Pontificia Universidad Católica del Perú

Estudios Generales Ciencias

1INF01 - Fundamentos de Programación

Guía de laboratorio #4

Programación Modular

25 de septiembre de 2021

Índice general

Historial de Revisiones

Revisión	Fecha	Autor(es)	Descripción
1.0	30.08.2019	C.Aguilera	Versión inicial.
1.0	30.08.2019	D.Allasi	Versión inicial.
2.0	10.05.2020	J.Baldeón	Se actualizó la parte teórica y los ejemplos.
2.1	10.05.2020	C.Aguilera	Se actualizaron los ejercicios propuestos.
2.2	11.05.2020	I.Brossard	Se actualizaron los ejercicios propuestos.
2.3	12.05.2020	J.Baldeón	Se realizó la integración y revisión de los ejercicios a la guía.
2.4	29.09.2020	J.Baldeón	Se incrementó la cantidad de problemas propuestos y se clasificaron en las categorías de nivel básico, nivel intermedio y nivel avanzado.
2.5	14.04.2021	J.Zárate	Se incrementó la cantidad de problemas propuestos.
2.6	24.09.2021	J.Zárate	Se incrementó la cantidad de problemas propuestos de nivel avanzado.

Siglas

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

Capítulo 1

Guía de Laboratorio #4

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 "Programación Modular".

Se busca que el alumno resuelva paso a paso las indicaciones dadas en esta guía contribuyendo de esta manera a los objetivos de aprendizaje del curso, en particular con la comprensión de la programación modular. Al finalizar el desarrollo de esta guía, complementado con la realización del correspondiente laboratorio, se espera que el alumno:

- Construya programas apoyados en la programación modular que utilice el paso de parámetros por valor y simule el paso de parámetros por referencia.

1.2. Materiales y métodos

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

1.3. Programación modular

Hasta el momento, la construcción de cada programa se ha realizado en una única función (*main*) que contiene todas las instrucciones. Y cada vez que se requiere reutilizar bloques de código se suele copiar, pegar y editar dichos bloques. A pesar de que esta técnica reduce las posibilidades de cometer errores, hay una mejor manera [13].

Cuando el código de un programa se hace muy largo, se vuelve necesario optar por una estrategia que divida el problema en subproblemas que luego sean abordados puntual e independientemente, enfocándose en resolver tareas específicas, que contribuyan al cumplimiento del objetivo general del programa principal.

La programación modular es un enfoque que consiste en dividir el desarrollo de un programa en elementos más pequeños como subprogramas para así lograr resolver problemas más simples que sean fáciles de afrontar [1]. Cabe señalar que estos módulos o subprogramas se estructuran e integran jerárquicamente, ver figura 1.1,

[1] http://pseint.sourceforge.net/
[2] http://www.bloodshed.net/dev/devcpp.html

como si fuera el organigrama de una empresa en el que se presenta la relación jerárquica de funciones en sus miembros [3].

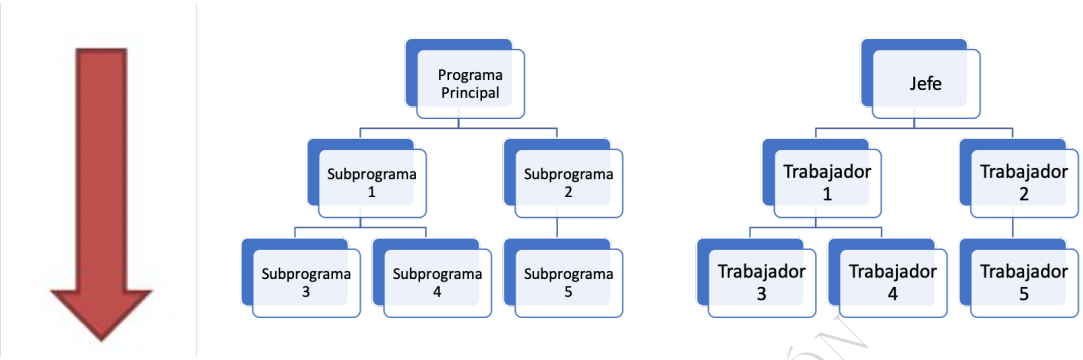


Figura 1.1: Diseño Top/Down: Diagrama de estructura o de módulos - Organigrama de una empresa.

Un módulo es una abstracción que proporciona encapsulación lógica alrededor de una declaración, modelando una o más abstracciones de datos y los subprogramas correspondientes que realizan ciertas operaciones útiles [1]. Según Louden [9], un módulo es una unidad de programa con una interfaz pública y una implementación privada.

Cuando se proponen subprogramas que resuelvan abstracciones del programa, se habla de un diseño modular, que permite compartir funcionalidades con otros desarrolladores, permite la compilación separada y brinda la capacidad de expansión cuando se agregan nuevos módulos al sistema [2]. Adicionlamente, Kernighan [7] destaca las siguientes ventajas en el uso de subprogramas:

- Divide tareas grandes en varias pequeñas.
- Permiten nuevos desarrollos sobre lo ya hecho, en lugar de comenzar desde cero.
- Ocultan los detalles de operación de las partes del programa que no necesitan saber acerca de ello.
- Permiten la legibilidad en la totalidad del código.
- Facilitan la realización de cambios en el código.

1.4. Tipos de subprogramas

Los subprogramas son los bloques de construcción fundamentales de programas [14] y tienen las siguientes características:

- Cada subprograma tiene un único punto de entrada.
- La unidad del programa que invoca se suspende durante la ejecución del subprograma llamado, lo que implica que solo hay un subprograma en ejecución en un momento dado.
- El control siempre vuelve al que lo invoca cuando finaliza la ejecución del subprograma.

Además, los subprogramas permiten al desarrollador considerar una secuencia de acciones como una acción única que se puede invocar desde muchos otros puntos de un programa [9].

De acuerdo con Sebesta [14], hay dos tipos fundamentales de subprogramas:

- Funciones, que retornan un valor.
- Procedimientos, que no retornan ningún valor.

Para modularizar programas en el lenguaje C se utilizan las funciones [3], debido que no incluye a los procedimientos como una forma separada de subprogramas. Es por ello que en el lenguaje C las funciones pueden ser definidas sin valores de retorno para que puedan ser usadas como procedimientos [14].

1.5. Funciones

Encontraremos algo común en los programas bien escritos: la función. Hemos venido utilizando en casi todos nuestros programas las rutinas *printf* y *scanf*, que son ejemplos de funciones. De hecho, cada uno de los programas también usó una función llamada *main*. Y es que una función proporciona el mecanismo para producir programas que sean fáciles de escribir, leer, comprender, depurar, modificar y mantener [8].

En el lenguaje C, una función es un bloque de código en el que ocurre la actividad de un subprograma [12]. Cada programa en C puede considerarse como una colección de estas funciones. Usar una función es algo así como contratar a una persona para que haga un trabajo específico para usted. A veces la interacción con esta persona es muy simple, y a veces es compleja [6].

Supongamos que tenemos una tarea que siempre se realiza exactamente de la misma manera, como por ejemplo el lavado de nuestro vehículo en un taller ya conocido. Nos dirigimos al taller y diríamos algo así: "Lávelo como siempre por favor". No necesitaríamos dar más instrucciones, y estaríamos despreocupados, porque el personal conoce su trabajo y simplemente recibiríamos el vehículo limpio. Una función en el lenguaje C opera similar al taller de lavado.

La representación de una función en el lenguaje C se puede apreciar en el programa 1.1.

Programa 1.1: Forma general de una función en el lenguaje C

```
1 tipo_retorno nombre_de_función(lista de parámetros, si los hay)
2 {
3     declaraciones;
4     instrucciones;
5 }
```

El *tipo_retorno* especifica el tipo de dato que devuelve la función. La *lista de parámetros* es una lista separada por comas de nombres de variables y sus tipos asociados. Los parámetros reciben los valores de los argumentos cuando se invoca a la función. Una función puede estar sin parámetros, en cuyo caso la lista de parámetros está vacía. Además, una lista de parámetros vacía se puede especificar explícitamente como tal colocando la palabra reservada *void* dentro de los paréntesis [3].

La implementación de funciones en el lenguaje C debe considerar lo siguiente:

1.5.1. Declaración de funciones

Consiste en declarar los prototipos (hacer un catálogo) de las funciones sin programarlas y que serán utilizadas en el programa. Se hace previo a la definición de la función *main*, y su sintaxis se aprecia en el programa 1.2.

Programa 1.2: Forma general de la declaración de una función en el lenguaje C

```
1 tipo_de_retorno nombre_de_la_función(lista_de_parámetros);
```

Donde:

- *tipo_de_retorno*: es el tipo de dato del valor devuelto por la función. Si la función no devuelve ningún valor, se usa la palabra reservada *void*.
- *nombre_de_la_función*: es el nombre o identificador de la función.
- *lista_de_parámetros*: es la lista, separada por comas, de los parámetros que la función necesita para operar. Si la función no necesita parámetros, se puede colocar la palabra reservada *void* o dejar vacío.

En el prototipo de una función no se especifican las sentencias que forman parte de la misma, sino sus características. Veamos el ejemplo de declaración de la función *calcular_factorial* en el programa 1.3:

Programa 1.3: Declaración de la función calcular_factorial en el lenguaje C

```
1 int calcular_factorial(int numero);
```

El motivo por el que se necesita la declaración de la función, es porque luego el compilador compara la invocación a la función *calcular_factorial* con el prototipo de la función para asegurar que:

- El número de argumentos sea el correcto.
- Los argumentos coinciden con los tipos definidos en el prototipo.
- Los tipos de argumentos están en el orden correcto.
- El tipo de retorno es consistente con el contexto en el cual la función es invocada.

1.5.2. Definición de funciones

Corresponde a la implementación de una función luego de su declaración. En el programa 1.4 se aprecia su sintaxis.

Programa 1.4: Forma general de la definición de una función en el lenguaje C

```
tipo_de_retorno nombre_de_la_función (lista_de_parámetros) {
    conjunto de instrucciones de la función;
    return <expresión>; /*Opcional si el tipo es void*/
}
```

Veamos el ejemplo de definición de la función *calcular_factorial* en el programa 1.5:

Programa 1.5: Definición de la función calcular_factorial en el lenguaje C

```
int calcular_factorial(int numero) {
        int i=1, fact=1;
        while (i<=numero){
                fact = fact * i;
                i++;
        }
        return fact;
}
```

1.5.3. Accesos a una función

Para que una función realice la tarea para la cual fue creada, se le debe de invocar desde un programa o subprograma. En el programa 1.6 se aprecia el ejemplo de invocación a la función *calcular_factorial* desde la función *main()*.

Programa 1.6: Ejemplo de invocación a la función calcular_factorial en el lenguaje C

```
#include <stdio.h>

int calcular_factorial(int numero) {
        int i=1, fact=1;
        while (i<=numero){
                fact = fact * i;
                i++;
        }
        return fact;
}

int main () {
        int num,fact;
        printf("Ingrese un numero: ");
        scanf(" %d",&num);
        fact = calcular_factorial(num);
        printf("El factorial de %d es %d",num,fact);
        return 0;
}
```

Importante

Los parámetros que se declaran en la definición del subprograma se denominan parámetros formales y se comportan como cualquier otra variable local dentro de la función [12].
Los parámetros que se pasan al subprograma en la invocación se denominan parámetros actuales o reales [4].

Ahora veamos otro ejemplo, analicemos el programa 1.7 en el lenguaje C desde dos perspectivas, desde el cómo se invoca y el cómo se define :

Programa 1.7: Ejemplo de cómo se invoca y define una función en el lenguaje C

```c
#include <stdio.h>
/*Prototipo o declaración de la función: */
void message();

int main() {
    message();
    printf("¡Divide y vencerás!\n");
    return 0;
}
void message(){
    printf("Si el problema es complejo, recuerda...\n");
}
```

La salida de este programa sería:

```
Si el problema es complejo, recuerda...
¡Divide y vencerás!
```

Concentrémonos en el desarrollo de la función *main()*, que a su vez invoca a la función *message()*. ¿Qué significa cuando decimos que *main()* "invoca" a la función *message()*? Queremos decir que el control pasa a la función *message()*. La actividad de *main()* se suspende temporalmente; se queda dormido mientras la función *message()* se despierta y se pone a trabajar. Cuando la función *message()* se queda sin instrucciones por ejecutar, el control vuelve a *main()*, que se despierta y comienza a ejecutar su código en el punto exacto donde lo dejó. Por lo tanto, *main()* se convierte en la función "invocante", mientras que *message()* se convierte en la función "invocada".

Si se entendió el concepto de "invocar" una función, estamos listos para invocar a más de una función. Analicemos ahora el programa 1.8 en el lenguaje C:

Programa 1.8: Otro ejemplo de cómo se invoca y define una función en el lenguaje C

```c
#include <stdio.h>
void message1();
void message2();
void message3();

int main() {
    printf("Se ejecuta la función main.\n");
    message1();
    message2();
    message3();
    return 0;
}
void message1(){
    printf("FunPro es divertido.\n");
}
void message2(){
    printf("La clave está en practicar.\n");
}
void message3(){
    printf("¡Y así podré triunfar!\n");
}
```

La salida de este programa sería:

```
Se ejecuta la función main.
FunPro es divertido.
La clave está en practicar.
¡Y así podré triunfar!
```

Para este último programa se pueden obtener las siguientes conclusiones [6]:

- Cualquier programa en el lenguaje C contiene al menos una función.
- Si un programa contiene solo una función, debe ser *main()*.
- Si un programa en el lenguaje C contiene más de una función, entonces una (y solo una) de estas funciones debe ser *main()*, porque la ejecución del programa siempre comienza con *main()*.
- No hay límite en el número de funciones que pueden estar presentes en un programa en el lenguaje C.
- Cada función en un programa es invocada en la secuencia especificada por la función que hizo el llamado a la función (típicamente la invocación se hace desde la función *main()*, pero también es posible invocar funciones dentro de otras funciones).
- Después de que cada función haya hecho lo suyo, el control vuelve a la función que hizo la invocación, que en este caso es la función *main()*. Cuando *main()* se quede sin instrucciones o invocaciones a función, el programa finalizará.

Nota

La biblioteca estándar de C proporciona una variada colección de funciones para realizar cálculos matemáticos, manipulación de cadenas y caracteres, entrada / salida y muchas otras operaciones útiles. A este conjunto de funciones se les conoce también como funciones de biblioteca. Esto facilita el trabajo del desarrollador, porque proporcionan muchas de las capacidades que necesita. Mientras que las funciones definidas por el programador se les conoce como funciones de usuario [3].

1.5.4. Argumentos de las funciones, llamadas por valor y por referencia

En el lenguaje C siempre los argumentos se pasan **por valor** a una función [3]. Este método consiste en copiar el valor de un argumento en el parámetro formal de la función. Entonces, los cambios hechos al parámetro no tienen efecto sobre el argumento [12].

En el programa 1.6 se puede apreciar el paso de parámetro por valor cuando se invoca a la función *Calcular-Factorial*, porque lo que interesa para la ejecución de la función es contar con solo el valor del argumento.

La segunda forma de pasar argumentos a un subprograma es mediante la invocación **por referencia**. En este método, la dirección de un argumento se copia en el parámetro. Dentro del subprograma, la dirección se utiliza para acceder al argumento real utilizado en la llamada. Esto significa que los cambios realizados en el parámetro afectan el argumento [12].

