Pontificia Universidad Católica del Perú

Estudios Generales Ciencias

1INF01 - Fundamentos de Programación

Guía de laboratorio #8

Diseño Estructurado

5 de noviembre de 2021

Índice general

Historial de Revisiones

Revisión	Fecha	Autor(es)	Descripción
1.0	04.11.2018	G.Colchado	Versión inicial.
1.0	08.11.2018	L.Hirsh	Versión inicial.
1.0	09.11.2018	E.García	Versión inicial.
1.1	07.06.2019	F.Huamán	Revisión e incremento de ejercicios.
1.1	08.06.2019	D.Allasi	Revisión e incremento de ejercicios.
2.0	03.11.2019	G.Colchado	Cambio de nombre de Guía #9 de Funciones a Diseño Estructurado y reestructuración del contenido. Adición de Técnica de Refinamientos Sucesivos con ejemplo de resta de dos fechas. Se tomaron 11 ejercicios propuestos de la Guía #5 de Programación Modular elaborada por C.Aguilera y D.Allasi. el 30.08.2019.
2.0	03.11.2019	J.Zárate	Revisión.
3.0	25.05.2021	S.Vargas	Modificación del número de la guía, guía 8. Actualización de la técnica de refinamiento sucesivos y del ejemplo de resta de fechas. Revisión, adición de casos de prueba y clasificación de ejercicios en nivel básico, intermedio y avanzado. Se añadieron ejercicios de los laboratorios del año 2020.
3.1	01.11.2021	J.Berrocal / S.Vargas	Se añadieron ejercicios de los laboratorios del ciclo 2021-1.

Siglas

EEGGCC	Estudios Generales Ciencias
IDE	Entorno de Desarrollo Integrado
PUCP	Pontificia Universidad Católica del Perú

Capítulo 1

Guía de Laboratorio #8

1.1. Introducción

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

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

- Comprenda el concepto de diseño estructurado.
- Comprenda la Técnica de refinamientos sucesivos.
- Implemente diseño estructurado en el lenguaje de programación C.

1.2. Materiales y métodos

Como lenguaje de programación imperativo se utilizará el lenguaje C. Como Entorno de Desarrollo Integrado (IDE) para el lenguaje C se utilizará Dev C++[1]. No obstante, es posible utilizar otros IDEs como Visual Studio Code.

1.3. Nociones Previas: Módulo o función

1.3.1. Definiendo un módulo o función

Existen ciertos casos donde queremos modularizar ciertas acciones y otras en las que tenemos bloques de instrucciones que necesitamos utilizar repetidas veces. Un ejemplo de este uso repetido de bloques de instrucciones son las funciones implementadas en las librerias stdio.h y math.h. Cada vez que necesitamos calcular la potencia de un número usamos la función predefinida `pow` sin necesidad de implementarla, pero esto lo podemos hacer únicamente porque alguien ya creó esas funciones y las agrupó en librerías. ¿Qué es lo que sucede si nosotros queremos definir nuestros propios bloques de acciones? En esos casos trabajamos con las funciones. Hasta ahora solo hemos utilizado funciones predefinidas en las librerías math.h y stdio.h como es el caso del `scanf`, `printf`, y `pow` para realizar tareas específicas (lectura de datos, impresión en pantalla, y

[1] http://sourceforge.net/projects/orwelldevcpp

cálculo de la potencia). Éstas funciones predefinidas han sido empleadas en distintos lugares de nuestro código y solo fue necesario conocer el tipo de dato de retorno, el nombre de la función, los parámetros y el orden de los mismos. Sin embargo, también es posible definir nuestras propias funciones, como bien se mencionó en el párrafo anterior, para encapsular un conjunto de instrucciones y, posteriormente, usar estas funciones en otras partes de nuestros programas.

Una función, por lo tanto, proporciona una forma conveniente de encapsular una tarea o un conjunto de instrucciones. Además, nos permite modularizar nuestros programas.

1.3.2. Implementación de un módulo o función en lenguaje C

En la figura 1.1 se puede apreciar la forma de implementación de una función en el lenguaje C. Debemos indicar un nombre para la función, así como también definir los parámetros que serán usados. Adicionalmente, es necesario indicar el tipo de retorno de la función.

```
tipo_de_retorno nombre_de_funcion(parametros){
    /*conjunto de instrucciones*/
    return <<variable_de_retorno>>
}
```

Figura 1.1: Lenguaje C: Estructura de una función

Donde:

- El tipo de retorno puede ser double, int, char, void.
- nombre_de_funcion es el nombre descriptivo que nos ayudará a identificar la función.
- Los parámetros son aquello que necesita la función para llevar a cabo el conjunto de instrucciones, por ejemplo, en el caso de la funcion potencia se necesitarían como parámetros la base y el exponente. Los parámetros pueden ser de dos tipos: por valor (recibe un valor) y por referencia (recibe una dirección de memoria).
- El conjunto de instrucciones son todas las acciones que se deben llevar a cabo con el fin de obtener el valor a retornar.
- Las llaves indican el inicio y fin del conjunto de instrucciones.

Tener en cuenta que:

Es importante prestar atención a las variables que declaremos dentro de las funciones, pues estas no estarán disponibles fuera de las mismas. Los parámetros de la función son también variables y pueden usarse con cualquier instrucción.

1.3.3. Tipos de retorno de un módulo o función en lenguaje C

Las funciones pueden tener los siguientes tipos de retorno: double, int, char, void.

Es importante tener en cuenta que para asignar el valor de retorno de una función a una variable, ambos deben ser del mismo tipo. Así, por ejemplo, una función con tipo de retorno double deberá retornar el valor de una variable de tipo double.

Tener en cuenta que:

void es un tipo de retorno *vacio* que permite que la función no devuelva un valor en particular. Por lo tanto, las funciones que tengan este tipo de retorno no hacen uso de la instrucción *return*.

1.3.4. Invocar a un módulo o función en lenguaje C

Para utilizar una función que ya ha sido definida es necesaria invocarla o llamarla. Lo más frecuente es asignar a una variable el valor que es devuelto por la función y a la función se le debe llamar dándole los parámetros que requiera pero sin incluir los tipos de datos de dichos parámetros, ver figura 1.2.

```
nombre_de_variable = nombre_de_funcion ( parametros );
```

Figura 1.2: Lenguaje C: Invocación a función

Nota: En caso el tipo de retorno de la función sea **void**, se llama directamente a la función sin asignarla a una variable.

1.4. Nociones Previas: Análisis - Diagrama de módulos

Como habíamos visto anteriormente en clase, las fases para resolver problemas son:

- **Definición**, consiste en entender claramente el problema.
- **Análisis**, busca describir el problema en función de los datos de entrada, los datos de salida y las precondiciones(que deben cumplir los datos de entrada).
- **Diseño**, se realiza el diseño del algoritmo propuesto para la solución. Dentro de las formas que se conocen para representar algoritmos hemos utilizado diagramas de flujo y pseudocódigos. En esta fase se usa la estrategia `divide y vencerás`, para dividir un problema en módulos o funciones más pequeñas y fáciles de resolver.
- **Codificación**, se implementa el diseño de la solución en un programa. Para esto se utiliza un lenguaje de programación, en este curso hemos usado el lenguaje de programación C.
- **Ejecución y pruebas**, se ejecuta el programa y se realizan diversas pruebas para verificar que el programa funciona correctamente. Se suelen usar tablas de verificación, ejecución paso a paso, entre otras herramientas.

La división de un problema en módulos o funciones más pequeños durante la fase de **Diseño**, se puede representar mediante los diagramas de módulos que permiten estructurar los programas de modo que sean más fáciles de implementar y entender, permitiendo la identificación de bloques de instrucciones que pueden encapsularse en funciones. De este modo, un programa puede estructurarse como un conjunto de módulos y relaciones tal como se aprecia en la figura 1.3.

Es importante tener en cuenta que los nombres de los módulos deben ser descriptivos y expresar la acción que se va a llevar a cabo en ellos. Además, debemos recordar que las flechas indican el orden de invocación de cada módulo.

En el siguiente ejemplo se puede apreciar como un problema se puede dividir en módulos y representarse mediante un diagrama de módulos.

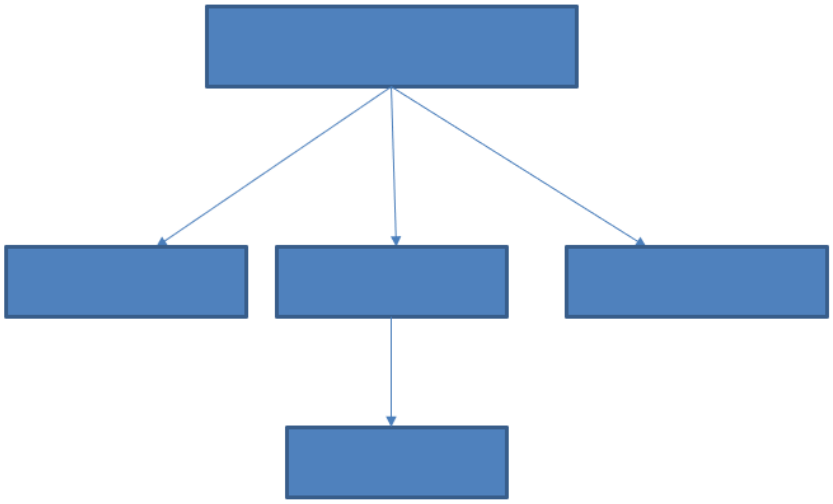

Figura 1.3: Representación de un diagrama de módulos

1.4.1. Cálculo del crecimiento exponencial

El número e es un número muy usado en las ciencias[2], jugando un papel importante en el cálculo de la función exponencial. Las primeras referencias a este número datan del año 1618 en un trabajo de John Napier sobre logaritmos. Al igual que π, el número e es un número irracional del cual no podemos conocer su valor exacto porque tiene infinitas cifras decimales. El valor aproximado de este número es $2,71828$ (aproximado a 5 decimales).

Existen varias definiciones del número e siendo la más común el $\lim_{n\to\infty}\left(1+\frac{1}{n}\right)^n$. Este límite puede aproximarse con la siguiente serie $1+\frac{1}{1!}+\frac{1}{2!}+\frac{1}{3!}+\frac{1}{4!}+\frac{1}{5!}+\frac{1}{6!}+\cdots$

Una de las múltiples aplicaciones en biología del número e es el crecimiento exponencial. Este tiene lugar en situaciones en las que no hay factores que limiten el crecimiento de la magnitud de interés (por ejemplo, la cantidad de bacterias en un medio determinado).

El valor de la magnitud, por lo tanto, crece cada vez más rápido en el tiempo, de acuerdo con la siguiente fórmula:

$$Magnitud\_instante\_mayor\_0 = Magnitud\_instante\_igual\_0 \cdot e^t$$

Donde:
$Magnitud\_instante\_mayor\_0$: valor de la magnitud en el instante $t > 0$
$Magnitud\_instante\_igual\_0$: valor de la magnitud en el instante $t = 0$

Teniendo en cuenta este concepto, podemos implementar una aplicación que nos permita calcular el valor final de la magnitud en el tiempo $t > 0$. En la imagen 1.4 se propone un diagrama de módulos para resolver el problema.

[2]Para entender un poco más a profundidad qué es el número e, se recomienda el siguiente video https://www.youtube.com/watch?v=Z5czpA-fyMU.

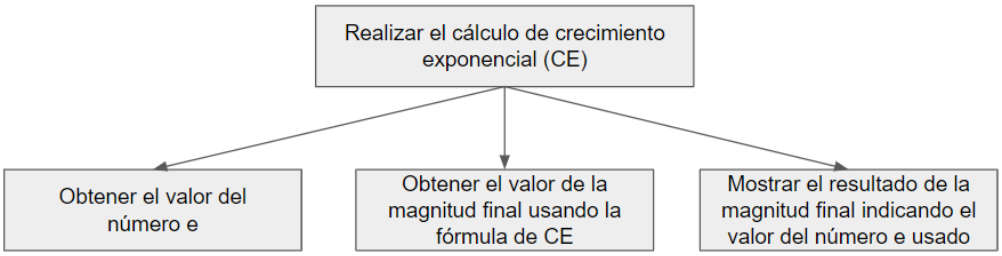

Figura 1.4: Módulos del programa de cálculo del Crecimiento Exponencial

Solución propuesta para el cálculo del crecimiento exponencial

En el Programa 1.1 se muestra la solución propuesta al problema del cálculo del crecimiento exponencial según el diagrama de módulos de figura 1.4. Note que cada módulo al final de las flechas del diagrama de módulos se corresponde con una función del programa y la función main corresponde al módulo de más arriba que llama a los otros tres módulos.

Programa 1.1: Cálculo del crecimiento exponencial

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

double obtener_numero_e(int );
double obtener_crec_expo(int , double , double );
void mostrar_magnitud_final_con_e(int , double , double );

int main() {
    int tiempo;
    double magnitud_instante_mayor_0, magnitud_instante_igual_0, valor_numero_e;

    /*Obtenemos el valor del número e usando la fúncion obtener_numero_e()*/
    valor_numero_e = obtener_numero_e(100);

    printf("Ingrese el valor de la magnitud en el instante 0: ");
    scanf("%lf", &magnitud_instante_igual_0);

    printf("Ingrese el tiempo final (t>0): ");
    scanf("%d", &tiempo);

    if (tiempo > 0 && magnitud_instante_igual_0 > 0) {

        /*Obtenemos el valor de la magnitud en tiempo > 0 usando la fúncion obtener_crec_expo()*/
        magnitud_instante_mayor_0 = obtener_crec_expo(tiempo, magnitud_instante_igual_0, valor_numero_e);

        /*Mostramos el valor de la magnitud final (tiempo >0) con valor del número e usado */
        mostrar_magnitud_final_con_e(tiempo, magnitud_instante_mayor_0, valor_numero_e);
    } else {
        printf("Los valores para el tiempo y la magnitud inicial deben ser positivos\n");
    }
}

double obtener_numero_e(int num_terminos_serie) {
    int i = 0;
    double suma = 0;
    double factorial = 1;
    double termino = 1;
    while (i <= num_terminos_serie) {
        if (i == 0)
            suma = suma + 1;
        else {
            factorial = factorial * i;
            termino = (double) 1 / factorial;
            suma = suma + termino;
```

```
        }
        i++;
    }
    return suma;
}

double obtener_crec_expo(int valor_tiempo, double valor_magnitud_instante_igual_0, double valor_numero_e) {
    double magnitud_instante_mayor_0;
    magnitud_instante_mayor_0 = valor_magnitud_instante_igual_0 * pow(valor_numero_e, valor_tiempo);
    return magnitud_instante_mayor_0;
}

void mostrar_magnitud_final_con_e(int tiempo, double magnitud_instante_mayor_0, double valor_numero_e){

    printf("El valor de la magnitud en el instante t=%d es: %lf\n", tiempo, magnitud_instante_mayor_0);
    printf("El valor aproximado del número e que ha usado es: %lf", valor_numero_e);
}
```

A continuación sigue un ejemplo de una posible salida de este programa:

```
Ingrese el valor de la magnitud en el instante 0: 5
Ingrese el tiempo final (t>0): 3
El valor de la magnitud en el instante t=3 es: 100.427685
El valor aproximado del número e que ha usado es: 2.718282
```

1.5. Diseño Estructurado

El diseño estructurado tiene las características de la programación modular donde se divide un problema en módulos más pequeños y fáciles de resolver.

En el diseño estructurado se divide un problema en módulos siguiendo el diseño descendente (Top-down) mediante la **técnica de refinamientos sucesivos**.

Usando la **técnica de refinamientos sucesivos**, un problema se divide en dos o más subproblemas o módulos más fáciles de resolver, sin embargo cuando el diseñador analiza cada nuevo módulo se puede dar cuenta de que algunos de ellos debe dividirlos más y así sucesivamente hasta que se obtengan módulos pequeñosque sean fáciles de programar y en ese momento detiene el diseño descendente o Top-down.

Mientras el diseñador aplica la **técnica de refinamientos sucesivos** va elaborando el diagrama de módulos que muestra de forma jerárquica o descendente los diferentes módulos para resolver el problema.

En el siguiente ejemplo se puede apreciar como se aplica la **técnica de refinamientos sucesivos** a la solución de un problema.

1.5.1. Resta de dos fechas

Elabore un programa en lenguaje C que permita ingresar dos fechas, restarlas y mostrar el resultado en días. Para las fechas se ingresarán en forma independiente los días, meses y años (Ejemplo: 1 3 2019).

Antes de realizar la resta, debe comprobar que las fechas sean válidas y que la segunda fecha sea mayor que la primera. En el caso del año debe validar que sean años comprendidos en el Siglo XX y XXI (Del 1900 al 2099).

Considere que los años bisiestos tienen 1 día más en el mes de Febrero, es decir 29 días. Un año es bisiesto si cumple lo siguiente: es divisible por 4 y no lo es por 100, o es divisible por 400.

A continuación se muestran los casos de prueba para este programa:

```
Ingrese fecha inicial: 1 11 2019
Ingrese fecha final: 16 11 2019
La resta en días es 15
```

```
Ingrese fecha inicial: 1 2 2019
Ingrese fecha final: 28 10 2019
La resta en días es 269

Ingrese fecha inicial: 1 1 2019
Ingrese fecha final: 1 1 2022
La resta en dias es 1096

Ingrese fecha inicial: 32 13 4000
Año no válido
Mes no válido
Día no válido

Ingrese fecha inicial: 1 1 2019
Ingrese fecha final: 32 40 3001
Año no válido
Mes no válido
Día no válido

