valor de n ingresado por el usuario.

CODIFICACIÓN



```
C ProblemaPerrin.c > ...
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      int n, i, p0=3, p1=0, p2=2, perrin;
7      do
8      {
9          printf("Ingrese n:"); // Entrada
10         scanf("%d", &n);
11         if (n < 0)
12             printf("Error debe ser >=0");
13     } while (n < 0);
14
15     // Salida primeros términos
16     if (n == 0)
17         printf("(%d)", p0);
18     else if (n == 1)
19         printf("(%d)", p1);
20     else if (n == 2)
21         printf("(%d)", p2);
22     else
23     {
24         if (n == 2)
25             printf("(%d)", p2);
26         // Términos desde p3 hasta pn
27         for (i = 3; i <= n; i++)
28         {
29             // Generación nuevo término
30             perrin = p1 + p0;
31             // Reasignación
32             p0 = p1;
33             p1 = p2;
34             p2 = perrin;
35         }
36         // Salida
37         printf("(%d)", perrin);
38     }
39     return EXIT_SUCCESS;
40 }
```



```

ProblemaPerrin.py > ...
1  def main():
2      #Inicialización primeros términos p0, p1 y p2
3      p0 = 3
4      p1 = 0
5      p2 = 2
6      n = int(input("Ingrese n: "))#Entrada
7      if n == 0:
8          print(f"({p0})")
9      elif n == 1:
10         print(f"({p1})")
11     elif n == 2:
12         print(f"({p2})")
13     else:
14         perrin = 0
15         for i in range(3, n + 1):
16             perrin = p1 + p0 #Generación nuevo término
17             #Reasignación
18             p0 = p1
19             p1 = p2
20             p2 = perrin
21
22         print(f"({perrin})")#Salida
23
24
25 if __name__ == "__main__":
26     main()

```

PRUEBAS

Entrada (n)	Salida (Perrin)	Estado
3	3	😊
6	5	😊
0	3	😊
1	0	😊
2	2	😊
8	10	😊

9 Los cubos – dificultad: media

Considera la siguiente propiedad descubierta por Nicómaco de Gerasa⁸:

“Sumando el primer impar, se obtiene el primer cubo;
Sumando los 2 siguientes impares, se obtiene el segundo cubo;
Sumando los 3 siguientes, se obtiene el tercer cubo, etc.”



Desarrolle un programa que lea el valor de n y calcule el n -ésimo cubo.

Solución ADCP

ANÁLISIS

Los cubos de Nicómaco consisten en la suma de los N números impares. Por ejemplo:

$$\begin{aligned} 1^3 &= 1 \\ 2^3 &= 3 + 5 = 8 \\ 3^3 &= 7 + 9 + 11 = 27 \\ 4^3 &= 13 + 15 + 17 + 19 = 64 \end{aligned}$$

Entrada: Un número n

Proceso mental: Se utilizará una estrategia interesante (entre muchas):



Casos particulares

Para $n = 1$, se considera el primer impar.

Para $n = 2$, Se consideran los 2 siguientes impares.

Para $n = 3$, considerar los 3 siguientes impares.

A partir de lo anterior, para obtener el cubo de n , se deben sumar n impares. Además, notar que, si se desea obtener el cubo de 2, se debe partir la suma desde el 2º término.

Para $n = 4$, partir desde el 7º término.

Para obtener la posición de inicio, simplemente restar n a la cota superior.

Ejemplo: $n = 4$, cota superior: 10, cota inferior: 6

Como ya se tiene la cota inferior, se forma el número impar que se genera en esta posición: 1, 3, 5, 7, 9, 11.

A partir del 11, se suma los 4 siguientes impares: 13, 15, 17, 19. Sumando ambos se obtiene 64, ¡Eureka!

Salida: n^3

Restricciones: $n > 0$

⁸ Vivió en Palestina siglo I y II de nuestra era. Escribió “Arithmetike eisagoge”, texto utilizado por más de 1000 años para enseñar aritmética.

DISEÑO

1. Sigue el diseño:
2. Verifica que n sea un número entero positivo mayor que cero.
3. Calcula la suma de los primeros N números impares.
4. Verifica si esta suma es un cubo perfecto.
5. Si la suma es un cubo perfecto, muestra el número n y la suma total obtenida.

CODIFICACIÓN



- versión 1

```

C ProblemaLosCubosver1.c > ...
1   #include <stdio.h>
2   #include <stdlib.h>
3   int main()
4   {
5       int n, superior, inferior, i, impar, cubo;
6       do
7       {
8           scanf("%d", &n); //Lectura
9           if (n <= 0)
10              printf("Ingrese un entero positivo\n");
11       } while (n <= 0); //Validación
12
13       impar = 1;
14       // cota superior
15       superior = (n * (n + 1)) / 2;
16       // cota inferior, ya que la suma de impares se hace n veces
17       inferior = (superior - n);
18       // posición de partida
19       printf("cota inferior es: %d\n", inferior);
20       // posición de partida
21       printf("cota superior es: %d\n", superior);
22       // el impar base (del cual voy a partir)
23       for (i = 0; i < inferior; i++)
24       {
25           impar += 2;
26       }
27       cubo = 0;
28       for (i = 0; i < n; i++)
29       {
30           cubo += impar;
31           impar += 2;
32       }
33       printf("%d al cubo es %d", n, cubo);
34       return EXIT_SUCCESS;
35   }

```



- versión 2

```
C ProblemaLosCubosver2.c > ...
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 /*Fórmula mágica
5  para n=2 -> 2*(2-1)+1=2*1+1=3
6  para n=3 -> 3*(3-1)+1=3*2+1=7
7  para n=4 -> 4*(4-1)+1=4*3+1=13
8  y para n=k? -->k*(k-1)+1
9 */
10 int main()
11 {
12     int n, cubo, impar, i;
13     cubo = 0;
14     //Lectura y validación
15     do
16     {
17         scanf("%d", &n);
18         if (n <= 0)
19             printf("Ingrese un entero positivo\n");
20     } while (n <= 0);
21
22     //usando la fórmula mágica
23     impar = (n * (n - 1)) + 1;
24
25     for (i = 0; i < n; i++)
26     {
27         cubo += impar;
28         impar += 2;
29     }
30     printf("el cubo es: (%d)", cubo);
31
32     return EXIT_SUCCESS;
33 }
```



– versión 3

```
C ProblemaLosCubosver3.c > ...
1 #include <stdio.h>
2 #include <stdlib.h>
3 int main()
4 {
5     int n, i, suma, cubo;
6     // Entrada
7     do
8     {
9         scanf("%d", &n);
10        if (n <= 0)
11            printf("Ingrese un entero positivo\n");
12    } while (n <= 0);
13    // Valor inicial acumulador
14    suma = (n * (n + 1)) / 2;
15    cubo = 0;
16    // Considerando el valor inicial y condición dentro del for
17    for (i = (((suma - 1) * 2) + 1); i >= ((suma - n)) * 2 + 1; i -= 2)
18        cubo += i;
19    printf("(%d)", cubo);
20    return EXIT_SUCCESS;
21 }
```



– a partir de la versión 2

```
🐍 ProblemaLosCubos.py > ...
1 #Fórmula mágica
2 #para n=2 -> 2*(2-1)+1=2*1+1=3
3 #para n=3 -> 3*(3-1)+1=3*2+1=7
4 # para n=4 -> 4*(4-1)+1=4*3+1=13
5 #y para n=k? -->k*(k-1)+1
6 n = -1
7 while n <= 0:
8     #Entrada y validación
9     n = int(input("Ingrese un entero positivo: "))
10    if n <= 0:
11        print("Debe ser positivo!")
12
13    cubo = 0
14    #Uso la fórmula mágica
15    impar = (n * (n - 1)) + 1
16
17    for i in range(n):
18        cubo += impar
19        impar += 2
20    #Salida
21    print(f"El cubo es: ({cubo})")
```

PRUEBAS

Entrada	Salida	Estado
0	Ingrese un número >0	😊
25	15625	😊
33	35937	😊
-1	Ingrese un número >0	😊

10 Sucesión la Bicicleta – dificultad: media

$$b_n = \frac{b_{n-1} + 1}{b_{n-2}}, \quad \forall n \geq 3$$

Esta sucesión se denomina así ya que para avanzar a través de sus términos basta con hacer "rodar" dos consecutivos. Por ejemplo, si tomamos los primeros términos de la sucesión 2 y 3 se obtiene que el siguiente es 2, sorprendentemente la generación es cíclica:

$$2, 3, 2, 1, 1, 2, 3, \dots$$



Dado un $n \geq 3$, construya un programa que genere los primeros n términos de la Sucesión Bicicleta.

Solución ADCP

ANÁLISIS



Entradas: Entero n

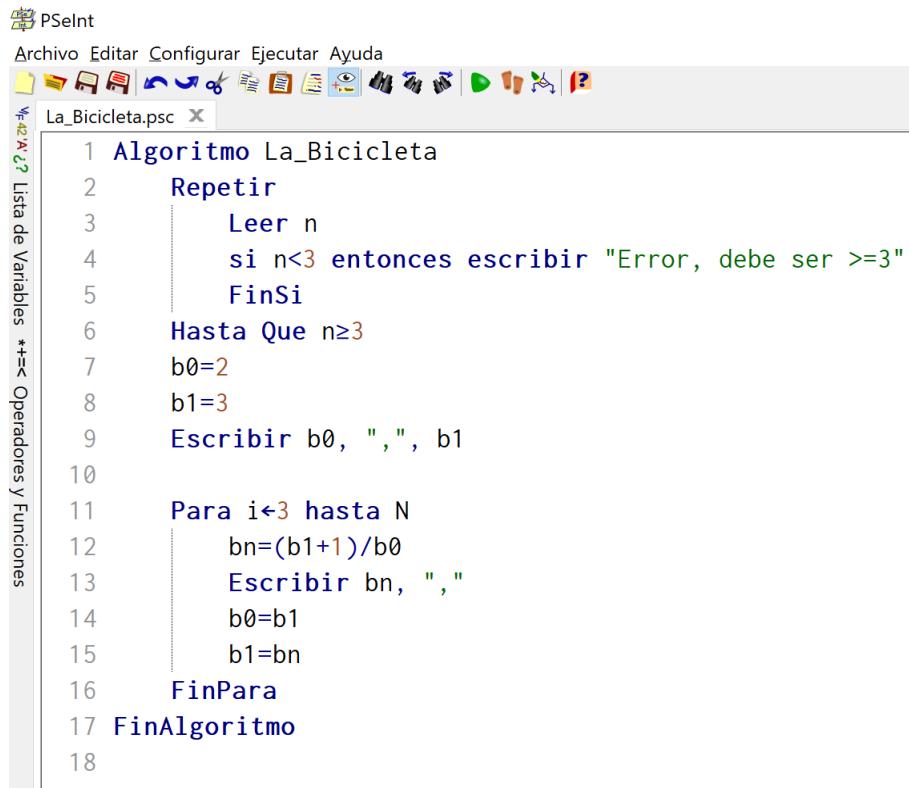
Proceso mental: Tal como en sucesión de Fibonacci, el término n -ésimo de la sucesión de la bicicleta se obtiene a partir de los 2 términos anteriores, es por ésto por lo que es necesario partir con los términos b_0 y b_1 :

$$b_n = \frac{b_{n-1} + 1}{b_{n-2}}, \quad \forall n \geq 3$$

Salidas: N primeros términos de la sucesión de la bicicleta ($b_0, b_1, b_2 \dots b_n$)