En el programa 1.9, se puede apreciar la definición de la función *intercambiar* que, mediante la técnica de paso de parámetros por referencia, se logra modificar los valores de cada variable. Pero lo que en realidad se definen como parámetros formales son variables de tipo dirección de memoria o *punteros*, y será así como se logrará acceder a los valores en memoria de las variables.

Programa 1.9: Intercambiar los valores de dos variables

```
1 #include <stdio.h>
2
3 void intercambiar(int *p, int *q);
```

```
4
5  int main() {
6          int a,b;
7          a = 666;
8          b = 999;
9          printf("a y b antes del intercambio: %d %d\n", a, b);
10         intercambiar(&a, &b); /* Se pasan las direcciones de a y b */
11         printf("a y b luego del intercambio: %d %d\n", a, b);
12 }
13
14 void intercambiar(int *p, int *q) {
15         int temp;
16         temp = *p; /* Se guarda en temp el valor apuntado por la dirección p */
17         *p = *q; /* Se asigna el valor apuntado por la dir. q en la variable apuntado por la dir. p */
18         *q = temp; /* Se asigna el valor de temp en la variable apuntado por la dirección q */
19         return;
20 }
```

La salida de este programa sería:

```
a y b antes del intercambio: 666 999
a y b luego del intercambio: 999 666
```

En este ejemplo podemos ver, en la línea 3, el uso del operador * para la definición de argumentos por referencia y, en la línea 10, el uso del operador & para realizar la llamada a la función con parámetros por referencia.

Al implementar programas en el lenguaje C, se pueden combinar ambas llamadas (valor y referencia) sin ningún problema, siendo ésta una práctica muy usual en programación.

Operadores de puntero:

Es posible lograr el paso por referencia utilizando el operador de dirección (&) y el operador de indirección (*).
El &, u operador dirección, es un operador unario que retorna la dirección de su operando. Por ejemplo, si asumimos las definiciones

```
int a = 666;
int *aPtr;
```

la instrucción

```
aPtr = &a;
```

asigna la dirección de la variable *a* a la variable puntero *aPtr*. La variable *aPtr* se dice que "apunta a" *a*. La figura 1.2 muestra una representación esquemática de la memoria después de que la última intrucción de asignación se ha ejecutado.
La representación del puntero en memoria se muestra en la figura 1.3, asumiendo que la variable entera es almacenada en la ubicación 700000, y la variable puntero aPtr es almacenada en la ubicación 600000.
Tenga en cuenta que el operando del operador dirección debe ser una variable, el operador dirección no puede ser aplicado a constantes o expresiones.
El operador unario *, conocido también como el operador indirección u operador desreferencia, retorna el valor del objeto al cual su operando (puntero) apunta. Por ejemplo, la sentencia

```
printf("%d", *aPtr);
```

imprime el valor de la variable *a* (666). Usando * en esta manera se denomina desreferenciando un puntero.

1.6. Procedimientos

De acuerdo con Sebesta [14], los procedimientos son un tipo subprogramas que ejecutan un proceso en específico y no tiene un valor de retorno. Mediante los procedimientos se pueden ejecutar un conjunto de instrucciones que

Figura 1.2: Representación gráfica de un puntero apuntando a una variable entera en memoria.

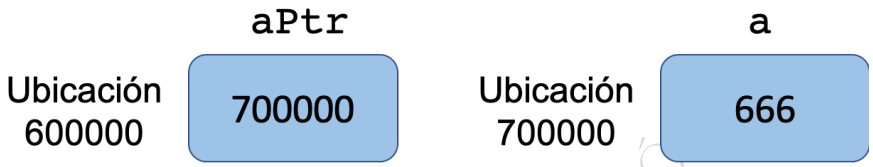


Figura 1.3: Representación de a y aPtr en memoria.

no devuelven ningún valor. En el lenguaje C los procedimientos se consideran como funciones que devuelven un resultado de tipo *void*.

A continuación, en el programa 1.10, que calcula la suma de dos números, se presenta un ejemplo del procedimiento *mostrar_ datos* que imprime en pantalla los datos ingresados y el resultado.

Programa 1.10: Cálculo de la suma de dos números que utiliza un procedimiento para mostrar el resultado

```
1  #include <stdio.h>
2
3  void mostrar_datos(int a, int b,int c){
4          printf("La suma de %d y %d es %d",a,b,c);
5  }
6
7  int main() {
8          int a,b,c;
9          printf("Ingrese dos números: ");
10         scanf("%d %d",&a,&b);
11         c = a+b;
12         mostrar_datos(a,b,c);
13 }
```

La salida que genera este programa es la siguiente:

```
Ingrese dos números: 123 543
La suma de 123 y 543 es 666
```

1.7. Cálculo del número *e*

El número e es un número muy usado en las ciencias[3], 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:

[3]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.

$$1+\frac{1}{1!}+\frac{1}{2!}+\frac{1}{3!}+\frac{1}{4!}+\frac{1}{5!}+\frac{1}{6!}+\cdots$$

A continuación, se muestra el programa 1.11 en el lenguaje C que permite el cálculo aproximado del número e usando la fórmula antes descrita aproximando el número a 6 decimales:

Programa 1.11: Cálculo del número e

```c
#include <stdio.h>

int main() {
    int i = 0;
    double suma = 0;
    double factorial = 1;
    double termino=1;
    while (termino>=0.0000001) {
        if (i == 0)
            suma = suma + 1;
        else {
            factorial = factorial*i;
            termino = 1 / factorial;
            suma = suma + termino;
        }
        i++;
    }
    printf("El valor del número e es = %1.6lf", suma);
    return 0;
}
```

La salida que genera este programa es la siguiente:

```
El valor del número e es = 2.718282
```

Recordar que:

El desarrollo en el lenguaje C del algoritmo presentado en el programa 1.11 se encuentra ampliamente descrito en las guías anteriores.

Si surge la necesidad de usar el valor de e que acabamos de generar como parte de otros programas, o incluso, se desea obtener el valor de e con diferentes precisiones, entonces, se puede definir un bloque de código que facilite su obtención y dicho bloque sería parte de una función que permita obtener su valor de acuerdo con el número de iteraciones que reciba como parámetro en su invocación.

A continuación, desarrollaremos una función en el lenguaje C que permita obtener el valor de e después de una determinada cantidad de iteraciones, dicha cantidad será parámetro de la función. Tengamos en cuenta que mientras mayor sea la cantidad de iteraciones, más exacto será el valor que obtengamos. Sin embargo, más iteraciones también significan más tiempo de procesamiento.

1.7.1. Cálculo del número e en una función

Podemos modificar el programa 1.11 de manera que el cálculo del número e se realice dentro de una función *numero_e* que retorne el valor aproximado de e para su posterior uso. La modificación se presenta en el programa 1.12.

Programa 1.12: Cálculo del número e dentro de una función

```c
#include<stdio.h>
/* Prototipo de la función numero_e */
double numero_e(int iteraciones);

int main(){
```

```
6         double valor_numero_e;
7         int num_iteraciones;
8         printf("Ingrese la cantidad de iteraciones: ");
9         scanf("%d",&num_iteraciones);
10        /* Se invoca a la función numero_e pasando, como parámetro real y por valor, el valor de la variable
              num_iteraciones */
11        valor_numero_e = numero_e(num_iteraciones);
12        printf("El valor del numero e es = %.6lf\n", valor_numero_e);
13        return 0;
14  }
15
16  /* Definición de la función numero_e con el parámetro formal iteraciones de tipo entero */
17  double numero_e(int iteraciones){
18        double suma, factorial, termino;
19        int i;
20        i=0;
21        suma=0;
22        factorial=1;
23        termino=1;
24        while (i<iteraciones){
25              if (i==0)
26                    suma = suma+1;
27              else{
28                    factorial = factorial*i;
29                    termino=(double)1/factorial;
30                    suma=suma+termino;
31              }
32              i++;
33        }
34        return suma;
35  }
```

Vemos que la función *main* cambió considerablemente, en la línea 6 vemos la declaración de la variable *valor_numero_e* y en la linea 11 se asigna el valor devuelto por la función *numero_e* a la variable *valor_numero_e* . A partir de este momento, podemos obtener el valor del número *e* con solo llamar a la función que acabamos de crear, pasando como parámetro la cantidad de iteraciones que queremos realizar.

La salida que genera este programa es la siguiente:

```
Ingrese la cantidad de iteraciones: 100
El valor del número e es = 2.718282
```

Si revisamos la estructura de nuestro programa, podemos apreciar que:

- El prototipo de la función *numero_e* declarada en la línea 3 tiene tipo de retorno *double*.
- La función *numero_e* recibe solo un parámetro de tipo entero, sin embargo, debemos recordar que otras funciones podrían recibir una cantidad diferente de parámetros.
- La invocación a la función *numero_e* en la línea 11 coincide con la declaración del prototipo en el número y tipo de parámetros.
- Como el retorno de la invocación a la función *numero_e* en la línea 11 es un valor de tipo *double*, éste es asignado a la variable *valor_numero_e* del mismo tipo *double*.
- La definición de la función *numero_e* en la línea 17 se hace luego de la definición de la función *main*. Sin embargo, también se podría haber definido antes de la función *main*.
- El parámetro usado en la definición de la función *numero_e* en la línea 17 indica que el número *e* se calculará usando una cantidad de iteraciones en particular.

Sobre la solución realizada y para ejemplificar aún más el paso de parámetros por valor, vamos a modificar el programa 1.12 para que la tarea de mostrar el resultado sea delegada a una función encargada para ello.

A continuación, crearemos la función *imprime_resultado* que muestre en la pantalla del usuario el mensaje final con el resultado. En el programa 1.13 se puede apreciar la versión del programa 1.12 que define y utiliza la función *imprime_resultado* definida por el usuario para dicha tarea.

Programa 1.13: Cálculo del número *e* dentro de una función con la impresión del resultado en otra función

```
1  #include<stdio.h>
2  /* Prototipos de las funciones numero_e e imprime_resultado */
3  double numero_e(int iteraciones);
4  void imprime_resultado(double numero_e);
5
6  int main(){
7          double valor_numero_e;
8          int num_iteraciones;
9          printf("Ingrese la cantidad de iteraciones: ");
10         scanf("%d",&num_iteraciones);
11         /* Se invoca a la función numero_e pasando, como parámetro real y por valor, el valor de la variable
                num_iteraciones */
12         valor_numero_e = numero_e(num_iteraciones);
13         imprime_resultado(valor_numero_e);
14         return 0;
15 }
16
17 /* Definición de la función numero_e con el parámetro formal iteraciones de tipo entero */
18 double numero_e(int iteraciones){
19         double suma, factorial, termino;
20         int i;
21         i=0;
22         suma=0;
23         factorial=1;
24         termino=1;
25         while (i<iteraciones){
26                 if (i==0)
27                         suma = suma+1;
28                 else{
29                         factorial = factorial*i;
30                         termino=(double)1/factorial;
31                         suma=suma+termino;
32                 }
33                 i++;
34         }
35         return suma;
36 }
37
38 /* Definición de la función imprime_resultado con el parámetro formal numero_e de tipo double */
39 void imprime_resultado(double numero_e){
40         printf("El valor del número e es = %.6lf\n", numero_e);
41         return;
42 }
```

La salida que genera este programa es la siguiente:

```
Ingrese la cantidad de iteraciones: 100
El valor del número e es = 2.718282
```

Se puede observar que la salida es la misma. Sin embargo, si revisamos la estructura de nuestro programa, podemos apreciar que:

- Se ha añadido la declaración de la función *imprime_resultado* en la zona de prototipos.
- El prototipo de la función *imprime_resultado* declarada en la línea 4 tiene tipo de retorno *void*.
- La función *imprime_resultado* recibe solo un parámetro de tipo *double*.
- La invocación a la función *imprime_resultado* en la línea 13 coincide con la declaración del prototipo en el número y tipo de parámetros.

- Como la invocación a la función *imprime_resultado* en la línea 13 no tiene un valor de retorno no se asigna a ninguna variable.
- La definición de la función *imprime_resultado* en la línea 39 se hace luego de la definición de la función *main*. Sin embargo, también se podría haber definido antes de la función *main*.
- El parámetro usado en la definición de la función *imprime_resultado* en la línea 39 indica que el número *e* se mostrará en la pantalla del usuario acompañado de un mensaje final.

Sobre el programa 1.13 y para ejemplificar el paso de parámetros por referencia vamos a modificar el programa para que la tarea de leer los datos de entrada sea delegada a una función encargada para ello.

A continuación, crearemos la función *leer_datos* que muestre en la pantalla del usuario el mensaje solicitándole el ingreso de los datos y permita la captura del valor ingresado por teclado en una variable. En el programa 1.14 se puede apreciar la nueva versión del programa 1.13 que define y utiliza la función *leer_datos* definida por el usuario para dicha tarea.

Programa 1.14: Cálculo del número *e* dentro de una función con la lectura de datos en otra función simulando un paso de parámetros por referencia

```
1  #include<stdio.h>
2  /* Prototipos de las funciones numero_e, imprime_resultado y leer_datos */
3  double numero_e(int iteraciones);
4  void imprime_resultado(double numero_e);
5  void leer_datos(int *dir_num_iteraciones);
6
7  int main(){
8          double valor_numero_e;
9          int num_iteraciones;
10         /* Se invoca a la función leer_datos pasando, como parámetro real, la dirección de la variable num_iteraciones. */
11         leer_datos (&num_iteraciones);
12         valor_numero_e = numero_e(num_iteraciones);
13         imprime_resultado(valor_numero_e);
14         return 0;
15 }
16
17 double numero_e(int iteraciones){
18         double suma, factorial, termino;
19         int i;
20         i=0;
21         suma=0;
22         factorial=1;
23         termino=1;
24         while (i<iteraciones){
25                 if (i==0)
26                         suma = suma+1;
27                 else{
28                         factorial = factorial*i;
29                         termino=(double)1/factorial;
30                         suma=suma+termino;
31                 }
32                 i++;
33         }
34         return suma;
35 }
36
37 void imprime_resultado(double numero_e){
38         printf("El valor del número e es = %.6lf\n", numero_e);
39         return;
40 }
41
42 /* Definición de la función leer_datos con el parámetro formal dir_num_iteraciones de tipo dirección a un entero. */
43 void leer_datos(int *dir_num_iteraciones){
44         printf("Ingrese la cantidad de iteraciones: ");
45         scanf(" %d", dir_num_iteraciones);
46 }
```

La salida que genera este programa es la siguiente:

```
Ingrese la cantidad de iteraciones: 100
El valor del número e es = 2.718282
```

Se puede observar que la salida es la misma. Sin embargo, si revisamos la estructura de nuestro programa, podemos apreciar que:

- Se ha añadido la declaración de la función *leer_ datos* en la zona de prototipos.
- El prototipo de la función *leer_ datos* declarada en la línea 5 tiene tipo de retorno *void*.
- La función *leer_ datos* recibe solo un parámetro de tipo dirección a un entero.
- La invocación a la función *leer_ datos* en la línea 11 coincide con la declaración del prototipo en el número y tipo de parámetros.
- Como la invocación a la función *leer_ datos* en la línea 11 no tiene un valor de retorno no se asigna a ninguna variable.
- La definición de la función *leer_ datos* en la línea 43 se hace luego de la definición de la función *main*. Sin embargo, también se podría haber definido antes de la función *main*.
- El parámetro usado en la definición de la función *leer_ datos* en la línea 43 indica que el parámetro *dir_ num_ iteraciones* es una dirección a un entero, y será mediante esa dirección que se puede acceder al valor para luego modificarla.
- La invocación a la función *scanf* en la línea 45 ya no requerirá que se le anteponga el operador dirección (&) al segundo parámetro, porque la variable *dir_ num_ iteraciones* ya es una dirección.
- Esta técnica de paso de direcciones como parámetros a una función permite lo que se conoce como paso de parámetros por referencia, debido a que lo que se pretende es modificar el valor de la variable definida en el programa o subprograma que invoca a la función.