Ingrese fecha inicial: 1 1 2020
Ingrese fecha final: 23 12 2019
La fecha final no es mayor a la inicial
```

1.5.2. Técnica de Refinamientos Sucesivos

Refinamiento 1

El problema de la resta de dos fechas podemos dividirlo en 3 subproblemas o módulos que resuelvan todo el problema del enunciado tal como se aprecia en la figura 1.5.

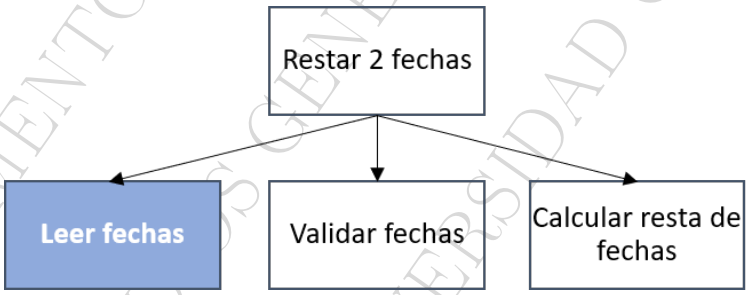

Figura 1.5: Refinamiento 1 - Diagrama de módulos

Si analizamos cada nuevo módulo vemos lo siguiente:

- El módulo "Leer fechas" es pequeño, tiene un objetivo específico y tiene claramente definido lo que se desarrollará. Lo sombrearemos en celeste. Podemos declararlo de esta manera:

  void leerFechas(int *diaIni, int *mesIni, int *anioIni, int *diaFin, int *mesFin, int *anioFin)

- El módulo "Validar fechas", debe realizar varias tareas, aún no tiene un solo objetivo específico, por lo que podemos seguir refinándolo. Podemos declararlo de esta manera:

  int validarFechas(int diaIni, int mesIni, int anioIni, int diaFin, int mesFin, int anioFin)

- El módulo "Calcular resta de fechas", debe realizar varias tareas, aún no tiene un solo objetivo específico, por lo que podemos seguir refinándolo. Podemos declararlo de esta manera:

  int calcularRestaFechas(int diaIni, int mesIni, int anioIni, int diaFin, int mesFin, int anioFin)

Por lo tanto necesitamos seguir refinando módulos.

Refinamiento 2

Se empieza a realizar el refinamiento solamente de los módulos pendientes del último nivel del diagrama de módulos. El módulo "Validar fechas" podemos dividirlo en el módulo:

- El módulo "Validar una fecha", debe realizar varias tareas, aún no tiene un solo objetivo específico, por lo que podemos seguir refinándolo. Podemos declararlo de esta manera: **int validarFecha(int dia, int mes, int anio)**

El módulo "Calcular resta de fechas" podemos dividirlo en los siguientes módulos:

- El módulo "Calcular días iniciales del mes inicio", debe realizar varias tareas, aún no tiene un solo objetivo específico, por lo que podemos seguir refinándolo. Podemos declararlo de esta manera:

  int calcularDiasMesIni(int diaIni,int mesIni,int anioIni)

- El módulo "Calcular días entre meses", debe realizar varias tareas, aún no tiene un solo objetivo específico, por lo que podemos seguir refinándolo. Podemos declararlo de esta manera:

  int calcularDiasEntreMeses(int mesIni,int anioIni,int mesFin,int anioFin)

En la figura 1.6 podemos ver el diagrama de módulos luego del segundo refinamiento.

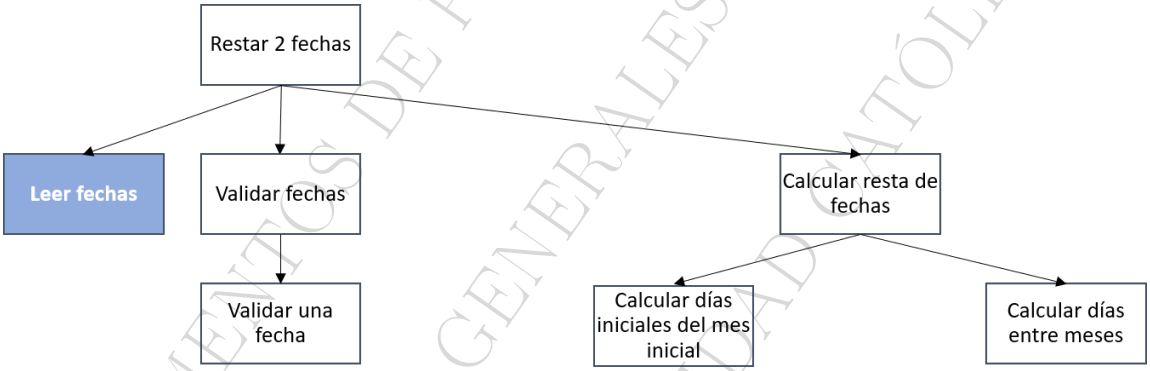

Figura 1.6: Refinamiento 2 - Diagrama de módulos

Por lo tanto necesitamos seguir refinando módulos.

Refinamiento 3

Se empieza a realizar el refinamiento solamente de los módulos pendientes del último nivel del diagrama de módulos.

El módulo "Validar una fecha" podemos dividirlo en el siguiente módulo:

- El módulo "Calcular días del mes", debe realizar varias tareas, aún no tiene un solo objetivo específico, por lo que podemos seguir refinándolo. Por ejemplo, si el mes es febrero se debe identificar si el año es bisiesto. Podemos declararlo de esta manera:

  int calcularDiasMes(int mes, int anio)

Para los módulos "Calcular días iniciales del mes inicio" y "Calcular días entre meses" identificamos que requieren usar el módulo anterior que hemos definido "Calcular días del mes".

En la figura 1.7 podemos ver el diagrama de módulos luego del tercer refinamiento.

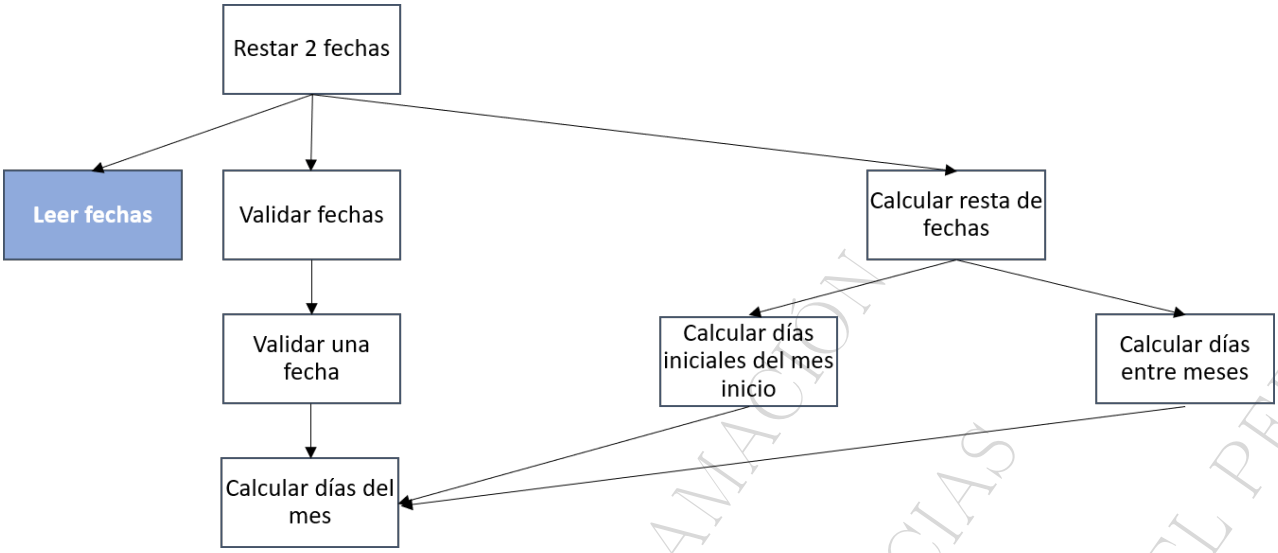

Figura 1.7: Refinamiento 3 - Diagrama de módulos

Refinamiento 4

El único módulo que queda pendiente para refinar es "Calcular días del mes", el cual lo dividimos en el módulo:

- El módulo "Validar año bisiesto" permitirá identificar si el año es bisiesto. Por lo tanto es pequeño, tiene un objetivo específico y tiene claramente definido lo que se desarrollará. Lo sombrearemos en celeste. Podemos declararlo de esta manera:

  int validarAnioBisiesto(int anio)

En la figura 1.8 podemos ver el diagrama de módulos luego del cuarto refinamiento y podemos observar que todos los módulos de los últimos niveles están en celeste y por lo tanto se detiene el refinamiento sucesivo. En este momento ya está definido claramente lo que hay que hacer por lo que se puede empezar a programar los módulos.

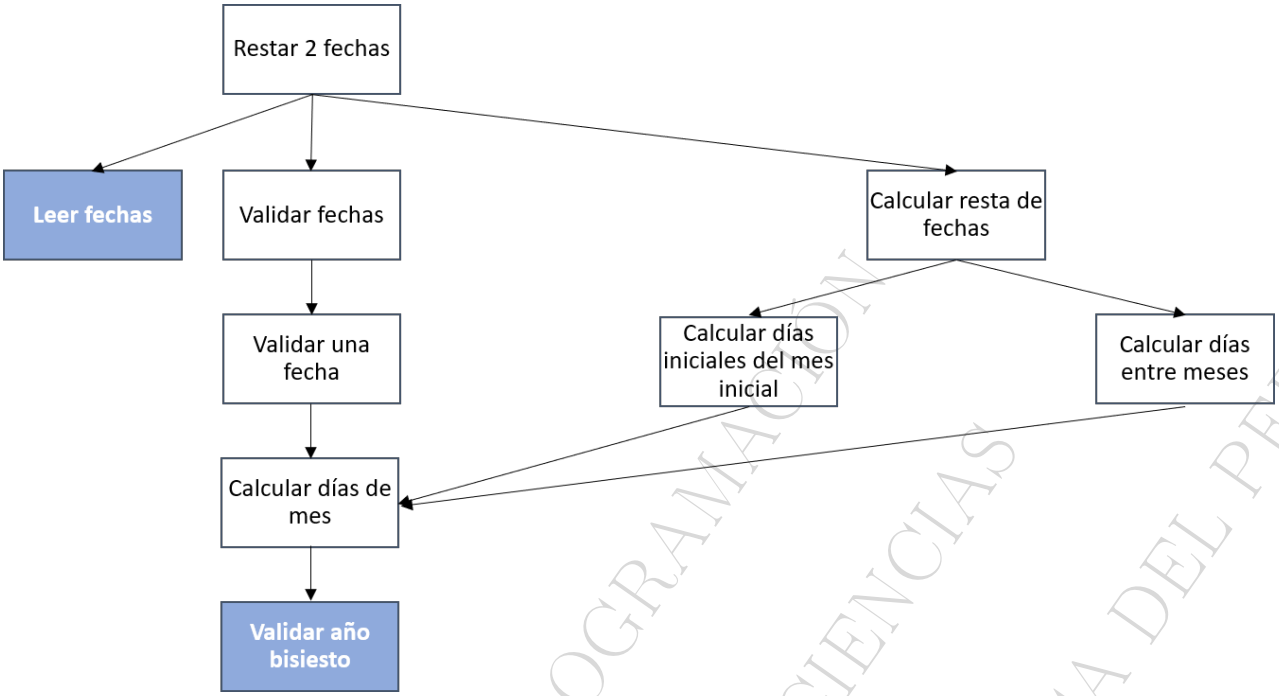

Figura 1.8: Refinamiento 4 - Diagrama de módulos

Programa en lenguaje C

En el Programa 1.2 se muestra la solución propuesta al problema de la resta de dos fechas de acuerdo al diseño del diagrama de módulos final del refinamiento 4. Note que la función main corresponde al módulo de nivel superior en el diagrama de módulos "Restar dos fechas" que a su vez sólo llama a los módulos "Leer fechas", "Validar fechas" y "Calcular resta de fechas" y así sucesivamente siguiendo el diseño descendente (Top-down).

Programa 1.2: Resta de dos fechas

```
#include <stdio.h>
#define ANNO_MIN 1900 /* Siglo XX */
#define ANNO_MAX 2099 /* Siglo XXI */

void leerFechas(int *, int *, int *, int *, int *, int *);
int validarFechas(int , int , int , int diFin, int , int );
int validarFecha(int , int , int );
int calcularDiasMes(int , int );
int validarAnioBisiesto(int );
int calcularRestaFechas(int , int , int , int , int , int );
int calcularDiasMesIni(int ,int ,int );
int calcularDiasEntreMeses(int ,int ,int ,int );

int main() {
        int diaIni, mesIni, anioIni, diaFin, mesFin, anioFin,restaDias;
        leerFechas(&diaIni, &mesIni, &anioIni, &diaFin, &mesFin, &anioFin);
        if (validarFechas(diaIni, mesIni, anioIni, diaFin, mesFin, anioFin)){
                restaDias=calcularRestaFechas(diaIni, mesIni, anioIni, diaFin, mesFin, anioFin);
                printf("La resta en días es  %d\n", restaDias);
        }
        else
                printf("La fecha final no es mayor a la inicial\n");
        return 0;
}

void leerFechas(int *diaIni, int *mesIni, int *anioIni, int *diaFin, int *mesFin, int *anioFin){
        printf("Ingrese el dia, mes y a de la fecha inicial: ");
        scanf("%d %d %d",diaIni,mesIni,anioIni);
        printf("Ingrese el dia, mes y a de la fecha final: ");
```

```
30          scanf("%d %d %d",diaFin,mesFin,anioFin);
31  }
32
33  int validarFechas(int diaIni, int mesIni, int anioIni, int diaFin, int mesFin, int anioFin){
34          int fecIniValida,fecFinValida,fecIniMayor;
35          fecIniValida=validarFecha(diaIni,mesIni,anioIni);
36          fecFinValida=validarFecha(diaFin,mesFin,anioFin);
37          if (fecIniValida && fecFinValida){
38                  fecIniMayor=(anioFin*10000 + mesFin*100 + diaFin) > (anioIni*10000 + mesIni*100 + diaIni);
39                  if (fecIniMayor)
40                          return 1;
41                  else{
42                          printf("Las fecha final no es mayor que la inicial\n");
43                          return 0;
44                  }
45          }
46          else{
47                  printf("Las fechas ingresadas no son correctas\n");
48                  return 0;
49          }
50          return validarFecha(diaIni,mesIni,anioIni) && validarFecha(diaFin,mesFin,anioFin);
51  }
52
53  int validarFecha(int dia, int mes, int anio) {
54          int diaValido,mesValido,anioValido;
55          diaValido=dia>=1 && dia<=calcularDiasMes(mes,anio);
56          mesValido=mes>=1 && mes<=12;
57          anioValido=anio>=ANNO_MIN && anio<=ANNO_MAX;
58          if (!diaValido) {
59                  printf("Día no válido\n");
60                  return 0;
61          }
62          else
63                  if (!mesValido) {
64                          printf("Mes no válido\n");
65                          return 0;
66                  }
67                  else
68                          if (!anioValido) {
69                                  printf("Año no válido\n");
70                                  return 0;
71                          }
72                          else
73                                  return 1;
74  }
75
76
77  int calcularDiasMes(int mes, int anio) {
78          if (mes==1 || mes==3 || mes==5 || mes==7 || mes==8 || mes==10 || mes==12)
79                  return 31;
80          else
81                  if (mes==4 || mes==6 || mes==9 || mes==11)
82                          return 30;
83          else
84                  if (validarAnioBisiesto(anio))
85                          return 29;
86                  else
87                          return 28;
88  }
89
90  int validarAnioBisiesto(int anio) {
91          return (anio % 4 == 0 && anio % 100 != 0) || anio % 400 == 0;
92  }
93
94  int calcularRestaFechas(int diaIni, int mesIni, int anioIni, int diaFin, int mesFin, int anioFin){
95          int diasMesIni,diasMeses=0,diasMesFin=0;
96          if (mesIni==mesFin && anioIni==anioFin)
97                  return diaFin-diaIni;
98          else{
99                  diasMesIni=calcularDiasMesIni(diaIni,mesIni,anioIni);
100                 diasMeses=calcularDiasEntreMeses(mesIni+1,anioIni,mesFin-1,anioFin);
```

```
101                     return diasMesIni+diasMeses+diaFin;
102             }
103     }
104
105     int calcularDiasMesIni(int diaIni,int mesIni,int anioIni){
106             int diasMaxMesIni;
107             diasMaxMesIni=calcularDiasMes(mesIni,anioIni);
108             return diasMaxMesIni-diaIni;
109     }
110
111     int calcularDiasEntreMeses(int mesIni,int anioIni,int mesFin,int anioFin){
112             int mesFinalAnioIni,mesInicialAnoFin,i,j,diasAnioIni=0,diasEntreAnios=0,diasAnioFin;
113             if (anioFin>anioIni){
114                     mesFinalAnioIni=12;
115                     mesInicialAnoFin=1;
116             }
117             else{
118                     mesFinalAnioIni=mesFin;
119                     mesInicialAnoFin=13;
120             }
121
122             for (i=mesIni;i<=mesFinalAnioIni;i++)
123                             diasAnioIni+=calcularDiasMes(i,anioIni);
124
125             for (j=anioIni+1;j<=anioFin-1;j++)
126                     for (i=1;i<=12;i++)
127                                     diasEntreAnios+=calcularDiasMes(i,j);
128
129             for (i=mesInicialAnoFin;i<=mesFin;i++)
130                     diasAnioFin+=calcularDiasMes(i,anioFin);
131
132             return  diasAnioIni+diasEntreAnios+diasAnioFin;
133     }
```

Capítulo 2

Ejercicios propuestos

Para los siguientes ejercicios propuestos, utilice la técnica de refinamientos sucesivos y proponga un diagrama de módulos. Luego implemente el diagrama de módulos en un programa con funciones en lenguaje C.

2.1. Nivel básico

2.1.1. Movimiento Armónico Simple

El movimiento armónico simple (MAS) es un movimiento rectilíneo realizado por un móvil que es oscilatorio y periódico, donde su aceleración siempre señala hacia la posición de equilibrio y su magnitud es directamente proporcional a la distancia del móvil a la posición de equilibrio.

Algunas fórmulas que se utilizan para realizar cálculos de este movimiento son las siguientes:

- $T = \frac{2*\pi}{w}$. , en s
- $f = \frac{1}{T}$. , en Hz

Donde T = período, f = frecuencia de oscilación y w = frecuencia cíclica.

Se le pide implementar un programa en lenguaje C que solicite al usuario ingresar la frecuencia de oscilación de un MAS, calcule y muestre su período y frecuencia cíclica.

Utilice los siguientes casos de prueba para verificar su solución:

```
Ingrese la frecuencia de oscilación: 12.3
Periodo: 0.081301 s, Frecuencia Cíclica: 77.283163 rad/s

Ingrese la frecuencia de oscilación: 7.8
Periodo: 0.128205 s, Frecuencia Cíclica: 49.008835 rad/s

Ingrese la frecuencia de oscilación: 18.6
Periodo: 0.053763 s, Frecuencia Cíclica: 116.867222 rad/s
```

2.1.2. El pentágono regular

Un pentágono regular es aquel que tiene todos sus lados iguales y sus ángulos internos congruentes. Ver figura 2.1.

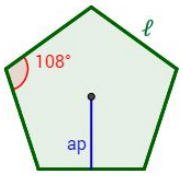

Figura 2.1: Pentágono Regular

De la figura 2.1 se puede apreciar que l es la longitud de sus lados, el ángulo interno es 108° y ap representa a la apotema del pentágono.

Para realizar el cálculo del área del pentágono regular se utiliza la siguiente fórmula:

- $Area = \frac{5*l*ap}{2}$.
- Para el cálculo del apotema se utiliza la siguiente fórmula: $ap = \sqrt{r^2 - (\frac{l}{2})^2}$, donde r es la distancia que hay desde el centro del polígono a uno de los vértices del polígono.

Se le pide implementar un programa en lenguaje C que solicite al usuario ingresar el lado del pentágono regular y la distancia que hay desde el centro del pentágono a uno de sus vértices, calcule y muestre su apotema y el área del pentágono regular.

Utilice los siguientes casos de prueba para verificar su solución:

```
Ingrese el lado y la distancia al centro del pentágono: 4 3
Apotema=2.236068, Área=22.360680

Ingrese el lado y la distancia al centro del pentágono: 8.2 5.5
Apotema=3.666061, Área=75.154241

Ingrese el lado y la distancia al centro del pentágono: 16.2 12.78
Apotema=9.885262, Área=400.353101
```

2.1.3. Coordenadas Polares y Cartesianas

En el sistema de coordenadas existen dos muy conocidas como son las coordenadas polares y las coordenadas cartesianas.

- Un punto P en coordenadas polares se representa por un par de números (r;θ) donde r es la distancia del polo al punto dado y donde θ es el ángulo de inclinación del radio vector OP con respecto al semi-eje positivo llamado eje polar, tal como lo muestra la figura 2.2.
- Un punto P en coordenadas cartesianas se representa por un par de número (a,b) donde a es una posición en el plano X y b es una posición en el plano Y.

Para transformar un punto P(x,y) del plano cartesiano en su representación polar se utiliza la siguiente fórmula:

- $r^2 = x^2 + y^2$
- $\theta = arctg(\frac{y}{x})$
- Finalmente (r;θ) = (r*cos(θ);r*sen(θ)) = (x,y)

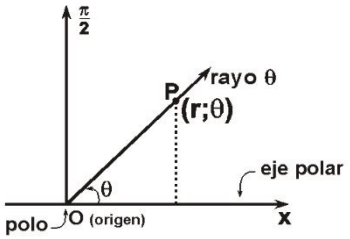

Figura 2.2: Punto en el sistema de coordenadas polar

Se le pide implementar un programa en lenguaje C que lea un punto P(x,y), calcule y muestre su representación en coordenadas polares.

Utilice los siguientes casos de prueba para verificar su solución:

```
Ingrese el punto en coordenadas cartesianas (x,y):5 5
Punto en coordenadas polares (r,ángulo): (7.071068,0.785398)

Ingrese el punto en coordenadas cartesianas (x,y):12.3 3.78
Punto en coordenadas polares (r,angulo): (12.867727,0.298156)

Ingrese el punto en coordenadas cartesianas (x,y):6.78  12.45
Punto en coordenadas polares (r,ángulo): (14.176421,1.072125)
```

2.1.4. Las rectas paralelas y perpendiculares

Cuando graficas dos o más ecuaciones lineales en el plano de coordenadas, generalmente se cruzan en algún punto. Sin embargo, cuando dos rectas en un plano coordenado nunca se cruzan, se llaman *rectas paralelas*. También veremos el caso cuando dos rectas en el plano de coordenadas se cruzan en un ángulo recto. Estas se llaman *rectas perpendiculares*. Las pendientes de las gráficas en cada uno de los casos tienen una relación especial entre ellas:

- Dos rectas son paralelas si sus pendientes son iguales.
- Dos rectas son perpendiculares si el producto de sus pendientes da el valor de -1.

Se le pide que desarrolle un programa en lenguaje C que permita leer el valor de 4 puntos P(x1,y1), Q(x2,y2), R(x3,y3), S(x4,y4) y determine si la recta formada por los puntos P y Q es paralela, perpendicular o simplemente se cruza con la recta formada por los puntos S y R. Debe mostrar un mensaje indicando que tipos de rectas son.

Utilice los siguientes casos de prueba para verificar su solución:

```
Ingresa los valores x e y para el punto P: 3 7
Ingresa los valores x e y para el punto Q: -5 -1
Ingresa los valores x e y para el punto R: -3 5
Ingresa los valores x e y para el punto S: 4 -4
La recta formada por los puntos P y Q es perpendicular a la recta formada por los puntos R y S.

Ingresa los valores x e y para el punto P: 1 2
Ingresa los valores x e y para el punto Q: 3 0
Ingresa los valores x e y para el punto R: 3 3
Ingresa los valores x e y para el punto S: 6 0
La recta formada por los puntos P y Q es paralela a la recta formada por los puntos R y S.
```

```
Ingresa los valores x e y para el punto P: 6 4
Ingresa los valores x e y para el punto Q: 3 1
Ingresa los valores x e y para el punto R: 3 3
Ingresa los valores x e y para el punto S: 6 0
La recta formada por los puntos P y Q cruza a la recta formada por los puntos R y S.
```

Sugerencia

En el refinamiento sucesivo puede considerar los siguientes módulos:

- Calcular pendiente de una recta
- Determinar si dos rectas son paralelas
- Determinar si dos rectas son perpendiculares

2.1.5. Los polinomios

Las expresiones algebraicas que se forman a partir de la unión de dos o más variables y constantes, vinculadas a través de operaciones de multiplicación, resta o suma, reciben el nombre de polinomios. Existen varios tipos de polinomios, siendo uno de ellos el polinomio completo. Un polinomio se denomina completo si tiene todos los términos desde el término independiente hasta el término de mayor grado. El grado de un polinomio es el mayor exponente al que se encuentra elevada una variable.

Se le pide implementar un programa en lenguaje C que lea el grado de un polinomio, lea todos los coeficientes asociados a los términos del polinomio de menor a mayor grado y determine si se trata de un polinomio completo. En caso el coeficiente que se ingrese sea 0, se considera que dicho termino no se encuentra en el polinomio.

Utilice los siguientes casos de prueba para verificar su solución:

Para el polinomio $3x^3 + 2x + 5$ sería:

```
Ingrese el grado del polinomio: 3
Ingrese el coeficiente del termino 0: 5
Ingrese el coeficiente del termino 1: 2
Ingrese el coeficiente del termino 2: 0
Ingrese el coeficiente del termino 3: 3
El polinomio ingresado NO es completo.
```

Para el polinomio $5x^4 - 2x^3 + x^2 + 8x - 5$ sería:

```
Ingrese el grado del polinomio: 4
Ingrese el coeficiente del termino 0: -5
Ingrese el coeficiente del termino 1: 8
Ingrese el coeficiente del termino 2: 1
Ingrese el coeficiente del termino 3: -2
Ingrese el coeficiente del termino 4: 5
El polinomio ingresado SI es completo.
```

Sugerencia:

- Al ser un programa muy sencillo y como se debe utilizar la técnica de refinamientos sucesivos, se le sugiere elaborar módulos para la lectura e impresión de datos.

2.1.6. Función Gaussiana

La función Gaussiana (en honor a Carl Friedrich Gauss) es una función ampliamente usada en estadística cuya gráfica tiene una foma característica de campana como puede apreciarse en la figura 2.3.

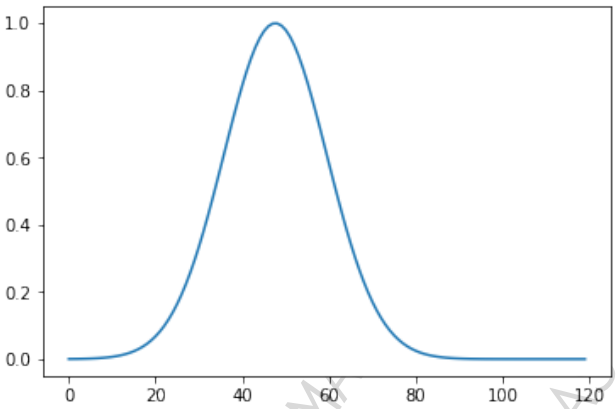

Figura 2.3: Curva de distribución normal - Campana de Gauss

A esta curva también se le conoce como curva de distribución normal dado que permite modelar muchos fenómenos naturales, sociales y psicológicos.

La función Gaussiana está definida por la siguiente fórmula:

$$f(x) = e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Donde:

- $f(x)$ es el valor de la función para el punto x
- μ es media de los valores en el conjunto de datos $X = \{x_1, x_2, ..., x_n\}$
- σ es el valor de la desviación estándar

Se le solicita que elabore el diseño y la implementación de un programa que permita conocer el valor de la función $f(x)$ para un punto dado x considerando que se conocen los valores de la media μ y desviación estándar σ.

El programa en cuestión deberá solicitar al usuario los valores de la media y desviación estándar solo una vez, pero solicitará el valor de x las veces que sea necesario en caso el usuario desee calcular múltiples valores de $f(x)$ (por ejemplo, para graficar varios puntos en un plano catersiano). El programa deberá finalizar cuando el usuario ingrese el valor de -999.999 como valor de x. Para el valor de e puede usar el valor 2.71828.

Use el siguiente caso de prueba para verificar su solución:

```
Ingrese la media y la desviacion estandar: 1.1 0.1
Ingrese el valor de x: 1.05
x=1.050000, f(x)=0.882497
Ingrese el valor de x: 1.1
x=1.100000, f(x)=1.000000
Ingrese el valor de x: 1.22
x=1.220000, f(x)=0.486752
Ingrese el valor de x: 0.99
x=0.990000, f(x)=0.546075
Ingrese el valor de x: 1.08
```

```
x=1.080000, f(x)=0.980199
Ingrese el valor de x: 999.999
Fin de la evaluación
```

2.1.7. Calculadora de conversión de unidades

Se solicita implementar un programa que muestre un menú con las siguientes opciones de forma iterativa hasta que usuario elija la opción 5. Salir:

- 1. Conversión de grados sexagesimales a radianes
- 2. Conversión de grados centígrados a grados farenheit
- 3. Conversión de centímetros a pulgadas
- 4. Conversión de metros a pies
- 5. Salir

Según la opción elegida se deberá pedir al usuario ingresar el valor a convertir en las unidades respectivas y mostrar el valor convertido en las unidades respectivas. Las fórmulas para realizar la conversión de unidades son las siguientes:

- grado sexagesimal = $\pi/180$ radianes
- grado farenheit = $9/5$(grado centígrado) + 32
- centímetro = $1/2.54$ pulgadas
- metro = 3.28084 pies

Use el siguiente caso de prueba para verificar su solución:

```
Calculadora de conversion de unidades
1. Conversión de grados sexagesimales a radianes
2. Conversión de grados centígrados a grados Farenheit
3. Conversión de centímetros a pulgadas
4. Conversión de metros a pies
5. Salir
Ingrese la opcion para realizar la conversión: 1
Ingrese dimension a convertir: 90
90 grados sexagesimales son equivalentes a 1.570796 radianes
Ingrese la opcion para realizar la conversión: 2
Ingrese dimension a convertir: 120
120 grados centígrados son equivalentes a 248 grados Farenheit
Ingrese la opcion para realizar la conversión: 3
Ingrese dimension a convertir: 55.7
55.7 centímetros son equivalentes a 21.929134 pulgadas
Ingrese la opcion para realizar la conversión: 4
Ingrese dimension a convertir: 89.74
89.74 centímetros son equivalentes a 294.422582 pies
Ingrese la opcion para realizar la conversión: 5
Fin de la conversión
```

2.1.8. Sumatorias de números naturales

Se solicita implementar un programa que muestre un menú con las siguientes opciones de forma iterativa hasta que usuario elija la opción 3. Salir:

- 1. Suma numeros naturales compuestos
- 2. Suma cuadrados de numeros naturales
- 3. Salir

Se debe solicitar el ingreso de dos números naturales a y b, ambos mayores que cero y que se cumpla que $a < b$.

Si se elige la opción 1 se deberá calcular y mostrar la suma de los números naturales compuestos comprendidos entre las posiciones a y b. Un número compuesto es cualquier número natural no primo, a excepción del 1. Es decir, tiene uno o más divisores distintos a 1 y a sí mismo. Los primeros 10 números compuestos son: 4, 6, 8, 9, 10, 12, 14, 15, 16, 18.

Si se elige la opción 2 se deberá deberá calcular y mostrar la suma de los cuadrados de los números naturales comprendidos entre las posiciones a y b. Los primeros 10 números naturales son: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 y sus cuadrados son 1, 4, 9, 16, 25, 36, 49, 64, 81, 100.

Utilice los siguientes casos de prueba para verificar su solución:

```
1. Suma numeros naturales compuestos
2. Suma cuadrados de numeros naturales
3. Salir

Ingrese la opción: 2
Ingrese el rango de números: 2 4
Suma cuadrados de numeros naturales: 29

Ingrese la opción: 1
Ingrese el rango de números: 1 3
Suma numeros naturales compuestos: 18

Ingrese la opción: 3

1. Suma numeros naturales compuestos
2. Suma cuadrados de numeros naturales
3. Salir

Ingrese la opción: 1
Ingrese el rango de números: 5 8
Suma numeros naturales compuestos: 51

Ingrese la opción: 2
Ingrese el rango de números: 5 10
Suma cuadrados de numeros naturales: 355

Ingrese la opción: 3
```

2.1.9. Los intervalos

Se denomina intervalo a un conjunto de números reales comprendidos entre un valor llamado límite inferior y otro valor llamado límite superior. Si el intervalo incluye estos valores se dice que el intervalo es cerrado. Se desea que implemente un programa en lenguaje C que permita procesar intervalos de la siguiente manera:

- El usuario ingresa los límites inferior y superior de dos intervalos A y B.
- Debe verificar si dichos intervalos (A y B) son válidos.
- En el caso que ambos intervalos sean válidos, se debe comprobar si los intervalos se intersecan. Debe imprimir un mensaje con el resultado de su validación.
- En el caso que los intervalos se intersequen, debe obtenerse los límites inferior y superior del intervalo resultante de la intersección (Intervalo C).
- Si alguno de los intervalos no es válido, debe mostrar un mensaje de error indicando claramente cual fue el intervalo inválido y debe solicitar al usuario que ingrese dos nuevos intervalos.
- El programa termina cuando se ingresa el valor de 0 para los límites superior e inferior de los intervalos A y B.

Recordar que:

- Un intervalo es válido si su límite superior es mayor o igual a su límite inferior.

Un ejemplo de la ejecución del programa sería el siguiente:

```
Ingrese límite inferior y superior de intervalo A: -12.6    34.7
Ingrese límite inferior y superior de intervalo B: 5.4    63.9
El intervalo A es válido.
El intervalo B es válido.
¿Los intervalos A y B se intersecan? Si.
Los datos del intervalo C son:
Límite inferior: 5.4
Límite Superior: 34.7
Ingrese límite inferior y superior de intervalo A: 0    0
Ingrese límite inferior y superior de intervalo B: 0    0
Fin de la ejecución.
```

2.2. Nivel intermedio

2.2.1. Clases de números primos

Un número natural es primo si solamente es divisible entre el mismo número y 1.

Se solicita implementar un programa que permita ingresar un número n mayor que 0 y muestre los primeros n pares de números primos de Sophie Germain y los primeros n pares de números primos Gemelos.

Números primos de Sophie Germain

Un número primo p es un número primo de Sophie Germain si $2p + 1$ también es un número primo. Ejemplo: Con $p = 2$, $2 * 2 + 1 = 5$ que también es un número primo. Los números primos de Sophie Germain recibieron su nombre por la matemática francesa que demostró que el teorema de Fermat era cierto para estos números.

Números primos Gemelos

Dos números p y q son números primos gemelos si es que tanto p como q son números primos y además la diferencia entre ambos es exactamente 2.

Utilice los siguientes casos de prueba para verificar su solución:

```
Ingrese cantidad de números: 2
Pares de números primos de Sophie Germain: (2,5) (3,7)
Pares de números primos Gemelos: (3,5) (5,7)

Ingrese cantidad de números: 5
Pares de números primos de Sophie Germain: (2,5) (3,7) (5,11) (11,23) (23,47)
Pares de números primos Gemelos: (3,5) (5,7) (11,13) (17,19) (29,31)
```

2.2.2. Sucesiones de números

Las sucesiones de números son secuencias de números naturales que siguen unas reglas de formación determinadas. Existen algunas sucesiones que tienen utilidad práctica como la sucesión de Padovan. La sucesión de Padovan [1] fue nombrada por el matemático Richard Padovan y tiene varios usos como por ejemplo el mostrado en la figura 2.4.

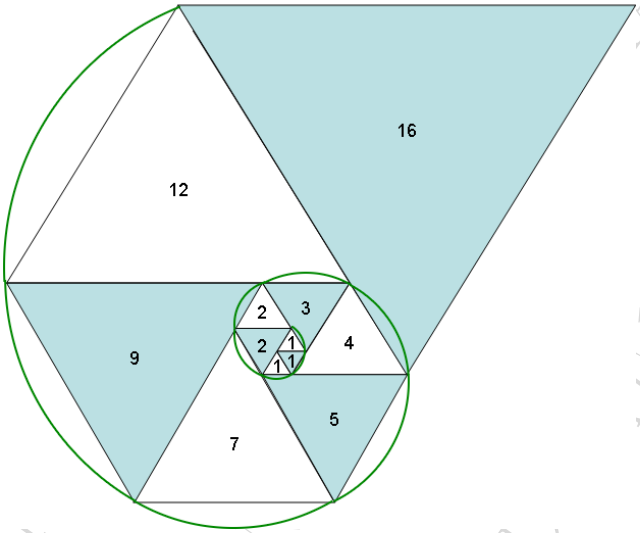

Figura 2.4: Espiral de triángulos equiláteros dónde la longitud de los lados siguen la sucesión de Padovan

Para calcular los números de la sucesión de Padovan se utiliza esta función $padovan(n)$.

$$padovan(n) = \begin{cases} 1 & n = 0 \\ 1 & n = 1 \\ 1 & n = 2 \\ padovan(n-2) + padovan(n-3) & \text{n} \geq 3 \end{cases}$$

Aquí se muestran los primeros 20 números de la sucesión de Padovan: 1, 1, 1, 2, 2, 3, 4, 5, 7, 9, 12, 16, 21, 28, 37, 49, 65, 86, 114, 151.

También existe la sucesión de Perrin que [2] es una sucesión con la misma regla de formación que la de Padovan, pero que utiliza diferentes valores iniciales, fue estudiada por el matemático francés Edouard Lucas en 1876. En 1899 sus ideas fueron desarrolladas por R. Perrin y por ello la sucesión se conoce como sucesión de Perrin.

Para calcular los números de la sucesión de Perrin se utiliza esta función $perrin(n)$.

$$perrin(n) = \begin{cases} 3 & n = 0 \\ 0 & n = 1 \\ 2 & n = 2 \\ perrin(n-2) + perrin(n-3) & \text{n} \geq 3 \end{cases}$$

[1] http://diposit.ub.edu/dspace/bitstream/2445/1323/1/217.pdf
[2] http://diposit.ub.edu/dspace/bitstream/2445/1323/1/217.pdf

Aquí se muestran los primeros 20 números de la sucesión de Perrin: 3, 0, 2, 3, 2, 5, 5, 7, 10, 12, 17, 22, 29, 39, 51, 68, 90, 119, 158, 209.

Se solicita implementar un programa que permita al usuario elegir mostrar los primeros n números ya sea de la sucesión de Padovan(P) o de la sucesión de Perrin(E), el programa deberá solicitar al usuario siempre la elección anterior hasta que se ingrese el caracter X. El número natural n ingresado debe ser mayor que cero.

Utilice los siguientes casos de prueba para verificar su solución:

```
Ingrese tipo de sucesión: P
Ingrese cantidad de números: 5
1 1 1 2 2

Ingrese tipo de sucesión: E
Ingrese cantidad de números: 3
3 0 2

Ingrese tipo de sucesión: X


Ingrese tipo de sucesión: P
Ingrese cantidad de números: 15
1 1 1 2 2 3 4 5 7 9 12 16 21 28 37

Ingrese tipo de sucesión: E
Ingrese cantidad de números: 15
3 0 2 3 2 5 5 7 10 12 17 22 29 39 51

Ingrese tipo de sucesión: X
```

2.2.3. Función definida por tramos

Se solicita implementar un programa que permita calcular el resultado de la función $f(n, m)$. El programa deberá solicitar el ingreso de dos números naturales n y m, ambos mayores que cero y $n < m$, calcular y mostrar el resultado de la función $f(n, m)$:

$$f(n,m) = \begin{cases} \sum_{i=n}^{m} Primo(i) & \text{si } 2 <= m <= 8 \\ \sum_{i=n}^{m} Fibonacci(i) & \text{si } m > 8 \end{cases}$$

Donde:

- $Primo(i)$ es el número primo de la posición i. Un número es primo si solamente es divisible entre el mismo número y 1. Ejemplo: Si los primeros números primos son $2, 3, 5, 7, 11, 13, 17, 19$, entonces $Primo(3) = 5$ y $Primo(5) = 11$.
- $Fibonacci(i)$ es el número fibonnaci de la posición i. Ejemplo: Si los primeros números de la sucesión de Fibonacci son: $1, 1, 2, 3, 5, 8, 13, 21, 34, 55$, entonces $Fibonacci(4) = 3$ y $Fibonacci(9) = 34$. Para calcular los números de Fibonacci se utiliza esta función $fib(n)$:

$$fib(n) = \begin{cases} 1 & n = 0 \\ 1 & n = 1 \\ fib(n-1) + fib(n-2) & \text{n} \geq 2 \end{cases}$$

Por ejemplo: Para $f(3,5)$ se deberá mostrar 23 que equivale a la suma de $5, 7, 11$ y para $f(6,9)$ se deberá mostrar 76 que equivale a la suma de $8, 13, 21, 34$.

Utilice los siguientes casos de prueba para verificar su solución:

```
Ingrese dos números : 3 5
f(3,5)=23

Ingrese dos números : 6 9
f(6,9)=76
```

2.2.4. Fórmula de Stokes

La ley de Stokes se refiere a la fuerza de fricción experimentada por objetos esféricos moviéndose en el seno de un fluido viscoso en un régimen laminar de bajos números de Reynolds. En general la ley de Stokes es válida en el movimiento de partículas esféricas pequeñas moviéndose a velocidades bajas. La ley de Stokes puede escribirse como:

$$F_d = 6\pi R\eta v$$

Figura 2.5: Ley de Stokes

Donde R es el radio de la esfera, ν su velocidad y η la viscosidad del fluido.

La condición de bajos números de Reynolds implica un flujo laminar lo cual puede traducirse por una velocidad relativa entre la esfera y el medio inferior a un cierto valor crítico. En estas condiciones la resistencia que ofrece el medio es debida casi exclusivamente a las fuerzas de rozamiento que se oponen al deslizamiento de unas capas de fluido sobre otras a partir de la capa límite adherida al cuerpo. La ley de Stokes se ha comprobado experimentalmente en multitud de fluidos y condiciones.

Si las partículas están cayendo verticalmente en un fluido viscoso debido a su propio peso puede calcularse su velocidad de caída o sedimentación igualando la fuerza de fricción con el peso aparente de la partícula en el fluido.

$$V_s = \frac{2}{9}\frac{r^2 g(\rho_p - \rho_f)}{\eta}$$

Figura 2.6: Velocidad de caida

- V_s es la velocidad de caída de las partículas (velocidad límite)
- g es la aceleración de la gravedad,
- ρ_p es la densidad de las partículas
- ρ_f es la densidad del fluido.
- η es la viscosidad del fluido.
- r es el radio equivalente de la partícula.

Elabore el diseño e implemente el código en lenguaje C que permita al usuario ingresar repetidas veces el material seleccionado por el usuario para la esfera, el radio de la esfera (en cm), el fluido, el radio de la particula (en cm) y calcule para cada juego de valores el valor de F_d y el valor de V_s para finalmente mostrar los valores calculados con los respectivos datos de entrada utilizados. Se dará por concluída la ejecución cuando el usuario ingrese el caracter * para el material.

Puede usar para Π el valor de 3.141592 y para la gravedad el valor de 9.81 m/s^2

Recuerde que debe convertir las unidades apropiadamente. En las tablas 2.1 y 2.2 se encuentra los datos de la densidad por material y la densidad y viscosidad por fluído.

Tabla 2.1: Densidad de esfera según el material

Material de la esfera	Densidad (g/cm3)
Hierro (H)	7.88
Aluminio (A)	2.70
Cobre (C)	8.93
Plomo (P)	11.35
Volframio (V)	19.34

Tabla 2.2: Densidad y viscosidad

Fluido	Densidad (g/cm3)	Viscosidad (kg/m·s)
Agua (A)	1.0	0.00105
Glicerina (G)	1.26	1.3923
Benceno (B)	0.88	0.000673
Aceite (C)	0.88	0.391

Utilice los siguientes casos de prueba para verificar su solución:

```
Ingrese el material de la esfera: H
Ingrese el radio de la esfera en cm: 5
Ingrese el fluído: A
Ingrese el radio de la partícula en cm: 1.5
Velocidad: 3213.942857 m/s
Fuerza de fricción: 31805.226081 kgm/s2
Ingrese el material de la esfera: P
Ingrese el radio de la esfera en cm: 67
Ingrese el fluído: B
Ingrese el radio de la partícula en cm: 1.2
Velocidad: 4883.718276 m/s
Fuerza de fricción: 415089.265922 kgm/s2
Ingrese el material de la esfera: *
Ingrese el radio de la esfera en cm: 3
Ingrese el fluído: C
Ingrese el radio de la partícula en cm: 1

Ingrese el material de la esfera: Z
Ingrese el radio de la esfera en cm: 3
Ingrese el fluído: H
Ingrese el radio de la partícula en cm: 1
Datos ingresados incorrectos
Ingrese el material de la esfera: *
Ingrese el radio de la esfera en cm: 5
Ingrese el fluído: H
Ingrese el radio de la partícula en cm: 2
```

2.2.5. Lejos de los primos

Un número primo es un entero mayor que 1 que no tiene divisores positivos otros que 1 y a si mismo. Los primeros números primos son: $2, 3, 5, 7, 11, 13, 17$. El número N se considera lejano de un número primo si no

existe un número primo entre $N-10$ y $N+10$, inclusive, por ejemplo todos los números $N-10, N-9, ..., N-1, N, N+1, ..., N+9, N+10$ no son primos.

Elabore un programa en lenguaje C que dado un entero A y un entero B. Devuelva el número de números lejanos de primos entre A y B, inclusive. Considere los números $A = 3328$ y $B = 4100$, existen 4 números lejanos de primos que son 3480, 3750, 3978 y 4038.

La entrada consiste en dos números enteros $0 \leq A$, $B \leq 100000$ y $0 \leq (B - A) \leq 1000$. En la salida imprima un solo número entero indicando la cantidad de números lejano de un número primo que existen entre A y B.

Utilice los siguientes casos de prueba para verificar su solución:

```
Ingrese los valores del rango: 3328 4100
Cantidad de números lejano: 4

Ingrese los valores del rango: 10 1000
Cantidad de números lejano: 0
```

2.2.6. Los primos truncables

El número n=3797 tiene una propiedad muy interesante. Siendo **n** un número primo es posible quitar continuamente sus dígitos de izquierda a derecha en forma continúa y cada uno de los números remanentes también es primo. En cada etapa : 3797, 797, 97 y 7 también son números primos. Similarmente si trabajamos de derecha a izquierda: 3797, 379, 37 y 3.

Tome en cuenta que los números 2, 3, 5 y 7 no se consideran truncables. Note que el número 1 no es primo.

Elabore un programa en lenguaje C que dados dos números a, b donde $0 \leq a, b \leq 10^7$ liste los primos truncables que hay en ese rango.

Como salida se debe imprimir los números primos truncables que hay en el rango a y b inclusive. Si no hay ningún primo truncable imprima un mensaje indicando lo sucedido.

Utilice los siguientes casos de prueba para verificar su solución:

```
Ingrese los valores del rango: 0 50
23 31 37

Ingrese los valores del rango: 3700 3800
3797
```

2.2.7. Rotando números

Implemente un programa que reciba como datos dos números, el primer sera el número a rotar y el segundo la cantidad de dígitos a rotar ; además recibirá una letra que indicará la dirección de rotación. Si la dirección es D, deberá rotar de derecha a izquierda. Si dirección es I deberá rotar de izquierda a derecha.

Utilice los siguientes casos de prueba para verificar su solución:

```
Ingresar número: 789456
Ingresar cantidad de dígitos a rotar:2
Ingresar dirección de rotación: I
El resultado es: 567894

Ingresar número: 789456
Ingresar cantidad de dígitos a rotar:2
Ingresar dirección de rotación: D
El resultado es: 945678
```

En el caso propuesto, se cumple ver Figura 2.7.

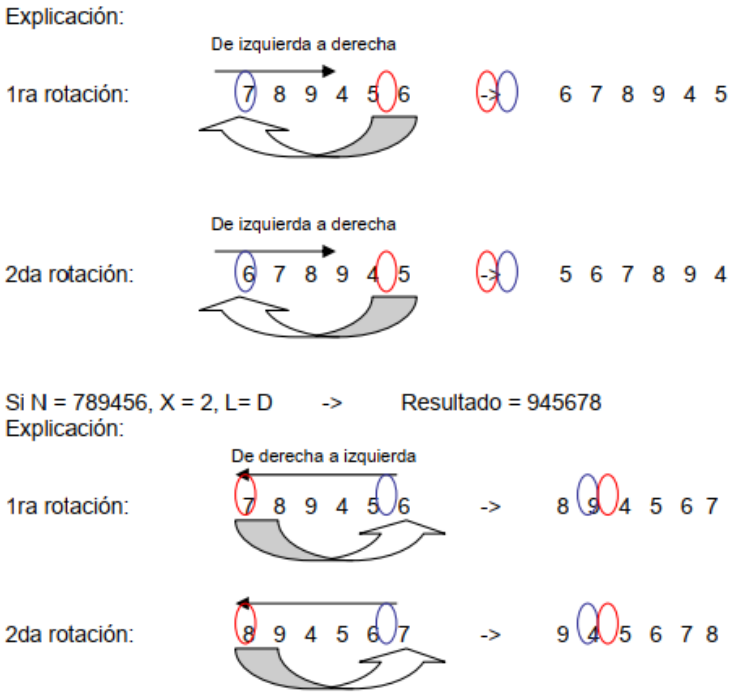

Figura 2.7: Rotando número

2.2.8. Resta de dos horas

Elabore un programa en lenguaje C que permita ingresar y restar dos horas del mismo día y muestre el resultado en el formato hh:mm:ss (Ejemplo: 02:59:30).

Las dos horas deben ingresarse en el formato hh:mm:ss (Ejemplo: 6:20:44). Debe comprobar que las horas sean válidas y que la segunda hora sea mayor que la primera.

Considere que las horas están comprendidas entre 0 y 23, los minutos entre 0 y 59 y los segundos entre 0 y 59. Además 1 hora equivale a 60 minutos y 1 minuto equivale a 60 segundos.

Utilice los siguientes casos de prueba para verificar su solución:

```
Ingrese hora inicial en formato hh:mm:ss : 6:20:44
Ingrese hora final en formato hh:mm:ss   : 6:20:59
La resta es 00:00:15

Ingrese hora inicial en formato hh:mm:ss : 6:20:44
Ingrese hora final en formato hh:mm:ss   : 6:21:49
La resta es 00:01:05

Ingrese hora inicial en formato hh:mm:ss : 6:20:44
Ingrese hora final en formato hh:mm:ss   : 8:21:49
La resta es 02:01:05

Ingrese hora inicial en formato hh:mm:ss : 25:65:70
Hora no válida
Minutos no válidos
Segundos no válidos

Ingrese hora inicial en formato hh:mm:ss : 6:20:44
```

```
Ingrese hora final en formato hh:mm:ss   : 28:69:75
Hora no válida
Minutos no válidos
Segundos no válidos