Restricciones: $n \geq 3$

DISEÑO



The screenshot shows the PSeInt software interface with the file 'La_Bicicleta.psc' open. The pseudocode is as follows:

```

1 Algoritmo La_Bicicleta
2   Repetir
3     Leer n
4     si n<3 entonces escribir "Error, debe ser >=3"
5     FinSi
6     Hasta Que n≥3
7     b0=2
8     b1=3
9     Escribir b0, ",", b1
10
11    Para i<3 hasta N
12      bn=(b1+1)/b0
13      Escribir bn, ","
14      b0=b1
15      b1=bn
16    FinPara
17 FinAlgoritmo
18

```

On the left side of the window, there are toolbars for File, Edit, Configure, Execute, Help, and various icons for file operations. A vertical sidebar on the left contains buttons for 'Lista de Variables', 'Operadores y Funciones', and 'Ayuda'.

CODIFICACIÓN



```

C ProblemaBicicleta.c > ...
1 #include <stdio.h>
2 #include <stdlib.h>
3 int main()
4 {
5     float b0 = 2, b1 = 3, bn;
6     int i, n;
7     do
8     {
9         scanf("%d", &n); // Entrada n
10        if (n < 3)
11        {
12            printf("Error, debe ser >=3\n");
13        }
14    } while (n < 3); // validación entrada mínima
15
16    printf("%3.2f\n%3.2f\n", b0, b1);
17    // n-2 términos de la secuencia dado que se imprimieron 2 fuera
18    for (i = 1; i <= n - 2; i += 1)
19    {
20        // Generando nuevo término
21        bn = (b1 + 1) / b0;
22        printf("%3.2f\n", bn);
23        // Reasignación para la siguiente iteración
24        b0 = b1;
25        b1 = bn;
26    }
27    return EXIT_SUCCESS;
28 }

```



```
ProblemaBicicleta.py > ...
1  def main():
2      b0, b1, bn = 2.0, 3.0, 0.0
3      i, n = 0, 0
4
5      while n<3:
6          n = int(input("Ingrese un número (debe ser >= 3): ")) # Entrada n
7          if n >= 3: #Validación
8              break
9          else:
10             print("Error, debe ser >= 3")
11
12     b0, b1 = 2.0, 3.0
13     print("{}\n{}".format(b0, b1))
14
15     # n-2 términos de la secuencia dado que se imprimieron 2 fuera
16     for i in range(1, n - 1):
17         # Generando nuevo término
18         bn = (b1 + 1) / b0
19         print("{}".format(bn))
20         # Reasignación para la siguiente iteración
21         b0, b1 = b1, bn
22
23 if __name__ == "__main__":
24     main()
```

PRUEBAS

Entrada	Salida	Estado
0	Error debe ser >=3	😊
3	2, 3, 2	😊
7	2, 3, 2, 1, 1, 2, 3...	😊
-1	Error debe ser >=3	😊

11 Euclides – dificultad: baja



Obtener el máximo común divisor de 2 números usando el algoritmo de Euclides.

Solución ADCP

ANÁLISIS

Entradas: 2 números enteros (m, n)

Proceso mental: ¿Euclides⁹? ¿máximo común divisor? El máximo común divisor (MCD) es el mayor número entero que divide exactamente a dos o más números enteros sin dejar residuo. En otras palabras, es el número más grande que es un divisor común a todos los números dados.

El MCD es una noción importante en la teoría de números y tiene varias aplicaciones en matemáticas y ciencias computacionales, como en simplificación de fracciones, criptografía, algoritmos de optimización, entre otros. El algoritmo de Euclides es un método antiguo y eficaz para calcular el MCD.



¿Qué es la divisibilidad? se dice que m es divisible por i si ($m \bmod i = 0$) o sea no queda residuo después de dividir.

Salidas: MCD

Restricciones: Ninguna

DISEÑO (Pseudocódigo – Pseint)

```

PSeint
Archivo Editar Configurar Ejecutar Ayuda
Euclides.psc x
1 Algoritmo Euclides
2   leer m, n
3   Si n≤m Entonces
4     final=n
5   Sino
6     final=m
7   FinSi
8   Para i=1 hasta final hacer //excluir el 1
9     Si ((n MOD i = 0) y (m MOD i=0)) entonces
10       maximo=i;
11     FinSi
12   FinPara
13   Escribir "Máximo comun: ", maximo;
14 FinAlgoritmo
15

```

¿Cómo se describiría en lenguaje natural?

1. Leer m, n
2. Verificar cuál de los 2 números es mayor, para definir la cota superior del divisor
3. Calcular el resto de la división de m y n por el iterador i hasta que este llegue a la cota superior
4. Guardar el valor de i en caso de cumplir condición de divisibilidad para m y simultáneamente para n

⁹ Euclides nació en Alejandría alrededor del 330 AC. Se le conoce como "El Padre de la Geometría".

5. Imprimir resultado

CODIFICACIÓN

– versión 1

```

C ProblemaEuclidesver1.c > ...
1   #include <stdio.h>
2   #include <stdlib.h>
3
4   int main()
5   {
6       int final, i, m, n, maximo;
7       //Entradas
8       scanf("%d", &m);
9       scanf("%d", &n);
10      //Establecer el mayor
11      if (n <= m)
12      {
13          final = n;
14      }
15      else
16      {
17          final = m;
18      }
19      //Verificación de divisibilidades
20      for (i = 1; i <= final; i += 1)
21      {
22          if ((n % i == 0 && m % i == 0))
23          {
24              maximo = i;
25          }
26      }
27      printf("Máximo común: %d\n", maximo);
28      return EXIT_SUCCESS;
29 }
```



– versión 2

```
C ProblemaEuclidesver2.c > ...
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     int m, n, mod;
7     scanf("%d %d", &m, &n); //Entrada
8     //Verificando si es una de las entradas
9     if (m % n == 0)
10    {
11        if (m < n)
12            printf("%d\n", m);
13        if (n < m)
14            printf("%d\n", n);
15    }
16    else
17    { //Detectando divisibilidades
18        while (m % n != 0)
19        {
20            mod = m % n; //calculando el resto
21            m = n;
22            n = mod;
23        }
24
25        printf("%d", mod); //Salida
26    }
27    return EXIT_SUCCESS;
28 }
```



```

Problema_Euclides.py > main
1 def main():
2     m, n=0, 0
3     while True:
4         try:#Lectura multiple de 2 enteros separados por " " (por defecto split())
5             m, n = map(int, input("Ingrese dos números separados por espacio: ").split())
6
7             if m != 0 and n != 0: #validación
8                 if m % n == 0:#en caso de m/n tener residuo cero
9                     if m < n:# el menor es el MCM
10                        print(m) # salida caso particular
11                     if n < m:# el menor es el MCM
12                        print(n) # salida caso particular
13                 else:
14                     while m % n != 0:
15                         mod = m % n #calculo del residuo
16                         m = n
17                         n = mod
18                         print(mod)#salida caso general
19                     break
20                 else:
21                     print("ERROR, deben ser != 0")#mensaje en caso de falla
22             except ValueError:
23                 print("ERROR, entrada no válida. Ingrese números enteros.")
24
25             if __name__ == "__main__":
26                 main()

```

PRUEBAS

Entrada	Salida	Estado
3 4	1	😊
12 3	3	😊
24 14	2	😊
21 7	7	😊

12 Agua en botella – dificultad: baja

En algún lugar del mundo existen n personas en una fila para obtener agua, la i -ésima quiere llenar su botella de c_i litros. Cuando se acaba el balde de agua del dispensador de agua (cada balde de agua proporciona C litros de agua), quien esté utilizando el dispensador de agua (incluido el último) sustituirá el balde del dispensador por un balde nuevo o agua, incluso si tiene suficiente agua. Luego se va inmediatamente, por lo que uno puede irse con menos agua de la que quiere. Ahora se requiere conocer cuántos baldes de agua se necesitan (incluido el primer balde).



Entrada

La primera línea contiene un único número entero T ($1 \leq T \leq 100$) que corresponde al número de casos de prueba.

Cada caso de prueba contiene dos líneas.

La primera línea contiene dos números enteros n, C ($1 \leq n \leq 106, 1 \leq C \leq 103$)

La segunda línea contiene n números enteros c_i ($1 \leq c_i \leq 103$), $i \in [1, n]$

```
4
3 1
6 4 5
4 1
1 6 4 5
2 10
5 2
5 10
8 6 7 10 2
```

Salida

Para cada caso de prueba, imprima un número entero positivo que represente la cantidad de baldes necesarios.

```
4
5
1
3
```

Fuente: Codeforces 2021 (adaptación del Water problem)

Solución ADCP

ANÁLISIS



Entradas: T : número de casos de prueba, n : número de personas, C : capacidad en litros del balde, c_i : requerimientos de agua de cada persona, $i = 1, n$

Salidas: B : cantidad de baldes de agua necesarios para cubrir necesidades.

Proceso: Podemos centrarnos en hacer una pregunta sencilla, ¿queda suficiente agua en el balde para llenar algo de la botella de la persona?, o dicho como una diferencia $(C - Ci) > 0$, si eso se cumple entonces el balde cubre necesidad y se debe reducir según requerimiento, pero si no se cubre toda la botella se necesitará otro balde, aun cuando sobre agua.