Para pensar

A pesar que la última versión que utiliza funciones para leer los datos entrada, calcular el número e e imprimir el resultado posee más líneas de código, ¿qué versión es más entendible al ojo humano?

1.8. Cálculo del seno

El seno es una de las funciones trigonométricas más conocidas y usadas en las ciencias. Se utiliza por ejemplo para calcular la trayectoría de un proyectil.

El seno de un ángulo α se define como la razón entre el cateto opuesto a dicho ángulo α y la hipotenusa. Se puede calcular de diversas maneras, entre ellas, la serie de Taylor. Usando esta serie el seno de un grado x, expresado en radianes, se puede calcular de la siguiente manera : $x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \frac{x^{11}}{11!} + \frac{x^{13}}{13!} + \cdots$

A continuación, se muestra el programa 1.15 en el lenguaje C que calcula el valor aproximado del seno usando la serie de Taylor descrita previamente. El programa considera solamente términos que en su valor absoluto tengan hasta 6 dígitos significativos.

Recordar que:

Un radián (rad) es la medida de un ángulo central que subtiende un arco cuya longitud es igual al del radio de la circunferencia. Puesto que la longitud de una circunferencia de radio r es igual a 2π, se tiene que $360°=2\pi$ radianes.

Programa 1.15: Cálculo del seno

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

int main() {
    int i = 1, signo=1;
    double x, suma=0, factorial=1, termino=1;
    printf("Ingrese ángulo en radianes: ");
    scanf("%lf", &x);
    while (fabs(termino)>=0.0000001){
        factorial = factorial * i;
        if (i%2==1){
            termino = signo*pow(x,i)/factorial;
            suma = suma + termino;
            signo = signo*(-1);
        }
        i++;
    }
    printf("suma = %lf\n", suma);
    return 0;
}
```

La salida que genera este programa es la siguiente:

```
Ingrese ángulo en radianes: 1.57
suma = 1.000000
```

Recordar que:

El algoritmo presentado en el programa 1.15 se encuentra ampliamente descrito en las guías anteriores.

Si surge la necesidad de usar el valor del *seno* que acabamos de generar, como parte de otros programas, o incluso, se desea obtener el valor del *seno* con diferentes precisiones, entonces, se puede definir un bloque de código que facilite su obtención y dicho bloque sería parte de una función que permita obtener su valor de acuerdo con el número de dígitos de precisión que reciba como parámetro en su invocación.

A continuación, desarrollaremos una función en el lenguaje C que permita obtener el valor del *seno* de un ángulo en radianes con una cantidad de dígitos significativos; el ángulo y la cantidad de dígitos serán parámetros de la función.

1.8.1. Cálculo del *seno* con una función

Podemos modificar el programa 1.15 de manera que, el cálculo del *seno* de un ángulo en radianes se realice dentro de una función *seno* que recibe como parámetros el ángulo en radianes y la cantidad de dígitos significativos, y retorne el valor aproximado del *seno* para su posterior uso. En el programa 1.16 se puede apreciar la nueva versión del programa 1.15 que define y utiliza la función *seno* definida por el usuario para dicha tarea.

Programa 1.16: Cálculo del *seno* dentro de una función

```c
#include <stdio.h>
#include <math.h>
/* Prototipo de la función seno */
double seno (double x, int num_digitos_signif);

int main() {
    int num_digitos_signif;
    double x, valor_seno;
    printf("Ingrese ángulo en radianes y un número para la cantidad de dígitos significativos: ");
    scanf("%lf %d", &x, &num_digitos_signif);
    valor_seno = seno(x, num_digitos_signif);
    printf("Seno(%.2lf) = %lf\n", x, valor_seno);
    return 0;
```

```
14 }
15
16 /* Definición de la función seno con parámetros formales, el valor del ángulo en grados sexagesimales y el número de dígitos
       significativos. */
17 double seno (double x, int num_digitos_signif){
18     int i = 1, signo = 1;
19     double suma = 0, factorial = 1, termino = 1, limite;
20     limite = 1 / pow(10, num_digitos_signif+1);
21     /* Si num_digitos_signif es 6, entonces límite = 0.0000001 */
22     while (fabs(termino)>=limite){
23         factorial = factorial * i;
24         if (i%2==1){
25             termino = signo*pow(x,i)/factorial;
26             suma = suma + termino;
27             signo = signo*(-1);
28         }
29         i++;
30     }
31     return suma;
32 }
```

Vemos que la función *main* del programa 1.16 cambió considerablemente, en la línea 7 vemos la declaración de una variable *num_digitos_signif*, en la línea 8 la declaración de la variable *valor_seno*, y en la linea 11 se asigna el valor devuelto por la función *seno* a la variable *valor_seno* considerando los dos parámetros que requiere para el cálculo. A partir de este momento, podemos obtener el valor del *seno* con solo llamar a la función que acabamos de crear, pasando como parámetros reales y por valor, el *valor del ángulo en radianes* y el *número de dígitos significativos*.

La salida que genera este programa es la siguiente:

```
Ingrese ángulo en radianes y un número para la cantidad de dígitos significativos: 1.57 6
Seno(1.57) = 1.000000
```

Si revisamos la estructura de nuestro programa, podemos apreciar que:

- El prototipo de la función *seno* declarada en la línea 4 tiene tipo de retorno *double*.
- La función *seno* recibe dos parámetros por valor, el primero es un parámetro de tipo *double* y el segundo es de tipo entero.
- La invocación a la función *seno* en la línea 11 coincide con la declaración del prototipo en el número y tipo de parámetros.
- Como el retorno de la invocación a la función *seno* en la línea 11 es un valor de tipo *double*, éste es asignado a la variable *valor_seno* del mismo tipo *double*.
- La definición de la función *seno* en la línea 17 se hace luego de la definición de la función *main*. Sin embargo, también se podría haber definido antes de la función *main*.
- Los parámetros de la definición de la función *seno* en la línea 17, que coinciden con el prototipo de la función, indican que el primer parámetro corresponde con el ángulo x en radianes, y el segundo parámetro corresponde con el número de dígitos significativos $num\_digitos\_signif$ que debe tener cada término como máximo en su valor absoluto.
- El desarrollo del cuerpo de la función *seno* se realiza en las líneas 18 hasta la 31.

Para poner en práctica

- Realice el desarrollo del programa 1.16 considerando funciones que permitan la lectura de datos, el cálculo del seno y la impresión del resultado, similar a la estrategia tomada en el programa 1.14

1.9. Cálculo del crecimiento exponencial

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 figura 1.4 se propone un diagrama de estructura o de módulos.

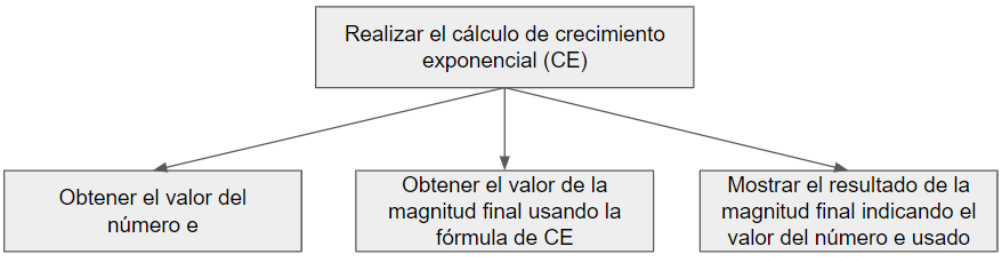


Figura 1.4: Módulos del programa de cálculo del Crecimiento Exponencial

Diagrama de estructura o de módulos:

El diagrama de estructura o de módulos sirve para el modelamiento top-down de un programa y sus subprogramas mediante un árbol de invocación de módulos [5, 11, 16]

1.9.1. Leyendo los datos de entrada

Para resolver el problema usando la fórmula de crecimiento exponencial, será necesario conocer los valores M_0 y t. Estos valores serán ingresados a través de la consola para luego ser usados en la fórmula $M_t = M_0 \cdot e^t$. Dado que necesitamos calcular una potencia, incluiremos la libreria math.h en nuestro programa, que inicialmente se vería como el programa 1.17.

Programa 1.17: Estructura inicial del programa

```
1  #include <stdio.h>
2  #include <math.h>
3
4
5  int main() {
6      int tiempo;
7      double magnitud_instante_mayor_0, magnitud_instante_igual_0;
8
9      printf("Ingrese el valor de la magnitud en el instante 0: ");
10     scanf(" %lf", &magnitud_instante_igual_0);
11
12     printf("Ingrese el tiempo final (t>0): ");
13     scanf(" %d", &tiempo);
```

```
14
15      if (tiempo > 0 && magnitud_instante_igual_0 > 0) {
16          /*Conjunto de instrucciones para el
17          cálculo del valor final de la magnitud*/
18      } else {
19          printf("Los valores para el tiempo y la magnitud inicial deben ser positivos");
20      }
21  }
```

Hasta este punto hemos leído los datos para las variables *tiempo* y *magnitud_instante_igual_0* pero podemos notar que la fórmula además hace uso del número e. Por lo tanto, necesitamos una forma de obtener el valor del número e antes de poder aplicar la fórmula.

1.9.2. Usando la función *numero_e*

Ya que la fórmula que usaremos necesita del número e y antes ya habíamos implementado una función que calcule su valor con un determinado número de iteraciones, podemos incluir dicha función en nuestro programa y llamarla desde donde sea necesario.

Podemos ver que en la línea 27 del programa 1.18 se le asigna el valor de la función *numero_e(100)* a la variable *valor_numero_e*. A partir de ese momento, esta variable tendrá el valor aproximado del número e después de 100 iteraciones.

Programa 1.18: Cálculo del crecimiento exponencial

```
1   #include <stdio.h>
2   #include <math.h>
3
4   double numero_e(int iteraciones) {
5       int i = 0;
6       double suma = 0;
7       double factorial = 1;
8       double termino = 1;
9       while (i <= iteraciones) {
10          if (i == 0)
11              suma = suma + 1;
12          else {
13              factorial = factorial * i;
14              termino = (double) 1 / factorial;
15              suma = suma + termino;
16          }
17          i++;
18      }
19      return suma;
20  }
21
22  int main() {
23      int tiempo;
24      double magnitud_instante_mayor_0, magnitud_instante_igual_0, valor_numero_e;
25
26      /*Obtenemos el valor del número e usando la función f_numero_e()*/
27      valor_numero_e = numero_e(100);
28
29      printf("Ingrese el valor de la magnitud en el instante 0: ");
30      scanf("%lf", &magnitud_instante_igual_0);
31
32      printf("Ingrese el tiempo final (t>0): ");
33      scanf("%d", &tiempo);
34
35      if (tiempo > 0 && magnitud_instante_igual_0 > 0) {
36          /*Aplicamos la fórmula usando los datos de entrada y el valor del número e*/
37          magnitud_instante_mayor_0 = magnitud_instante_igual_0 * pow(valor_numero_e, tiempo);
38          printf("El valor de la magnitud para t=%d es %lf", tiempo, magnitud_instante_mayor_0);
39      } else {
40          printf("Los valores para el tiempo y la magnitud inicial deben ser positivos");
41      }
42  }
```

Si bien el programa 1.18 realiza el cálculo del crecimiento exponencial correctamente, su implementación no corresponde al diagrama de estructura o de módulos presentado inicialmente. Es necesario recordar que al momento de realizar la implementación, cada rectángulo del diagrama corresponde a un subprograma. Por lo tanto, tenemos que realizar el cálculo del valor de la magnitud M_t en el instante $t > 0$ en una nueva función.

Para tal fin, encapsularemos el cálculo del crecimiento exponencial en la función *calcular_ crecimiento_ exponencial*. Esta función deberá recibir los parámetros de la fórmula: tiempo, valor de la magnitud inicial, y valor del número *e*.

A continuación, se muestra la implementación del programa con la nueva función *calcular_ crecimiento_ exponencial*:

Programa 1.19: Cálculo del crecimiento exponencial

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

double numero_e(int iteraciones) {
    int i = 0;
    double suma = 0;
    double factorial = 1;
    double termino = 1;
    while (i <= iteraciones) {
        if (i == 0)
            suma = suma + 1;
        else {
            factorial = factorial * i;
            termino = (double) 1 / factorial;
            suma = suma + termino;
        }
        i++;
    }
    return suma;
}

double calcular_crecimiento_exponencial(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;
}

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

    /*Obtenemos el valor del número e usando la función numero_e()*/
    valor_numero_e = 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 función calcular_ crecimiento_ exponencial()*/
        magnitud_instante_mayor_0 = calcular_crecimiento_exponencial(tiempo, magnitud_instante_igual_0,
              valor_numero_e);
        printf("El valor de la magnitud para t= %d es  %lf", tiempo, magnitud_instante_mayor_0);
    } else {
        printf("Los valores para el tiempo y la magnitud inicial deben ser positivos");
    }
}
```

Si volvemos a revisar el diagrama de estructura o de módulos, podemos notar que nuestra implementación todavía no coincide con el diagrama. Para que nuestra solución esté completa, debemos realizar la salida del programa en una nueva función que llamaremos *mostrar_ magnitud_ final_ con_ parametro_ e*.

1.9.3. Solución propuesta para el cálculo del crecimiento exponencial

Nuestro último subprograma debe mostrar el resultado de la magnitud para el tiempo $t > 0$ (magnitud final), y además debe mostrar también cuál fue el valor del número e que se usó en la fórmula.

En el programa 1.20 se muestra la solución propuesta al problema del cálculo del crecimiento exponencial. Es importante notar que la última función *mostrar_magnitud_final_con_parametro_e* no devuelve ningún valor ya que solo imprime en pantalla los datos requeridos. Por lo tanto, la función tiene un tipo de retorno *void* y no hace uso de la instrucción *return*:

Programa 1.20: Cálculo del crecimiento exponencial

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

double numero_e(int iteraciones) {
    int i = 0;
    double suma = 0;
    double factorial = 1;
    double termino = 1;
    while (i <= iteraciones) {
        if (i == 0)
            suma = suma + 1;
        else {
            factorial = factorial * i;
            termino = (double) 1 / factorial;
            suma = suma + termino;
        }
        i++;
    }
    return suma;
}

double calcular_crecimiento_exponencial(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_parametro_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);
}

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

    /*Obtenemos el valor del número e usando la funcion f_numero_e()*/
    valor_numero_e = 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 función calcular_crecimiento_exponencial()*/
        magnitud_instante_mayor_0 = calcular_crecimiento_exponencial(tiempo, magnitud_instante_igual_0,
             valor_numero_e);

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

```
    }
}
```

La salida que genera este programa es la siguiente:

```
Ingrese el valor de la magnitud en el instante 0: 10
Ingrese el tiempo final (t>0): 5
El valor de la magnitud en el instante t=5 es: 1484.131591
El valor aproximado del numero e que ha usado es: 2.718282
```

Para pensar

¿Es posible aplicar la estrategia de dividir para vencer y obtener un diagrama de estructura o de módulos diferente al propuesto?