Ingrese hora inicial en formato hh:mm:ss : 6:20:44
Ingrese hora final en formato hh:mm:ss   : 4:21:49
La hora final no es mayor a la inicial
```

2.3. Nivel avanzado

2.3.1. Figuras geométricas en el plano cartesiano (adaptado del laboratorio 8 del ciclo 2020-2)

Se tienen en el plano cartesiano diversas figuras geométricas, cuadrados o triángulos equiláteros, para las cuales se requiere conocer el perímetro. Pero estas figuras deben cumplir ciertas condiciones, deben ser cuadrados o triángulos y sus vértices equidistantes al centro del plano cartesiano.

Se pide que desarrolle un programa en Lenguaje C que, solicite una lista de figuras geométricas, cada una con la identificación del tipo de figura; para cuadrados C o c, para triángulos T o t; los puntos de sus vértices; para un cuadrado 4 puntos y 3 puntos para un triángulo; valide si los datos ingresados son correctos y muestre el perímetro de cada figura geométrica. Para finalizar el ingreso de datos debe ingresar como tipo de figura X o x. Finalmente al terminar de ingresar lo datos debe indicar la cantidad de figuras ingresadas, las que se ingresaron correctamente y las que no; si por lo menos se ingresaron dos figuras correctamente indicar la figura geométrica con mayor y menor perímetro y el tipo de figura al que corresponden. Si solo existe una figura mostrar el tipo de figura con su perímetro, sino se ingresó ninguna figura correctamente indicarlo.

Al leer el tipo de figura se ingresa un carácter, si no corresponde a los caracteres válidos la figura se cuenta como incorrecta y se muestra el mensaje correspondiente. Al leer los vértices (se ingresan en sentido horario o antihorario) debe verificar que se vaya formando la figura correspondiente, si los lados que va calculando no son iguales o lo vértices no son equidistantes al centro del plano cartesiano debe indicar el error y no continuar con la lectura de los siguientes vértices; de ser así la figura se cuenta como incorrecta.

Debe usar el paradigma de programación modular y desarrollar como mínimo 4 módulos adicionales al principal. De los cuales como máximo debe tener uno para leer datos, uno para validar; además por lo menos uno de sus módulos debe utilizar parámetros por referencia y por lo menos uno debe devolver un valor.

Al comparar cálculos con resultados de números reales debe usar una precisión con valor de 0.001.

Caso de prueba para verificación de solución

Ingrese el tipo de figura (C o c para cuadrado y T o t para triángulo), para terminar ingrese X o x: c
Ingrese el punto 1 de la figura cuadrado: -2 2
Ingrese el punto 2 de la figura cuadrado: 2 2
Ingrese el punto 3 de la figura cuadrado: 2 -2
Ingrese el punto 4 de la figura cuadrado: -2 -2
Perímetro del cuadrado: 16.000000
Ingrese el tipo de figura (C o c para cuadrado y T o t para triángulo), para terminar ingrese X o x: T
Ingrese el punto 1 de la figura triángulo: -2 -1.1547
Ingrese el punto 2 de la figura triángulo: 2 -1.1547
Ingrese el punto 3 de la figura triángulo: 0 2.3094
Perímetro del triángulo: 12.000000
Ingrese el tipo de figura (C o c para cuadrado y T o t para triangulo), para terminar ingrese X o x: c
Ingrese el punto 1 de la figura cuadrado: -4 3
Ingrese el punto 2 de la figura cuadrado: 3 2
Los vértices no son equidistantes con el origen de coordenadas
Ingrese el tipo de figura (C o c para cuadrado y T o t para triangulo), para terminar ingrese X o x: x
Se terminó el ingreso de las figuras
Resultados:
La cantidad de figuras ingresadas fue: 3
La cantidad de figuras erradas fue: 1
La cantidad de figuras correctas fue: 2
La figura con mayor perímetro fue un cuadrado, el perímetro calculado es: 16.000000
La figura con menor perímetro fue un triángulo, el perímetro calculado es: 12.000000

Caso de prueba para verificación de solución

Ingrese el tipo de figura (C o c para cuadrado y T o t para triangulo), para terminar ingrese X o x: f
Figura errada
Ingrese el tipo de figura (C o c para cuadrado y T o t para triangulo), para terminar ingrese X o x: g
Figura errada
Ingrese el tipo de figura (C o c para cuadrado y T o t para triangulo), para terminar ingrese X o x: c
Ingrese el punto 1 de la figura cuadrado: 3 3
Ingrese el punto 2 de la figura cuadrado: 3 -3
Ingrese el punto 3 de la figura cuadrado: -3 3
La figura no tiene los lados iguales
Ingrese el tipo de figura (C o c para cuadrado y T o t para triangulo), para terminar ingrese X o x: t
Ingrese el punto 1 de la figura triangulo: 2 4
Ingrese el punto 2 de la figura triangulo: 5 3
La figura no es equidistante con el origen de coordenadas
Ingrese el tipo de figura (C o c para cuadrado y T o t para triangulo), para terminar ingrese X o x: x
Se terminó el ingreso de las figuras
Resultados:
La cantidad de figuras ingresadas fue: 4
La cantidad de figuras erradas fue: 4
La cantidad de figuras correctas fue: 0
No se ingresó ninguna figura correcta

2.3.2. Números equidigital, frugal y no económico (adaptado del laboratorio 8 del ciclo 2020-2)

Un número n se llama económico si el número de dígitos en su factorización de primos (incluyendo exponentes mayores a 1) no es mayor que el número de dígitos de n. Al referirse a factorización de primos, se habla de los

divisores primos de un número, por ejemplo $2^2 \times 3 \times 5^3$ es la factorización de primos del número 1500.
En la práctica, los números económicos son la unión de los números equidigitales y frugales.

Los primeros números económicos son: 2, 3, 5, 7, 10, 11, 13, 14, 15, 16, 17, 19, 21, 23, 25, 27, 29, 31, 32, 35, 37, 41, 43, 47, 49, 53, 59 Un número frugal es aquel en el que la cantidad de dígitos usados en su factorización de primos (excluyendo exponentes iguales a 1) es menor a la cantidad de dígitos usados en el número. Por ejemplo, el número 128 es frugal en base 10 porque es escrito usando tres dígitos, mientras que su factorización de primos 2^7 usa dos dígitos. Por ejemplo 144 no es frugal ya que tiene tres dígitos y su factorización $2^3 \times 3^2$ usa cuatro dígitos. Los primeros números frugales son: 125, 128, 243, 256, 343, 512, 625, 729, 1024, 1029, 1215, 1250, 1280, 1331, 1369, 1458, 1536,
Mientras que los números equidigitales en base 10 son aquellos en los que la cantidad de dígitos usados en su factorización de primos (excluyendo exponentes iguales a 1) es igual a la cantidad de dígitos usados en el número. Por ejemplo el número 127 es equidigital en base 10 porque es escrito usando tres dígitos 127^1(recordemos que el exponente debe ser mayor a uno para ser considerado). Por ejemplo, 126 no es equidigital porque $2^1 \times 3^2 \times 7^1$, son 4 dígitos y el número es de tres dígitos. Los primeros números equidigitales son: 2, 3, 5, 7, 10, 11, 13, 14, 15, 16, 17, 19, 21, 23, 25, 27, 29, 31, 32, 35, 37, 41, 43, 47.

Se le pide que haciendo uso de lenguaje C elabore un programa que permita saber si un número es equidigital, frugal o no económico e imprima su factorización de primos. Para ello debe , utilizando módulos, poder:

- Identificar si un número es primo o no,
- Identificar cuál es el exponente al que se encuentra cierto divisor de un número (por ejemplo el 9 tendría para el divisor 3 un exponente de 2, el 6 tendría para el divisor 2 un exponente de 1, el 1024 tendría para el divisor 2 un exponente de 10) y a la vez se debe identificar cuántos dígitos tiene ese exponente (recuerde que el 1 se considera como 0 dígitos, así en los ejemplos anteriores este valdría 0, 0 y 2 respectivamente).
- Identificar la cantidad de dígitos que posee un número (ojo, no podrá usar fórmula alguna para identificar ésta cantidad, deberá utilizar una estructura iterativa para ello).

Su programa principal deberá, leer un número a evaluar, validar que sea positivo mayor a 0, de no serlo, debe mandar un mensaje de error. De ser válido, debe identificar la cantidad de dígitos usados en su factorización de primos (por ejemplo para el número 128 esta cantidad sería 2), identificar la cantidad de dígitos que posee el número (por ejemplo para el número 128 sería 3) y finalmente evaluar y mostrar el tipo de número que es o no es, de tal forma, el número podría ser Frugal, podría ser Equidigital o podría no ser Económico (El 128 sería Frugal ya que 2 es menor a 3). Para realizar estas acciones debe utilizar los módulos creados. Su solución debe presentar al menos 4 módulos incluído el principal.

Caso de prueba para verificación de solución

Ingrese el número a evaluar :128
2ˆ7 El número 128 es frugal

Caso de prueba para verificación de solución

Ingrese el número a evaluar :1250
2ˆ1 5ˆ4 El número 1250 es frugal

Caso de prueba para verificación de solución

Ingrese el número a evaluar :43
43ˆ1 El número 43 es equidigital

Caso de prueba para verificación de solución

Ingrese el número a evaluar :-54
Datos de entrada inválidos

Caso de prueba para verificación de solución

Ingrese el número a evaluar :8
2^3 El número 8 no es económico

2.3.3. Operaciones con números (adaptado del laboratorio 8 del ciclo 2020-2)

Si un número de cuatro cifras descendentes continuas en 1 se invierte y estos dos números se restan, el resultado es 3087. Este resultado es el mismo para cualquier operación realizada con todos los números de cuatro cifras descendentes continuas en 1. Por ejemplo:

- 7654-4567=3087
- 9876-6789=3087

Si se realiza la misma operación para un número de 3 cifras descendentes continuas en 1 el resultado siempre será 198. Note que la suma de las cifras de los números 3087 y 198 es múltiplo de 9.

- 3+0+8+7=18
- 1+9+8=18

Si en lugar que el número tenga cifras descendentes continuas en 1, tendría cifras ascendentes continuas en 1 e igual se invierte y se restan los números, el resultado es el mismo que el anterior pero con signo negativo. Multiplicando este número negativo por -1 se convierte en positivo y se cumpliría que la suma de sus cifras es múltiplo de 9.

- 4567-7654=-3087 * -1 = 3087 , 3+0+8+7 = 18
- 123-321=-198 * -1 = 198 , 1+9+8 = 18

Lo anteriormente descrito se cumple para cualquier número entero que tenga cifras ascendentes continuas en 1 o descendentes continuas en 1.

Se pide que, escriba un programa en Lenguaje C que reciba un rango de números enteros positivos de máximo 5 cifras y mínimo 2 cifras, el tipo de números a evaluar (ascendente, descendente o los dos tipos anteriores - ascendente y descendente) y muestre todos los números que cumplen la condición descrita previamente. Además, debe mostrar la cantidad total de números que cumplen la condición. Si alguno de los números ascendentes continuos en 1 o descendentes continuos en 1 no cumplen la condición se debe mostrar el número, e igualmente al finalizar indicar cuántos números con las características descritas no cumplen la condición.

Para evaluar los números ascendentes se debe ingresar a o A, para descendentes d o D o para evaluar los dos casos t o T. Si el rango o el tipo de número no son válidos se debe mostrar un mensaje apropiado y terminar el programa.

Debe usar el paradigma de programación modular y desarrollar como mínimo 5 módulos adicionales al principal. De los cuales como máximo: uno puede leer los datos, uno puede validarlos, por lo menos uno debe usar parámetros por referencia y por lo menos dos deben devolver un valor (diferentes a la validación). Debe usar una estructura iterativa en el programa principal. En todos los casos que requiera extraer los dígitos de un número debe usar estructuras iterativas.

Caso de prueba para verificación de solución

Ingrese rango: 200 300
Ingrese tipo de número: A
Ascendente: 234
En el rango [200-300], hay 1 números que cumplen la condición
Todos los números con cifras ascendentes continuas en 1 en el rango [200,300] cumplen la condición

Caso de prueba para verificación de solución

Ingrese rango:200 700
Ingrese tipo de número: d
Descendente: 210
Descendente: 321
Descendente: 432
Descendente: 543
Descendente: 654
En el rango [200-700], hay 5 números que cumplen la condición
Todos los números con cifras descendentes continuas en 1 en el rango [200,700] cumplen la condición

Caso de prueba para verificación de solución

Ingrese rango:10 100
Ingrese tipo de número: T
Descendente: 10
Ascendente : 12
Descendente: 21
Ascendente : 23
Descendente: 32
Ascendente : 34
Descendente: 43
Ascendente : 45
Descendente: 54
Ascendente : 56
Descendente: 65
Ascendente : 67
Descendente: 76
Ascendente : 78
Descendente: 87
Ascendente : 89
Descendente: 98
En el rango [10-100], hay 17 números que cumplen la condición
Todos los números con cifras descendentes y/o ascendentes continuas en 1 el rango [10,100] cumplen la condición

Caso de prueba para verificación de solución

Ingrese rango:10 100000
Ingrese tipo de número: A
Datos inválidos

Caso de prueba para verificación de solución

Ingrese rango: 100 90
Ingrese tipo de número: t
Datos inválidos

Caso de prueba para verificación de solución

Ingrese rango:135 7890
Ingrese tipo de número: x
Datos inválidos

Caso de prueba para verificación de solución

Ingrese rango:9 1000
Ingrese tipo de numero:A
Datos inválidos

2.3.4. Números Duffiniano, no compuestos y no duffiniano (adaptado del laboratorio 8 del ciclo 2020-2)

Un número n se llama de Duffiniano si el número es compuesto que no tiene factores primos en común con la suma de sus divisores. Asi, 35 es Duffiniano ya que la suma de sus divisores 1+5+7+35 = 48 no comparte ningún divisor primo con el 35 (48 no es divisible entre 5 ni entre 7). Los primeros números de Duffinianos son 4, 8, 9, 16, 21, 25, 27, 32, 35, 36, 39, 49, 50, 55, 57, 63, 64, 65, 75, 77, 81, 85, 93, 98.

Un número compuesto es aquel que no es primo y que tampoco es 1, así, por ejemplo el 5 no es un número compuesto por ser primo y el 4 si lo es al igual que el 66 por ser no primos y diferentes a 1.

Se le pide que haciendo uso de lenguaje C elabore un programa que permita saber si un número es Duffiniano, no compuesto o no duffiniano. Para ello debe , utilizando módulos, poder:

- Identificar cuál es el exponente al que se encuentra cierto divisor de un número (por ejemplo el 9 tendría para el divisor 3 un exponente de 2, el 18 tendría para el divisor 3 un exponente de 2).
- Identificar si un número es divisible únicamente entre si y el 1.
- Identificar si un número es duffiniano y/o compuesto, haciendo uso de los dos módulos anteriores

Su programa principal deberá, leer un rango de números a evaluar, verificar que dicho rango sea válido y mayor a 0, de no serlo deberá mostrar un mensaje de error. De ser válido, deberá para cada número en dicho rango identificar si es duffiniano, no compuesto o no duffiniano. Para realizar estas acciones debe utilizar uno o varios de los módulos creados. Su solución debe presentar al menos 4 módulos incluído el principal.

Casos de prueba para verificación de solución

Ingrese el rango de números a evaluar :3 10
El número 3 no es compuesto
El número 4 es Duffiniano
El número 5 no es compuesto
El número 6 no es Duffiniano
El número 7 no es compuesto
El número 8 es Duffiniano
El número 9 es Duffiniano
El número 10 no es Duffiniano

Casos de prueba para verificación de solución

Ingrese el rango de números a evaluar : -43 10
Datos de entrada inválidos

Casos de prueba para verificación de solución

Ingrese el rango de números a evaluar :54 23
El primer valor debe ser menor al segundo valor del rango

2.3.5. Representación de horas en grados (adaptado del laboratorio 8 del ciclo 2020-2)

Una actividad particular de los murciélagos ocurre en ciertos momentos del día. En este caso durante cuatro momentos donde se registra la hora exacta (horas, minutos y segundos):
23:00:17, 23:40:20, 00:12:45, 00:17:19

Usando la idea de que hay veinticuatro horas en un día, que es análogo a 360 grados en un círculo, mapee las horas del día desde y hacia los ángulos; y usando las ideas de promedio o la media de ángulos debe calcular y mostrar el tiempo promedio de la actividad nocturna con una precisión de un segundo de tiempo. Para calcular el promedio o el ángulo medio se debe tener en cuenta cómo se envuelven los ángulos de modo que cualquier ángulo en grados más cualquier múltiplo entero de 360 grados será una medida del mismo ángulo.
Para calcular el ángulo medio de varios ángulos se puede utilizar la siguiente fórmula (ver Figura 2.8), la cual dados los ángulos $a_1, \dots a_n$ la media se calcula mediante:

$$\bar{\alpha} = \operatorname{atan2}\left(\frac{1}{n}\cdot\sum_{j=1}^{n}\sin\alpha_j, \frac{1}{n}\cdot\sum_{j=1}^{n}\cos\alpha_j\right)$$

Figura 2.8: Fórmula ángulo medio de varios ángulos

Se pide que desarrolle un programa en Lenguaje C que dada una cantidad de horas del día a registrar puedan ser representados en ángulos en grados y luego se devuelve su ángulo medio el mismo que se debe mostrar en formato de horas del día (horas, minutos y segundos). En caso se introduzca un valor incorrecto de la hora se debe solicitar un nuevo registro hasta que sea una hora valida.
Debe usar el paradigma de programación modular y desarrollar como mínimo 4 módulos adicionales al principal. De los cuales como máximo debe tener uno para leer datos, uno para validar; además por lo menos uno de sus módulos debe utilizar parámetros por referencia y por lo menos uno debe devolver un valor.

Recuerde que:

- Se puede usar del carácter dos puntos (:) entre los dos especificadores de formato, %d y %d, de la instrucción scanf(). Esto hace que el usuario deba introducir los dos puntos entre los números que conforman el formato del tiempo. Estos números son capturados por la función mediante los especificadores %d. .
- double atan2(double y, double x)

Caso de prueba para verificación de solución

Ingresar el número de datos (horas) : 2
Ingrese los datos separados por un espacio entre cada unidad :
00:00:00
06:00:00
La hora es : 3:0:0

Caso de prueba para verificación de solución

Ingresar el número de datos (horas) : 4
Ingrese los datos separados por un espacio entre cada unidad :
23:00:17
23:40:20
00:12:45
00:17:19
La hora es : 23:47:43

Caso de prueba para verificación de solución

Ingresar el número de datos (horas) : 2
Ingrese los datos separados por un espacio entre cada unidad :
00:00:00
04:60:00
Minutos incorrectos!
Debe escribir la hora de manera correcta: 04:59:50
La hora es : 2:29:55

Caso de prueba para verificación de solución

Ingresar el número de datos (horas) : 2
Ingrese los datos separados por un espacio entre cada unidad :
00:00:00
04:59:70
Segundos incorrectos!
Debe escribir la hora de manera correcta: 04:59:50
La hora es : 2:29:55

2.3.6. Números atractivos (adaptado del laboratorio 8 del ciclo 2020-2)

Un número es un número atractivo si el número de sus factores primos (distintos o no) también es primo. Por ejemplo:

El número 20, cuya descomposición prima es $2 \times 2 \times 5$, es un número atractivo porque el número de sus factores primos (3) también es primo.
El número 22, cuya descomposición prima es 2×11, es un número atractivo porque el número de sus factores primos (2) también es primo.

Se pide que desarrolle un programa en Lenguaje C, que solicite un rango de números [a,b] que deben ser positivos y $a \leq b$, de no cumplir con estas condiciones se debe mostrar un mensaje de error. Además, también debe imprimir un número fijo de números obtenidos por línea impresa, en este caso 7 números por línea.
Finalmente se debe imprimir tres tipos de números atractivos: (1) los n primeros, (2) los n primeros pares y (3) los n primeros impares.

Debe usar el paradigma de programación modular y desarrollar como mínimo 4 módulos adicionales al principal. De los cuales como máximo debe tener uno para leer datos, uno para validar; además por lo menos uno de sus módulos debe utilizar parámetros por referencia y por lo menos uno debe devolver un valor.

Caso de prueba para verificación de solución

```
Ingrese el rango inicio y fin: 11 45
Los primeros números atractivos del tipo (1):
12 14 15 18 20 21 22
25 26 27 28 30 32 33
34 35 38 39 42 44 45
Los primeros números atractivos del tipo (2):
12 14 18 20 22 26 28
30 32 34 38 42 44
Los primeros números atractivos del tipo (3):
15 21 25 27 33 35 39
45
```

Caso de prueba para verificación de solución

```
Ingrese el rango inicio y fin: 41 10
Error en ingreso de datos
```

Caso de prueba para verificación de solución