Consideremos un ejemplo, $n = 4, C = 10, Ci = [2,6,7,10], i \in [1, 4]$

Litros remanentes (C)	Litros requeridos por persona (C_i)	$C - Ci$	$C - Ci < 0$	Baldes usados (B)
10	2	8	no	1
8	6	2	no	1
2	7	-5	si	2
10	10	0	si	3

DISEÑO

PSelnt

Archivo Editar Configurar Ejecutar Ayuda

La_Bicicleta.psc* X

```

1 ALgoritmo Agua_en_botella
2   Dimensionar C(MAX)
3   Leer T //Números de casos de prueba
4   Para j=1 hasta T Hacer //Por cada caso de prueba
5     Leer Cap, n //Capacidad por balde, cantidad de personas
6     C_uso=Cap
7     Baldes=1
8     Para i=1 hasta n hacer
9       Leer C[i] //Lectura de los requerimientos de agua por persona
10      FinPara
11      Para i=1 hasta n hacer
12        C_uso = C_uso-C[i] //Descontando los requerimientos
13        //individuales de la capacidad
14        SI C_uso ≤ 0 //Si se supera la disponibilidad se pasa a otro balde
15          Baldes=Baldes+1
16          C_uso = Cap //se reinicia con la capacidad completa
17          FINSI
18        FinPara
19        Escribir (Baldes)
20      FinPara
21  FinAlgoritmo
22

```

CODIFICACIÓN



```

1 #include <stdio.h>
2 #include <stdlib.h>
3 int main() {
4     int T; // Número de casos de prueba
5     scanf("%d", &T); // Entrada de T
6
7     for (int j = 0; j < T; j++) {
8         int n, c; // Número de elementos y capacidad del balde
9         scanf("%d %d", &n, &c); // Entrada de número persoans (n) y capacidad de balde (c)
10
11        int C_uso = c; // Variable para rastrear la capacidad actual en uso
12        int baldes = 1; // Inicializar el contador de baldes en 1 (contar el inicial)
13
14        // Crear un arrglo para almacenar las capacidades de cada balde
15        int Cn[n];
16        for (int i = 0; i < n; i++) {
17            scanf("%d", &Cn[i]); // Entrada requerimiento de agua
18        }
19
20        // Iterar sobre las capacidades de los baldes
21        for (int i = 0; i < n; i++) {
22            C_uso -= Cn[i]; // Restar al balde el requerimiento actual
23            if (C_uso <= 0) {
24                baldes++; // Incrementar el contador de baldes si la capacidad se agota
25                C_uso = c; // Reiniciar la capacidad en uso al valor original
26            }
27        }
28
29        // Imprimir el número de baldes necesarios
30        printf("%d\n", baldes);
31    }
32
33    return EXIT_SUCCESS;
34 }
```



```

 Aguap.py > ...
1  T = int(input())
2  for j in range(T):
3      # Entrada: número persoans (n) y capacidad de balde (c)
4      n,c = [int(i) for i in input().split(" ")];
5      C_uso = c;
6      baldes= 1
7      # Arreglo para almacenar los requerimientos de agua separados por espacio
8      Cn = [int(i) for i in input().split(" ")]
9      for i in Cn:
10          C_uso-= i;
11          #verificación si hay agua suficiente para el requerimiento
12          if C_uso <= 0:
13              baldes+=1;
14          C_uso = c;
15      print(baldes)#salida: cantidad de baldes
```

PRUEBAS

Entradas	Salidas	Estado
4	4	😊
3 1	5	😊
6 4 5	1	😊
4 1	3	😊
1 6 4 5		😊
2 1 0		
5 2		
5 1 0		
8 6 7 1 0 2		



13 El medio *hot-dog* - dificultad: media

Vicente es un informático muy importante con una cuenta bancaria muy importante. Dado que Vicente es tan exitoso con muchos clientes, deposita dinero en su cuenta cada mañana. Después de ir al banco y depositar dinero, Vicente va a teletrabajar. Y ahí reside la gran debilidad de Vicente (alias Bicho): una tienda de *hot-dog*¹⁰. Vicente es un adicto a los *hot-dog* que se recupera, y aunque no ha comido un *hot-dog* en años, no puede dejar de preguntarse cuántos *hot-dog* de USD \$1.00 podría comprar con el dinero de cuenta. Tener USD \$5.00 en su cuenta significa 5 *hot-dogs* que Vicente podría tener, pero ¿qué hay de USD \$4.50? Bueno, eso es más de 4 *hot-dogs* por supuesto, pero definitivamente menos de 5. ¿Cómo compraría uno incluso una cantidad no entera de *hot-dog*?

Ese concepto confunde a Vicente, así que cada vez que su saldo de cuenta no es un número entero, se detiene para reflexionar sobre la naturaleza de los *hot-dogs* no enteros y termina siendo tarde para trabajar. Ahora, Vicente ha llegado demasiado tarde muchas veces y está empezando a preocuparse de que perderá su trabajo. Él quiere saber cuántas veces llegará tarde a trabajar durante los próximos días, dado su saldo inicial en la cuenta y la cantidad de dinero que depositará cada día. Por favor, responda esto por él, o Vicente empezará a pensar de nuevo.

¹⁰ Hot-dog: Alimento que se genera con la combinación de una salchicha del tipo Frankfurt (frankfurter) o vienesa (wiener) hervida o frita, servida en un pan con forma alargada que suele acompañarse con algún aderezo, como salsa de tomate, mostaza, entre otros. Denominado completo en Chile, pancho en Argentina y perro caliente en México (dato inútil).

Entrada

La primera línea contiene un número entero N ($1 \leq N \leq 1000$), el número de días de análisis. Cada una de las siguientes líneas $N + 1$ contiene una cadena que representa una cantidad de dinero. La primera cadena es el saldo inicial de la cuenta de Vicente, mientras que las siguientes N cadenas son las cantidades que depositará en su cuenta en los diferentes días. Cada cadena tiene la forma $\$X.Y$ donde X es una sub-cadena de longitud 1 ó 2 indicando el dinero entero en la cantidad $\$X.Y$, mientras que Y es un subcadena de la longitud exactamente 2 que denota los centavos en la suma $\$X.Y$

Entrada (USD \$)	Salida
1	1
\$1.57	
\$3.14	
4	2
\$1.00	
\$0.01	
\$0.99	
\$10.00	
\$98.76	

Salida

Genere una sola línea con un número entero que indique cuántas veces Vicente llegará tarde al trabajo durante los siguientes N días.

Fuente: ICPC Latam Regional 2020 (Adaptación de Non-integer donuts problem)

Solución ADCP

ANÁLISIS

Entradas: Enteros N y $N + 1$ reales de la forma $\$X.Y$

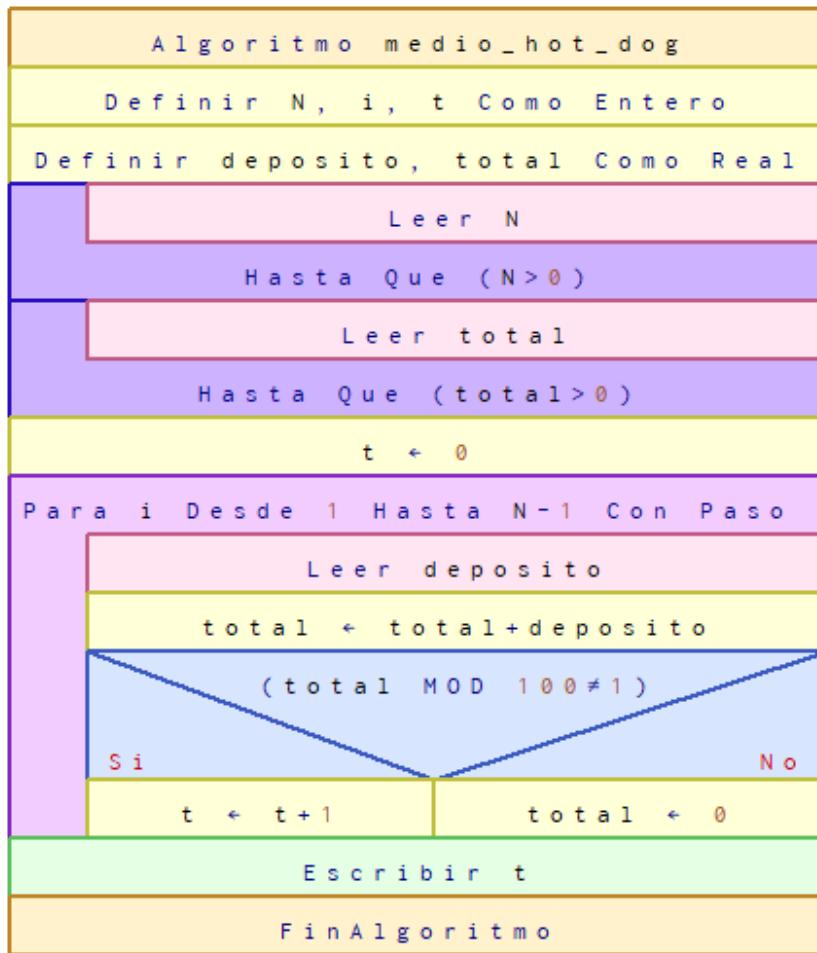


Proceso: La clave del problema está en detectar en qué momento Vicente entra en modo reflexión, esto es cuando la suma de su cuenta más el dinero depositado da un número real y no entero. Dicho de otro modo, Vicente llegará tarde al detectar parte decimal distinta de cero de su cuenta distinta de cero. Se asume que cuando tiene un saldo $\$S$ de dinero se compra S hotdogs (si esto no ocurre no importaría, ya que se restaría una cantidad entera de dinero).

Salidas: Veces en que llegará tarde (t)

Restricciones: $N > 0, \$X.Y > 0$

DISEÑO (Diagrama N-S)



Note que la lectura de la entrada se simplificó al leer cada componente de las entradas de la forma \$X.Y por separado. De esta forma evitamos el realizar un preprocessamiento denominado *parsing*¹¹, recordar que, si se está trabajando con una plataforma de programación competitiva con juez virtual para subir problemas resueltos, debiese incorporarse dicho preprocessamiento.

Te invito a crear una función especial para leer y preprocessar la entrada, te dejo una idea:

1. Verifica que la cadena comience con \$.
2. Elimina \$ y separa la parte entera (x) de la decimal (y).
3. Convierte las partes a números y combina la parte entera y decimal para obtener el número real.