1.10. Los números casi primos

En diversos concursos de programación, la definición del número casi primo está presente y suele variar algunas veces. Un número casi primo es aquel número que no es primo y que dentro de sus divisores solo tiene un número primo.

Por ejemplo:

- El número 4 es un número casi primo, debido a que no es primo y dentro de sus divisores que son el 1, 2 y 4 solo tiene un número primo, que es el 2.
- El número 6 no es un número casi primo, debido a que si bien es cierto no es un número primo, dentro de sus divisores que son el 1, 2, 3 y 6 tiene 2 divisores que son primos, el 2 y 3.

El usuario ingresará 2 números *a* y *b*, donde *a>0* y textitb>0 y textita<b, con lo cual se busca representar un rango de números. El programa debe mostrar los números casi primos que existen en ese rango e indicar cuántos son.

1.10.1. Leyendo los datos de entrada

Para resolver el problema será necesario conocer los valores de *a* y *b* que representan el rango de números a evaluar. Estos valores serán ingresados a través de la consola para luego ser usados en nuestro programa. Para ello, crearemos una función *leer_ datos* que permita leer los valores de *a* y *b*. Al ser *leer_ datos* una función que devuelve 2 valores, los mismos deben ser argumentos por referencia para que puedan modificarse al momento de realizar el *scanf* y volver al *main* con los valores leídos. Una versión inicial de nuestro programa se encuentra en el programa 1.21.

Programa 1.21: Lectura y validación de datos

```
#include <stdio.h>

void leer_datos(int *a, int *b);

int main(){
        int i,a,b;
        leer_datos(&a,&b);
        if (a>0 && b>0 && a<b){
                /* Completar */
        }
        else
```

```
12                  printf("Los datos ingresados no son correctos");
13          return 0;
14  }
15
16  void leer_datos(int *a, int *b){
17          printf("Ingrese el rango a y b: ");
18          scanf("%d %d",a,b);
19  }
```

1.10.2. Creamos la función *evaluar_casi_primo*

Lo que se solicita en el problema es hallar los números casi primos que existen en un rango, por lo que sería una buena idea encapsular la evaluación de si un número es casi primo en una función, la cual recibiría como parámetro el número a evaluar y devolvería 1 si el número en evaluación es casi primo o 0 si no lo es. Esta función sería utilizada en la función *main*.

Una segunda versión de nuestro programa podría ser la presentada en el programa 1.22.

Programa 1.22: Invocación de la función *evaluar_casi_primo* para cada número del rango

```
1   #include <stdio.h>
2
3   void leer_datos(int *a, int *b);
4   int evaluar_casi_primo(int numero);
5
6   int main(){
7           int i,a,b,cont_casi_primos=0,es_casi_primo;
8           leer_datos(&a,&b);
9           if (a>0 && b>0 && a<b){
10                  /*Aquí recorremos con la variable i todos los números que están en el rango de a y b*/
11                  i=a;
12                  while (i<=b) {
13                          es_casi_primo = evaluar_casi_primo(i);
14                          if (es_casi_primo){
15                                  printf("El número %d es casi primo\n", i);
16                                  cont_casi_primos++;
17                          }
18                          i++;
19                  }
20                  printf("En el rango de %d a %d existen %d números casi primos",a,b,cont_casi_primos);
21          }
22          else
23                  printf("Los datos ingresados no son correctos");
24          return 0;
25  }
26
27  void leer_datos(int *a, int *b){
28          printf("Ingrese el rango a y b: ");
29          scanf("%d %d",a,b);
30  }
31
32  int evaluar_casi_primo(int numero) {
33          int es_casi_primo=1; /*Partiremos asumiendo que el número sí es casi primo*/
34          /* Debemos evaluar las condiciones de casi primo utilizando los conceptos de números primos*/
35          /* Completar */
36
37          return es_casi_primo;
38  }
```

De la definición de un número casi primo podemos inferir que, para determinar si un número es casi primo debemos considerar dentro de la evaluación si dicho número es primo o no, y en caso no lo fuese debemos saber cuantos divisores primos tiene. Entonces vemos claramente que dentro de esta función utilizaremos una función que permita evaluar si el número es primo, esta función será *evaluar_si_es_primo*.

1.10.3. Usando la función *evaluar_si_es_primo*

Dentro de la evaluación de si un número es casi primo se necesita evaluar si dicho número no es primo y si tiene exactamente solo 1 divisor primo.

Entonces se implementará la función *evaluar_si_es_primo* que evalúa si un número es primo. Esta función será invocada dentro de la lógica de la función *evaluar_casi_primo*, además, puede ser invocada donde sea necesaria.

Con este cambio, la versión final del programa se presenta en el programa 1.23.

Programa 1.23: Presenta los números casi primos en un determinado rango

```c
#include <stdio.h>

void leer_datos(int *a, int *b);
int evaluar_casi_primo(int numero);
int evaluar_si_es_primo(int numero);

int main(){
	int i,a,b,cont_casi_primos=0,es_casi_primo;
	leer_datos(&a,&b);
	if (a>0 && b>0 && a<b){
		/*Aqui recorremos con la variable i todos los números que están en el rango de a y b*/
		for (i=a; i<=b; i++){
			es_casi_primo = evaluar_casi_primo(i);
			if (es_casi_primo){
				printf("El número %d es casi primo\n",i);
				cont_casi_primos++;
			}
		}
		printf("En el rango de %d a %d existen %d números casi primos",a,b,cont_casi_primos);
	}
	else
		printf("Los datos ingresados no son correctos");
	return 0;
}

void leer_datos(int *a, int *b){
	printf("Ingrese el rango a y b: ");
	scanf("%d %d",a,b);
}

int evaluar_si_es_primo(int numero){
	int cant_divisores=0, i, es_primo=0;
	if (numero>1){
		i=1;
		while (i<=numero){ /*Se calcula la cantidad de divisores del número*/
			if (numero%i==0)
				cant_divisores++;
			i++;
		}
		if (cant_divisores==2)
			es_primo = 1;
	}
	return es_primo;
}

int evaluar_casi_primo(int numero) {
	int es_casi_primo=0,cant_div_primos=0, i, es_primo, es_divisor_primo;
	es_primo = evaluar_si_es_primo(numero);
	if (!es_primo){
		i=1;
		while (i<=numero) {
			if (numero%i==0){
				es_divisor_primo = evaluar_si_es_primo(i);
				if (es_divisor_primo==1)
					cant_div_primos++;
			}
```

```
57                      i++;
58              }
59              /*Se cumple ambas condiciones, el número no es primo y solo tiene 1 divisor primo*/
60              if (cant_div_primos==1)
61                      es_casi_primo=1;
62      }
63      return es_casi_primo;
64  }
```

La salida que genera este programa es la siguiente:

```
Ingrese el rango a y b: 1 20
El número 4 es casi primo
El número 8 es casi primo
El número 9 es casi primo
El número 16 es casi primo
En el rango de 1 a 20 existen 4 números casi primos
```

Para pensar

¿Será posible obtener una solución distinta a la propuesta?

Capítulo 2

Ejercicios propuestos

Para cada uno de los ejercicios propuestos se solicita que implemente un programa en el lenguaje C conforme con los temas revisados en las guías *preliminar*, #1, #2 y #3 del curso de Fundamentos de Programación. Además, debe de utilizar el paradigma de programación modular revisado en esta guía.

2.1. Nivel básico

2.1.1. Los polinomios

Un polinomio es una expresión algebraica formada a partir de dos o más términos que contienen coeficientes y variables. Estos términos están vinculados por medio de operadores como la suma o la resta. Se dice que un polinomio es un polinomio completo cuando existen todos sus términos (contando desde el término independiente hasta el término de mayor grado), es decir, cuando todos las coeficientes de todos los términos son diferentes a cero.

Se le pide implementar un programa en el lenguaje C que lea el grado de un polinomio, junto con los coeficientes asociados a cada término, y determine si se trata de un polinomio completo o no. En caso un coeficiente que se ingrese sea 0, se considerará que dicho término no se encuentra en el polinomio. Se deberán ingresar los coeficientes comenzando por el término independiente.

Recuerde que el grado de un polinomio es el mayor exponente al que se encuentra elevada una variable del polinomio.

Su solución deberá incluir 3 funciones adicionales a la función *main*. Una función para la lectura de datos, una función para validar si un coeficiente es diferente de 0, y una función para la salida del mensaje final, tal y como se muestra en los ejemplos de ejecución.

Un ejemplo de ejecución para el polinomio $3x^3 + 2x + 5$ sería:

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

Otro ejemplo de ejecución, esta vez para el polinimo $5x^4 - 2x^3 + x^2 + 8x - 5$ sería:

```
Ingrese el grado del polinomio: 4
Ingrese el coeficiente del término 0: -5
```

```
Ingrese el coeficiente del término 1: 8
Ingrese el coeficiente del término 2: 1
Ingrese el coeficiente del término 3: -2
Ingrese el coeficiente del término 4: 5
El polinomio ingresado SÍ es completo.
```

Sugerencia:

- Su función de lectura puede contener el ingreso de todos los valores de su solución, incluyendo tanto el grado como los coeficientes (estos últimos de manera iterativa) y su función de validación puede ser usada inmediatamente después de la lectura de cada coeficiente.

2.1.2. Movimiento armónico simple

El movimiento armónico simple (M.A.S.), llamado también movimiento vibratorio armónico simple (M.V.A.S.) es un movimiento caracterizado por ser periódico (y oscilatorio) en ausencia de fricción. El móvil que realiza un M.A.S oscilará alejándose y acercándose de algún punto situado en el centro de su trayectoria.

Algunas fórmulas usadas para realizar cálculos de este movimiento son se presentan a continuación:

- $T = \frac{2 * \pi}{w}$.
- $f = \frac{1}{T}$.

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

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

Su programa deberá incluir una función para la lectura de la frecuencia de oscilación f, una función para el cálculo del período T (a partir de la frecuencia de oscilación ingresada), una función para el cálculo de la frecuencia cíclica (a partir del período calculado) y una función de salida que imprima los 2 valores solicitados en el párrafo anterior.

Sugerencia:

- Para trabajar con el número π, usted debería utilizar una directiva de preprocesamiento que realice una sustitución simbólica en base a una secuencia de caracteres.

2.1.3. Máximo común divisor de 2 números

Existen varias formas de calcular el máximo común divisor (MCD) de 2 números. Una de estas formas es listar todos los divisores de cada uno de estos 2 números y determinar cuál es la mayor coincidencia entre estas 2 listas.

Una variante de esta técnica, es solamente listar los divisores del menor de los 2 números a evaluar, y, trabajando con esta lista de atrás hacia adelante (es decir, partiendo del mayor divisor, y terminando en el menor divisor), evaluar si cada uno de los elementos de esta lista son divisores del otro número a evaluar.

Se le pide implementar un programa en el lenguaje C que contenga:

- La función main
- Una función que permita determinar la cantidad de divisores que tiene un número.

- Una función que retorne el divisor número n de algún entero positivo. Su función deberá recibir como parámetros el entero positivo a evaluar, y un número correspondiente al orden o posición del divisor a devolver. Por ejemplo, los divisores del entero positivo 18 son 1, 2, 3, 6, 9 y 18. Por lo que su función deberá poder devolver 6 resultados diferentes, en base al valor de n ingresado. En nuestro ejemplo, si ingresa n=6, entonces el valor a retornar sera 18, si ingresa n=5, entonces el valor a retornar será 9.
- Una función que reciba 2 números enteros como parámetros, y determine si el primero es divisor del segundo. Por ejemplo, si recibe 6 y 48, dado que 6 es divisor de 48, la función retornará 1, en cambio, si recibe 9 y 48, dado que 9 no es divisor de 48, la función retornará 0.

Su programa deberá leer 2 números A y B, y hacer uso de una estructura iterativa que itere tantas veces como divisores tenga el número menor a evaluar, permitiendo hallar el MCD entre A y B. Por ejemplo, si A <B, entonces en la primera vuelta, deberá generar el maximo divisor de A y evaluar si ese divisor de A es, además, divisor de B. En la segunda vuelta, deberá generar el segundo mayor divisor A y evaluar si dicho divisor de a es divisor de B, y así sucesivamente. La primera incidencia que encuentre, será el MCD de A y B.

A continuación, un ejemplo de ejecución para los números 48 y 18:

```
Ingrese dos números (A B): 48 18
el MCD de 48 y 18 es 6
```

Su programa determinará que el número 48 tiene 10 divisores (1, 2, 3, 4, 6, 8, 12, 16 ,24, 48) y el número 18 tiene 6 divisores (1, 2, 3, 6, 9, 18). En seguida, dado que 48 es mayor a 18, su programa se dispondrá a generar los 6 divisores de 18, empezando por el mayor. El mayor divisor de 18 es el mismo 18, por lo tanto, su programa comparará este 18 (es decir, el sexto elemento de la lista de divisores de 18) contra el número 48 preguntándose si 18 es divisor de 48.

Dado que 18 no es divisor de 48, su programa comparará el siguiente máximo divisor de 18 (es decir, el quinto elemento de la lista de divisores de 18) y repetirá el mismo proceso, es decir, trabajará con el 9, y se preguntará si 9 es divisor de 48.

Nuevamente, dado que el número evaluado no es divisor de 48, el programa generará ahora el tercer mayor divisor de 18, que vale 6 (es decir, el cuarto elemento de la lista de divisores de 18) y al enfrentarlo contra 48, encontrará que ese 6 sí es divisor de 48. En ese momento su programa dejará de iterar y mostrará un mensaje indicando cuál es el MCD entre 48 y 18.

Sugerencia

- Debido a lo extenso que es este ejercicio, no considerar el caso de 2 números que no tienen MCD. Es decir, asuma que el usuario ingresará siempre una pareja de 2 números que tienen MCD.
- Asuma que el usuario ingresará siempre valores mayores o iguales a 2. No se preocupe por esta validación.

2.1.4. Mínimo común múltiplo de 2 números

Existen varias formas de calcular el mínimo común múltiplo (mcm) de 2 números. Una de estas formas es comenzar a listar todos los posibles múltiplos de cada uno de estos 2 números y determinar cuál es la menor coincidencia entre estas 2 listas infinitas.

Una variante de esta técnica es comenzar a generar solamente los múltiplos del mayor de los 2 números a evaluar, y determinar si cada número generado es múltiplo del otro número. El primer número generado que cumpla la condición de ser múltiplo del otro, será el mcm.

Se le pide implementar un programa en el lenguaje C que contenga:

- La función *main*

- Una función que determine si un número A es múltiplo de otro número B (debe retornar 1 si es que A es múltiplo de B, y 0 si es que A no es múltiplo de B).

Su programa deberá contener una estructura iterativa que, partiendo de dos números ingresados por teclado, evalúe los 100 primeros múltiplos del mayor de ellos para determinar, haciendo uso de la función señalada en la viñeta anterior, el mcm de los números ingresados.