```
Ingrese el rango inicio y fin: 1 20
Los primeros números atractivos del tipo (1):
4 6 8 9 10 12 14
15 18 20
Los primeros números atractivos del tipo (2):
4 6 8 10 12 14 18
20
Los primeros números atractivos del tipo (3):
9 15
```

2.3.7. Números abundantes y no abundantes (adaptado del laboratorio 8 del ciclo 2020-2)

Un número n es abundante si la suma de sus divisores propios es mayor al mismo número n. Por ejemplo 12 tiene por divisores propios 1,2,3 4, 6 los cuales suman 16, valor mayor a 12. Por lo que el 12 se considera como

abundante. Existen infinitos números abundantes y algunos de ellos pueden ser obtenidos mediante la suma de dos números abundantes y en algunos casos la suma de 3 números abundantes.

Se le pide que haciendo uso de lenguaje C elabore un programa que permita para un rango de valores identificar si cada valor en el rango puede obtenerse de sumar dos números abundantes, de sumar tres números abundantes, si es abundante o si no es abundante y no puede obtenerse por medio de la suma de dos o tres números abundantes. Para ello debe , utilizando módulos, poder:

- Identificar Si un número es o no abundante, debe utilizar estructuras iterativas.
- Identificar si un número puede ser obtenido de la suma de dos números abundantes y recordar cuáles son estos dos números. Solo la primera coincidencia.
- Identificar si un número puede ser obtenido de la suma de tres números abundantes y recordar cuáles son estos tres números. Solo la primera coincidencia.

Su programa principal deberá, leer un rango de números a evaluar, verificar que este rango sea válido, mostrar un mensaje de error de no serlo. Si el rango es válido para cada uno de los números en el rango identificar si se puede obtener sumando dos números abundantes y mostrar estos tres números. De no ser posible, identificar si se puede obtener sumando tres números abundantes y mostrar estos cuatro números. De no ser posible, evaluar si el número es abundante y mostrarlo, finalmente si el número no es abundante, deberá mostrar el mensaje: no es abundante y no puede ser representado como la suma de abundantes.

Casos de prueba para verificación de solución

Ingrese el rango de números a evaluar : 991 995
991 no es abundante y no puede ser representado como la suma de abundantes
El número 992 = 12 + 980
El número 993 = 48 + 945
El número 994 = 40 + 954
El número 995 = 945 + 20 +30

Casos de prueba para verificación de solución

Ingrese el rango de números a evaluar : 20123 20120
El primer valor debe ser menor al segundo valor del rango

Casos de prueba para verificación de solución

Ingrese el rango de números a evaluar : 20120 20126
El número 20120 = 20 + 20100
El número 20121 = 276 + 19845
El número 20122 = 18 + 20104
El número 20123 = 5355 + 14768
El número 20124 = 12 + 20112
El número 20125 = 280 + 19845
El número 20126 = 20 + 20106

Casos de prueba para verificación de solución

Ingrese el rango de números a evaluar : 50 60
El número 50 = 20 + 30
51 no es abundante y no puede ser representado como la suma de abundantes
El número 52 = 12 + 40
53 no es abundante y no puede ser representado como la suma de abundantes
El número 54 = 12 + 42
55 no es abundante y no puede ser representado como la suma de abundantes
El número 56 = 20 + 36
57 no es abundante y no puede ser representado como la suma de abundantes
El número 58 = 18 + 40
59 no es abundante y no puede ser representado como la suma de abundantes
El número 60 = 12 + 48

Casos de prueba para verificación de solución

Ingrese el rango de números a evaluar : 10 15
10 no es abundante y no puede ser representado como la suma de abundantes
11 no es abundante y no puede ser representado como la suma de abundantes
12 es abundante
13 no es abundante y no puede ser representado como la suma de abundantes
14 no es abundante y no puede ser representado como la suma de abundantes
15 no es abundante y no puede ser representado como la suma de abundantes

2.3.8. Suma de números en diferentes bases (adaptado del laboratorio 8 del ciclo 2020-2)

Existen diferentes sistemas de numeración, el que utilizamos comúnmente para cálculos matemáticos es el sistema decimal. También existen otros sistemas como el binario, octal, hexadecimal, etc.
Cuando los dígitos de un número con base mayor a la decimal exceden al número 10, se utilizan letras para su representación.
En la figura 2.9 se puede ver una equivalencia entre los componentes del sistema de numeración decimal y del hexadecimal y el uso de letras para la representación de dígitos mayores a 9.

Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Hexadecimal	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Figura 2.9: Equivalencia entre sistema de numeración decimal y hexadecimal

Para sumar números de un dígito en cualquier base, primero se convierte cada dígito en su repesentación en base 10; se suman los dígitos y el dígito menos significativo del resultado es el residuo de la suma entre la base y el dígito más significativo (si existe) es el cociente de la suma entre la base. Por ejemplo:

- 7+7 en base 8. 7 en base 10 es igual 7, 7+7=14, residuo de 14/8 es 6 y cociente de 14/8 es 1, entonces la suma es 16 en base 8.
- E+A en base 16. E en base 10 es 14, A en base 10 es 10, 10+14=24, residuo de 24/16 es 8 y cociente de 24/16 es 1, entonces la suma es 18 en base 16.

Se pide que desarrolle un programa en Lenguaje C que solicite una lista de bases y dos números de 4 cifras; y muestre la suma de los números en la misma base. Al finalizar debe ingresar cuántas sumas realizadas fueron

válidas y cuantas no.

Para ello debe validar que la base ingresada esté entre la base binaria y la base hexadecimal, de no ser así debe mostrar el mensaje adecuado. Cada número se ingresará dígito por dígito empezando del dígito menos significativo, se debe validar que cada dígito corresponda a la base. Si el número tiene menos de 4 cifras se ingresará 0 para completarlas. La suma se muestra al final de ingresar los números de 4 cifras, pero solo se muestran los dígitos más significativos que sean mayores a 0, por ejemplo si en base 10 se sumarán los números 12 y 13 se debe mostrar 25 no 00025. Para terminar de ingresar los datos, el primer dígito menos significativo del número puede ser x o X. Los dígitos que son letras se pueden ingresar en mayúsculas o minúsculas.

La suma la debe realizar dígito por dígito, pero solo se muestra el resultado al final de ingresados todos los dígitos de los números. No puede leer todos los dígitos, formar los números y sumarlos.

Debe usar el paradigma de programación modular y desarrollar como mínimo 5 módulos adicionales al principal. De los cuales: uno debe leer y validar la base ingresada utilizando por lo menos un parámetro por referencia, por lo menos dos deben devolver un valor y por lo menos uno debe usar parámetros por referencia (diferente a la lectura y validación). Debe usar una estructura iterativa en el programa principal.

Caso de prueba para verificación de solución

Ingrese la base de los números: 16
Leer dígito 1 del primer número: E
Leer dígito 1 del segundo número: 1
Leer dígito 2 del primer número: A
Leer dígito 2 del segundo número: B
Leer dígito 3 del primer número: 2
Leer dígito 3 del segundo número: 6
Leer dígito 4 del primer número: 0
Leer dígito 4 del segundo número: A
La suma es A95F
Ingrese la base de los números: 3
Leer dígito 1 del primer número: 1
Leer dígito 1 del segundo número: 2
Leer dígito 2 del primer número: 0
Leer dígito 2 del segundo número: 2
Leer dígito 3 del primer número: 1
Leer dígito 3 del segundo número: 0
Leer dígito 4 del primer número: 1
Leer dígito 4 del segundo número: 0
La suma es 1200
Ingrese la base de los números: 4
Leer dígito 1 del primer número:7
No se puede continuar con la lectura del primer número porque el dígito ingresado no corresponde a la base
Ingrese la base de los números: 5
Leer dígito 1 del primer número: x

Terminó el ingreso de base y números
Resultados:
Se ingresaron 3 sumas de las cuales 2 fueron correctas.

Caso de prueba para verificación de solución

Ingrese la base de los números: 14
Leer dígito 1 del primer número: a
Leer dígito 1 del segundo número: 5
Leer dígito 2 del primer número: B
Leer dígito 2 del segundo número: 7
Leer dígito 3 del primer número: 9
Leer dígito 3 del segundo número: 0
Leer dígito 4 del primer número: C
Leer dígito 4 del segundo número: 0
La suma es CA51
Ingrese la base de los numeros:20
La base ingresada no es valida
Ingrese la base de los números:7
Leer dígito 1 del primer número:6
Leer dígito 1 del segundo número:9
No se puede continuar con la lectura del segundo número porque el dígito ingresado no corresponde a la base
Ingrese la base de los números:10
Leer dígito 1 del primer número:9
Leer dígito 1 del segundo número:9
Leer dígito 2 del primer número:9
Leer dígito 2 del segundo número:9
Leer dígito 3 del primer número:9
Leer dígito 3 del segundo numero:9
Leer dígito 4 del primer número:9
Leer dígito 4 del segundo número:9
La suma es 19998
Ingrese la base de los números:10
Leer dígito 1 del primer número:1
Leer dígito 1 del segundo número:2
Leer dígito 2 del primer número:0
Leer dígito 2 del segundo número:0
Leer dígito 3 del primer número:0
Leer dígito 3 del segundo número:0
Leer dígito 4 del primer número:0
Leer dígito 4 del segundo número:0
La suma es 3
Ingrese la base de los números: 4
Leer dígito 1 del primer número: X

Terminó el ingreso de base y números
Resultados: Se ingresaron 4 sumas de las cuales 3 fueron correctas.

2.3.9. Números brasileños (adaptado del laboratorio 8 del ciclo 2020-2)

Los números brasileños se definen como el conjunto de números enteros positivos donde cada número N tiene al menos un número natural B donde $1 < B < N - 1$ donde la representación de N en la base B tiene todos los dígitos iguales.
Por ejemplo:
1, 2 y 3 no pueden ser brasileños; no hay base B que satisfaga la condición $1 < B < N - 1$.
4 no es brasileño; 4 en la base 2 es 100. Los dígitos no son todos iguales.
5 no es brasileño; 5 en la base 2 es 101, en la base 3 es 12. No hay representación donde los dígitos son iguales.

6 no es brasileño; 6 en la base 2 es 110, en la base 3 es 20, en la base 4 es 12 . No hay representación donde los dígitos son iguales.
7 es brasileño; 7 en la base 2 es 111. Hay al menos una representación donde los dígitos son todos iguales.
8 es brasileño; 8 en la base 3 es 22. Hay al menos una representación donde los dígitos son todos iguales.

Todos los enteros pares $2P >= 8$ son brasileños porque 2P = 2 (P-1) + 2, que es 22 en la base P-1 cuando $P - 1 > 2$. Eso se vuelve cierto cuando $P >= 4$.

Se pide que desarrolle un programa en Lenguaje C, que solicite la cantidad de números a calcular la cual debe ser positiva y menor o igual a 10, de no cumplir con estas condiciones se debe mostrar un mensaje de error. Además, también se solicita la cantidad de números a imprimir por fila la cual debe ser menor o igual a la cantidad de números solicitados, igualmente de no cumplir se debe mostrar un mensaje de error. Finalmente se debe imprimir tres tipos de números brasileño: (1) los n primeros, (2) los n primeros impares y (3) los n primeros primos.
Debe usar el paradigma de programación modular y desarrollar como mínimo 4 módulos adicionales al principal. De los cuales como máximo debe tener uno para leer datos, uno para validar; además por lo menos uno de sus módulos debe utilizar parámetros por referencia y por lo menos uno debe devolver un valor.

Caso de prueba para verificación de solución

```
Ingresar la cantidad de números a calcular: 7
Ingresar la cantidad de números por fila: 3
Los primeros 7 números brasileños del tipo (1):
7 8 10
12 13 14
15

Los primeros 7 números brasileños del tipo (2):
7 13 15
21 27 31
33

Los primeros 7 números brasileños del tipo (3):
7 13 31
43 73 127
157
```

Caso de prueba para verificación de solución

```
Ingresar la cantidad de números a calcular: 8
Ingresar la cantidad de números por fila: 9
Error en el valor de cantidad de números por fila
```

Caso de prueba para verificación de solución

```
Ingresar la cantidad de números a calcular: 8
Ingresar la cantidad de números por fila: 8
Los primeros 8 números brasileños del tipo (1):
7 8 10 12 13 14 15 16
Los primeros 8 números brasileños del tipo (2):
7 13 15 21 27 31 33 35
Los primeros 8 números brasileños del tipo (3):
7 13 31 43 73 127 157 211
```

Caso de prueba para verificación de solución

Ingresar la cantidad de números a calcular: 0
Ingresar la cantidad de números por fila: 2
Error en el valor de la cantidad de números a calcular.

2.3.10. Números de Aquiles, poderosos y de potencia perfecta (adaptado del laboratorio 8 del ciclo 2020-2)

Un número n se llama de Aquiles si el número es poderoso pero no de potencia perfecta. Los primeros números de Aquiles son: 72, 108, 200, 288, 392, 432, 500, 648, 675, 800, 864, 968, 972, 1125, 1152, 1323, 1352.
Un número se considera poderoso, si al obtener sus divisores primos todos ellos tienen un exponente mayor o igual a 2. Por ejemplo, el 72 es un número poderoso, ya que sus divisores primos $2^3 \times 3^2$, ambos tienen exponente mayor igual a 2. Los primeros números poderosos son: 1, 4, 8, 9, 16, 25, 27, 32, 36, 49, 64, 72, 81, 100, 108, 121, 125, 128, 144, 169, 196.

Un número es de potencia perfecta cuando existen dos números naturales $m > 1$ y $k > 1$ tales que $n = m^k$, así el número 128 es de potencia perfecta porque su divisor primo es 2^7 y tanto 2 como 7 son mayores a 1. Otro ejemplo de número de potencia perfecta es el 100 donde $2^2 \times 5^2$ ya que todos los divisores primos tienen el mismo exponente, en este caso 2, haciendo que el número se pueda ver de la forma 10^2.

Se le pide que haciendo uso de lenguaje C elabore un programa que permita saber si un número es de Aquiles, poderoso o de potencia perfecta. Para ello debe , utilizando módulos, poder:

- Identificar cuál es el exponente al que se encuentra cierto divisor de un número (por ejemplo el 9 tendría para el divisor 3 un exponente de 2, el 6 tendría para el divisor 3 un exponente de 1)
- Identificar si un número es primo o no,
- Identificar si un número es poderoso y/o de potencia perfecta, haciendo uso de los dos módulos anteriores.

Su programa principal deberá, leer un rango de números a evaluar, verificar que este rango sea válido, mostrar un mensaje de error de no serlo. Si el rango es válido y considera valores positivos mayores a 0, para cada uno de los números en el rango identificar si es poderoso y/o perfecto y a continuación se muestre que tipo de número es, el número podría ser de Aquiles, podría ser poderoso o de potencia perfecta. Para realizar estas acciones debe utilizar uno o varios de los módulos creados. Su solución debe presentar al menos 4 módulos incluído el principal.

Caso de prueba para verificación de solución

Ingrese el rango de números a evaluar :71 81
El número 71 es potencia perfecta
El número 72 es Aquiles
El número 73 es potencia perfecta
El número 74 es potencia perfecta
El número 75 no es Aquiles
El número 76 no es Aquiles
El número 77 es potencia perfecta
El número 78 es potencia perfecta
El número 79 es potencia perfecta
El número 80 no es Aquiles
El número 81 es poderoso

Caso de prueba para verificación de solución

Ingrese el rango de números a evaluar :654 24
Datos de entrada inválidos

Caso de prueba para verificación de solución

Ingrese el rango de números a evaluar :-5 -54
Datos de entrada inválidos

2.3.11. Triples pitagóricos (adaptado del laboratorio 8 del ciclo 2020-2)

Un triple pitagórico se define como tres enteros positivos (a, b, c) dónde $a < b < c$ y $a^2 + b^2 = c^2$.

Se llaman triples primitivos si a, b, c son coprimos, es decir, si sus máximos divisores comunes por pares MCD (a, b) = MCD (a, c) = MCD (b, c) = 1.

Debido a su relación a través del teorema de Pitágoras, a, b y c son coprimos si a y b son coprimos (MCD (a, b) = 1).

Cada triple forma la longitud de los lados de un triángulo rectángulo, cuyo perímetro es P = a + b + c.

Determinar cuántos triples pitagóricos hay con un perímetro no mayor a 100 y el número de estos que son primitivos.

Se pide que desarrolle un programa en Lenguaje C, que solicite el valor máximo del perímetro que deben ser positivos y menor o igual a 100, de no cumplir con estas condiciones se debe mostrar un mensaje de error. Finalmente se debe imprimir todas los tripletes pitagóricos e indicar cuales son primitivos. Además de indicar el número de tripletas y cuantas son primitivas.

Debe usar el paradigma de programación modular y desarrollar como mínimo 4 módulos adicionales al principal. De los cuales como máximo debe tener uno para leer datos, uno para validar; además por lo menos uno de sus módulos debe utilizar parámetros por referencia y por lo menos uno debe devolver un valor.

Caso de prueba para verificación de solución

Ingresar valor de perímetro máximo: 100
9 16 25 Es primitivo
25 144 169 Es primitivo
36 64 100
49 576 625 Es primitivo
64 225 289 Es primitivo
81 144 225
81 1600 1681 Es primitivo
100 576 676
144 256 400
144 1225 1369 Es primitivo
225 400 625
225 1296 1521
256 900 1156
324 576 900
400 441 841 Es primitivo
441 784 1225
576 1024 1600
Hasta 100, hay 17 tripletas, de las cuales 7 son primitivas

Caso de prueba para verificación de solución

Ingresar valor de perímetro máximo: 0
Error perímetro cero o negativo

Caso de prueba para verificación de solución

Ingresar valor de perímetro máximo: 120
Error perímetro mayor al máximo permitido

2.3.12. Números casi-amigos y abundantes especiales (adaptado del laboratorio 8 del ciclo 2020-2)

En las matemáticas los números se han clasificado de diversas maneras de acuerdo con sus características: primos, perfectos, abundantes, amigos, casi-amigos entre otros.

Lo números casi-amigos son una pareja de números, en la cual cada uno de ellos es igual a la suma de los divisores del otro número menos 1. Por ejemplo los números 48 y 75 son casi-amigos porque 48=1+3+5+15+25-1 y 75=1+2+3+4+6+8+12+16+24-1.

Los números abundantes son aquellos que la suma de los divisores es mayor al propio número. Por ejemplo el número 12 es un numero abundante 1+2+3+4+6>12. Un tipo especial de números abundantes son aquellos que adicionalmente cumplen que la suma de sus divisores menos un divisor nunca es igual al número. Por ejemplo para el número 30, sus divisores son 1,2,3,5,6,10 y 15, la suma es mayor a 30 y las sumas de sus divisores restando uno de ellos son:
1+2+3+5+6+10=27, sin considerar el 15
1+2+3+5+6+15=32; sin considerar el 10
1+2+3+5+10+15=36, sin considerar el 6
1+2+3+6+10+15=37, sin considerar el 5
1+2+5+6+10+15=39, sin considerar el 3
1+3+5+6+10+15=40, sin considerar el 2
2+3+5+6+10+15=41, sin considerar el 1

Tenga en cuenta que el mismo número no se toma en cuenta en la suma de los divisores.

Se pide que desarrolle un programa en Lenguaje C que, solicite un rango de números enteros positivos, el tipo de números a mostrar y muestre la lista de números que cumplan con el tipo solicitado que se encuentran dentro del rango. Debe validar que el rango de números enteros positivos [inicio, fin] sea un rango válido: inicio<fin. El tipo de números que se puede ingresar es c o C para los números casi-amigos, e o E para los números abundantes especiales y t o T si desea mostrar los dos tipos de números. En caso el rango no sea válido o no se ingresen las letras designadas para el tipo de números, debe mostrar un mensaje adecuado y terminar el programa. Debe mostrar también la cantidad del tipo de números solicitados en el rango.

Debe usar el paradigma de programación modular y desarrollar como mínimo 4 módulos adicionales al principal. De los cuales como máximo debe tener uno para leer datos, uno para validar; además por lo menos uno de sus módulos debe utilizar parámetros por referencia y por lo menos uno debe devolver un valor. Solo puede tener un módulo donde sume y reste los divisores de un número.

Caso de prueba para verificación de solución

Ingrese inicio y fin:1 100
Ingrese el tipo de números que desea mostrar: e
Números abundantes especiales:
1. 30
2. 36
3. 42
4. 48
5. 54
6. 60
7. 66
8. 70
9. 72
10. 78
11. 80
12. 84
13. 90
14. 96
15. 100
En el rango [1,100] existen 15 números abundantes especiales

Caso de prueba para verificación de solución

Ingrese inicio y fin:40 80
Ingrese el tipo de números que desea mostrar: C Parejas de números casi amigos:
1. (48,75)
2. (75,48)
En el rango [40,80] existen 2 parejas de números casi-amigos

Caso de prueba para verificación de solución

Ingrese inicio y fin:30 10
Ingrese el tipo de números que desea mostrar: a
El rango y/o el tipo de números no es correcto

Caso de prueba para verificación de solución

Ingrese inicio y fin:1 200
Ingrese el tipo de números que desea mostrar: T
Parejas de números casi amigos:
1. (48,75)
2. (75,48)
3. (140,195)
4. (195,140)
En el rango [1,200] existen 4 parejas de números casi-amigos
Números abundantes especiales:
1. 30
2. 36
3. 42
4. 48
5. 54
6. 60
7. 66
8. 70
9. 72
10. 78
11. 80
12. 84
13. 90
14. 96
15. 100
16. 102
17. 108
18. 112
19. 114
20. 120
21. 126
22. 132
23. 138
24. 140
25. 144
26. 150
27. 156
28. 160
29. 162
30. 168
31. 174
32. 176
33. 180
34. 186
35. 192
36. 198
37. 200
En el rango [1,200] existen 37 números abundantes especiales

2.3.13. Secuencia de números de Van der Corput (adaptado del laboratorio 8 del ciclo 2020-2)

Una secuencia de Van der Corput es una secuencia sobre el intervalo unitario publicada por primera vez en 1935 por el matemático holandés Van der Corput. Se construye invirtiendo la representación en base b de la secuencia de números naturales (0, 1, 2, 3, ...), donde el 0 a menudo no se incluye.
Un caso particular es la secuencia binaria de Van der Corput que se puede escribir como:
$0_2, 0.1_2, 0.01_2, 0.11_2, 0.001_2, 0.101_2, 0.011_2, 0.111_2, 0.0001_2, 0.1001_2, 0.0101_2, 0.1101_2, 0.0011_2, 0.1011_2,$
$0.0111_2, 0.1111_2$
o, en forma fraccionada, como:

$\frac{0}{1}, \frac{1}{2}, \frac{1}{4}, \frac{3}{4}, \frac{1}{8}, \frac{5}{8}, \frac{3}{8}, \frac{7}{8}, \frac{1}{16}, \frac{9}{16}, \frac{5}{16}, \frac{13}{16}, \frac{3}{16}, \frac{11}{16}, \frac{7}{16}, \frac{15}{16}...$

Donde los numeradores, excepto el primer término, son necesariamente impares, mientras que los denominadores son potencias de dos, por lo tanto, coprimos, y todas las fracciones anteriores ya están en forma reducida. Para entender mejor el problema tenemos en la Figura 2.10 la siguiente representación binaria:

Binario	Representación
0.	$0*2^0 = 0$
1.	$1*2^0 = 1$
10.	$1*2^1+0*2^0 = 2$
11.	$1*2^1+1*2^0 = 3$

Figura 2.10: Ejemplar de representación binaria

Para la secuencia de números de Van der Corput tenemos que se puede tener dígitos binarios a la derecha del "punto", al igual que en el sistema numérico decimal (ver Figura 2.11).

Binario	Representación
.0	$0*2^0 = 0$
.1	$1*2^{-1} = 1/2$
.01	$0*2^{-1}+1*2^{-2} = 1/4$
.11	$1*2^{-1}+1*2^{-2} = 3/4$

Figura 2.11: Ejemplar de representación binaria

Se pide que desarrolle un programa en Lenguaje C, que pueda reproducir la secuencia de números de Van der Corput en base 2 (binario) y base 3 de los "n" primeros números entre 1 y 9.

Debe usar el paradigma de programación modular y desarrollar como mínimo 4 módulos adicionales al principal. De los cuales como máximo debe tener uno para leer datos, uno para validar; además por lo menos uno de sus módulos debe utilizar parámetros por referencia y por lo menos uno debe devolver un valor.

Caso de prueba para verificación de solución

Ingresar el valor de n: 4
base 2: 0 1/2 1/4 3/4
base 3: 0 1/3 2/3 1/9

Caso de prueba para verificación de solución

Ingresar el valor de n: 7
base 2: 0 1/2 1/4 3/4 1/8 5/8 3/8
base 3: 0 1/3 2/3 1/9 4/9 7/9 2/9

Caso de prueba para verificación de solución

Ingresar el valor de n: 0
Error en el dato de entrada, valor fuera del rango permitido

Caso de prueba para verificación de solución

Ingresar el valor de n: 10
Error en el dato de entrada, valor fuera del rango permitido

2.3.14. Números de Woodall (adaptado del laboratorio 9 del ciclo 2020-2)

En teoría de números, un número de Woodall (W_x), para cualquier número natural n, es cualquier número natural de la forma: $W_n = n * 2^n - 1$

Los primeros números de Woodall son: 1, 7, 23, 63, 159, 383, 895, ...

Los primeros en estudiar los números de Woodall fueron Allan J. C. Cunningham y H. J. Woodall en 1917, inspirados por los estudios iniciales de James Cullen sobre los similarmente definidos números de Cullen.

Los números de Woodall que también son números primos se les denomina números primos de Woodall; los primeros exponentes n a los cuales corresponden números primos de Woodall W_n son 2, 3, 6, 30, 75, 81, 115, 123, 249, 362, 384, ...

Se pide que desarrolle un programa en Lenguaje C, que permita validar una cantidad de números de woodall. Para ello debe ingresar la cantidad de números de woodall a encontrar. Esta cantidad debe ser mayor que cero, en caso no sea así debe terminar el programa. Si la cantidad es correcta, debe pedirle al usuario que ingrese una lista de números a evaluar. Por cada número ingresado debe evaluar si el número es natural, si no es así, debe mostrar un mensaje de error y pasar al siguiente número. Si el número es natural, debe evaluar si es de woodall y debe mostrar un mensaje relacionado a dicha evaluación (ver casos de prueba). Para el caso que si se trate de un número de woodall, debe determinar si dicho número es primo de woodall, esto debido a que al final del proceso debe mostrar cuál fue el mayor número primo de woodall encontrado con su respectivo exponente.

Se dejan de ingresar números a evaluar cuando ya se hayan encontrado la cantidad de números de woodall solicitados.

Debe usar el paradigma de programación modular y desarrollar como mínimo 4 módulos adicionales al principal. De los cuales como máximo debe tener uno para leer datos, uno para validar; además por lo menos uno de sus módulos debe utilizar parámetros por referencia y por lo menos uno debe devolver un valor.

Caso de prueba para verificación de solución

Ingrese la cantidad de números primos de woodall a encontrar: -2
La cantidad de números primos de woodall ingresado no es correcto.

Caso de prueba para verificación de solución

Ingrese la cantidad de números primos de woodall a encontrar: 3
Ingrese el número a evaluar: -5
El número ingresado no es natural.
Ingrese el número a evaluar: 1540
El número 1540 no es de woodall.
Ingrese el número a evaluar: 895
El número 895 es de woodall con exponente 7.
Ingrese el número a evaluar: 159
El número 159 es de woodall con exponente 5.
Ingrese el número a evaluar: 654
El número 654 no es de woodall.
Ingrese el número a evaluar: 63
El número 63 es de woodall con exponente 4.
Ninguno de los números de woodall encontrados fueron primos.

Caso de prueba para verificación de solución

Ingrese la cantidad de números primos de woodall a encontrar: 2
Ingrese el número a evaluar: 383
El número 383 es de woodall con exponente 6.
Ingrese el número a evaluar: 15
El número 1540 no es de woodall.
Ingrese el número a evaluar: 895
El número 895 es de woodall con exponente 7.
EL mayor número primo de woodall encontrado fue 383 con exponente 6.

2.3.15. Distancia entre planetas (adaptado del laboratorio 9 del ciclo 2020-2)

La unidad astronómica (abreviada UA) es una unidad de longitud igual a 149'597,870.7 km. Esta longitud es equivalente a la distancia media entre la Tierra y el Sol (para este tipo de situaciones, se usa la distancia media porque las órbitas de los planetas no son circulares, sino elípticas).

Las distancias medias entre el Sol y el resto de planetas se muestra a continuación:

- Mercurio: 0.39 UA
- Venus: 0.72 UA
- Marte: 1.52 UA
- Ceres: 2.77 UA
- Júpiter: 5.20 UA
- Saturno: 9.58 UA
- Urano: 19.23 UA
- Neptuno: 30.10 UA
- Plutón: 39.3 UA
- Eris: 96.4 UA

Se solicita implementar un programa en lenguaje C que permita medir la distancia entre dos o más planetas.

Dado que los planetas giran alrededor del Sol, estos difícilmente se encontrarán en línea recta (motivo por el que no puede calcular la distancia entre dos planetas solamente restando sus respectivas distancias con respecto al Sol).

Para solucionar esto, el programa deberá solicitar al usuario un ángulo (en grados sexagesimales) que indique la posición de cada planeta. Viendo el Sistema Solar en un plano cartesiano con el Sol en el origen de coordenadas, si el ángulo de un planeta es 0 o 180, esto significará que está exactamente en el eje X, Si el ángulo es 90 o 270, esto significará que está en el eje Y.

Por ejemplo, si la tierra se encuentra a 75 grados con respecto al eje X, y Marte a solo 40 grados, entonces la distancia entre estos dos planetas será 0.905637 UA (ver Figura 2.12).

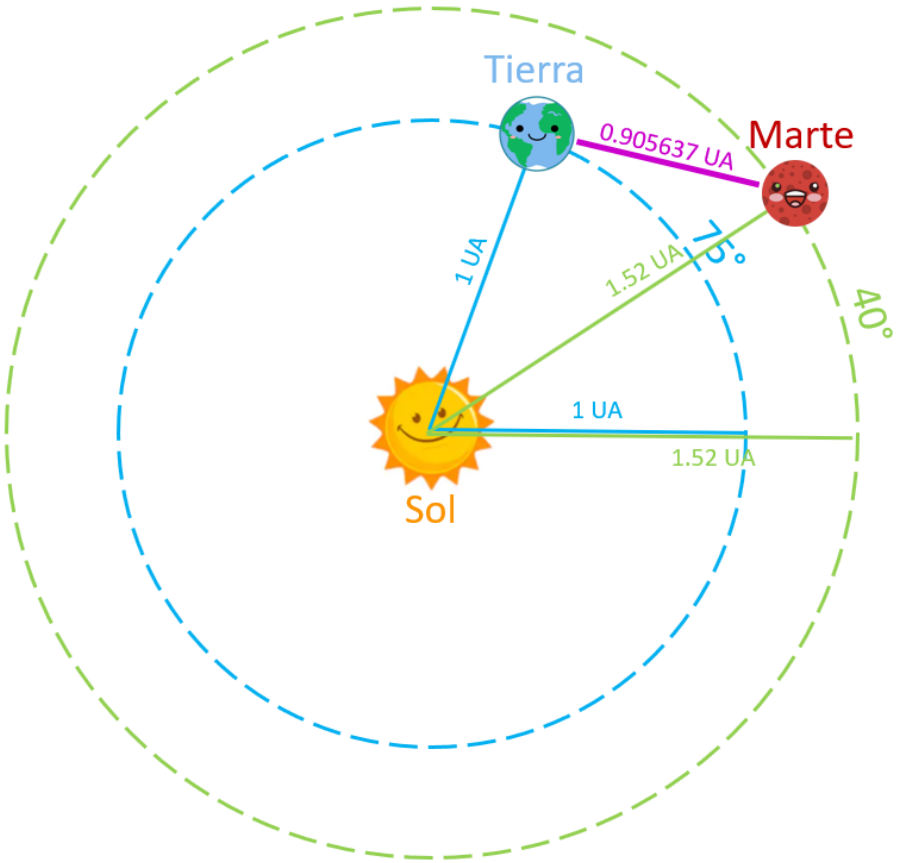

Figura 2.12: distancia entre la Tierra y Marte a 75 y 40 grados con respecto al eje X

El programa se deberá comportar de la siguiente manera:

- Solicitar al usuario un primer planeta y su ángulo con respecto al eje x, este deberá ser M, V, T, m, C, J, S, U, N, P, E (notar que se usa M mayúscula para Mercurio y m minúscula para Marte). Si el primer planeta es diferente a estos valores, el programa deberá terminar.
- Para cada primer planeta, el programa deberá solicitar al usuario varios segundo planeta. Para cada Segundo planeta, se deberá ingresar la letra de dicho segundo planeta y su ángulo con respecto al eje X.

  Para cada pareja válida de primer planeta y segundo planeta, el programa deberá imprimir la distancia entre estos dos.

  El programa deberá seguir solicitando otro segundo planeta con su respectivo ángulo y calcular la distancia entre el primer planeta y el nuevo segundo planeta hasta que se ingrese un segundo planeta diferente

de M, V, T, m, C, J, S, U, N, P, E.

Asuma que el primer planeta y el segundo planeta siempre son planetas diferentes. Por otro lado, al momento de ingresar las letras del segundo planeta, se puede repetir el mismo segundo planeta varias veces y se puede repetir también el ángulo (ver casos de prueba).

- Una vez que se ingrese un segundo planeta diferente de M, V, T, m, C, J, S, U, N, P, E, el programa deberá indicar cuál fue la mayor distancia de todas (no es necesario que imprima la letra de los planetas que generaron esta distancia). En caso de que no haya habido ningún segundo planeta válido (y por lo tanto ninguna distancia, no deberá imprimir nada).
- Se deberá volver a pedir otro primer planeta y repetir el proceso hasta que se ingrese un primer planeta diferente de M, V, T, m, C, J, S, U, N, P, E.

Deberá utilizar una función para calcular la distancia entre dos puntos, una función para leer el ángulo (las lecturas de las letras del primer planeta y del segundo planeta deberán ser implementadas en main), una función para obtener la distancia media de un planeta con respecto al Sol (a través de su letra, por ejemplo, 0.39 para M, 0.75 para V, 1 para T, 1.52 para m, 2.77 para C, 5.2 para J, etc.), y una función para calcular las coordenadas de un planeta, dada su distancia con respecto al Sol y su ángulo con respecto al eje X. Por ejemplo, Para Marte, si su ángulo con respecto al eje X es 0 grados, su posición será (1.52, 0), si su ángulo es 90 grados con respecto al eje X, entonces su posición será (0, 1.52), en cambio, si su ángulo es 180, su posición será (-1.52, 0). Considere que el Sol se encuentra en el origen de coordenadas (0,0).

Caso de prueba para verificación de solución