¹¹ En programación y ciencias de la computación, el *parsing* permite convertir datos en una forma que el sistema pueda utilizar, generalmente desglosando los datos de entrada en componentes más pequeños para su procesamiento.

CODIFICACIÓN



```
1 #include <stdio.h>
2 #include <stdlib.h>
3 int main()
4 {
5     int N, i, t = 0;
6     float deposito, total;
7     // Entrada y validación de cantidad de días
8     do
9     {
10         scanf("%d", &N);
11     } while (N <= 0);
12
13     // Entrada y validación de USD $
14     do
15     {
16         scanf("%f", &total);
17     } while (total <= 0);
18
19     // Para todos los días
20     for (i = 0; i < N; i++)
21     {
22         scanf("%f", &deposito);
23         total += deposito;
24         // verificando si total almacena una parte decimal 0
25         if ((int)total / total != 1)
26             t++;
27         else
28             total = 0;
29     }
30     printf("%d", t);
31     EXIT_SUCCESS;
32 }
```



```

1 # Entrada de días (N)
2 N = int(input())
3
4 # Leer total y extraer la parte decimal
5 total = int(input().split(".")[1]) # Convertir el input en un número
6
7 # Inicializar el contador
8 t = 0
9
10 # Ciclo
11 for i in range(N):
12     # Leer depósito diario y extraer la parte decimal
13     deposito = int(input().split(".")[1])
14
15     # Sumar el valor ingresado al total
16     total += deposito
17
18     # Verificar si el total tiene una parte decimal igual a 0
19     if total % 100 == 0:
20         total = 0 # Reiniciar el total si la parte decimal es 0
21     else:
22         t += 1 # Incrementar el contador t si la parte decimal no es 0
23
24 # Imprimir contador
25 print(t)

```

Observe que en lenguaje Python el *parsing* de la entrada se logra implementar con el método **SPLIT** (ver línea 13 en el código *.py).

PRUEBAS

Entrada	Salida	Estado
1	1	😊
1.57		
3.14		
4	2	😊
1.00		
0.01		
0.99		
10.00		
98.76		

14 Los viajeros – dificultad: alta

Alice, Bob y Cindy han decidido irse de viaje por las fiestas! Mientras el resto de sus amigos han ido a vacacionar a Japón y Europa, estos tres amigos han decidido una alternativa más divertida (y económica), explorar las filipinas en grupo. Por pura coincidencia los 3 compraron la misma maleta para llevarla al viaje. Por buena suerte todos decidieron rotular sus maletas con su nombre (estas etiquetas distinguen entre mayúsculas y minúsculas). Por desgracia en estas etiquetas se han ido borrando algunas letras con el tiempo, al menos tienes asegurado que el largo de la palabra se mantiene. Ayudémosles a que pasen unas vacaciones libres de estrés escribiendo un programa que identifique al dueño de la maleta una etiqueta poco legible asumiendo que el dueño es Alice, Bob o Cindy.



Entrada	Salida
Ali.e	Alice
bob	SOMETHING'S WRONG
.i...	Cindy

Solución ADCP

ANÁLISIS

Entradas

Un *string* o cadena de caracteres **S** conteniendo caracteres alfabéticos y/o punto(.)

Proceso



Lo primero que se puede filtrar, es el caso de que el nombre ingresado devuelva "SOMETHING'S WRONG", en primer momento se puede solucionar de forma muy sencilla con una pregunta, ¿el *string* tiene 3 ó 5 caracteres? si la respuesta es no, se imprime "SOMETHING'S WRONG" y se terminaría el ejercicio, si la respuesta en un sí, por otro lado, se podría ver como progresá, primero viendo con el caso de que fuera de sólo 3 caracteres de largo.

Se evaluarán distintos escenarios que pueden ocurrir, para ello se utiliza Bob como caso de prueba inicial, Bob sólo tiene 3 caracteres pudiendo comparar los posibles casos de uno en uno.

B	.	.
B	o	b
si	puede ser	puede ser

En este caso como la cantidad de caracteres es 3 y el único carácter que da la pista es correcto, podemos asumir que es la maleta de bob.

b	o	b
B	o	b

NO	si	si
----	----	----

En este caso la maleta no es de bob porque el escribe su nombre con una mayúscula al principio por lo que imprimiríamos *SOMETHING'S WRONG*.

.	.	.
B	o	b
tal vez	tal vez	tal vez

En este caso la respuesta sería Bob puesto que es la única posibilidad con tres caracteres de los tres nombres, ahora bien, en caso de que el largo sea de 5 tenemos un poco más de variedad, para eliminar la posibilidad de que sea cualquiera, se descarta la posibilidad de que sea cualquiera de los 2 nombres por lo que, si diera que entregan, se sabrá al instante que se debe imprimir *CAN'T TELL*.

Si ese no fuera el caso habría que ver como reconocer a que persona corresponde, para eso podríamos ir comparando letra a letra de nuevo, pero ahora viendo que palabra forma de forma que si es un punto asumimos que es la letra correcta y si en algún momento aparece una letra que no calza intentamos con la otra persona posible.

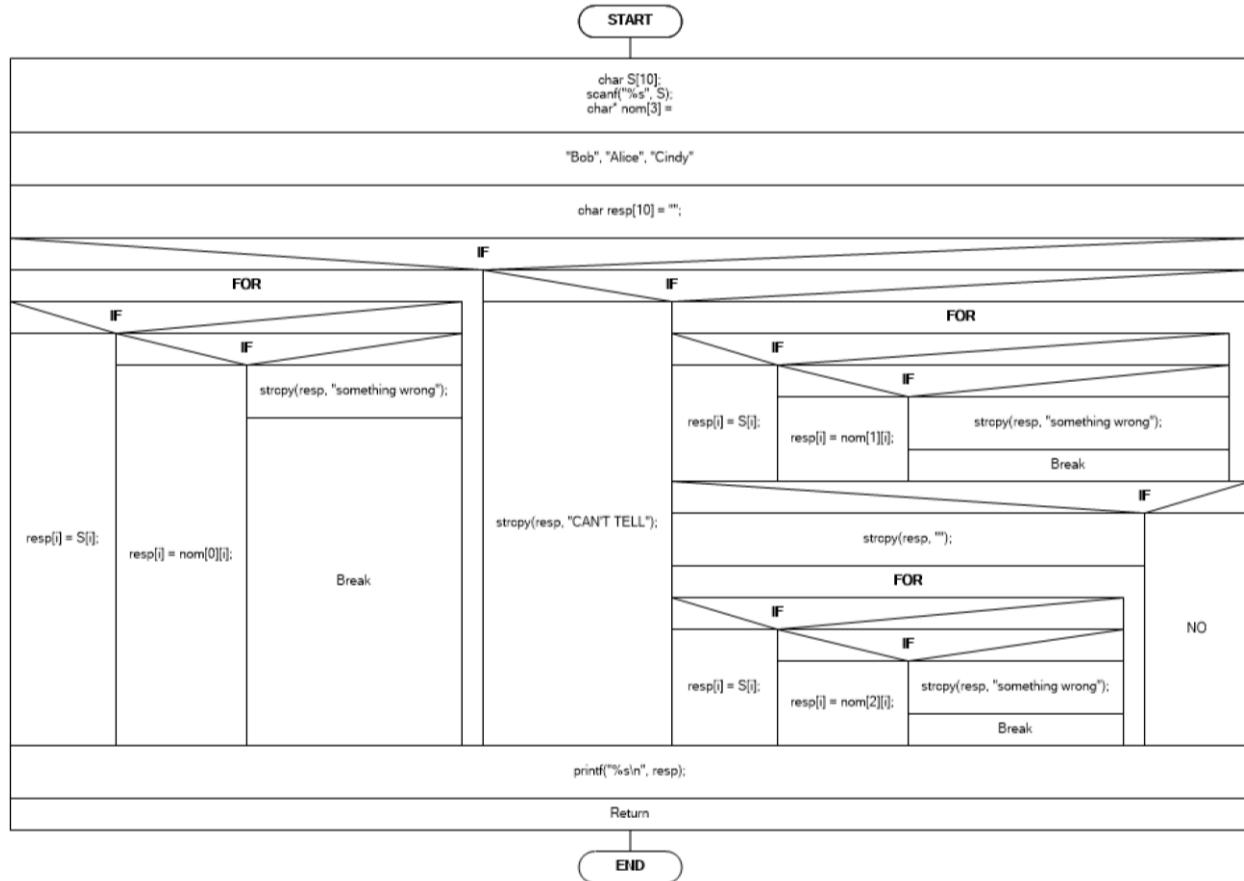
Salida

Se asume el dueño de la maleta que puede ser Alice, Cindy o Bob entonces para la salida tenemos 3 opciones posibles:

1. Si con las letras visibles solo puede ser uno de los 3 nombres entonces se imprime el nombre.
2. Si existiera más de una posibilidad se imprime *CAN'T TELL*
3. Si el nombre no coincide con ninguno de los 3 se imprime *SOMETHING'S WRONG*

Restricciones

El *string S* debe tener entre 1 a 5 caracteres, estos caracteres deben ser caracteres alfabéticos en mayúsculas o minúsculas o puede ser un punto.