A continuación, un ejemplo de ejecución para los números 6 y 8:

```
Ingrese dos números (A B): 6 8
el mcm de 6 y 8 es 24
```

En este ejemplo de ejecución, dado que 8 es mayor que 6, su programa comenzará a generar todos los posibles múltiplos de 8, y cada uno de ellos será evaluado para determinar si se cumple que también es múltiplo de 6. El primer valor múltplo de 8 generado que cumpla con la condición de ser múltiplo de 6 fue el número 24.

Sugerencia

- Asuma que todos los números ingresados permitirán encontrar un mcm dentro de los 100 primeros múltiplos comparados.
- Asuma que el usuario ingresará siempre valores mayores o iguales a 2. No se preocupe por esta validación.
- La solución natural de este problema es mediante el uso de una estructura iterativa controlada por contador. En caso de que usted decida trabajar la iteración por medio de un centinela, no olvide la condición de generar solo los 100 primeros múltiplos del número mayor, para evitar iteraciones infinitas.

2.1.5. El pentágono regular

Se dice que un pentágono es regular cuando todos sus lados son iguales y, además, todos sus ángulos internos son congruentes. Ver Figura2.1.

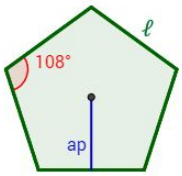


Figura 2.1: Pentágono Regular

En 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 el 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.

Su solución deberá incluir una función para el cálculo del área del pentágono, la cual será llamada desde la función *main*, y otra función para el cálculo del apotema, la cual será llamada desde la función que calcula el área del pentágono.

Sugerencia

Adicionalmente, puede considerar el uso de un módulo para la lectura de datos y otro módulo para la impresión de los valores calculados.

2.1.6. 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 le pide que implemente un programa en el lenguaje C que realice las siguientes operaciones con intervalos cerrados:

- El usuario podrá ingresar los límites inferior y superior de dos intervalos.
- Debe verificar, mediante una función, si dichos intervalos son válidos, es decir, deberá controlar que el límite superior sea mayor o igual al límite inferior. Si alguno de los intervalos no es válido, debe mostrar un mensaje de error indicando claramente cuál fue el intervalo inválido.
- En el caso que ambos intervalos sean válidos, se deberá comprobar (mediante una función) si los intervalos se intersecan. Esta comprobación deberá incluir mensajes hacia el usuario para informarle si hay intersección o no.
- En el caso que los intervalos se intersequen, debe obtenerse los límites inferior y superior del intervalo resultante de la intersección.

Para la implementación utilice el diagrama de estructura o de módulos de la figura 2.2.

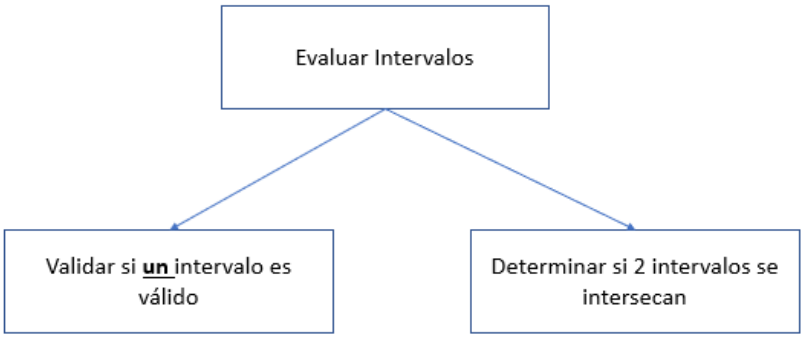


Figura 2.2: Diagrama de estructura o de módulos - Los intervalos

A continuación, se muestra un ejemplo de la ejecución del programa solicitado:

```
Ingrese límite inferior y superior del primer intervalo: -12.6    34.7
Ingrese límite inferior y superior del segundo intervalo: 5.4    63.9
El intervalo A es válido.
El intervalo B es válido.
¿Los intervalos A y B se intersecan? Sí.
Los datos del intervalo resultante de la intersección son:
Límite inferior: 5.4
Límite Superior: 34.7
```

```
Ingrese límite inferior y superior del primer intervalo: -12.6    5.4
Ingrese límite inferior y superior del segundo intervalo: 34.7    63.9
El intervalo A es válido.
El intervalo B es válido.
¿Los intervalos A y B se intersecan? No.
```

2.1.7. Implementación de un menú con diferentes operaciones

En determinadas situaciones, vamos a requerir que nuestros programas realicen diferentes actividades, según la necesidad del usuario. Para atender diferentes necesidades con un solo programa, es posible crear un menú en el que se le presenta al usuario las diferentes funcionalidades de nuestros programas, para que ellos elijan qué hacer.

Para explicar este concepto, vamos a apoyarnos de dos ejercicios muy sencillos, el primero correspondiente a la distancia entre dos puntos, y el segundo correspondiente a la distancia entre un punto y una recta.

Como se puede apreciar, estas dos tareas corresponden a diferentes cálculos de distancias en el plano cartesiano. Usted podría elaborar un solo programa en el que se le muestre al usuario un menú con las dos posibles formas de calcular distancias, y el usuario pueda elegir entre las opciones del menú según su necesidad.

Recordar que:

En el plano cartesiano, un punto P está representado por una coordenada x y una coordenada y, denotándose de la siguiente forma: *P(x,y)*. En geometría analítica, las líneas rectas en un plano pueden ser expresadas mediante una ecuación del tipo $Ax + By + C = 0$, donde x e y son coordenadas de un punto en un plano cartesiano.

Utilizando ambos conceptos, podemos calcular la distancia que existe entre 2 puntos y la distancia mínima que existe de un punto a una recta.

- Para calcular la distancia que existe entre 2 puntos $P(x1, y1)$ y $Q(x2, y2)$ se cuenta con la siguiente fórmula :

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

- Para calcular la distancia mínima de un punto $P(x1, y1)$ a una recta con ecuación $Ax+By+C = 0$ se tiene la siguiente fórmula:

$$D_p l = \frac{|Ax_1 + By_1 + C|}{\sqrt{A^2 + B^2}}$$

Se le pide elaborar un programa en el lenguaje C que le permita calcular la distancia entre 2 puntos y la distancia mínima de un punto a una recta. Para ello deberá imprimir un texto que le presente al usuario las opciones que tiene su programa de cálculo de distancias y le permita seleccionar la funcionalidad que quiera.

Un mensaje apropiado puede ser el siguiente: Ingrese 1 para calcular la distancia entre dos puntos, o ingrese 2 para calcular la distancia entre un punto y una recta.

Dependiendo del número que ingrese el usuario, su programa deberá realizar lo siguiente:

- Si ingresa la opción 1 se calculará e imprimirá la distancia entre 2 puntos, para realizar esta tarea, su programa deberá leer las coordenadas correspondientes a 2 puntos, es decir, leerá cuatro coordenadas. Además su pograma deberá determinar (mediante el uso de una función) si es que es mayor la distancia de las abscisas (el par de coordenadas x), o la distancia de las ordenadas (el par de coordenadas y). Deberá imprimir el mensaje correspondiente, considerando también la posibilidad de que ambas distancias sean iguales.

- Si ingresa la opción 2, se calculará la distancia mínima de un punto a una recta (para ello se ingresará las coordenadas x e y de un punto, y luego los valores para los coeficientes A, B y C correspondientes a la recta). En seguida, deberá reemplazar estos 5 valores en la fórmula correspondiente e imprimir la distancia solicitada.

Considere el siguiente diagrama de estructura o de módulos para resolver el problema:

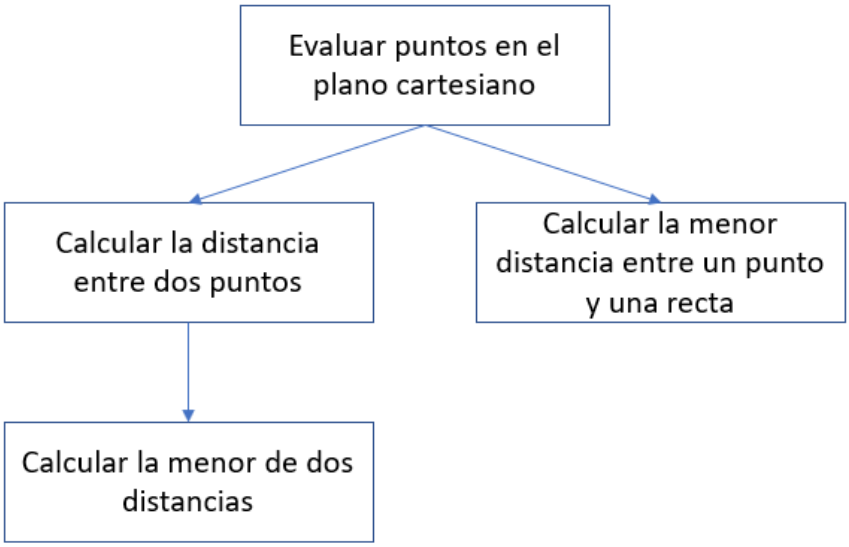


Figura 2.3: Diagrama de estructura o de módulos - Puntos en el plano cartesiano.

A continuación, se muestran algunos ejemplos de ejecución:

```
Ingrese 1 para calcular la distancia entre dos puntos, o ingrese 2 para calcular la distancia
entre un punto y una recta: 1
Ingrese las coordenadas del primer punto (x1 y1): 4 1
Ingrese las coordenadas del segundo punto (x2 y2): 7 5
La distancia entre el punto (4,1) y el punto (7,5) es 5
La diferencia mayor estuvo en las coordenadas Y
```

```
Ingrese 1 para calcular la distancia entre dos puntos, o ingrese 2 para calcular la distancia
entre un punto y una recta: 2
Ingrese las coordenadas del punto (x y): 4 3
Ingrese los 3 coeficientes de la recta Ax+By+C=0: 5 -12 -10
La menor distancia entre el punto (4,1) y la recta (5)x+(-12)y+(-10) = 0 es: 3
```

Sugerencia:

- El cálculo de cada distancia deberá ser implementado en una función aparte.
- Usted puede agregar dos funciones de lectura de datos (una para leer 2 puntos, y otra para leer un punto y una recta), y dos funciones para la salida de datos (una para la distancia entre 2 puntos y la diferencia mayor, y otra para la menor distancia entre un punto y una recta).
- Usted puede agregar una tercera opción a su menú que diga: Para finalizar el programa, ingrese 9, y programar una impresión de mensaje de finalización en caso el usuario ingrese dicho número.

A continuación, se muestran un ejemplo de ejecución con el cambio propuesto:

```
Ingrese 1 para calcular la distancia entre dos puntos, ingrese 2 para calcular la distancia
entre un punto y una recta, o ingrese 9 para finalizar: 9

Gracias por usar nuestros servicios, hasta luego.
```

2.1.8. Tres números consecutivos

Se le pide realizar un programa en el lenguaje C que solicite al usuario una lista de 20 números positivos y que indique el mensaje *¡Felicitaciones, ha ingresado 3 números consecutivos!* cada vez que se identifique que se han introducido 3 números consecutivos. Después de haber ingresado los 20 números, el programa deberá indicar cuántas veces se imprimió el mensaje de felicitaciones. No considerar una misma lectura de datos para dos triadas diferentes, es decir, si ingresa los números 3, 4, 5, 6, 7, 1 solo debe considerar la secuencia 3, 4, 5 en su mensaje de bienvenida e ignorar las triadas 4, 5, 6 y 5, 6, 7. Por lo tanto, en este ejemplo solo se imprime 1 mensaje de felicitaciones. En cambio, si se ingresan los números 3, 4, 5, 6, 7, 8, entonces se imprimirían 2 mensajes de felicitaciones.

Su solución debe incluir las siguientes funciones:

- La función *main*.
- Una función que lea 1 número.
- Una función que recuerde los 3 últimos números leídos.
- Una función que verifique si 3 números son consecutivos.
- Otras funciones que usted considere importantes tanto para la impresión de mensajs, como para el conteo de coincidencias.

A continuación se presenta un ejemplo de ejecución:

```
Ingrese un número: 3
Ingrese un número: 4
Ingrese un número: 6
Ingrese un número: 7
Ingrese un número: 8
¡Felicitaciones, ha ingresado 3 números consecutivos!
Ingrese un número: 9
Ingrese un número: 4
Ingrese un número: 1
Ingrese un número: 2
Ingrese un número: 5
Ingrese un número: 3
Ingrese un número: 4
Ingrese un número: 8
Ingrese un número: 9
Ingrese un número: 10
¡Felicitaciones, ha ingresado 3 números consecutivos!
Ingrese un número: 9
Ingrese un número: 14
Ingrese un número: 1
Ingrese un número: 22
Ingrese un número: 5
Se imprimió un total de 2 mensajes de felicitaciones.
```

Sugerencia

Puede probar una variante con estructura iterativa controlada por centinela, en la que se soliciten infinitos números y el programa termine cuando el usuario ingrese el valor de -1. En este caso, el número de intentos ya no estaría limitado por el número 20.

A continuación se presenta un ejemplo de ejecución con esta variante:

```
Ingrese un número: 6
Ingrese un número: 7
Ingrese un número: 8
¡Felicitaciones, ha ingresado 3 números consecutivos!
Ingrese un número: 9
Ingrese un número: 4
Ingrese un número: 5
Ingrese un número: 6
¡Felicitaciones, ha ingresado 3 números consecutivos!
Ingrese un número: 7
Ingrese un número: 8
Ingrese un número: 4
Ingrese un número: -1
Se imprimió un total de 2 mensajes de felicitaciones.
```

2.1.9. 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 2.4.

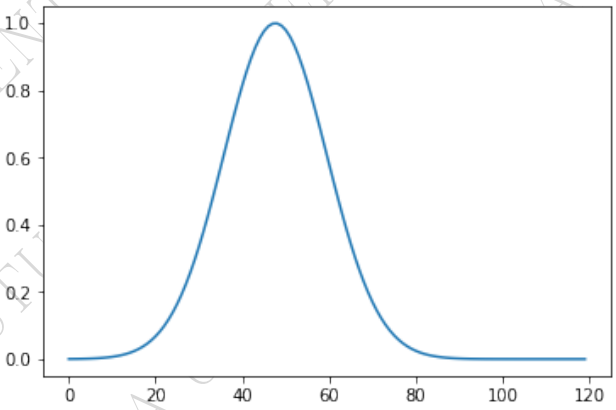


Figura 2.4: Curva de distribución normal - Campana de Gauss

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:

$$f(x) = ae^{-\frac{(x-b)^2}{2c^2}}$$

Donde:

- a determina el valor del punto más alto de la campana

- b determina el centro de la campana
- c determina el ancho de la campana

En Estadística, se utiliza mucho una función gaussiana en la que:

$$a = \frac{1}{c\sqrt{2\pi}}$$

Pues, esta función Gaussiana permite modelar la función de densidad de una variable aleatoria, con media $\mu = b$ y varianza $\sigma^2 = c^2$. Tendríamos, entonces, una función Gaussiana con las siguientes características:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Donde:

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

Se le pide implementar un programa en el lenguaje C 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. Su solución deberá incluir una función para el cálculo de $f(x)$ y otra para la impresión de un mensaje que indique claramente cuál es el resultado de la función, y cuál valor de x se usó para calcular dicho resultado. El valor de π debe ser tratado como una constante simbólica con su respectiva directiva de pre procesamiento.