```
Ingrese primer planeta: T
ingrese ángulo: 45
Ingrese segundo planeta: Q
Planeta desconocido.

Ingrese primer planeta: T
ingrese ángulo: 75
Ingrese segundo planeta: m
ingrese ángulo: 40
la distancia es: 0.905637 UA
Ingrese segundo planeta: m
ingrese ángulo: 40
la distancia es: 0.905637 UA
Ingrese segundo planeta: m
ingrese ángulo: 75
la distancia es: 0.52 UA
Ingrese segundo planeta: X
Planeta desconocido.
La mayor distancia fue 0.905637 UA

Ingrese primer planeta: W
Planeta desconocido.
```

2.3.16. Particularidades de los números (adaptado del laboratorio 9 del ciclo 2020-2)

Una particularidad que existe en los números es que todos sus dígitos pueden ser consecutivos, ya sea de forma ascendente o descendente.

Por ejemplo, el número 8765 tiene todos sus dígitos de forma consecutiva en orden descendente, así como el número 34567 tiene todos sus dígitos de forma consecutiva en orden ascendente.

También tenemos estos ejemplos, el número 321098 tiene todos sus dígitos de forma consecutiva en orden descendente pues después del dígito 0 continúa el dígito 9 y sigue el orden descendente. Otro ejemplo sería el número 89012, el cuál tiene todos sus dígitos de forma consecutiva en orden ascendente pues después del dígito 9 continúa el dígito 0 y sigue el orden descendente

Esta claro que el orden de descendente o ascendente solo se puede definir si los números tienen 2 o más dígitos, por ello no se considerarán los números de un dígito.

Se pide que desarrolle un programa en Lenguaje C, que permita validar una cantidad de números y verifique por cada uno de ellos si todos sus dígitos son consecutivos y en caso de serlo debe indicar si el orden es ascendente o descendente. Para ello debe ingresar cada número a evaluar. Si el número ingresado es menor o igual a 0 o tiene solo un dígito entonces el programa termina. Si el número es correcto, debe evaluar lo solicitado y mostrar los mensajes correspondientes, de acuerdo a lo mostrado en los casos de prueba. Al finalizar debe mostrar cuantos números ingresados tienen sus dígitos de forma consecutiva, tanto en orden ascendente como descendente.

Debe usar el paradigma de programación modular y desarrollar como mínimo 4 módulos adicionales al principal. De los cuales como máximo debe tener uno para leer datos, uno para validar; además por lo menos uno de sus módulos debe utilizar parámetros por referencia y por lo menos uno debe devolver un valor.

Caso de prueba para verificación de solución

Ingrese el número a evaluar: -2
Se tienen 0 números con dígitos consecutivos de forma ascendente Se tienen 0 números con dígitos consecutivos de forma descendente

Caso de prueba para verificación de solución

Ingrese el número a evaluar: 34567
El número tiene todos sus dígitos consecutivos de forma ascendente.
Ingrese el número a evaluar: 1540
El número no tiene todos sus dígitos consecutivos.
Ingrese el número a evaluar: 32109
El número tiene todos sus dígitos consecutivos de forma descendente.
Ingrese el número a evaluar: 43210
El número tiene todos sus dígitos consecutivos de forma descendente.
Ingrese el número a evaluar: 8
Se tienen 1 números con dígitos consecutivos de forma ascendente
Se tienen 2 números con dígitos consecutivos de forma descendente

2.3.17. Triángulos y Teorema de Pitágoras (adaptado del laboratorio 9 del ciclo 2020-2)

Se tiene un triángulo rectángulo con lado 1 horizontal y lado 2 vertical. Se desea dividir este triángulo en N piezas, de modo que el último trapecio formado tenga un ancho de a unidades, el penúltimo trapecio tenga un ancho de $2a$ unidades, y así sucesivamente hasta llegar a la región triangular que tenga $n \times a$ unidades.

Por ejemplo, para N = 8, la región triangular llegaría hasta $8a$ unidades y, en total, el lado 1 tendría una longitud de $36a$ (ver Figura 2.13):

Por ejemplo, si el lado 1 del triángulo tuviera una longitud de 10, y este se quisiera dividir en 8 partes, entonces el último trapecio de la derecha tendría sería 10/36 unidades de ancho, el penúltimo trapecio de la derecha sería 2(10/36) unidades de ancho, y así sucesivamente hasta llegar a la región triangular de la izquierda que sería 8(10/36) unidades de ancho.

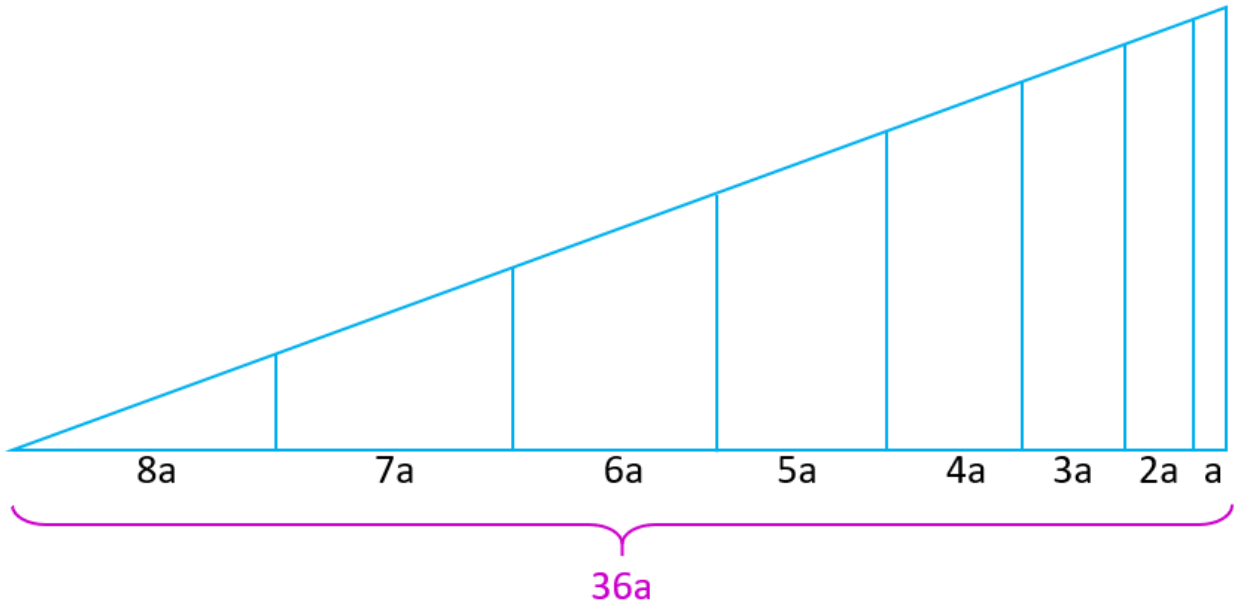

Figura 2.13: triángulo rectángulo dividido en 8 partes

Para llegar a estos valores, se dividió el lado 1 del triángulo en 8 partes, de modo que la última parte de la derecha mida 1a, la penúltima parte mida 2a, y así, hasta llegar a la región triangular de la izquierda que mide 8a, de tal modo que 8a + 7a + 6a + 5a + 4a + 3a + 2a + a = 10.

Se solicita: Implementar un programa en lenguaje C que permita al usuario ingresar el lado 1 de un triángulo rectángulo (el lado 1 será el lado horizontal de la parte inferior), el lado 2 el triángulo (el lado 2 será el lado vertical de la parte derecha), y calcule e imprima el tercer lado (para esto, deberá hacer uso del Teorema de Pitágoras). Asuma que los lados siempre son positivos.

En seguida, el programa deberá permitir que el usuario divida ese triángulo en N partes, según la lógica explicada párrafos arriba.

El número de partes deberá ser una variable ingresada por teclado. El triángulo dividido en N=8 partes fue solo un ejemplo. Por favor, entienda que no todos los triángulos se van a dividir en 8 partes sino en la cantidad de partes (N) que el usuario ingrese por teclado.

El programa deberá calcular el área de cada región y, al terminar de calcular las n áreas, deberá indicar cuál fue la región con mayor área de todas y volver a imprimir dicha área.

En seguida, el programa deberá imprimir el área total del triángulo ingresado inicialmente, y volver a pedir otro valor de N para repetir el proceso.

El programa deberá seguir pidiendo más valores de N hasta que se ingrese un valor menor a 2. En ese momento, el programa deberá imprimir el mensaje `"Deben haber por lo menos 2 partes"` y finalizar la ejecución inmediatamente.

El programa deberá incluir los siguientes módulos:

- El módulo principal
- Un módulo que calcule el denominador a utilizar para determinar el ancho de cada parte. Por ejemplo, si n es 8, entonces el denominador a usar será 36, debido a que 8+7+6+5+4+3+2+1 es 36 y esto significa que los lados tendrán un ancho de 8/36, 7/36, 6/36, 5/36, 4/36, 3/36, 2/36 y 1/36.

  Este valor de denominador deberá ser retornado a main. Una vez retornado este valor a main, el módulo principal contará podrá calcular el ancho de cada región (para la primera región hay que dividir el lado 1 entre el denominador y multiplicar dicho valor por n (en el ejemplo con n=8, la primera región tiene

un ancho de 8/36 del valor del lado 1).

- Un módulo que permita calcular el área de un triángulo rectángulo como el mostrado en la Figura (con un lado completamente horizontal y otro lado completamente vertical).
- Un módulo que valide si el valor de n es mayor o igual a 2. Este módulo retornará 1 si es que el valor de n es mayor o igual a 2, y retornará 0 si es que no lo es.

Caso de prueba para verificación de solución

```
Ingrese lado 1: 10
Ingrese lado 2: 4
EL lado 3 mide: 10.770330

ingrese numero de partes: 8
El área 1 es: 0.987654
El área 2 es: 2.484568
El área 3 es: 3.333333
El área 4 es: 3.626543
El área 5 es: 3.456790
El área 6 es: 2.916667
El área 7 es: 2.098765
El área 8 es: 1.095679
El área total es: 20.000000
El área mayor fue el área 4 con un valor de 3.626543

ingrese numero de partes: 1
Deben haber por lo menos 2 partes
```

Caso de prueba para verificación de solución