DISEÑO (Diagrama N-S)**CODIFICACIÓN**

(1/3)

```

1 #include<stdio.h>
2 #include<string.h>
3 #include<stdlib.h>
4
5 int main() {
6     char S[10];
7     // Solicitar al usuario que ingrese una cadena de caracteres
8     scanf("%s", S);
9
10    // Nombres predefinidos
11    char* nom[3] = {"Bob", "Alice", "Cindy"};
12    // Variable para almacenar la respuesta
13    char resp[10] = "";
  
```



(2/3)

```

// Verificar si la longitud de la cadena es 3
if (strlen(S) == 3) {
    // Comparar cada carácter de la cadena con el primer nombre
    for (int i = 0; i < 3; i++) {
        if (S[i] == nom[0][i]) {
            resp[i] = S[i];
        } else if (S[i] == '.') {
            resp[i] = nom[0][i];
        } else {
            // Si hay un carácter diferente, asignar un mensaje de error
            strcpy(resp, "algo salió mal");
            break;
        }
    }
} else if (strcmp(S, ".....") == 0) {
    // Si la cadena es ".....", asignar un mensaje especial
    strcpy(resp, "NO SE PUEDE DETERMINAR");
} else {
    // Comparar cada carácter de la cadena con el segundo nombre
    for (int i = 0; i < 5; i++) {
        if (S[i] == nom[1][i]) {
            resp[i] = S[i];
        } else if (S[i] == '.') {
            resp[i] = nom[1][i];
        } else {
            // Si hay un carácter diferente, asignar un mensaje de error
            strcpy(resp, "algo salió mal");
            break;
        }
    }
}

```



(3/3)

```

// Si la respuesta no coincide con el segundo nombre, intentar con el tercero
if (strcmp(resp, nom[1]) != 0) {
    // Reiniciar la respuesta
    strcpy(resp, "");
    // Comparar cada carácter de la cadena con el tercer nombre
    for (int i = 0; i < 5; i++) {
        if (S[i] == nom[2][i]) {
            resp[i] = S[i];
        } else if (S[i] == '.') {
            resp[i] = nom[2][i];
        } else {
            // Si hay un carácter diferente, asignar un mensaje de error
            strcpy(resp, "algo salió mal");
            break;
        }
    }
}

// Imprimir la respuesta
printf("%s\n", resp);
return EXIT_SUCCESS;
}

```



(1/2)

```

# Solicitar al usuario que ingrese una cadena de caracteres
S = input()

# Nombres predefinidos
nom = ["Bob", "Alice", "Cindy"]

# Variable para almacenar la respuesta
resp = ""

# Verificar si la longitud de la cadena es 3
if len(S) == 3:
    # Comparar cada carácter de la cadena con el primer nombre
    for i in range(3):
        if S[i] == nom[0][i]:
            resp = resp + S[i]
        elif S[i] == ".":
            resp = resp + nom[0][i]
        else:
            # Si hay un carácter diferente, asignar un mensaje de error
            resp = "algo salió mal"
            break
    # Verificar si la cadena es "...."
elif S == "....":
    resp = "NO SE PUEDE DETERMINAR"
else:
    # Comparar cada carácter de la cadena con el segundo nombre
    for i in range(5):
        if S[i] == nom[1][i]:
            resp = resp + S[i]
        elif S[i] == ".":
            resp = resp + nom[1][i]
        else:
            # Si hay un carácter diferente, asignar un mensaje de error
            resp = "algo salió mal"
            break

```



(2/2)

```

# Si la respuesta no coincide con el segundo nombre, intentar con el tercero
if resp != nom[1]:
    resp = ""
    # Comparar cada carácter de la cadena con el tercer nombre
    for i in range(5):
        if S[i] == nom[2][i]:
            resp = resp + S[i]
        elif S[i] == ".":
            resp = resp + nom[2][i]
        else:
            # Si hay un carácter diferente, asignar un mensaje de error
            resp = "algo salió mal"
            break

    # Imprimir la respuesta
print(resp)

```

15 Amigos – dificultad: alta

Dos números son amigos, si cada uno de ellos es igual a la suma de los divisores del otro (sin incluir al mismo número en la suma). Por ejemplo, 220 y 284 son amigos, ya que:



Suma de divisores de 220: $1 + 2 + 4 + 5 + 10 + 11 + 20 + 22 + 44 + 55 + 110 = 284$

Suma de divisores de 284: $1 + 2 + 4 + 71 + 142 = 220$

Implemente un algoritmo que permita obtener las parejas de números amigos menores o iguales a m , siendo m un número ingresado por teclado.

Solución ADCP

ANÁLISIS

Entradas: Cota superior (m)

Proceso mental:



- ¿Cómo se obtiene el divisor de un número?, si $(K \text{ MOD } i = 0)$ con $i = 5$, n significa que k es divisible por 5
 - ¿cómo se obtienen números amigos k, i ? si suma de divisores de $k = \text{suma}$ de divisores de i con $(k \leq m)$ y $(i < k)$
- Recuerde que: $(a \text{ MOD } b)$ es el resto de la división entera entre a y b , también conocido como operador módulo.
- ¿cómo generar números entre 1 y m ?

Para $k = 1, m$ hacer //generará números de $[1, m]$

Para $i = 1$ hasta $k - 1$ hacer //generará números entre $[1, k - 1]$

//Y aplicando la regla de divisores

Si $(k \text{ MOD } i = 0)$ Entonces

$S1 = S1 + i;$ //se acumulará a los divisores de k

FinSi

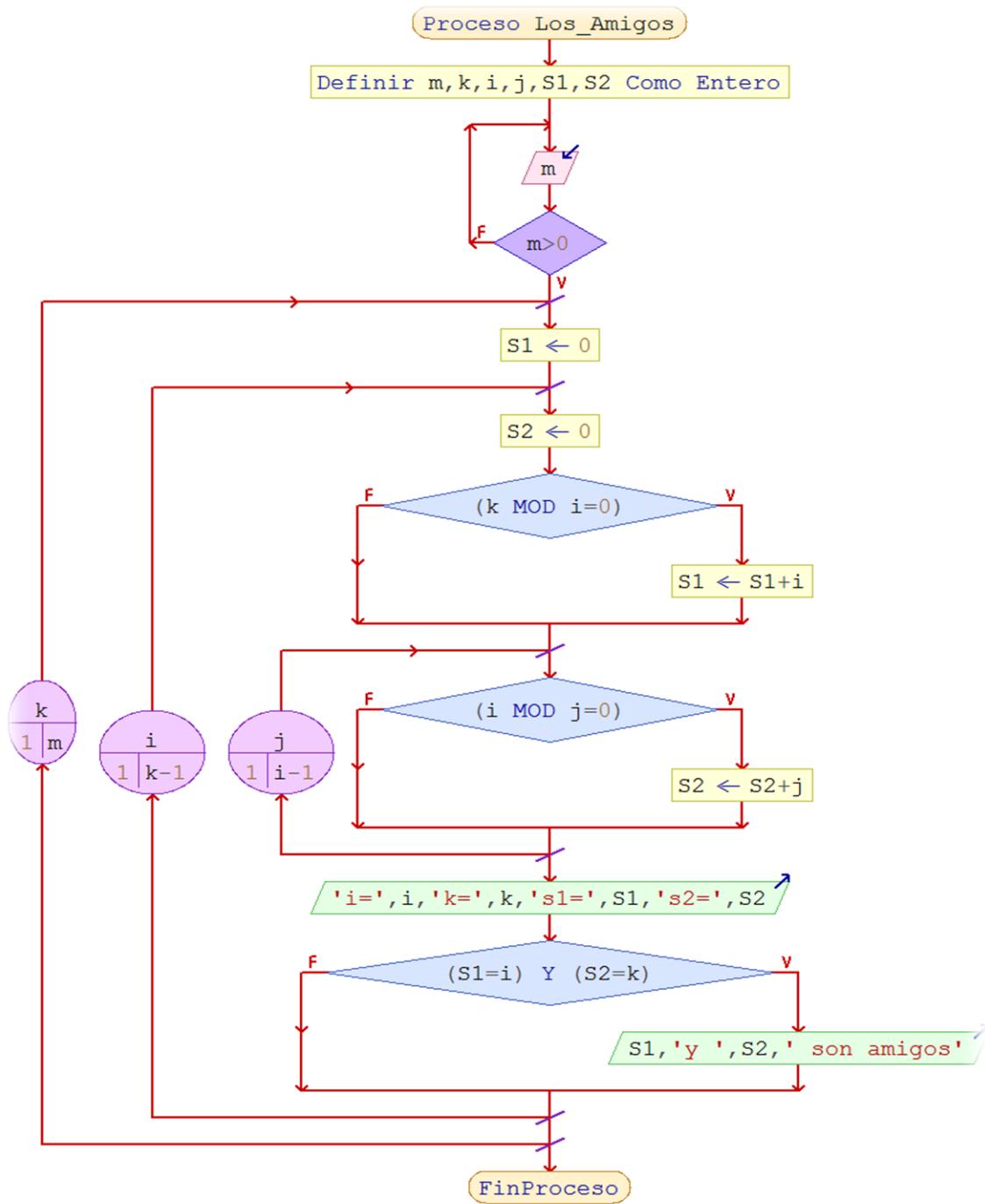
Fin Para// i

FinPara// k

Salidas: Números amigos (i, k)

Restricciones: $m > 0$

DISEÑO (Diagrama de Flujo)



CODIFICACIÓN



```

1 ~ #include <stdio.h>
2   #include <stdlib.h>
3
4 ~ int main() {
5     int i, j, k, m, S1 = 0, S2 = 0;
6     // Solicitar al usuario que ingrese un número entero positivo
7 ~ do {
8     |   scanf("%d", &m);
9     } while (m <= 0);
10
11    // Iterar sobre los números hasta el valor ingresado por el usuario
12 ~ for (k = 1; k <= m; k++) {
13     |   S1 = 0;
14     |   // Calcular la suma de los divisores propios de k
15 ~     for (i = 1; i < k; i++) {
16         |   S2 = 0;
17         |   // Verificar si i es un divisor propio de k
18 ~         if (k % i == 0) {
19             |   S1 = S1 + i;
20             }
21         // Calcular la suma de los divisores propios de i
22 ~         for (j = 1; j < i; j++) {
23             |   if (i % j == 0) {
24                 |       S2 = S2 + j;
25                 }
26             }
27         // Imprimir información sobre los valores de i, k, S1 y S2
28 ~         printf("\ni=%d\tk=%d\tS1=%d\tS2=%d", i, k, S1, S2);
29
30         // Verificar si i y k son números amigos
31 ~         if ((S1 == i) && (S2 == k)) {
32             |   printf("\nlos numeros %d y %d son numeros amigos", i, k);
33             |   getchar();
34             }
35         }
36     }
37 ~     return EXIT_SUCCESS;
38 }

```



```

1  def main():
2      # Inicializar variables
3      i=0
4      j=0
5      k=0
6
7      # Solicitar al usuario que ingrese un número entero positivo
8      m = int(input("Ingrese un entero positivo: "))
9      while m <= 0:
10         if (m<=0):print("Error! m debe ser >0 ")
11         m = int(input("Ingrese un entero positivo: "))
12
13     # Iterar sobre los números hasta el valor ingresado por el usuario
14     for k in range(1, m + 1):
15         S1 = 0
16
17         # Calcular la suma de los divisores propios de k0
18         for i in range(1, k):
19             S2 = 0
20
21             # Verificar si i es un divisor propio de k
22             if k % i == 0:
23                 S1 = S1 + i
24
25             # Calcular la suma de los divisores propios de i
26             for j in range(1, i):
27                 if i % j == 0:
28                     S2 = S2 + j
29
30         # Imprimir información sobre los valores de i, k, S1 y S2
31         print(f"\ni= {i} \tk={k} \tS1={S1} \tS2={S2}")
32
33         # Verificar si i y k son números amigos
34         if S1 == i and S2 == k:
35             print(f"Los números {i} y {k} son números amigos")
36
37     if __name__ == "__main__":
38         main()

```

PRUEBAS

m	amigo1	amigo2	Estado
290	220	284	😊
1300	220 1184	284 1210	😊
-1	¡Error! m debe ser >0		😊
2950	220 1184 2620	284 1210 2924	😊
100000	220	284	😊
	1184	1210	
	2620	2924	
	5020	5564	
	6232	6368	
	10744	10856	
	12285	14595	
	17296	18416	
	63020	76084	
	66928	66992	
	67095	71145	
	69615	87633	
	71145	67095	
	76084	63020	
	79750	88730	
	87633	69615	
	88730	79750	

16 Problema clasificación de personajes de anime – dificultad: media



En un mundo lleno de anime, hay diferentes tipos de personajes con habilidades únicas. Los investigadores de la Academia de Ciencias de Anime están interesados en desarrollar un algoritmo de *machine learning* para clasificar automáticamente los personajes en dos categorías: "Héroe" o "Villano", basándose en sus características.