Con el objetivo de entender mejor cómo funciona el paradigma de programación modular, utilice la función descrita en la sección 1.7 que permite el cálculo del número e.

En la figura 2.5, se muestra el diagrama de estructura o de módulos sugerido para el diseño del programa.

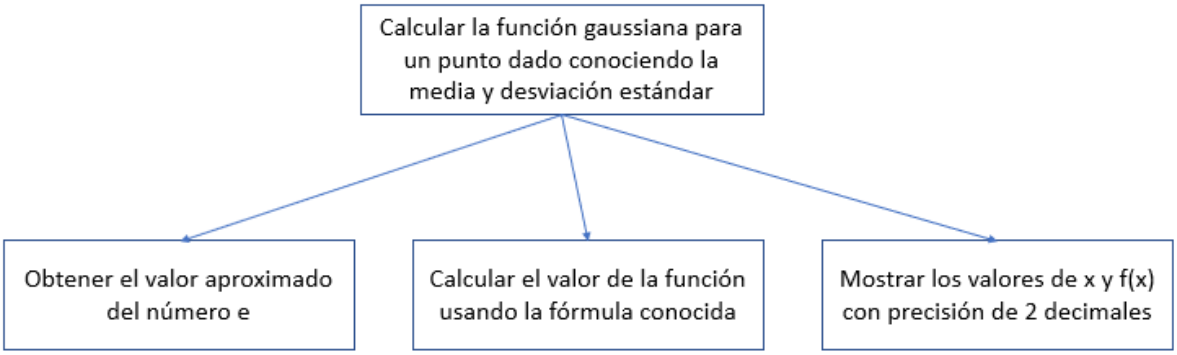


Figura 2.5: Diagrama de estructura o de módulos - Función Gaussiana

Sugerencia

Con el objetivo de entender mejor cómo funciona el paradigma, considere también implementar una función ficticia para el cálculo del número π, es decir, en vez de usar una directiva de pre procesamiento, deberá cargar su variable π mediante una función que lo único que hace es retornar 3.141593. Más adelante, trabajaremos una función que estima el valor de este número y la implementaremos como parte de nuestros programas, pero este ejercicio le ayudará a comprender mejor cómo es que se comunican 2 o más funciones para cumplir un objetivo.

2.1.10. Cálculo del número Pi

El cálculo del número π siempre ha sido una tarea desafiante. Para este fin se han utilizado distintas series como la de Gregory, Newton, Sharp, Euler, entre otras. En 1655, Jhon Wallis descubrió una productoria (conocida como el producto de Wallis) que aproxima el valor de π.

El producto de Wallis establece que

$$\prod_{n=1}^{\infty}\left(\frac{2n}{2n-1}\cdot\frac{2n}{2n+1}\right)=\frac{2}{1}\cdot\frac{2}{3}\cdot\frac{4}{3}\cdot\frac{4}{5}\cdot\frac{6}{5}\cdot\frac{6}{7}\cdot\frac{8}{7}\cdot\frac{8}{9}=\frac{\pi}{2}$$

Utilizando el producto de Wallis se puede calcular fácilmente el valor de π. A medida que se incrementa el número de términos de la productoria, se obtienen valores de π con menos error de aproximación.

Se le pide que desarrolle un programa en el lenguaje C que permita imprimir el valor aproximado de π usando el producto de Wallis con 200 términos y 2000 términos. Su solución deberá incluir una función que retorne la aproximación de π en base a un número determinado de términos. Deberá hacer un llamado a esta función tanto para la aproximación con 200 términos y la aproximación con 2000 términos.

A continuación, se presenta un ejemplo de una posible salida de este programa:

```
El número PI (200 términos) es: 3.137678
El número PI (2000 términos) es: 3.141200
```

2.2. Nivel intermedio

2.2.1. La conjetura de Collatz

En 1937, el matemático alemán Lothar Collatz, enunció la conjetura de Collatz, también conocida como el problema de Ulam, conjetura 3n + 1, entre otros. Collatz enunció que, a partir de cualquier número natural, siempre se puede llegar a la unidad siguiendo el mismo procedimiento: Si el número es par, hay que dividirlo entre 2, en cambio, si no es par, solo hay que multiplicarlo por 3 y sumarle 1. Repetir este procedimiento hasta obtener 1 como respuesta.

Por ejemplo, si se toma el número 13, se obtiene el siguientes resultado: 40, 20, 10, 5, 16, 8, 4, 2, 1; y si se toma el número 6, se obtiene el siguiente resultado: 3, 10, 5, 16, 8, 4, 2, 1

Se le pide que desarrolle un programa en el lenguaje C que permita imprimir el valor de la serie cuando se ingresa un número entero positivo. Su solución deberá contener una función que reciba un número n, y retorne *n/2* si se trata de un número par, y *3n+1* si se trata de un número impar. Para eso, esta función deberá hacer un llamado a otra función que valide si un número es par o impar.

Deberá hacer uso de una estructura iterativa que le permita generar la serie de Collatz de un número ingresado por teclado, invocando múltiples veces a las funciones descritas en el párrafo anterior.

Sugerencia

Si decide usar una función con parámetros por valor, deberá iterar con dos variables que almacenen el término actual y el término de la iteración anterior. Si decide usar una función que simule el paso de parámetros por referencia, deberá sobrescribir repetidas veces el valor a evaluar. No olvide imprimirlo antes de volver a sobrescribirlo.

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

```
Ingresa número a evaluar: 6
La serie es: 3,10,5,16,8,4,2,1.
```

2.2.2. Las rectas paralelas y perpendiculares

Es muy frecuente que, al graficar dos ecuaciones lineales en un plano de coordenadas, estas se crucen. En el caso de que se crucen en un ángulo recto, se dice que estas rectas son *perpendiculares*. Para saber si se cruzarán formando un ángulo recto, basta con calcular el producto de las pendientes de ambas rectas, si este producto es igual a -1, podemos afirmar que estamos frente a dos *rectas perpendiculares*.

Por otro lado, existen también algunas rectas que nunca se cruzan. Estas son llamadas *rectas paralelas* y pueden identificarse rápidamente comparando sus pendientes. Dos rectas serán paralelas si sus pendientes son iguales.

Se le pide desarrollar un programa en el 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 de cuál de estos posibles 3 casos se trata.

A continuación, se muestra un ejemplo de una posible salida de este programa para 4 puntos:

```
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.
```

Sugerencia

Puede considerar las siguientes funciones:

- Una función que calcule la pendiente de una recta.
- Una función que determine si dos rectas son paralelas.
- Una función que determine si dos rectas son perpendiculares.
- Una función que lea las 2 coordenadas de un punto. Esta función deberá imprimir un mensaje como el que se muestra en el ejemplo de ejecución, por lo que deberá estar preparada para imprimir el nombre del punto a leer, en base a un parámetro dado (de tipo caracter).

2.2.3. Los números catalanes

Los números catalanes son los números que están asociados a la fórmula:

$$C_i = \frac{1}{i+1}\binom{2i}{i}$$

Recordemos que para trabajar con combinatorias sin repetición, se puede usar la siguiente fórmula:

$$\binom{n}{p} = \frac{n!}{p!\,(n-p)!}$$

Por lo tanto:

$$\binom{2i}{i} = \frac{(2i)!}{i!\,(2i-i)!} = \frac{(2i)!}{i! \times i!}$$

Reemplazando en nuestra fórmula de número catalán, teneos que:

$$C_i = \frac{1}{i+1}\binom{2i}{i} = \frac{1}{i+1}\left(\frac{(2i)!}{i! \times i!}\right) = \frac{(2i)!}{(i+1)! \times i!}$$

Siguiendo esta fórmula, es posible determinar que los primeros números catalanes son: 1,1,2,5,14,42,132,429...

Se le pide implementar un programa en el lenguaje C que al ingresar un número i, calcule e imprima todos los números catalanes desde C_0 hasta C_i

Su solución deberá incluir una función para el cálculo del factorial de un número, y una función para el cálculo de un número catalán. Estas funciones deberán ser llamadas repetidas veces desde una estructura iterativa con entrada controlada.

A continuación, se muestran dos ejemplos de ejecución:

```
Ingresar un número: 0
Los números catalanes correspondientes son: 1
```

```
Ingresar un número: 4
Los números catalanes correspondientes son: 1, 1, 2, 5, 14
```

2.2.4. Lejos de los primos

Un número primo es un entero mayor que 1 cuyos únicos divisores son él mismo, y la unidad. Algunos ejemplos de números primos son: $2, 3, 5, 7, 11, 13y17$. Un número N se considera lejano de un número primo si es que no existen números primos desde $N-10$ hasta $N+10$. Es decir, ninguno de los números $N-10, N-9, ..., N-1, N, N+1, ..., N+9, N+10$ deberá ser primo para considerar a N como un número lejano de los primos.

Se le pide: dado un entero A y un entero B. Devuelva la cantidad de números lejanos de primos contenidos entre A y B inclusive. Por ejemplo, si se ingresaran los números $A = 3328$ y $B = 4100$, encontraríamos que existen 4 números lejanos de primos que son los números 3480, 3750, 3978 y 4038.

En la salida deberá imprimir la cantidad de números lejano de primos que existen entre A y B, acompañada de un mensaje, tal y como se aprecia en el ejemplo de ejecución. Su solución deberá incluir una función que determine si un núnro es primo o no, la cual será invocada para evaluar a todos los números comprendidos entre $N-10$ y $N+10$ inclusive.

A continuación se muestra un ejemplo de una posible salida de este programa:

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

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

2.2.5. 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 o de derecha a izquierda, y cada uno de los números remanentes también será primo. Por ejemplo, de izquierda a derecha, después de cada retiro de dígitos, se obtendrían los valores 3797, 797, 97 y 7, los cuales son todos números primos. Similarmente si trabajamos de derecha a izquierda, se obtendrían los valores 3797, 379, 37 y 3, todos primos.

Sugerencia

No considerar números de 1 cifra, es decir, los números 2, 3, 5 y 7 no se consideran primos truncables. Asimismo, recuerde que el número 1 no es primo.

Se le pide implementar un programa que solicite al usuario el ingreso de un rango de dos números enteros mayores que 1, y retorne un mensaje con todos los números primos truncables que hay dentro del rango ingresado. Si no hay ningún primo truncable su programa deberá imprimir un mensaje indicando lo sucedido.

Su solución deberá implementar una función que realice el retiro de dígitos de izquierda a derecha y otra función que realize el retiro de dígitos de derecha a izquierda:

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

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

2.2.6. La conjetura de Goldbach

La conjetura de Goldbach es un teorema que señala que todo número par mayor que 2 puede escribirse como la suma de dos números primos. Por ejemplo, el número 8 puede escribirse escribirse como la suma de los primos 3 y 5.

Se le solicita implementar un programa en el lenguaje C que determine si un número ingresado por teclado cumple con la conjetura de Goldbach. Usted debe asumir que el usuario únicamente ingresará números pares mayores a 2, es decir, no debe validar que el número ingresado cumpla esta condición, y deberá únicamente validar si el número ingresado cumple con la conjetura o no.

Al evaluar si su número cumpla la conjetura deberá, además, imprimir los números primos que permiten que se cumple dicha conjetura. En caso un número pueda ser representado por dos o más sumas diferentes de primos, todas estas deberán ser impresas (cada una en una línea diferente).

Su solución deberá incluir una función que determine si un número es primo o no, la cual deberá ser invocada repetidas veces dentro de una estructura iterativa que genere todas las posibles parejas de sumas de primos a comparar con el valor ingresado por teclado. En caso de que un número par mayor a 2 no cumpla con esta conjetura, su programa deberá imprimir un mensaje de alerta, caso contrario, deberá imprimir el mensaje "Se cumple la conjetura de Goldbach".

A continuación, se muestra un ejemplo de ejecución:

```
Ingrese un número: 10
10 = 3 + 7
10 = 5 + 5
10 = 7 + 3
Se cumple la conjetura de Goldbach
```

2.2.7. Encontrar el capicúa más cercano

Un número N es capicúa, cuando se lee igual de izquierda a derecha que de derecha a izquierda. Se le solicita implementar un programa en el lenguaje C que lea un número y determine si es capicúa o no. En caso de que no lo sea, su solución deberá encontrar el capicúa más cercano, indicando cuánto se tiene que restar o sumar para hallar dicho número capicúa.

Su solución deberá hacer uso de una estructura algorítmicas iterativa para realizar la búsqueda del número capicúa más cercano, generando nuevos números mayores o menores al número ingresado por teclado.

Su solución deberá incluir una función que verifique si un número es capicúa o no. Esta función deberá volver a ser llamada cada vez que se generan números mayores o menores al número ingresado por teclado, para que sean evaluados también. En caso de que un número ingresado por teclado esté igual de cerca de dos números capicúas, su programa deberá imprimir los dos.

En el caso propuesto, se cumple ver Figura 2.6.

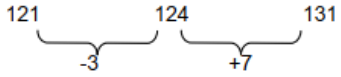


Figura 2.6: Ejemplo capicúa cercano

A continuación se muetran dos ejemplos de ejecución de este programa:

```
Ingresar número: 124
Capicúa más cercano es 121, se tiene que restar 3.
```

```
Ingresar número: 126
Capicúa más cercano es 121, se tiene que restar 5.
Capicúa más cercano es 131, se tiene que sumar 5.
```

```
Ingresar número: 121
El número ingresado es capicúa.
```

2.2.8. Rotación de números

Llamemos rotación de un número al proceso por el cual tomamos un número y corremos una posición todos sus dígitos (para el caso del dígito que quede fuera, este toma el lugar de la posición que quedó vacía al hacer el cambio). Por ejemplo, el número 45678 puede rotar hacia la derecha, moviendo sus primeros 4 dígitos una posición hacia la derecha, y el quinto dígito (el número 8) a la posición inicial, es decir, quedaria 84567. Asimismo, este número podría rotar hacia la izquierda si corremos sus últimos 4 digitos una posición hacia la izquierda, y movemos el primer dígito a la posición de la unidad, de la siguiente manera: 56784.

Se le pide implementar un programa que reciba como datos un número a rotar, el número de veces que se quiere hacer la rotación, y un caracter que permita al usuario indicar hacia qué dirección desea hacer la rotación: rotando hacia la derecha si el usuario ingresa la letra D y hacia la izquierda si el usuario ingresa la letra I (considere mayúsculas y minúsculas).

Su solución deberá hacer uso de una función que, simulando el paso de parámetros por referencia, permita realizar la rotación dentro de una misma variable, es decir, su función deberá sobrescribir al número ingresado cada vez que este sea rotado.

A continuación, un ejemplo de ejecución:

```
Ingrese el número a rotar: 789456
Ingresar la cantidad de veces:2
Ingresar sentido de la rotación: I
El resultado es: 945678
```

Sugerencia

Puede considerar una función que convierta la «D» mayúscula en «d» minúscula, y la «I» mayúscula por «i» minúscula para facilitar la elección de la dirección de la rotación. De hacer esto, luego solo tendría que validar contra minúsculas.