```
ingrese numero de partes: -5
Deben haber por lo menos 2 partes
```

2.3.18. Números primos cíclicos (adaptado del laboratorio 9 del ciclo 2020-2)

Un número primo circular es un número primo con la propiedad de que el número generado en todas las rotaciones de sus dígitos producen números primos.

Por ejemplo: 197 es un número primo circular, ya que 197, 971 y 719 también son primos.
1193 es un número primo circular, ya que 1931 , 9311 y 3119 también son primos.

Tenga en cuenta que un número que es generado de una permutación cíclica de un número primo circular más pequeño no se considera en sí mismo un número primo circular. Entonces 13 es un primo circular, pero 31 no lo es.

Se pide que desarrolle un programa en Lenguaje C, que pueda reproducir los "n"primeros números primos circulares. Para ello se debe ingresar un número positivo pero menor o igual a 20 de no cumplir con esa condición debe emitir un mensaje de error.
Debe usar el paradigma de programación modular y desarrollar como mínimo 4 módulos adicionales al principal. De los cuales como máximo debe tener uno para leer datos, uno para validar; además por lo menos uno de sus módulos debe utilizar parámetros por referencia y por lo menos uno debe devolver un valor.

Caso de prueba para verificación de solución

Ingresar la cantidad de primos circulares a obtener: 13
Los primeros 13 primos circulares :
2, 3, 5, 7, 11, 13, 17, 37, 79, 113, 197, 199, 337

Caso de prueba para verificación de solución

Ingresar la cantidad de primos circulares a obtener: 21
Error valor no permitido

Caso de prueba para verificación de solución

Ingresar la cantidad de primos circulares a obtener: 0
Error valor no permitido

Caso de prueba para verificación de solución

Ingresar la cantidad de primos circulares a obtener: 5
Los primeros 5 primos circulares :
2, 3, 5, 7, 11

2.3.19. Números curiosos (adaptado del laboratorio 9 del ciclo 2020-2)

En matemáticas existen números curiosos que tienen la siguiente propiedad: Si multiplicamos el número por 2, 3, 4, 5, 6, 7, 8 o 9, al menos una de las multiplicaciones nos dará resultado un número que tiene exactamente los mismos dígitos del original pero en diferente orden.

Por ejemplo, para el número 142857:

142857 x 2 = 285714
142857 x 3 = 428571
142857 x 4 = 541728
142857 x 5 = 714285
142857 x 6 = 857142
142857 x 7 = 999999
142857 x 8 = 1142856
142857 x 9 = 1285713

Aquí vemos que al multiplicar el número por 2, 3, 4, 5 y 6, el resultado de dichas multiplicaciones cumple con la propiedad de que dará como resultado un número que tiene exactamente los mismos dígitos del original pero en diferente orden.

Se pide que desarrolle un programa en Lenguaje C, que pueda leer un número, valide que el mismo sea mayor que 0, en caso no lo sea, debe mostrar un mensaje de error y el programa termina. En caso la validación sea correcta, debe evaluar si el número es curioso porque cumple con la propiedad indicada. En caso cumpla, debe indicar con que número o números al multiplicar se cumple la propiedad.

Debe usar el paradigma de programación modular y desarrollar como mínimo 4 módulos adicionales al principal. De los cuales como máximo debe tener uno para leer datos, uno para validar; además por lo menos uno

de sus módulos debe utilizar parámetros por referencia y por lo menos uno debe devolver un valor.

Caso de prueba para verificación de solución

Ingresar número a evaluar: -4
El número ingresado es incorrecto.

Caso de prueba para verificación de solución

Ingresar número a evaluar: 142857
El número cumple la propiedad al multiplicarlo por 2 que da como resultado 285714
El número cumple la propiedad al multiplicarlo por 3 que da como resultado 428571
El número cumple la propiedad al multiplicarlo por 4 que da como resultado 571428
El número cumple la propiedad al multiplicarlo por 5 que da como resultado 714285
El número cumple la propiedad al multiplicarlo por 6 que da como resultado 857142

Caso de prueba para verificación de solución

Ingresar número a evaluar: 1457
El número no cumple la propiedad.

2.3.20. Operaciones a nivel de bits (adaptado del laboratorio 8 del ciclo 2020-1)

La información en el computador se almacena en bytes, cada byte está compuesto por 8 bits. Por ejemplo, la representación del número 4078 en 3 bytes, se muestra en la Figura 2.33.

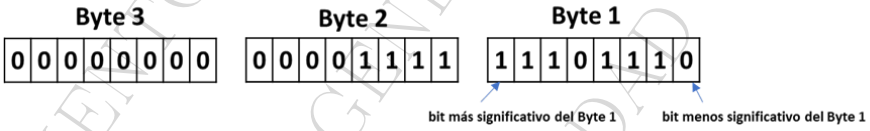

Figura 2.14: Representación del número 4078 en 3 bytes

Esta representación se realiza en el sistema binario, por lo cual para convertir un número en base 10 a base 2 debe utilizar el método de divisiones sucesivas hasta que el cociente sea cero y luego se forma el número (en base 2) con los restos obtenidos de abajo hacia arriba, como se muestra en la Figura 2.34.

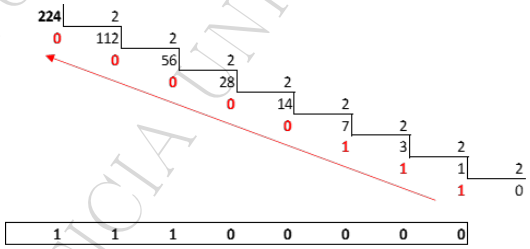

Figura 2.15: Divisiones sucesivas para convertir el número 4078 en base decimal a base 2

El número 224 se representa en 3 bytes como se muestra en la Figura 2.35.

Con los bits se pueden realizar diversas operaciones lógicas AND, OR, XOR, entre otras. La operación lógica AND, devuelve 1 si los dos bits tienen el valor de 1, en otro caso devuelve 0 como se muestra en la Figura 2.36.

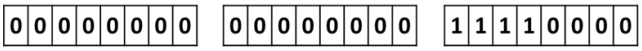

Figura 2.16: Representación del número 224 en 3 bytes

bit1	bit2	bit1 AND bit2
1	1	1
1	0	0
0	1	0
0	0	0

Figura 2.17: Operación AND

La operación lógica OR, devuelve 1 si uno de los dos bits tiene el valor de 1, en otro caso devuelve 0 como se muestra en la Figura 2.37.

La operación lógica XOR, devuelve 1 si el valor de los bits es diferente, en otro caso devuelve 0 como se muestra en la Figura 2.26.

Se pide que desarrolle un programa en Lenguaje C que reciba una lista de: dos números enteros positivos y una operación lógica (AND: A a, OR: O o, XOR: X x), y muestre la representación en 3 bytes de cada número, el resultado de la operación lógica y el número decimal que forma el resultado de la operación. Para terminar de ingresar la lista de números, por lo menos uno de ellos debe ser negativo. Antes de realizar cualquier operación debe validar que los números ingresados sean de 16777215 ($2^{24} - 1$).

Para el desarrollo del programa debe desarrollar como mínimo 6 módulos adicionales al main. De estos módulos, por lo menos 2 deben simular el uso de parámetros por referencia, ninguno de ellos debe usarse para leer datos. Además, debe desarrollar como mínimo 3 módulos que devuelvan un valor.

Se sugiere que cada byte sea una variable de tipo entero.

Recuerde que para transformar un número entero de cualquier base a base decimal debe usar el método de multiplicaciones sucesivas. Cada dígito en la base original se multiplica por la potencia de la base original elevada a la posición del dígito, como se muestra en la Figura 2.27.

Use los siguientes casos de prueba para verificar si su solución está correcta.

```
000000000 000000000 00010010
Número 2 en bytes:
000000000 000000000 00011000
Resultado de la operación AND a nivel de bits:
000000000 000000000 00010000
El resultado en base 10 es 16
```

bit1	bit2	bit1 OR bit2
1	1	1
1	0	1
0	1	1
0	0	0

Figura 2.18: Operación OR

bit1	bit2	bit1 XOR bit2
1	1	0
1	0	1
0	1	1
0	0	0

Figura 2.19: Operación XOR

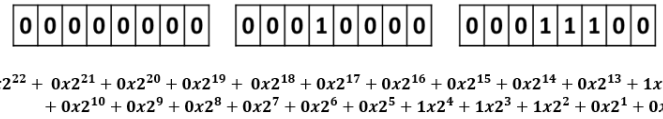

Figura 2.20: Multiplicaciones sucesivas para la representación en 3 bytes del número 4124

```
Ingrese los números a procesar: 18 24
Ingrese la operación a realizar: o
Número 1 en bytes:
000000000 000000000 00010010
Número 2 en bytes:
000000000 000000000 00011000
Resultado de la operación OR a nivel de bits:
000000000 000000000 00011010
El resultado en base 10 es 26
Ingrese los números a procesar: 18 24
Ingrese la operación a realizar: X
Número 1 en bytes:
000000000 000000000 00010010
Número 2 en bytes:
000000000 000000000 00011000
Resultado de la operación XOR a nivel de bits:
000000000 000000000 00001010
El resultado en base 10 es 10
Ingrese los números a procesar: -2 5
Fin de los cálculos


Ingrese los números a procesar: 16777215 8
Ingrese la operación a realizar: x
Número 1 en bytes:
11111111 11111111 11111111
Número 2 en bytes:
000000000 000000000 00001000
Resultado de la operación XOR a nivel de bits:
11111111 11111111 11110111
El resultado en base 10 es 16777207
Ingrese los números a procesar: 456 16777210
Ingrese la operación a realizar: O
Número 1 en bytes:
000000000 00000001 11001000
Número 2 en bytes:
11111111 11111111 11111010
Resultado de la operación OR a nivel de bits:
11111111 11111111 11111010
El resultado en base 10 es 16777210
Ingrese los números a procesar: -6 3
```

```
Fin de los cálculos


Ingrese los números a procesar: 16777999   456
Números fuera del rango
Ingrese los números a procesar: 897  65
Ingrese la operación a realizar: s
Operación incorrecta
Ingrese los números a procesar: 5 -4
Fin de los cálculos
```

2.3.21. Rotación de bits - Complemento a 2 (adaptado del laboratorio 8 del ciclo 2020-1)

La información en el computador se almacena en bytes, cada byte está compuesto por 8 bits. Por ejemplo, la representación del número 4078 en 3 bytes, se muestra en la Figura 2.33.

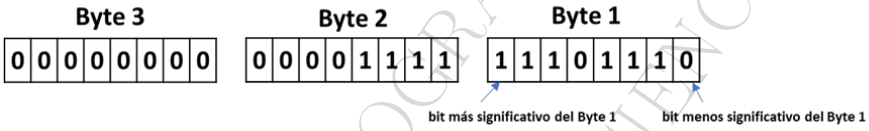

Figura 2.21: Representación del número 4078 en 3 bytes

Esta representación se realiza en el sistema binario, por lo cual para convertir un número en base 10 a base 2 debe utilizar el método de divisiones sucesivas hasta que el cociente sea cero y luego se forma el número (en base 2) con los restos obtenidos de abajo hacia arriba, como se muestra en la Figura 2.34.

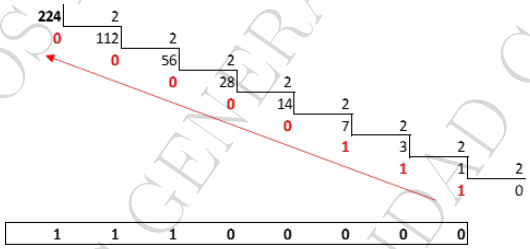

Figura 2.22: Divisiones sucesivas para convertir el número 4078 en base decimal a base 2

El número 224 se representa en 3 bytes como se muestra en la Figura 2.35.

Figura 2.23: Representación del número 224 en 3 bytes

En la Figura 2.36, se muestra el resultado de la rotación hacia la izquierda de 3 bits del número 224 (almacenado en 3 bytes), transformando estos bytes a un número decimal, se forma el número 28.

En la Figura 2.37 se muestra paso a paso como se realiza cada rotación a la izquierda de 3bits para el número 224. En la 1era rotación a la izquierda, se realiza lo siguiente:

- Se saca el bit menos significativo de cada byte y se deja el byte con los 7 bits más significativos:
  - Para el 1er byte, el bit menos significativo es 0 y el 1er byte se quedará con 1111000
  - Para el 2do byte, el bit menos significativo es 0 y el 2do byte se quedará con 0000000
  - Para el 3er byte, el bit menos significativo es 0 y el 3er byte se quedará con 0000000

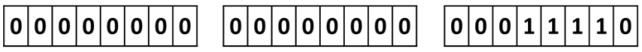

Figura 2.24: Rotación de 3 bits a la izquierda del número 224, forma el número 28

Figura 2.25: Detalle de la rotación de 3 bits a la izquierda del número 224

- El bit menos significativo del 1er byte se coloca como el más significativo del 3er byte. Entonces, el 3er byte queda con 00000000
- El bit menos significativo del 3do byte se coloca como el más significativo del 2do byte, el 2do byte queda con 00000000.
- El bit menos significativo del 2do byte se coloca como el más significativo del 1er byte, el 1er byte queda con 01111000.

Los mismos pasos se repiten para la 2da y 3era rotación.

En la Figura 2.26, se muestra paso a paso como se realiza cada rotación de bits hacia la derecha, rotando 2 bits del número 1031, transformando estos bytes a un número decimal, se forma el número 4124.

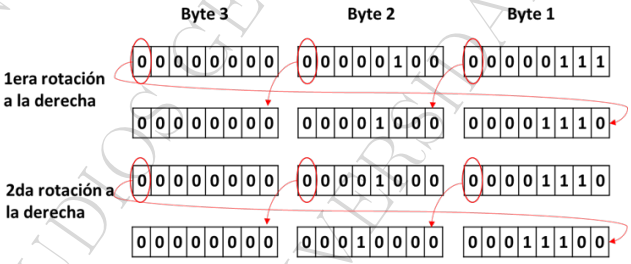

Figura 2.26: Detalle de la rotación de 2 bits a la derecha del número 1031

En la 1era rotación a la derecha, se realiza lo siguiente:

- Se saca el bit más significativo de cada byte y se deja el byte con los 7 bits menos significativos:
  - Para el 1er byte, el bit más significativo es 0 y el 1er byte se quedará con 0000100
  - Para el 2do byte, el bit más significativo es 0 y el 2do byte se quedará con 0000000
  - Para el 3er byte, el bit más significativo es 0 y el 3er byte se quedará con 0000000
- El bit más significativo del 1er byte se coloca como el menos significativo del 2do byte. Entonces, el 2do byte queda con 00001000
- El bit más significativo del 2do byte se coloca como el menos significativo del 3er byte, el 3er byte queda con 00000000
- El bit más significativo del 3er byte se coloca como el menos significativo del 1er byte, el 1er byte queda con 00001110

Los mismos pasos se repiten para la 2da rotación.

Se pide que desarrolle un programa en Lenguaje C que reciba una lista de números decimales enteros positivos, para cada número solicite el tipo de rotación derecha (D o d) o izquierda (I o i) y la cantidad de posiciones que se rotarán los bits (cantidad positiva menor o igual a 25) y muestre la representación en 3 bytes del número ingresado, la representación en 3 bytes del número rotado y el valor decimal del número rotado. La lista de números se terminará ingresando un número negativo.

Antes de realizar cualquier operación debe validar los datos de entrada: el número ingresado debe ser positivo y no mayor a 16777215 (que es el mayor número entero positivo que se puede representar con 3 bytes, $2^{24}-1$), el tipo de rotación debe ser d, D, i o I y la cantidad de posiciones debe ser positiva y menor a 25).

Para el desarrollo del programa debe desarrollar como mínimo 5 módulos adicionales al main. De estos módulos, 2 deben simular el uso de parámetros por referencia, ninguno de ellos debe usarse para leer datos. Además debe desarrollar como mínimo 2 módulos que devuelvan un valor.

Se sugiere que cada byte sea una variable de tipo entero.

Recuerde que para transformar un número entero de cualquier base a base decimal debe usar el método de multiplicaciones sucesivas. Cada dígito en la base original se multiplica por la potencia de la base original elevada a la posición del dígito, como se muestra en la Figura 2.27.

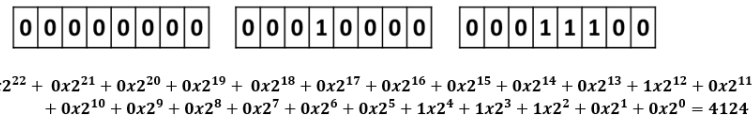

Figura 2.27: Multiplicaciones sucesivas para la representación en 3 bytes del número 4124

Use los siguientes casos de prueba para verificar si su solución está correcta.

```
Ingrese un número: 567845
Ingrese la cantidad de posiciones para rotar los bits: 5
Ingrese la dirección de rotación: d
El número en representado en 3 bytes: 00001000 10101010 00100101
El número con los bits rotados: 00010101 01000100 10100001
El número rotado en representación decimal: 1393825

Ingrese un número: 345
Ingrese la cantidad de posiciones para rotar los bits: 2
```

```
Ingrese la dirección de rotación: i
El número en representado en 3 bytes: 000000000 00000001 01011001
El número con los bits rotados: 01000000 000000000 01010110
El número rotado en representación decimal: 4194390

Ingrese un número: 99999999
Número ingresado fuera del límite

Ingrese un número: -45



Ingrese un número: 1378
Ingrese la cantidad de posiciones para rotar los bits: 10
Ingrese la dirección de rotación: I
El número en representado en 3 bytes: 000000000 00000101 01100010
El número con los bits rotados: 01011000 10000000 00000001
El número rotado en representacin decimal: 5799937

Ingrese un número: 456
Ingrese la cantidad de posiciones para rotar los bits: 45
La cantidad de posiciones ingresada es incorrecta

Ingrese un número: 865
Ingrese la cantidad de posiciones para rotar los bits: -2
La cantidad de posiciones ingresada es incorrecta

Ingrese un número: 5634
Ingrese la cantidad de posiciones para rotar los bits: 8
Ingrese la dirección de rotación: Z
La dirección de rotación es incorrecta

Ingrese un número: -1
```

2.3.22. Representación de números negativos en bytes - Complemento a 2 (adaptado del laboratorio 8 del ciclo 2020-1)

La información en el computador se almacena en bytes, cada byte está compuesto por 8 bits. Por ejemplo, la representación del número 4078 en 3 bytes, se muestra en la Figura 2.33.

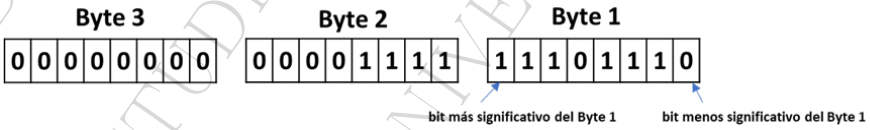

Figura 2.28: Representación del número 4078 en 3 bytes

Esta representación se realiza en el sistema binario, por lo cual para convertir un número en base 10 a base 2 debe utilizar el método de divisiones sucesivas hasta que el cociente sea cero y luego se forma el número (en base 2) con los restos obtenidos de abajo hacia arriba, como se muestra en la Figura 2.34.

El número 224 se representa en 3 bytes como se muestra en la Figura 2.35.

Para almacenar los números negativos enteros en el computador, se utiliza el **método de Complemento a 2**. Este método consiste en:

- Representar el número negativo como el complemento del número positivo. El complemento consiste en cambiar los bits que tienen 0 por 1 y los que tienen 1 por 0.

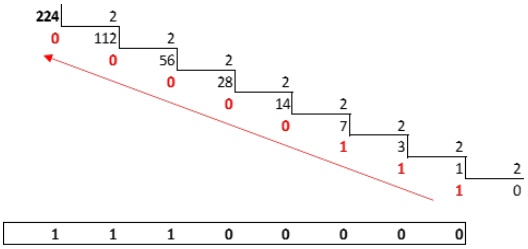

Figura 2.29: Divisiones sucesivas para convertir el número 4078 en base decimal a base 2

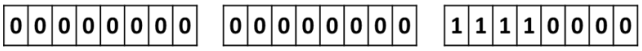

Figura 2.30: Representación del número 224 en 3 bytes

- Sumar a esta representación 1, la suma se realiza al bit menos significativo.

En la Figura 2.36, se muestra la representación del número negativo -335 en 3 bytes, usando el método de Complemento a 2. Al sumar 1 al bit menos significativo, el resultado es 1 porque se suma 0+1.

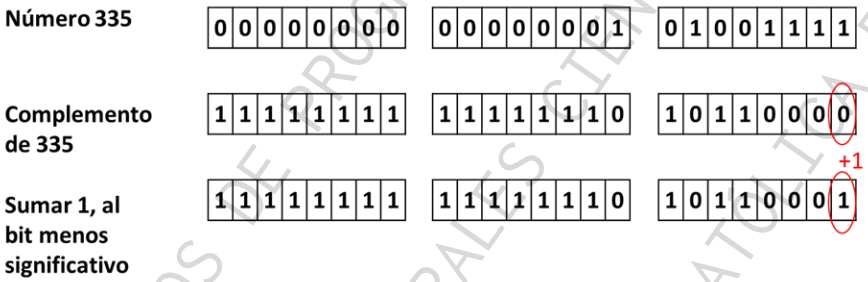

Figura 2.31: Representación del número -335 en 3 bytes usando el metodo de Complemento a 2

Para sumar bits, se deben seguir las siguientes reglas:

- $0 + 0 = 0$
- $0 + 1 = 1$
- $1 + 0 = 1$
- $1 + 1 = 0$, se acarrea uno para el siguiente bit más significativo.

En la Figura 2.37 se muestra la representación del número negativo -1038 en 3 bytes, usando el método de Complemento a 2. Al sumar el 1 al bit menos significativo, el resultado es 0 porque se suma 1+1, se acarrea 1 para el siguiente bit y se suma 1+0 cuyo resultado es 1.

Se pide que desarrolle un programa en Lenguaje C que reciba un rango de números enteros negativos, para cada número en el rango debe mostrar su representación en Complemento a 2 utilizando 3 bytes. Antes de realizar cualquier operación debe validar que el rango ingresado sea de números negativos y no puede ser menor a -8388607 = - $2^{23} + 1$.

Para el desarrollo del programa debe desarrollar como mínimo 5 módulos adicionales al main. De estos módulos, por lo menos 2 deben simular el uso de parámetros por referencia, ninguno de ellos debe usarse para leer datos. Además, debe desarrollar como mínimo 2 módulos que devuelvan un valor.

Se sugiere que cada byte sea una variable de tipo entero.

Use los siguientes casos de prueba para verificar si su solución está correcta.

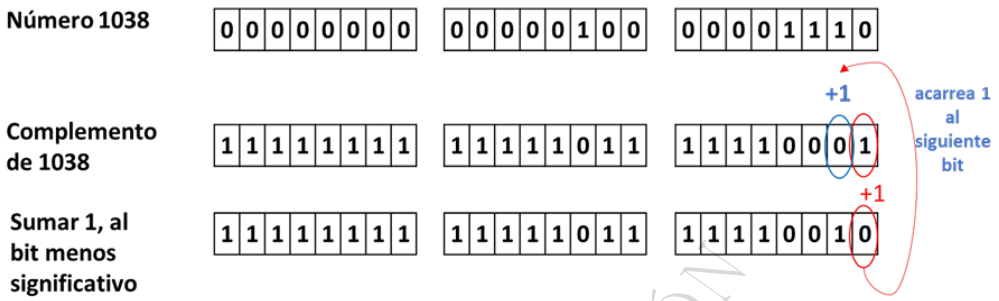

Figura 2.32: Representación del número -1038 en 3 bytes usando el metodo de Complemento a 2

```
Ingrese el rango de números: -1040    -1035
El número -1040 en complemento a 2 se representa como:
11111111  11111011  11110000
El número -1039 en complemento a 2 se representa como:
11111111  11111011  11110001
El número -1038 en complemento a 2 se representa como:
11111111  11111011  11110010
El número -1037 en complemento a 2 se representa como:
11111111  11111011  11110011
El número -1036 en complemento a 2 se representa como:
11111111  11111011  11110100
El número -1035 en complemento a 2 se representa como:
11111111  11111011  11110101


Ingrese el rango de números: -512   -510
El número -512 en complemento a 2 se representa como:
11111111  11111110  000000000
El número -511 en complemento a 2 se representa como:
11111111  11111110  00000001
El número -510 en complemento a 2 se representa como:
11111111  11111110  00000010


Ingrese el rango de números: -8388600 -8388595
El número -8388600 en complemento a 2 se representa como:
10000000  000000000  00001000
El número -8388599 en complemento a 2 se representa como:
10000000  000000000  00001001
El número -8388598 en complemento a 2 se representa como:
10000000  000000000  00001010
El número -8388597 en complemento a 2 se representa como:
10000000  000000000  00001011
El número -8388596 en complemento a 2 se representa como:
10000000  000000000  00001100
El número -8388595 en complemento a 2 se representa como:
10000000  000000000  00001101