Se te ha asignado la tarea de desarrollar una solución¹² que implemente un algoritmo de clasificación simple. Se proporciona un conjunto de datos de entrenamiento que contiene información sobre varios personajes de anime, incluyendo su nivel de poder, su actitud, y su historia pasada. Considere los siguientes datos de entrenamiento (Nombre, nivel de Poder, Actitud {1: Buena, -1: Mala}, Historia Pasada {0: Heroica, 1: Trágica})

Entradas

Naruto, 9000, 1, 0
 Goku, 12000, 1, 0
 Light, 5000, -1, 1
 Freeza, 8000, -1, 1
 Luffy, 7500, 1, 0
 Sasuke, 8500, -1, 1
 Vegeta, 11000, -1, 0

Reglas:

1. Si el nivel de poder es mayor a 9000, el personaje se considera un "Héroe".
2. Si la actitud es positiva (1) y la historia pasada es heroica (0), el personaje se considera un "Héroe".
3. En todos los demás casos, el personaje se considera un "Villano".
4. El algoritmo de clasificación debe implementarse en una función denominada **ClasificarPersonaje** (C y Python). Los resultados se imprimen en la consola, indicando si cada personaje es un "Héroe" o un "Villano".
5. Los datos de entrenamiento deben ser inicializados durante la codificación del programa utilizando un arreglo de estructuras (*struct*).

¹² Se considera de un nivel intermedio de dificultad porque requiere el uso de *struct* (C) y clases (Python) y una arquitectura de programa basado en funciones.

Solución ADCP

ANÁLISIS



Entradas: Datos de entrenamiento inicializados con la estructura (Nombre, nivel de Poder, Actitud {1: Buena, -1: Mala}, Historia Pasada {0: Heroica, 1: Trágica})

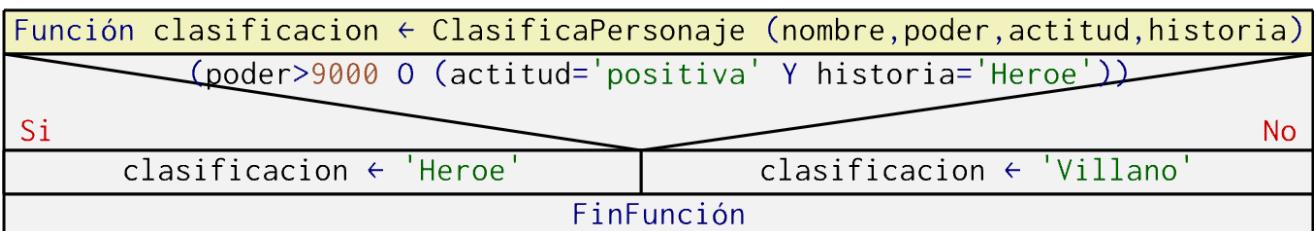
Proceso mental: En el contexto de *machine learning* o aprendizaje de máquina, el algoritmo de clasificación utilizado en este problema es bastante simple y se basa en reglas. Ilustra el concepto de tomar decisiones automáticas basadas en características específicas. El algoritmo de clasificación sigue estas reglas:

1. Si el nivel de poder del personaje es mayor a 9000, se clasifica como "Héroe".
2. Si la actitud del personaje es positiva (1) y su historia pasada es heroica (0), también se clasifica como "Héroe".
3. En todos los demás casos, el personaje se clasifica como "Villano".

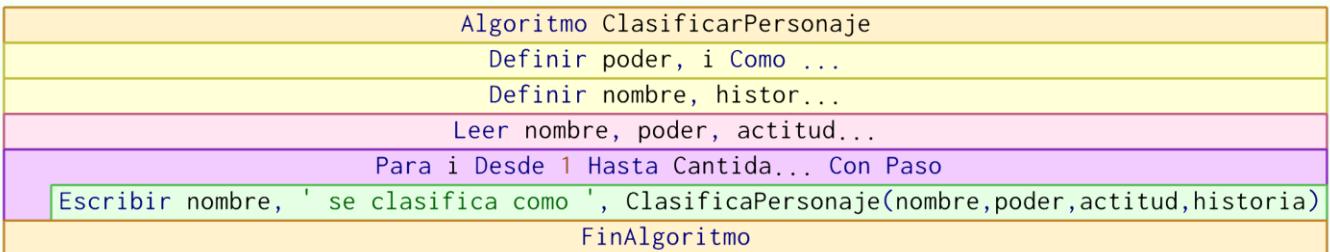
Estas reglas se establecen mediante la observación de los datos de entrenamiento proporcionados. El objetivo es hacer que el modelo, basado en reglas condicionales, capture patrones en los datos que permitan realizar predicciones sobre nuevos datos. En un contexto más amplio de *machine learning*, los algoritmos de clasificación a menudo aprenden automáticamente estas reglas a partir de datos de entrenamiento más extensos. Algoritmos más avanzados, como los clasificadores de regresión logística, máquinas de soporte vectorial (SVM) o redes neuronales (NN), utilizan métodos más complejos para ajustar los parámetros del modelo y adaptarse a patrones en los datos.

DISEÑO

De la función



Del programa principal



Observe que, en este caso, se declaró una función denominada **ClasificaPersonaje** con 4 parámetros de entrada: *nombre, poder, actitud, historia*, parámetros que son evaluados según las reglas de clasificación dadas para determinar si el personaje es *Héroe* o *Villano*, de esta manera la función retorna la clasificación realizada. Este resultado es recibido por algoritmo principal a partir de la llamada que realiza a la función en la penúltima fila del diagrama N-S o bien desde la línea 34 del código fuente en C. La idea de definir una función es externalizar una tarea específica, permitiendo que un futuro se pueda reusar las veces que sea necesaria sin reescribir código. Note que el nombre del algoritmo es diferente del nombre de la función.

CODIFICACIÓN



13

```

C ProblemaClasificacion.c > ...
1  #include <stdio.h>
2  #include <stdlib.h>
3  //Definición de estructura de epersonajes de anime
4  struct Personaje {
5      char nombre[20];
6      int nivelDePoder;
7      int actitud;
8      int historiaPasada;
9  };
10
11 //Función clasificadora
12 char clasificarPersonaje(struct Personaje *p) {
13     if (p->nivelDePoder > 9000 || (p->actitud == 1 && p->historiaPasada == 0)) {
14         return 'H'; // Héroe
15     } else {
16         return 'V'; // Villano
17     }
18 }
19
20 int main() {
21     //Definición de struct con atributos
22     struct Personaje personajes[] = {
23         {"Naruto - Naruto", 9000, 1, 0},
24         {"Goku - Dragon Ball", 12000, 1, 0},
25         {"Light - Death Note", 5000, -1, 1},
26         {"Freezer - Dragon Ball", 8000, -1, 1},
27         {"Luffy - One Peace", 7500, 1, 0},
28         {"Sasuke - Naruto", 8500, -1, 1},
29         {"Vegeta - Dragon Ball", 11000, -1, 0}
30     };
31
32     //Salida
33     for (int i = 0; i < sizeof(personajes) / sizeof(personajes[0]); i++) {
34         printf("%s es un %s\n", personajes[i].nombre, (clasificarPersonaje(&personajes[i]) == 'H') ? "Héroe" : "Villano");
35     }
36     return EXIT_SUCCESS;
37 }
    
```

¹³ Observe la línea 34 y la forma abreviada de condicionar. Investiga sobre la función `sizeof`, el uso de funciones y estructuras en lenguaje C y sus equivalentes en Python.



```

Problema 16_Clasicacion.py > ...
1  class Personaje:
2      #Constructor de la clase Personaje
3      def __init__(self, nombre, nivel_de_poder, actitud, historia_pasada):
4          #Definición de clase para personajes de anime con diversos atributos
5          self.nombre = nombre
6          self.nivel_de_poder = nivel_de_poder
7          self.actitud = actitud
8          self.historia_pasada = historia_pasada
9      #Clasificador
10     def clasificar_personaje(personaje):
11         if personaje.nivel_de_poder > 9000 or (personaje.actitud == 1 and personaje.historia_pasada == 0):
12             return 'H' # Héroe
13         else:
14             return 'V' # Villano
15
16     #Inicialización de los atributos
17     personajes = [
18         Personaje("Naruto", 9000, 1, 0),
19         Personaje("Goku", 12000, 1, 0),
20         Personaje("Light", 5000, -1, 1),
21         Personaje("Freeza", 8000, -1, 1),
22         Personaje("Luffy", 7500, 1, 0),
23         Personaje("Sasuke", 8500, -1, 1),
24         Personaje("Vegeta", 11000, -1, 0)
25     ]
26
27     #Salida
28     for personaje in personajes:
29         print(f"{personaje.nombre} es un {'Héroe' if clasificar_personaje(personaje) == 'H' else 'Villano'}")

```

PRUEBAS

Entradas	Salida	Estado
Naruto 9000 1 0	Héroe	😊
Goku 12000 1 0	Héroe	😊
Light 5000 -1 1	Villano	😊
Freeza 8000 -1 1	Villano	😊
Luffy 7500 1 0	Héroe	😊
Sasuke 8500 -1 1	Villano	😊
Vegeta 11000 -1 0	Héroe	😊

Parte B: Problemas propuestos

En el siguiente apartado se proponen distintos problemas para que ustedes mismos los resuelvan, algunos de ellos tienen pistas en el análisis para que tengas alguna ayuda en la estrategia a seguir. Para todos es necesario completar el [Análisis, Diseño, Codificación y Pruebas](#).