2.2.9. El reloj digital

En un reloj digital, la hora se representa por tres números enteros: las horas h, los minutos m y los segundos s. En cambio, en un reloj analógico se utilizan 3 agujas para representar la hora: el horario, el minutero y el segundero; cada una de ellas forma un ángulo respecto al origen (una aguja imaginaria que apunta hacia las 12). Si se conociera la hora del reloj digital, se podría determinar los ángulos que forma cada aguja del reloj analógico, con respecto al origen (aguja imaginaria que señala las 12):

- AngH = 30 h + m/2 + s/120
- AngM = 6m +s/10
- AngS = 6s

Donde AngH es el ángulo del horario, AngM es el ángulo del minutero y AngS es el ángulo del segundero. Se le solicita implementar un programa que requiere el ingreso de la hora en un reloj digital, y calcule en qué posición se deberán colocar las agujas de un relog analógico para que este señale la hora ingresada por teclado.

Deberá asumir que se trata de un reloj analógico nuevo, por lo que al comenzar su programa, las 3 agujas se encontrarán apuntando hacia las 12. Su salida de datos deberá indicar cuántos ángulos se debe mover cada aguja. Considere tres funciones para el cálculo de los tres ángulos finales en los que deberán estar cada una de las tres agujas (pues cada aguja obedece a una fórmula diferente) y, además, una función que calcule cuántos ángulos debe moverse una aguja cualquiera para llegar desde su posición inicial hasta su posición final.

Para este ejercicio, dado que las tres agujas se encuentran inicialmente apuntando hacia las 12, esta función siempre retornará un valor idéntico al ángulo correspondiente a la posición final de la aguja evaluada. Por ejemplo, si desea que el minutero señale 15 minutos (es decir, que apunte hacia las 3), esta función recibirá como parámetros la posición inicial en ángulos (cero, pues la aguja inicialmente apunta hacia las 12) y la posición final en ángulos (90, pues en su posición final debe apuntar hacia las 3), motivo por el cual la función retornará 90, ya que 90 - 0 = 90.

A continuación, se presenta un gráfico que muestra las tres agujas del reloj analógico, así como también una aguja ficticia que señala a las 12. Su programa deberá siempre calcular las diferencias entre el ángulo de cada aguja y el ángulo de la aguja imaginaria representada por una línea punteada. Ver figura 2.7.

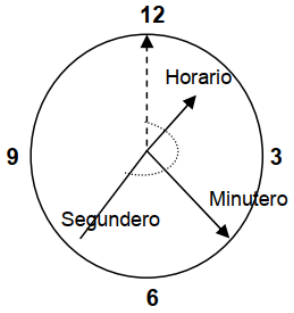


Figura 2.7: Reloj analógico

A continuación, un ejemplo de ejecución:

```
Ingrese una hora en formato hh:mm:ss 03:00:00
El horario debe moverse 90 grados
El minutero debe moverse 0 grados
El segundero debe moverse 0 grados
```

Sugerencia

Asuma que el usuario siempre ingresa valores entre 00:00:00 y 11:59:59. Asuma que el usuario siempre ingresará horas menores o iguales a 11, minutos menores o iguales a 59 y segundos menores o iguales a 59.

Para pensar

Considere implementar este mismo ejercicio para situaciones en las que las manecillas inicialmente no se encuentran apuntando hacia las 12. Puede agregar una lectura para el ingreso de la hora inicial (y calcular el ángulo en el que se encuentra cada aguja al comienzo del ejercicio) y otra lectura para el ingreso de la hora final (y calcular el ángulo en el que se deben encontrar cada una de las agujas al finalizar del ejercicio), e imprimir cuántos grados se debería mover cada aguja (puede escribir los grados negativos para el caso de las agujas que tengan que retroceder).

Para pensar

Considere implementar un programa que lea la hora inicial y múltiples horas finales. El programa indicará los grados que deben moverse las agujas frente a cada nuevo ingreso de datos. Su programa deberá terminar cuando el usuario ingrese la hora 99:99:99.

2.2.10. Coordenadas polares y cartesianas

Existen dos sitemas de coordenadas bidimensionales muy conocidos: 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 origen O al punto dado P y donde θ es el ángulo de inclinación del radio vector OP con respecto al eje polar (eje equivalente al eje x del sistema cartesiano), tal y como se muestra en la figura 2.8.
- 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) a su representación polar, o un punto P(r,θ) a su representación cartesiana, se pueden las siguientes fórmulas:

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

Se le pide implementar un programa en el lenguaje C que lea un punto P(x,y), calcule y muestre su representación en coordenadas polares, o un punto P(r,θ) y calcule y muestre su representación en coordenadas cartesianas, según lo solicite el usuario. Las fórmulas listadas en este enunciado solo son válidas para coordenadas cartesianas que correspondan a un punto ubicado en el primer cuadrante, por lo tanto, usted deberá asumir que el usuario siempre ingresa puntos ubicados en dicho cuadrante.

Su solución deberá incluir: la función main, una constante simbólica para establecer el valor de PI, una función que transforme un punto de coordenadas polares a coordenadas cartesianas, una función que transforme un punto de coordenadas cartesianas a coordenadas polares, y otras funciones que usted considere conveniente.

Su solución deberá, además, incluir un menú inicial para que el usuario elija qué tipo de conversión desea realizar. A continuación, se muestran dos ejemplos de ejecución:

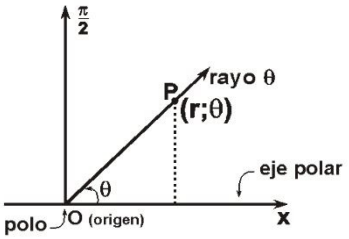


Figura 2.8: Punto en el sistema de coordenadas polar

```
Coordenadas Polares y Cartesianas
Ingrese 1 para convertir un punto de coordenadas cartesianas a polares
Ingrese 2 para convertir un punto de coordenadas polares a cartesianas
1
Eligió convertir un punto de coordenadas cartesianas a polares
Por favor, ingrese un punto X, Y: 12, 5
Resultado: P(13, 22.6)
```

```
Coordenadas Polares y Cartesianas
Ingrese 1 para convertir un punto de coordenadas cartesianas a polares
Ingrese 2 para convertir un punto de coordenadas polares a cartesianas
2
Eligió convertir un punto de coordenadas polares a cartesianas
Por favor, ingrese un punto r, zeta: 13, 22.6
Resultado: P(12, 5)
```

Sugerencia

Recuerde que la función *atan()* retorna el arcotangente de un argumento (ángulo en radianes). Usted deberá hacer las conversiones necesarias para la resolución de este problema.

2.3. Nivel avanzado

2.3.1. Números deficientes, abundantes, perfectos y semiperfectos

Un número natural es deficiente cuando la suma de sus divisores propios es menor que él. Se denomina divisor propio de un número natural n, a otro número también natural que es divisor de n, pero diferente de n. Por ejemplo:

- 10 es deficiente porque sus divisores propios, 1, 2 y 5 suman 8 que es menor que 10.

Un número natural es abundante cuando la suma de sus divisores propios es mayor que él. Por ejemplo:

- 12 es abundante porque sus divisores propios, 1, 2, 3, 4 y 6 suman 16 que es mayor que 12.

Un número natural es perfecto, cuando la suma de sus divisores propios es igual a él. Por ejemplo:

- 6 es perfecto porque sus divisores propios, 1, 2 y 3 suman 6 que es igual a 6.

Un número natural es semiperfecto, cuando la suma de algunos de sus divisores propios es igual a él. Para números naturales menores que 30, una forma para determinar si el número es semiperfecto es restar a la suma de divisores propios, uno de los divisores propios, si uno de esos resultados es igual al número, entonces se puede decir que el número es semiperfecto. Por ejemplo:

- 12 es semiperfecto porque algunos de sus divisores propios, 1, 2, 3 y 6 suman 12 que es igual a 12.

Los números perfectos son también semiperfectos.

Se le pide a usted que elabore un programa en ANSI C que, haciendo uso de la programación modular con estructuras algorítmicas selectivas e iterativas, realice lo siguiente:

- Realice la lectura de un número n $(0 < n < 30)$.
- Verifique que el número natural ingresado sea mayor que 0 y menor que 30. Si no está en el rango, muestra el mensaje `El valor ingresado no pertenece al rango 0<n<30` y el programa termina.
- Si el número ingresado pertenece al rango válido, imprima en pantalla si el número es deficiente, abundante o perfecto, si el número es también semiperfecto deberá indicarlo.

Su solución deberá incluir las siguientes funciones, además de la función `main`:

- `void leer_datos(int *n)` //Función que, mediante el paso de parámetro por referencia, lee el valor de n ingresado por teclado
- `int es_valido(int n)` //Función que verifica si el valor de n es mayor que 0 y menor que 30.
- `int sum_divisores_propios(int n)` //Función que retorna la suma de los divisores propios de n.
- `int es_deficiente(int n, int sum_div_propios)` //Función que evalúa si n es deficiente, y recibe como parámetros, el valor de n y el valor de la suma de divisores propios.
- `int es_abundante(int n, int sum_div_propios)` //Función que evalúa si n es abundante, y recibe como parámetros, el valor de n y el valor de la suma de divisores propios.
- `int es_perfecto(int n, int sum_div_propios)` //Función que evalúa si n es perfecto, y recibe como parámetros, el valor de n y el valor de la suma de divisores propios.
- `int es_semi_perfecto(int n, int sum_div_propios)` //Función que evalúa si n es semiperfecto, y recibe como parámetros, el valor de n y el valor de la suma de divisores propios.

Casos de prueba para verificación de solución

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

n	Impresión
30	No es un número válido.
6	6 es un número perfecto y semiperfecto.
10	10 es un número deficiente.
12	12 es abundante y semiperfecto.
18	18 es abundante y semiperfecto.
19	19 es un número deficiente.

2.3.2. Números felices e infelices

Los números, cómo las personas, pueden ser felices o infelices. Para determinar si un número es feliz se sigue el siguiente procedimiento: se suman los cuadrados de sus dígitos, y se repite el proceso cuantas veces sea necesario; si en algún momento obtenemos un 1, hemos terminado y entonces se puede decir que el número original es feliz :). En caso de que la cantidad de repeticiones del proceso sobrepase las 8 iteraciones, entonces se dice que el número original es infeliz :(. Por ejemplo:
7 es un número feliz, ya que:

- $7^2 = 49$
- $4^2 + 9^2 = 97$
- $9^2 + 7^2 = 130$
- $1^2 + 3^2 + 0^2 = 10$
- $1^2 + 0^2 = 1$

Si n no es feliz, la suma de los cuadrados luego de 8 iteraciones no llegaría a 1. Por ejemplo, 4 es infeliz, porque luego de 8 iteraciones no llega a 1:

$4, 16, 37, 58, 89, 145, 42, 20, 4, ...$

Algunos números felices de dos cifras son: 10, 13, 19, 23, ...

Cuando un número es feliz y a la vez es primo, se dice que es un primo feliz. Algunos primos felices de dos cifras son: 13, 19, 23, ...

Se le pide a usted que elabore un programa en el lenguaje C que, haciendo uso de la programación modular con estructuras algorítmicas selectivas e iterativas, determine si un número de dos cifras es feliz o infeliz; en el caso de ser primo, determine si es un primo feliz o infeliz. Su programa debe considerar lo siguiente:

- Realice la lectura de un número n de dos cifras.
- Verifique que el número natural ingresado sea de dos cifras.
- Imprima en pantalla si el número n es feliz o infeliz. Adicional a ello, si es feliz y primo se imprimirá que es un primo feliz, y si no es feliz y es primo, deberá imprimirse que es un primo infeliz.

Su solución deberá incluir las siguientes funciones, además de la función `main`:

- `void leer_datos(int *n)` //Función que lee el valor de n ingresado por teclado simulando el paso de parámetro por referencia.
- `int es_feliz(int n)` //Función que evalúa si n es feliz o no.
- `int sum_digits_al_cuadrado(int n)` //Función que retorna la suma de los cuadrados de los dígitos de n.
- `int cant_digitos(int n);` //Función que retorna la cantidad de dígitos de n, sabiendo que n a lo más puede tener 3 dígitos.
- `int es_primo(int n)` //Función que evalúa si n es primo.

Casos de prueba para verificación de solución

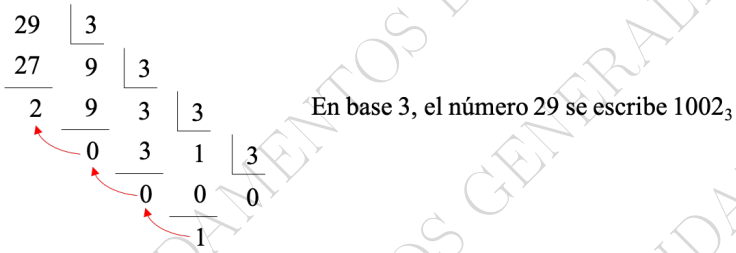
Use los siguientes casos para verificar si su solución es correcta.

n	Impresión
8	8 no es un número de dos cifras. (Y finaliza el programa)
10	10 es un número feliz.
11	11 es un primo infeliz.
12	12 es un número infeliz.
13	13 es un primo feliz.
94	94 es un número feliz.
97	97 es un primo feliz.
98	98 es un número infeliz.

2.3.3. Conversión de un número en base 10 a una base inferior

Para realizar la conversión de un número natural en base 10 a una base inferior se puede hacer mediante la técnica de divisiones sucesivas, en el que el número se divide entre la base destino, si el cociente entero no es cero, dicho cociente se vuelve a dividir entre la base destino, y así sucesivamente hasta lograr un cociente igual a cero. El número en la base destino se consigue concatenando las cifras de los residuos comenzando con el último residuo hasta el primer residuo. El número natural en base 10 a convertir debe ser menor a 1024 y la cantidad máxima de divisiones sucesivas es 10.

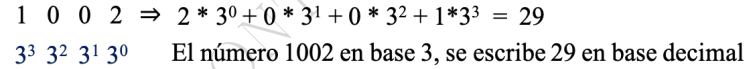
Por ejemplo: Si se desea convertir el número 29 en base 10 a base 3, a continuación se presenta la secuencia de divisiones sucesivas y la formación del nuevo número concatenando los residuos.



Para realizar la conversión de un número de base inferior a la base 10, se siguen los siguientes pasos:

1. Iniciamos por el lado derecho hasta el izquierdo del número en base inferior, cada cifra se multiplica por la base elevado a la potencia consecutiva (comenzando por la potencia 0, es decir; $base^0$).
2. Después de realizar cada una de las multiplicaciones, se suman todas y el número resultante será el equivalente al sistema decimal.

Por ejemplo: Si se desea convertir el número 1002 en base 3 a la base decimal, a continuación, se presenta la secuencia siguiendo los pasos del algoritmo previamente descrito.



Se le pide a usted que elabore un programa en ANSI C que, haciendo uso de la programación modular con estructuras algorítmicas selectivas e iterativas, realice lo siguiente:

- Realice la lectura del número a convertir.

```
Ingrese el número a convertir (0 <= número <= 1023):
```

- Luego realice la lectura de la base destino que tiene que ser menor que 10 y mayor que 1.

```
Ingrese la base destino (1 < base < 10):
```

- Realice la conversión del número en base 10 a la nueva base, debe validar que el número en base 10 sea mayor igual que 0 y menor igual que 1023, y también que la base destino debe ser menor que 10 y mayor que 1.
- Imprima en pantalla el número en la nueva base menor que 10.
- Inmediatamente después, realice la conversión del número obtenido previamente y que está en la nueva base, a la base decimal. Durante la conversión valide que el número obtenido previamente sea un número válido y que la base siga siendo mayor que 1 y menor que 10.
- Finalmente, imprima en pantalla el número en base decimal luego de la conversión.