Ingrese el rango de números:  -8388608    -8388600
Rango incorrecto


Ingrese el rango de números: -300   100
Rango incorrecto
```

2.3.23. Representación de números negativos en bytes - Invertir complemento a 2 (adaptado del laboratorio 8 del ciclo 2020-1)

La información en el computador se almacena en bytes, cada byte está compuesto por 8 bits. Por ejemplo, la representación del número 4078 en 3 bytes, se muestra en la Figura 2.33.

Figura 2.33: Representación del número 4078 en 3 bytes

Esta representación se realiza en el sistema binario, por lo cual para convertir un número en base 10 a base 2 debe utilizar el método de divisiones sucesivas hasta que el cociente sea cero y luego se forma el número (en base 2) con los restos obtenidos de abajo hacia arriba, como se muestra en la Figura 2.34.

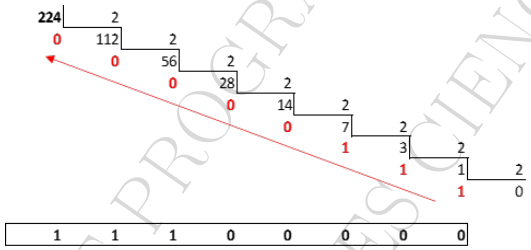

Figura 2.34: Divisiones sucesivas para convertir el número 4078 en base decimal a base 2

El número 224 se representa en 3 bytes como se muestra en la Figura 2.35.

Figura 2.35: Representación del número 224 en 3 bytes

Para almacenar los números negativos enteros en el computador, se utiliza el **método de Complemento a 2**. Este método consiste en:

- Representar el número negativo como el complemento del número positivo. El complemento consiste en cambiar los bits que tienen 0 por 1 y los que tienen 1 por 0.
- Sumar a esta representación 1, la suma se realiza al bit menos significativo.

En la Figura 2.36, se muestra la representación del número negativo -335 en 3 bytes, usando el método de Complemento a 2. Al sumar 1 al bit menos significativo, el resultado es 1 porque se suma 0+1.

Si se tiene un número negativo representado en 3 bytes usando el método de Complemento a 2, para poder obtener su representación en base 10, se siguen los siguientes pasos:

- Restar 1 al bit menos significativo.
- Invertir el complemento del número, cambiar los bits que tienen 1 por 0 y los que tienen 0 por 1.
- Usar el método de multiplicaciones sucesivas para transformar el número de base 2 a base 10.
- Multiplicar por -1 el número obtenido

Para restar bits, se deben seguir las siguientes reglas:

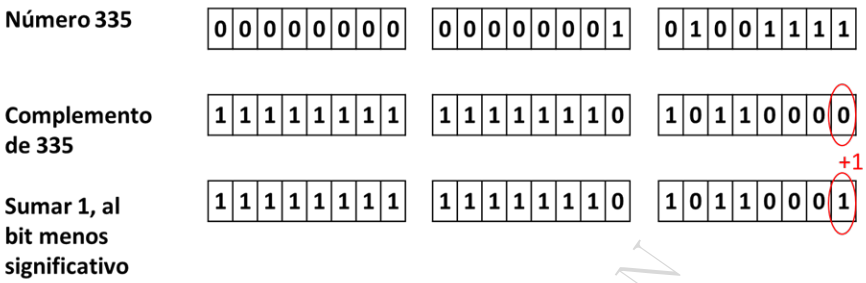

Figura 2.36: Representación del número -335 en 3 bytes usando el metodo de Complemento a 2

- 0 - 0 = 0
- 0 - 1 = 1, - 1 se acarrea menos uno para el siguiente bit más significativo
- 1 - 0 = 1
- 1 - 1 = 0

En la Figura 2.37 se muestra la transformación de los bytes 11111111 11111011 11110010 al número decimal -1038. Al restar 1 al bit menos significativo que tiene valor 0, el resultado es 1 y se acarrea -1 para el siguiente bit, luego se resta 1-1 cuyo resultado es 0.

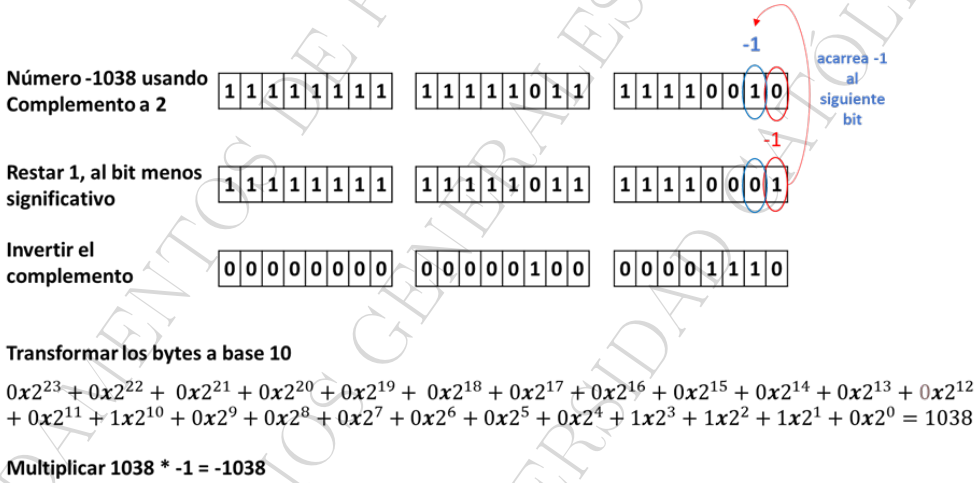

Figura 2.37: Representación del número -1038 en 3 bytes usando el metodo de Complemento a 2

Se pide que desarrolle un programa en Lenguaje C que reciba una lista de números negativos representados en 3 bytes (usando el método de complemento a 2), para cada número debe mostrar el número en base 10. Antes de realizar cualquier operación debe validar que los bytes ingresados sean correctos, esto significa que tengan máximo una longitud de 8 bits y que cada bit tenga un valor de 0 o 1. La lista de números se terminará ingresando 0 para los 3 bytes.

Para el desarrollo del programa debe desarrollar como mínimo 5 módulos adicionales al main. De estos módulos, por lo menos 1 debe simular el uso de parámetros por referencia y no debe usarse para leer datos. Además, debe desarrollar como mínimo 3 módulos que devuelvan un valor.

Se sugiere que cada byte sea una variable de tipo entero.

Use los siguientes casos de prueba para verificar si su solución está correcta.

```
Ingrese los tres bytes para calcular el número negativo en base 10:
10000000  000000000  00001001
El número en base 10 es: -8388599
Ingrese los tres bytes para calcular el número negativo en base 10:
11111111  11111110  000000000
El número en base 10 es: -512
Ingrese los tres bytes para calcular el número negativo en base 10:
11111111  11111011  11110101
El número en base 10 es: -1035
Ingrese los tres bytespara calcular el número negativo en base 10:0 0 0

Ingrese los tres bytes para calcular el número negativo en base 10:
111111111    1111    1
Los bytes ingresados son inválidos
Ingrese los tres bytes para calcular el número negativo en base 10:
111112    1121    33
Los bytes ingresados son inválidos
Ingrese los tres bytes para calcular el número negativo en base 10:
0 0 0
```

2.3.24. Operaciones con fracciones (adaptado del laboratorio 9 del ciclo 2020-1)

Se pide que elabore un programa en lenguaje C que permita operar una serie de fracciones. Para esto deberá:

- Leer la cantidad de fracciones a operar. La cantidad de fracciones que se deberá operar deberá ser un número mayor o igual a dos.
- Leer la operación a realizar. La operación deberá ser únicamente el valor de `+` o `-`.
- Validar que los datos antes ingresados sean correctos. En caso que algún dado leído no sea el esperado, se deberá emitir el mensaje `Los datos ingresados son inválidos`.
- Leer cada una de las fracciones y operarlas conforme a la operación ingresada. Todo esto deberá hacerlo en una única función de tipo `void`. Al final deberá de imprimir el valor de la fracción resultante. Asuma, para esta pregunta, que los denominadores serán siempre diferentes a cero.

Para operar las fracciones, deberá seguir necesariamente el siguiente procedimiento:

- Hallar el Mínimo Común Múltiplo (MCM) de los denominadores de los operandos. El MCM será el denominador de la fracción resultante de la operación.
- Para cada fracción se debe dividir el MCM entre el denominador original y el resultado lo multiplicamos por el numerador de esa misma fracción. Estos dos números sumados o restados, serán parte del numerador de la fracción resultante de la operación. Se sumarán o restarán dependiendo si la operación seleccionada es `+` o `-`.

Por ejemplo, si se quisiera operar las fracciones $\frac{2}{3} + \frac{4}{5}$, lo primero que se hace es hallar el MCM, en este caso es 15, este será el denominador del resultado, Luego este valor se divide entre el denominador de cada fracción y se multiplica por el numerador. En el caso de la fracción $\frac{2}{3}$, 15 entre 3 retorna 5 por lo que 5 se multiplica por 2 dando como resultado 10. En el caso de la fracción $\frac{4}{5}$, 15 entre 5 retorna 3 por lo que 3 se multiplica por 4 dando como resultado 12. Entonces el resultado será $\frac{10+12}{15} = \frac{22}{15}$.

Para esta pregunta asuma que ya existe una función que retorna el Máximo Común Divisor (MCD). Para que pueda probar su programa, le recomendamos que use la siguiente implementación, la cual ha sido ontenida usando la técnica de la Fuerza Bruta. Recuerde además que $MCD(a,b) \times MCM(a,b) = a \times b$.

Programa 2.1: Función para calcular el Máximo Común Divisor

```
int MCD(int a, int b){
        int divisor;
        if (a>=b)
                divisor=b;
        else
                divisor=a;
        while (divisor>=1){
                if (a % divisor==0 && b%divisor==0)
                        return divisor;
                divisor--;
        }
}
```

A continuación se presentan algunos ejemplos de ejecución del programa:

```
Ingrese cantidad de fracciones: 2
Ingrese operación: +

Ingrese numerador y denominador: 2 3
Ingrese numerador y denominador: 4 5

El resultado de la operación es: 22 / 15

Ingrese cantidad de fracciones: 3
Ingrese operación: -

Ingrese numerador y denominador: 1 2
Ingrese numerador y denominador: 4 7
Ingrese numerador y denominador: 8 9

El resultado de la operación es: -121 / 126

Ingrese cantidad de fracciones: 5
Ingrese operación: +

Ingrese numerador y denominador: 1 2
Ingrese numerador y denominador: 4 3
Ingrese numerador y denominador: 3 6
Ingrese numerador y denominador: 5 3
Ingrese numerador y denominador: 6 8

El resultado de la operación es: 114 / 24
```

2.3.25. Suma hexadecimal (adaptado del laboratorio 9 del ciclo 2020-1)

El sistema hexadecimal es el sistema de numeración posicional que tiene como base el 16. Su uso actual está muy vinculado a la informática ya que las operaciones del CPU suelen usar el byte como unidad básica de

memoria pues un byte, representa 2^8 valores $(0 - 255)$, y esto puede representarse como $2^8 = 2^4 * 2^4 = 16 * 16 = 16^2 = 1 * 16^2 + 0 * 16^1 + 0 * 16^0$, que equivale al número 100 en base 16 (100_{16}), es decir, tres dígitos hexadecimales en lugar de los ocho dígitos que hubiéramos usado en base 2. En principio, dado que el sistema usual de numeración es de base decimal y, por ello, solo se dispone de diez dígitos, se adoptó la convención de usar las seis primeras letras del alfabeto latino para suplir los dígitos que faltan. El conjunto de símbolos es el siguiente: $S = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E$. Se debe notar que $A = 10$, $B = 11$, $C = 12$, $D = 13$, $E = 14$ y $F = 15$.

Se pide que elabore un programa en lenguaje C que permita sumar números en hexadecimal. Podrán existir una cantidad variable de números hexadecimal por cada línea del flujo de entrada y además, podrá existir una cantidad variable de líneas del flujo de entrada. Para esto deberá:

- Leer números en hexadecimal considerando que pueden existir varios números por línea del flujo de entrada. La lectura finalizará cuando se lea un cambio de línea. Asuma que los números estarán separados por un espacio en blanco y que siempre se ingresarán dígitos en hexadecimal correctos. La cantidad de dígitos del número hexadecimal es variable y deberá considerar esto en su propuesta de solución.
- Leer varias líneas del flujo de entrada. La lectura finalizará cuando se lea el caracter `*`.
- Sumar todos los números leídos e imprimir dicha suma también en hexadecimal.
- Deberá considerar que los dígitos hexadecimales podrán estar tanto en mayúsculas como en minúsculas.
- Deberá utilizar por lo menos 5 funciones.

Recordar que:

Recuerde que el carácter 'A' es representado por el código ASCII 65, y tiene valor numérico 10. Por otro lado, el carácter '0' es representado por el código ASCII 48, y tiene valor numérico 0. Los códigos ASCII son valores sucesivos por lo que '1' tiene código ASCII 49 y 'B' tiene código ASCII 66. Recomendación: si se tiene el carácter 'A' y se desea obtener el valor numérico 10, bastaría con restarle 55 al código ASCII correspondiente al carácter 'A'.

A continuación se presentan algunos ejemplos de ejecución del programa:

```
Ingrese lista de números en hexadecimal: 12C 190
Ingrese lista de números en hexadecimal: 1F4 258 2bc
Ingrese lista de números en hexadecimal: *

La suma de números es: 9C4

Ingrese lista de números en hexadecimal: F 10 30
Ingrese lista de números en hexadecimal: ab AB 56*

La suma de números es: 1FB

Ingrese lista de números en hexadecimal: F*

La suma de números es: F
```

2.3.26. Polígonos (adaptado del laboratorio 9 del ciclo 2020-1)

A lo largo de los años, diversas culturas han resuelto problemas relacionados al cálculo de áreas contenidas tanto por las curvas como por los polígonos. Para ello, utilizaban diversas formas, entre ellas, técnicas de trazo y fórmulas que se encuentran en documentos históricos como papiros y tablas de arcilla. Posteriormente, se identificaron fórmulas particulares dependiendo de la figura.

Dentro de este contexto, actualmente existen fórmulas específicas dependiendo de la figura a evaluar. Utilizando algunas de estas fórmulas que se describirán posteriormente, se pide que desarrolle un programa en lenguaje C que muestre un menú con los datos de la figura y de acuerdo a la opción ingresada (polígono regular de más de 4 lados, figura circular o cuadrilátero), realice los cálculos correspondientes. Para terminar la ejecución del menú se ingresará uel carácter *. A continuación, el detalle del menú:

Opción	Figura
R	Polígono regular de más de 4 lados
C	Figura Circular
X	Cuadrilátero
*	Salir del menú

Por cada opción ingresada, se solicitará la cantidad de figuras para calcular y mostrar el valor en particular (sea perímetro, área o ambos), además se calculará el promedio de dichas áreas por tipo de figura.

- Cuando se elige la opción R, deberá calcularse e imprimirse el perímetro y área del polígono regular de más de 4 lados. El polígono podrá ser pentágono, hexágono u octógono y en base a la opción que seleccione el usuario podrá identificar la cantidad de lados:

Opción	Tipo de polígono	Cantidad de lados
P	Pentágono	5
H	Hexágono	6
O	Octógono	8

  Para realizar los cálculos, se solicitará el identificador del tipo de polígono (P, H, O), el valor de la longitud del lado, y el valor del apotema. La fórmula a aplicar para el perímetro y el área es:

  P =n×l

  A =1/2×P*apt

  Donde: P : perímetro A: área n: cantidad de lados apt: apotema l: longitud del lado

- Cuando se elige la opción C, deberá calcularse e imprimirse el área de una figura circular. Como opciones se tienen: sector circular, corona circular y trapecio circular. En base a la opción que seleccione el usuario podrá identificar el tipo de figura y se seleccionarán ciertos datos de entrada para aplicar la fórmula correspondiente:

Opción	Tipo de polígono	Datos a ingresar	Fórmula del área
1	Sector circular	radio, ángulo	$\pi radio^2 ang/360$
2	Corona circular	radio mayor, radio menor	$\pi(radioMayor^2 - radioMenor^2)$
3	Trapecio circular	radio mayor, radio menor,ángulo	$\pi(radMayor^2 - radMenor^2)ang/360$

- Cuando se elige la opción X, deberá calcularse e imprimirse el área del cuadrilátero que podrá ser rombo, romboide o trapecio y en base a la opción que seleccione el usuario podrá identificar el tipo de figura y se seleccionarán ciertos datos de entrada para aplicar la fórmula correspondiente:

Opción	Tipo de polígono	Datos a ingresar	Fórmula del área
R	Rombo	diagonal mayor, diagonal menor	$diagMayordiagMenor/2$
O	Romboide	base, altura	$basealtura$
T	Trapecio	diagonal mayor, diagonal menor, altura	$baseMayorbaseMenoraltura/2$

Su programa deberá utilizar el paradigma de programación modular con al menos 4 módulos adicionales al principal. Un módulo debe simular el uso de parámetros por referencia, es decir debe modificar los parámetros y no puede usarse para leer datos. De los otros módulos, por lo menos dos deben devolver un valor.

A continuación se presentan algunos ejemplos de ejecución del programa:

```
Ingresar la opción del menú·: R
──────── Polígonos regulares ────────
Ingrese la cantidad de polígonos regulares a evaluar: 3
Ingrese la letra que identifica el polígono regular #1: (P)Pentágono, (H)Hexágono,
(O) Octógono: P
Ingrese el valor de la longitud del lado del polígono regular:5
Ingrese la longitud de la apotema del polígono regular:7

..................Polígono regular #1 de 5 lados ..................
El perímetro es: 25.00
El área es: 87.50

Ingrese la letra que identifica el polígono regular #2: (P)Pentágono, (H)Hexágono,
(O) Octógono: H
Ingrese el valor de la longitud del lado del polígono regular:5.66
Ingrese la longitud de la apotema del polígono regular:5

.................. Polígono regular #2 de 6 lados ..................
El perímetro es: 33.96
El área es: 84.90

Ingrese la letra que identifica el polígono regular #3: (P)Pentágono, (H)Hexágono,
(O) Octógono: O
Ingrese el valor de la longitud del lado del polígono regular:7.5
Ingrese la longitud de la apotema del polígono regular:6

.................. Polígono regular #3 de 8 lados ..................
El perímetro es: 60.00
El área es: 180.00

El promedio de los 3 polígono regulares es: 117.47

Ingrese la opción del menú·: X
──────── Cuadriláteros ────────
Ingrese la cantidad de cuadriláteros a evaluar: 1
Ingrese la opcin de la figura cuadrilátera a evaluar: (R)Rombo, (O)Romboide, (T)Trapecio:
O
Ingrese la base y la altura:9.5
6
Cuadrilátero #1
.............................
El área es: 57.00

El promedio de áreas de los 1 cuadriláteros es: 57.00

Ingrese la opción del menú·: C
──────── Figuras circulares ────────
Ingrese la cantidad de figuras circulares a evaluar: 2
Ingrese la opción de la figura circular a evaluar: (1) Sector circular, (2) Corona circular,
(3) Trapecio circular: 1
Ingrese el radio y el ángulo:10 360
Sector circular #1
.............................
El área es: 314.16

Ingrese la opción de la figura circular a evaluar: (1) Sector circular, (2) Corona circular,
```

```
(3) Trapecio circular: 3
Ingrese el radio del círculo más grande y el radio del círculo más pequeño:7.5 3
Ingrese el ángulo:180
Sector circular #2
.............................
El área es: 74.22

El promedio de áreas de las 2 figuras circulares es: 194.19



Ingresar la opción del menú: *
```

2.3.27. Digito verificador del DNI (adaptado del laboratorio 9 del ciclo 2020-1)

Nuestro documento nacional de identidad (DNI) además de tener un número de 8 dígitos, cuenta con un dígito verificador.

Para calcular el dígito verificador se deben seguir los siguientes pasos:

- Tomaremos como ejemplo el DNI 17801146.
- Separamos cada uno de los dígitos 1, 7, 8, 0, 1, 1, 4, 6.
- Multiplicamos cada dígito por esta serie en el mismo orden 3, 2, 7, 6, 5, 4, 3, 2 de esta forma: 1 x 3, 7 x 2, 8 x 7, 0 x 6, 1 x 5, 1 x 4, 4 x 3, 6 x 2
- Sumamos todos los productos dándonos el resultado de 106
- Dividimos el resultado anterior entre 11 y tomamos el residuo: 106/11 = 9 sobrándonos 7 (9 x 11 = 99 para 106 nos faltaría 7)
- Al valor 11 (por defecto) le restamos el resultado anterior 7, lo que nos daría 4. Existe una excepción, si este resultado sería 11, es decir 11 – 0 = 11 (0 es el resultado del punto anterior, es decir se trata de una división exacta que no tiene residuo) entonces este resultado se modificar por 0 y no 11.
- Al resultado anterior le sumamos 1, es decir 4 + 1 = 5 lo que significa que vamos a buscar la 5ta posición en la serie NUMERICA (por defecto) 6, 7, 8, 9, 0, 1, 1, 2, 3, 4, 5 ó la 5ta posición en la serie ALFABÉTICA (por defecto) K, A, B, C, D, E, F, G, H, I, J. Entonces el dígito verificador puede ser un número, siendo el 0 o puede ser una letra, la D. Para los DNI's emitidos hasta el 14 de agosto del 2007 pertenecientes a personas de mayores de 60 años se consideraba como dígito verificador la letra y no el número calculado.

Se pide que desarrolle un programa en Lenguaje C que reciba una lista de: DNI, dígito verificardor y fecha de emisión; siga el algoritmo descrito; calcule el dígito verificador y lo compare con el ingresado indicando si es correcto o no.

Para leer el DNI, debe ingresar cada dígito carácter por carácter e ir formando el DNI en valor numérico. La lectura de cada dígito (incluido el dígito verificador) debe realizarse en una iterativa, debe verificar que cada dígito del DNI (incluido el dígito verificador) sea válido, si se ingresa * se termina la lista de ingreso de datos y el programa. Solo puede definir una variable para leer cada dígito del DNI ((incluido el dígito verificador). Si no realiza lo que se indica no se corregirá lo desarrollado y se asignará 0 en el laboratorio.

Debe asumir que la fecha de emisión se ingresa correctamente y dividida en 3 números, el primero representa al día, el segundo al mes y el tercero al año.

Luego de leer los datos de entrada debe mostrar el DNI ingresado y el dígito verificador, indicando si es letra o número.

En el caso que el DNI se haya emitido el 14 de agosto del 2007 o antes, se debe mostrar el cálculo del dígito verificador tanto para la letra como para el número.

Debe utilizar el paradigma de programación modular definiendo como mínimo 5 módulos, de los cuales uno de ellos modifique más de un parámetro y dos módulos que calculen y devuelvan un solo valor.

Use los siguientes casos de prueba para verificar si su solución está correcta.

```
Ingrese el DNI incluyendo el dígito verificador:
0
9
6
7
1
2
6
7
2
Ingrese la fecha de emisión del DNI:12 5 2005
DNI ingresado: 09671267
Dígito verificador ingresado: 2, número
El digito verificador calculado puede ser un número o una letra por la fecha de emisión del DNI
Los dígitos verificadores calculados: 2 G
El dígito verificador calculado coincide con el ingresado

Ingrese el DNI incluyendo el dígito verificador:
0
9
6
7
1
2
6
7
2
Ingrese la fecha de emisión del DNI:12 12 2020
DNI ingresado: 09671267
Dígito verificador ingresado: 2, número
El digito verificador calculado será un número
El dígito verificador calculado: 2
El dígito verificador calculado coincide con el ingresado

Ingrese el DNI incluyendo el dígito verificador:
*
Fin de evaluaciones de DNI


Ingrese el DNI incluyendo el dígito verificador:
0
7
4
5
2
6
7
8
A
Ingrese la fecha de emisión del DNI:12 3 2020
DNI ingresado: 07452678
Dígito verificador ingresado: A, letra
El digito verificador será un número
El dígito verificador calculado: 6
El dígito verificador calculado no coincide con el ingresado

Ingrese el DNI incluyendo el dígito verificador:
0
A
Error al ingresar los dígitos del DNI
Ingrese el DNI incluyendo el dígito verificador:
1
0
1
3
5
6
7
6
Z
El dígito verificador ingresado no es correcto

Ingrese el DNI incluyendo el dígito verificador:
1
```

```
0
1
3
5
6
7
6
A
Ingrese la fecha de emisión del DNI:12 3 2010
DNI ingresado: 10135676
Dígito verificador ingresado: A, letra
El digito verificador calculado será un número
El dígito verificador calculado: 6
El dígito verificador calculado no coincide con el ingresado

Ingrese el DNI incluyendo el dígito verificador:
2
3
4
5
*
Fin de evaluaciones de DNI
```