¡Suerte con el resto de las soluciones para los problemas propuestos! 

Problema fechas – dificultad: media

Verificar si una fecha ingresada en formato **dd/mm/aaaa** es válida o no considerando reglas de rango y calendario gregoriano. Considere el formato especificado. Lea la entrada usando una cadena de caracteres y haga *parsing*.



Pista ANÁLISIS

Entradas dd/mm/aaaa (considere almacenarlo en una cadena y transformar)



Proceso mental: Pues bien, a partir de esas reglas que se detallan más adelante se extraerán las validaciones de los meses, el caso particular es febrero que cada 4 años toma el valor 29 en vez de 28. El calendario gregoriano¹⁴ se basa en un año solar de aproximadamente 365.2425 días. Para mantenerse alineado con las estaciones del año, se agregó un día adicional (llamado año bisiesto) en años divisibles por 4, excepto aquellos divisibles por 100 a menos que también sean divisibles por 400.

Reglas:

El calendario gregoriano consta de 12 meses:

Enero (31 días)

Febrero (28 días, 29 en años bisiestos)

Marzo (31 días)

Abril (30 días)

Mayo (31 días)

Junio (30 días)

Julio (31 días)

Agosto (31 días)

Septiembre (30 días)

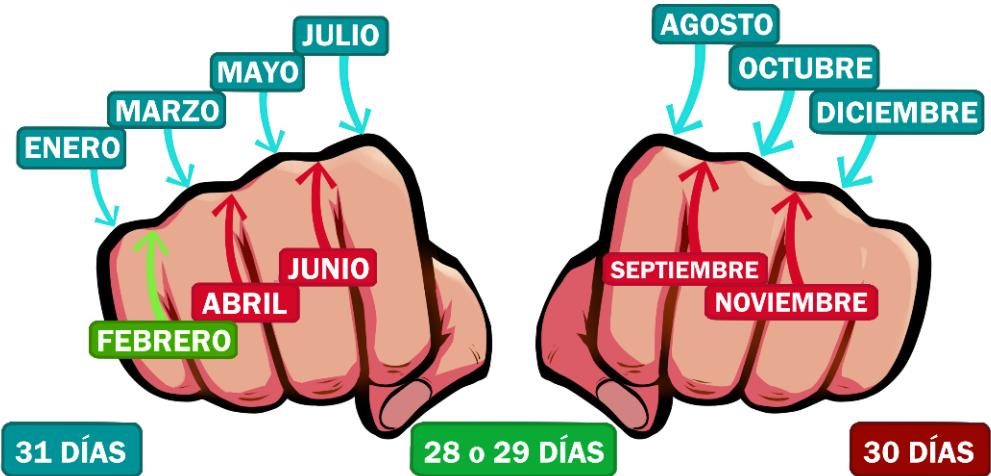
Octubre (31 días)

Noviembre (30 días)

Diciembre (31 días)

¹⁴ El calendario gregoriano es el calendario civil utilizado en la mayoría de los países del mundo hoy en día. Fue introducido por el papa Gregorio XIII en 1582 como una reforma del calendario juliano, con el objetivo de corregir ciertas imprecisiones en la medición del año solar.

Ahora bien, ¿Te acuerdas de la regla de los nudillos? Esta les apoyará para establecer las reglas de los días para cada mes.



Con esto, es posible saber cuándo un mes tiene 30, 31 ó 28 días. La excepción es febrero que tiene 28 ó 29 porque es un año bisiesto. ¿Qué es ser bisiesto y cómo saber si un año es bisiesto?

Año bisiesto (o año intercalar): Año que tiene 366 días en lugar de 365, en el que febrero tiene 29 días en lugar de 28; se repite cada cuatro años, excepto cuando el año acaba en dos ceros, en otras palabras, en años divisibles por 4, excepto aquellos divisibles por 100 a menos que también sean divisibles por 400. **Se observa que la dificultad en este problema está en procesar la entrada que debiese leerse en una cadena para luego extraer de allí cada componente de la fecha.**

Salida: Mensaje “Fecha Válida” en caso de éxito y “Fecha no válida” en caso de fracaso

Restricciones: $dd > 0$ y $dd \leq 31$, $mm > 0$ y $mm \leq 12$, $aaaa > 0$

Problema La AI en gestión de RR energéticos – dificultad: baja



La humanidad ha avanzado significativamente en la implementación de sistemas de Inteligencia Artificial (IA) para gestionar eficientemente los recursos energéticos del planeta. En este mundo paralelo, la principal fuente de energía es un tipo especial de cristal llamado Energon. Cada cristal tiene un número único que lo identifica, y la eficiencia energética de un Energon se correlaciona directamente con la suma de sus divisores, excluyendo el propio número.

En este contexto, los ingenieros han desarrollado un programa de IA que se encarga de analizar y optimizar la eficiencia de los cristales de Energon. Sin embargo, hay un problema particular que la IA debe resolver: calcular

la suma de los divisores distintos del número único asignado a cada cristal. Este valor es crucial para determinar la eficiencia real del cristal y su contribución al suministro global de energía.

La tarea de la IA es fundamental para garantizar que los recursos energéticos se utilicen de manera óptima, maximizando la producción de energía y minimizando el desperdicio. Cada cristal de Energon representa una pieza esencial en el rompecabezas de la sostenibilidad y la supervivencia de la humanidad.

Entrada	Salida
3	1
6	6
8	7
-1	¡Error!, debe ser >0
0	¡Error!, debe ser >0



SumaQTSuma++[∞] – dificultad: baja

Suponga que al mismo estudiante de 1º básico que ayudaste en el primer problema se le dio un nuevo desafío - calcular el cuadrado de un número no negativo - el pesar del estudiante sólo sabe hacer sumas simples y algo aprendió de multiplicación con tu ayuda. Apóyalo en el desarrollo de la solución con lo que sabe.

Entrada	Salida
3	9
0	¡Error!, debe ser >0

Pista ANÁLISIS

Entrada: Un número (n)



Proceso mental: Recuerden que en este apartado ustedes pueden incluir una oda, una fórmula, un mapa conceptual, un esquema, un gráfico, una secuencia numérica, un diagrama de eventos...un dibujo. Todo aquello que se relacione con sus experiencias de aprendizajes previas y que le ayude a entender el problema y a proponer una solución. Como el problema menciona a un estudiante de primero básico, que por currículum sólo sabe realizar sumas simples, debemos explicarle la operación elevado ($base^{exponente}$) a través de la siguiente analogía:

$$1^2 = 1 * 1 = 1$$

$$2^2 = 2 * 2 = 2 + 2$$

[∞] Antes de desarrollar este problema se recomienda desarrollar el Problema 1 SumaQTSuma

$$3^2 = 3 * 3 = 3 + 3 + 3$$

$$4^2 = 4 * 4 = 4 + 4 + 4 + 4$$

Generalizando: $n^2 = n * n = n + n + \dots + n$ veces

Salida: El cuadrado del número **n**

Restricciones: El número debe ser mayor que 0 (se extrae del enunciado)

Problema Ayudando al abuelo Laíno – dificultad: baja

No sabemos concretamente por qué, pero al abuelo Laíno no le agrada una de las vocales de la lengua castellana. Tal vez sea por él engoroso declamarla. En todo caso, afortunadamente esa vocal no está presente en la palabra “abuelo”, de manera que sus descendientes no se encuentran en apuros cuando le llaman afectuosamente de ese modo.



El abuelo Laíno trabajó con nulo descanso durante décadas, por lo que se prepara para tomar un justo receso de sus arduas tareas. En este lapso, desea emprender una aventura atravesando lugares lejanos, para lo cual está ahora empacando su maleta. El abuelo Laíno no desea llevar en ella objetos cuyos nombres contengan la vocal que tanto lo consterna, no vaya a ser que al verlos se vea forzado a pensar en la tan censurable letra durante su reposo. Su tarea es ayudarlo en esta labor, para lo cual deben aconsejarle sobre cuáles de los objetos que posee puede empacar.

Entrada

Una línea conteniendo una cadena no vacía de hasta 20 caracteres de la ‘a’ a la ‘z’ indicando el nombre de un objeto que posee el abuelo Laíno.

Salida

Imprimir en la salida una línea conteniendo un carácter que representa si el abuelo Laíno puede empacar el objetivo cuyo nombre aparece en la entrada. El carácter debe ser una ‘S’ si el abuelo Laíno puede empacarlo, y una ‘N’ caso contrario.

Entrada	Salida
remera	S
camisa	N
buey	S
i	N
abuelo	S
estenoporquetienelai	N

Fuente: Warmup TCP 2018

Problema Tropiconce – dificultad: baja

En el corazón del sur de Chile, rodeada por exuberante vegetación y bañada por suaves brisas del océano Pacífico, se encuentra Tropiconce, una ciudad mágica donde el sol y la lluvia bailan en armonía. En este rincón tropical de Chile, la vida se teje con las fluctuaciones meteorológicas diarias.



Cada mañana, los habitantes de Tropiconce despiertan con la emoción de descubrir qué caprichos les deparará el clima. Las fichas meteorológicas diarias se convierten en los relatos que marcan el pulso de la ciudad. La temperatura máxima y mínima son como los personajes principales de esta historia, el termómetro oscila entre el calor abrazador y las noches frescas que invitan a abrigarse.

Pero es el agua caída en milímetros la que realmente pinta la paleta de colores de Tropiconce. En esta tierra tropical, la lluvia es más que un fenómeno atmosférico; es el maestro que da vida a la exuberante vegetación y alimenta los ríos que serpentean a través de la ciudad. Los registros pluviométricos son como los versos de una poesía, a veces suave y melódica, otras veces intensa y apasionada.

Un mes en Tropiconce es un viaje a través de los cambios climáticos, donde la gente se adapta con gracia a la danza interminable del clima. Los habitantes de esta ciudad tropical celebran la lluvia como un regalo que nutre la tierra y alimenta sus sueños, mientras que el sol radiante los invita a disfrutar de playas cercanas y paisajes tropicales.

Dado un mes de registros, se pide determinar e informar por pantalla:

- El día más frío y cuál fue esa temperatura
- El día más cálido y cuál fue esa temperatura
- Los 3 días de mayor lluvia junto a la cantidad de lluvia caída estos días
- El total de lluvia caída

Entrada

30 tuplas de datos incluyendo la temperatura mínima (t_{\min}), temperatura máxima (t_{\max}) en °C y los milímetros de agua caída durante el mes.

Salida

- El día más frío y cuál fue esa temperatura
- El día más cálido y cuál fue esa temperatura
- Los 3 días de mayor lluvia junto a la cantidad de lluvia caída estos días
- El total de lluvia caída

Entrada			Salida
1	21	1.2	Día más frío -2.8°C, día 23
0	12	3.3	Día más cálido 26.0°C, día 7
-1	23	4.1	3 días más lluviosos
2	8	0.1	3 con 4.1 mm
15	0	0.0	día 12 con 5.6mm
22	4	0.9	día 22 con 6 mm, total agua caída 15.7 mm
2	26	1.0	Tota agua caída mes: 37 mm
3	21	1.2	
4	21	0.2	
2	21	0.0	
1	21	1.2	
0	12	5.6	
-1	23	3.1	
2	8	0.1	
15	0	0.0	
22	4	0.9	
2	25	1.0	
3	21	1.2	
4	21	0.2	
2	21	0.0	
1	21	1.2	
0	12	6.0	
-2.8	23	1.1	
2	8	0.1	
15	0	0.0	
22	4	0.9	
2	25	1.0	
3	21	1.2	
4	21	0.2	
2	21	0.0	

Problema Hottest Mountain – dificultad: baja



Dada la temperatura de un conjunto de N montañas ($1 \leq N \leq 100$), simplemente debes encontrar la más cálida de ellas. Si dos montañas tienen la misma temperatura, entonces debes elegir la que tenga la posición más alta en la especificación de entrada (dada última). También simple, ¿no?

Entrada

La primera línea de la entrada es N , $1 \leq N \leq 1000$ la cantidad de montañas para este problema. En cada una de las siguientes N líneas, hay un número real $T \leq 1000$, que representa la temperatura de la N -ésima montaña.

Salida

Imprima una sola línea con el número P , que representa la posición de la montaña más caliente.

Entrada	Salida
7	7
15.00	
35.54	
11.32	
11.32	
21.00	
15.00	
35.54	

Fuente: Adaptación de High Mountains Problem
Argentinian Programming Tournament 2012

Problema del Panagrama – dificultad: media



"Panagrama Show" es un nuevo y emocionante programa de televisión que ofrece suculentos premios en efectivo para los participantes que detectan correctamente si una oración es un Panagrama. Un Panagrama es una oración que contiene al menos una vez todas las letras de un alfabeto determinado. Un clásico ejemplo de Panagrama en español es:

**El veloz murciélagó hindú
comía feliz cardillo y kiwi,
la cigüeña tocaba el saxofón
detrás del palenque de paja.**

ronda.

A cada participante del show se le da una oración y él / ella debe indicar dentro de quince segundos si la oración dada es un Panagrama o no. Cuando un(a) competidor(a) falla, él/ella es eliminado(a) de la competencia, de lo contrario, continúa para la siguiente

Entrada

La entrada consta de una sola línea que contiene una cadena **S** que contiene al menos uno y como máximo 200 caracteres. Los caracteres en **S** son letras minúsculas del alfabeto español y espacios.

Salida

Genere una sola línea con la letra mayúscula "S" si la oración es un Panagrama y la letra mayúscula "N" de lo contrario.

Entrada	Salida
el veloz murciélagos hindú comía feliz kiwi piña y quinotos bajo un exquisito palmar	S
jovencillo emponzoñado de whisky qué figurota exhibes	S
el veloz murciélagos hindú comía feliz cardillo y kiwi	N

Fuente: Adaptación de *Pangram Problem* (Kereki, 2017)

Warmup ICPC 2017

Problema Batalla por la Galaxia – dificultad: media

En una galaxia muy, muy lejana, dos facciones, los Rebeldes y el Imperio Galáctico, se enfrentan en una batalla crucial por el control de sistemas planetarios clave. Durante la batalla, se descubre que los mensajes codificados que contienen estrategias cruciales para ambos bandos se han mezclado en el sistema de transmisión. Estos mensajes deben ser restaurados y entregados rápidamente a cada facción para mantener sus oportunidades de victoria.



Los mensajes están compuestos por datos cifrados en forma de secuencias numéricas que representan coordenadas de ataque, órdenes tácticas y ubicaciones estratégicas en la galaxia. Sin embargo, debido a la interferencia en la transmisión, las secuencias numéricas han llegado mezcladas y necesitan ser ordenadas correctamente para ser decodificadas y utilizadas en la batalla.

Cada facción necesita desesperadamente reconstruir sus mensajes para anticipar los movimientos del enemigo y llevar a cabo sus propios ataques con precisión. Ambos bandos buscan un *Jedi* que, con su capacidad de discernimiento y sabiduría, pueda resolver este enigma y ordenar las secuencias numéricas para decodificar los mensajes.

El desafío implica reconstruir las secuencias numéricas mezcladas para obtener las coordenadas y órdenes correctas que permitirán a cada facción tomar decisiones estratégicas precisas en la batalla por el control de la galaxia.

Entrada	Salida
560	230
230	310
780	420
420	560
310	670
670	780

Problema Emojis a la antigua :) – dificultad: alta

Considere que antes de la era de los sistemas operativos gráficos la forma de comunicarnos vía chat era usando texto formato ASCII¹⁵. Mediante los emoticonos ASCII era posible representar todo tipo de emociones. Considere que volvemos a esa era y necesitamos que, a partir de un emoji antiguo y un número entero n , que representa la intensificación de una emoción, determine como salida la cantidad de veces que se repitió el emoji seguido de un texto que representa el significado de éste. La entrada debe ser leída en una única cadena para luego preprocesarla y extraer sus componentes (emoji, intensificador), use arreglos paralelos para mantener la lista de emojis y sus significados disponibles para la búsqueda. Resuelva el problema descomponiéndolo en subtareas (use funciones), considere incluir la biblioteca `<string.h>` y buscar documentación para las funciones `fgets`, `strcmp`, `strcpy`, `strtok`.



Entrada	Salida
:) 2	2 veces Feliz
:(4	4 veces Triste
:D 1	1 vez Muy feliz
:O 1	1 vez Sorprendido
;) 1	1 vez Guiñando un ojo
:P 2	2 veces Sacando la lengua
:'(3	3 veces Llorando
: 3	3 veces Neutral
:/ 1	1 vez Indeciso o dudoso
:3 5	5 veces Coqueto
:* 2	2 veces Besucón

El código fuente de cada solución a los problemas de este libro digital los encuentras en el repositorio complementario y público https://github.com/cmartinezUCSC/Resolucion_problemas_con_ADCP. También se incluye allí, un apartado **Extra** con nuevos ejercicios para practicar, incluyendo algunas pistas para orientar tu avance.



¹⁵ El código ASCII (American Standard Code for Information Interchange) es un estándar de codificación de caracteres que asigna valores numéricos únicos a los caracteres utilizados en ordenadores y dispositivos eléctricos.

References

- Buck, D. (director). (2012). *Midiendo el mundo*. Alemania: x filme creative pool. Disponible en <https://youtu.be/lpnhkksqii>
- Codeforces. (2021). *Competitive Programming Platform*. Disponible en <https://codeforces.com>
- CoolMath© (s.f.). Gauss's problem and arithmetic series. Disponible en <https://www.coolmath.com/algebra/19-sequences-series/06-gauss-problem-arithmetic-series-02>
- Fibonacci y el número de oro. (s.f.). Red de cerebros. Disponible en <https://www.youtube.com/watch?v=8bcyiuiIf2k>
- Halim, S., & Halim, F. (2013). Programación competitiva 3: Manual para concursantes del ICPC y la IOI (3ra ed.). Lulu.com.
- Kereki, I. (2017). Pangram problem. *ACM International Collegiate Programming Contest 2017– Warmup Session* (p.5). Disponible en <https://maratona.sbc.org.br/hist/2017/resultados/warmup.pdf>
- Martínez-Araneda, C. and Muñoz, M. (2014). ADPT: An active learning method for a programming lab course. In *10th International CDIO Conference, UPC Barcelona Tech, Spain*. Disponible en http://www.cdio.org/files/document/cdio2014/38/38_paper.pdf
- Microsoft Corporation. (2024). *Visual Studio Code [Software]* release 1.94. Disponible en <https://code.visualstudio.com>
- Novara, P. (2024.). *PSeInt [Software]* release 20230517. Disponible desde <http://pseint.sourceforge.net>
- Pereira, P. (2020). Non-Integer Donuts problem. In *ICPC Latin American Regional Contest – 2020* (p.20). Disponible desde <https://icpcarchive.github.io/Latin American Contests/Latin American Regional Contest/2020 Latin American Regional Contest/problems.pdf>
- Rodríguez, G. (2019.). *Zinjal [Software]* release 20191006. Disponible en <http://zinjal.sourceforge.net>
- Such, O. (s.f.). Sucesión de Perrin. *Numerentur*. Recuperado desde <https://numerentur.org/> Junio 2024
- TAP2012. (2012). High Mountains problem (H). En *Torneo Argentino de Programación – ACM-ICPC 2012*. Disponible en <https://vjudge.net/contest/631115#problem/H>
- Van Heesch, D. (s.f.). *Doxxygen [Software]*. Disponible en <http://www.doxxygen.nl>

Anexo - Plantilla ADCP

Plantilla ADCP

Problema:

Recuerde completar cada sección antes de codificar:

- El problema en extenso
- El problema resumido
- Análisis
- Diseño
- Codificación
- Pruebas

Resumen (no más de 3 líneas):

Análisis	
Entradas	<i>Identifique Los datos de entrada</i>
Proceso	<i>Paso 1</i>
	<i>Paso 2</i>
	<i>Paso 3</i>
Salidas	
Restricciones	<i>Incluya Las restricciones o validaciones a Las entradas</i>
Diseño (lenguaje natural, pseudocódigo, flujograma o diagrama N-S)	

Codificación