Su solución deberá incluir las siguientes funciones, además de la función `main`:

- `void leer_datos(int *n, int *base)` //Función que lee el número n a convertir y la *base* destino, simulando el paso de parámetros por referencia.
- `int convertir_de_base_10_a_base_destino(int n, int base)` //Función que convierte el número n de base 10 a *base* destino. En el caso que el número n no está en el rango permitido, retorna -1. En el caso que la *base* sea incorrecta, retorna -2.
- `int convertir_de_base_origen_a_base_10(int n, int base)` //Función que convierte el número n de *base* origen a base 10. En el caso que el número ingresado sea -1 o -2 o la *base* sea incorrecta, retorna -1.

Casos de prueba para verificación de solución

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

Número a convertir	Base	Impresión
29	10	No se ha logrado convertir porque la base es mayor que 9 o es menor que 2.
1024	2	El número ingresado no está en el rango permitido.
29	3	El número 29 en base 10 tiene como equivalente a 1002 en base 3. Luego, el número 1002 en base 3 tiene como equivalente a 29 en base 10.
1023	2	El número 1023 en base 10 tiene como equivalente a 1111111111 en base 2. Luego, el número 1111111111 en base 2 tiene como equivalente a 1023 en base 10.
66	9	El número 66 en base 10 tiene como equivalente a 73 en base 9. Luego, el número 73 en base 9 tiene como equivalente a 66 en base 10.
69	2	El número 69 en base 10 tiene como equivalente a 1000101 en base 2. Luego, el número 1000101 en base 2 tiene como equivalente a 69 en base 10.
666	6	El número 666 en base 10 tiene como equivalente a 3030 en base 6. Luego, el número 3030 en base 6 tiene como equivalente a 666 en base 10.

2.3.4. Suma de números de máximo 3 cifras hasta la base 10

Los pasos para realizar la suma de dos números de varias cifras la conocemos todos desde el colegio. A continuación, se presentan algunos ejemplos de sumas de varias cifras en diferentes bases.

Base 10	666 +	666 +	666 +	Base 9	666_9 +	666_9 +	666_9 +
	66	6	666		66_9	6_9	666_9
	732	672	1332		743_9	673_9	1443_9
Base 8	666_8 +	666_8 +	666_8 +	Base 7	666_7 +	666_7 +	666_7 +
	66_8	6_8	666_8		66_7	6_7	666_7
	754_8	674_8	1554_8		1065_7	1005_7	1665_7

Se le pide a usted que elabore un programa en el lenguaje C que, haciendo uso de la programación modular con estructuras algorítmicas selectivas e iterativas, realice la suma de dos números de máximo 3 cifras en cualquier base menor o igual que 10 y mayor que 1. El procedimiento a seguir es el mismo que usted aplica para sumar dos números de varias cifras, sumando unidades con unidades, decenas con decenas, centenas con centenas, etc., y llevando una cifra cuando se tenga que llevar. En todo el programa NO ESTÁ PERMITIDO CONVERTIR A BASE 10 los números.

El programa debe considerar lo siguiente:

- Realice la lectura de la base de numeración, la base debe ser mayor que 1 y menor igual que 10.

  ```
  Ingrese la base (1 < base <= 10)):
  ```

- Realice la lectura del primer número.

  ```
  Ingrese el primer número:
  ```

- Realice la lectura del segundo número.

  ```
  Ingrese el segundo número:
  ```

- Si la base no es correcta o los números no son válidos, es decir, la cantidad de su cifras no corresponde a máximo 3 cifras, o sus cifras no corresponden al sistema de numeración permitidos por la base, se muestra un mensaje dependiendo de qué número es el incorrecto y el programa termina.

  ```
  Lo siento, error en la base. 0
  Lo siento, el primer número no es válido. 0
  Lo siento, el segundo número no es válido. 0
  Lo siento, los números no son válidos.
  ```

- Si la base y los números son válidos, se realiza la suma y se muestra el resultado en pantalla.

Su solución deberá incluir las siguientes funciones, además de la función `main`:

- `void leer_datos(int *base, int *n1, int *n2)` //Función que lee la *base* menor igual que 10 y mayor que 1, y los números n1 y n2, ambos de la misma *base*.
- `int es_valido(int base, int n)` //Función que valida si el número el número n es de máximo 3 cifras y está bien escrito en la base dada por *base*.
- `int suma(int base, int n1, int n2)` //Función que suma dos números en la base dada por *base*.

Casos de prueba para verificación de solución

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

Base	Núm. 1	Núm. 2	Impresión
11	666	777	Lo siento, error en la base. (Y finaliza el programa)
3	33	2	Lo siento, el primer número no es válido. (Y finaliza el programa)
9	666	99	Lo siento, el segundo número no es válido. (Y finaliza el programa)
8	8	99	Lo siento, los números no son válidos. (Y finaliza el programa)
9	666	666	La suma de 666 y 666 en base 9 da como resultado 1443 en base 9.
7	6	66	La suma de 6 y 66 en base 7 da como resultado 105 en base 7.
8	666	66	La suma de 666 y 66 en base 8 da como resultado 754 en base 8.
7	666	66	La suma de 666 y 66 en base 7 da como resultado 1065 en base 7.
3	1002	1002	La suma de 1002 y 1002 en base 3 da como resultado 2011 en base 3.

2.3.5. Resta de números de máximo 3 cifras hasta la base 10

Los pasos para realizar la resta o sustracción de dos números de varias cifras la conocemos todos desde el colegio. A continuación, se presentan algunos ejemplos de restas de varias cifras en diferentes bases.

Base 10	$654 - 66 = 588$	$654 - 6 = 648$	$654 - 646 = 8$	Base 9	$654_9 - 66_9 = 577_9$	$654_9 - 6_9 = 647_9$	$654_9 - 646_9 = 7_9$
Base 8	$654_8 - 66_8 = 566_8$	$654_8 - 6_8 = 646_8$	$654_8 - 646_8 = 6_8$	Base 7	$654_7 - 66_7 = 555_7$	$654_7 - 6_7 = 645_7$	$654_7 - 646_7 = 5_7$

Se le pide a usted que elabore un programa en el lenguaje C que, haciendo uso de la programación modular con estructuras algorítmicas selectivas e iterativas, realice la resta de dos números de máximo 3 cifras en cualquier base menor o igual que 10 y mayor que 1. El procedimiento a seguir es el mismo que usted aplica para restar dos números de varias cifras, restando unidades con unidades, decenas con decenas, centenas con centenas, etc., y prestando una cifra cuando el minuendo es menor que el sustraendo. En todo el programa NO ESTÁ PERMITIDO CONVERTIR A BASE 10 los números.

El programa debe considerar lo siguiente:

- Realice la lectura de la base de numeración, la base debe ser mayor que 1 y menor igual que 10.

  ```
  Ingrese la base (1 < base <= 10):
  ```

- Realice la lectura del primer número.

  ```
  Ingrese el primer número:
  ```

- Realice la lectura del segundo número.

  ```
  Ingrese el segundo número:
  ```

- Si la base ingresada no es válida, se muestra un mensaje y termina el programa. Si los números no son válidos, es decir, la cantidad de sus cifras excede a 3 o sus cifras no corresponden al sistema de numeración permitidos por la base, se muestra un mensaje dependiendo de qué número es el incorrecto y el programa termina. También debe validar que el primer número sea mayor que el segundo número. Los posibles mensajes son:

```
Lo siento, la base es incorrecta. 0
Lo siento, el primer número no es válido. 0
Lo siento, el segundo número no es válido. 0
Lo siento, los números no son válidos. 0
Lo siento, el primer número debe ser mayor que el segundo número.
```

- Si la base y los números son válidos, se realiza la resta y se muestra el resultado en pantalla.

Su solución deberá incluir las siguientes funciones, además de la función `main`:

- `void leer_datos(int *base, int *n1, int *n2)` //Función que lee la *base* menor igual que 10 y mayor que 1, y los números n1 y n2, ambos de la misma *base*.
- `int es_de_3_digitos_y_esta_en_su_base_correcta(int n, int base)` //Función que valida si el número n es de máximo 3 cifras y está bien escrito en la base dada por *base*.
- `int resta(int base, int n1, int n2)` //Función que resta dos números en la base dada por *base*.

Casos de prueba para verificación de solución

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

Base	Núm. 1	Núm. 2	Impresión
11	6	7	Lo siento, la base es incorrecta. (Y el programa finaliza)
9	76	69	Lo siento, el segundo número no es válido. (Y el programa finaliza)
8	758	99	Lo siento, los números no son válidos. (Y el programa finaliza)
7	66	666	Lo siento, el primer número tiene que ser mayor que el segundo número.
4	1001	1	Lo siento, el primer número no es válido. (Y el programa finaliza)
4	100	1001	Lo siento, el segundo número no es válido. (Y el programa finaliza)
9	654	66	La resta de 654 y 66, ambas en base 9, da como resultado 577.
8	654	66	La resta de 654 y 66, ambas en base 8, da como resultado 566.
7	654	66	La resta de 654 y 66, ambas en base 7, da como resultado 555.
3	21	12	La resta de 21 y 12, ambas en base 3, da como resultado 2.

2.3.6. Límite de una función mediante aproximaciones

El límite de una función f(x), cuando x tiende a c es L si y sólo si para todo $\varepsilon > 0$, existe un $\delta > 0$ tal que para todo número real x en el dominio de la función, si $0 < |x - c| < \delta$ entonces $|f(x) - L| < \varepsilon$.

Esto es:

$$\lim_{x \to c} f(x) = L \iff \forall \varepsilon > 0,\ \exists \delta > 0 \,|\, \forall x \in \mathrm{Dom}(f),\ 0 < |x - c| < \delta \longrightarrow |f(x) - L| < \varepsilon$$

Mediante aproximaciones sucesivas se desea hallar el límite para la siguiente función cuando x tiende a 2.

$$\lim_{x \to 2} \frac{2x^2 - 4x}{x - 2}$$

Si se considera un δ mínimo de 0.001 para el cálculo del límite y se desea hallar dicho límite en 5 aproximaciones, en la figura 2.10 se presenta una tabla con los datos por cada aproximación.

Se le pide a usted que elabore un programa en el lenguaje C que, haciendo uso de la programación modular con estructuras algorítmicas selectivas e iterativas, realice lo siguiente:

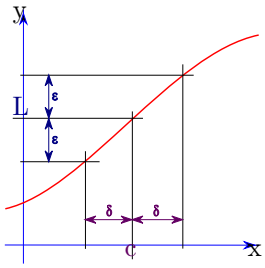


Figura 2.9: Definición de límite. Fuente: https://upload.wikimedia.org/wikipedia/commons/d/d1/Límite_01.svg

	1° aprox. c-16*δ	2° aprox. c - 8*δ	3° aprox. c - 4*δ	4° aprox. c - 2*δ	5° aprox. c - δ	c	5° aprox. c + δ	4° aprox. c + 2*δ	3° aprox. c + 4*δ	2° aprox. c + 8*δ	1° aprox. c+16*δ
x	1.984	1.992	1.996	1.998	1.999	2	2.001	2.002	2.004	2.008	2.016
y =f(x)	3.968	3.984	3.992	3.996	3.998	y	4.002	4.004	4.008	4.016	4.032

$$\frac{3.998 + 4.002}{2}$$

Figura 2.10: Tabla de aproximaciones para el cálculo del límite cuando x tiende a 2, con un δ mínimo de 0.001.

- Realice la lectura del valor al que tiende x (c), la cantidad de aproximaciones (n) para hallar el límite $(0 < n)$, y el valor del δ mínimo (d) para la aproximación final.
- Realice las n aproximaciones, y por cada aproximación (i) muestre los valores del número de aproximación i, $2^{n-i}\delta$, $c - 2^{n-i}\delta$, $f(c - 2^{n-i}\delta)$, $c + 2^{n-i}\delta$ y $f(c + 2^{n-i}\delta)$
- Solo si el valor de la diferencia entre $f(c + 2^{n-i}\delta)$ y $f(c - 2^{n-i}\delta)$ decrece o se mantienen iguales en todas las aproximaciones (se aproximan los respectivos $f(x)$), obtenga el límite de la función mediante el promedio de ambas y muestre dicho límite de la función cuando x tiende a c. En el caso que no decrezca en todas las aproximaciones muestre el mensaje que es probable que no haya un único límite de la función cuando x tiende a c.

Su solución deberá incluir las siguientes funciones, además de la función `main`:

- `void leer_datos(double *c, int *num_aproximaciones, double *delta_min)` //Función que, simulando el paso de parámetros por referencia, lee el valor de c, el número de aproximaciones y el delta mínimo.
- `void calcular_limite_funcion_cuando_x_tiende_a(double c, int n, double d, double *limite, int *se_aproxima)`
  /*Función que calcula el límite de una función dada por aproximaciones sucesivas y muestra dichas aproximaciones. Recibe cinco parámetros, los 3 primeros son por valor, y los 2 últimos simula el paso de parámetros por referencia. El primer parámetro es el valor de c, el segundo parámetro es la cantidad de aproximaciones n, el tercer parámetro es el delta mínimo d, mediante el cuarto parámetro se puede obtener el valor del límite solo si la diferencia de los f(x) en cada aproximación va decreciendo o se mantienen iguales, y mediante el último parámetro se obtiene si se aproxima o no a un único valor el valor de f(x) cuando x tiende a c. Por cada aproximación se imprime en pantalla los siguientes valores: número de aproximación (i), $2^{n-i}d$, $c - 2^{n-i}d$, $f(c - 2^{n-i}d)$, $c + 2^{n-i}d$ y $f(c + 2^{n-i}d)$ */
- `double funcion(double x)` //Función que calcula el valor de la función $f(x) = (2x^2 - 4x)/(x - 2)$ para un determinado x diferente de 2.

Casos de prueba para verificación de solución

Use los siguientes casos para verificar si su solución es correcta.

c	Núm. aprox. (n)	$\delta mín$ (d)	Impresión
2	5	0.001	Aprx.1: 16d=0.016, c-16d=1.984, f(1.984)=3.968, c+16d=2.016, f(2.016)=4.032 Aprx.2: 8d=0.008, c-8d=1.992, f(1.992)=3.984, c+8d=2.008, f(2.008)=4.016 Aprx.3: 4d=0.004, c-4d=1.996, f(1.996)=3.992, c+4d=2.004, f(2.004)=4.008 Aprx.4: 2d=0.002, c-2d=1.998, f(1.998)=3.996, c+2d=2.002, f(2.002)=4.004 Aprx.5: 1d=0.001, c-1d=1.999, f(1.999)=3.998, c+1d=2.001, f(2.001)=4.002 Como la diferencia de los f(x) en cada aproximación decrece o se mantienen iguales, entonces el valor del límite de la función cuando x tiende a 2.000 es 4.000.

2.3.7. Número de pasos para llegar a la constante de Kaprekar de 3 cifras

Si tomamos cualquier número de 3 cifras (que no sean todas iguales) y generamos el número más grande que se pueda formar con esas cifras, y el más pequeño, y los restamos, y hacemos este proceso repetidas veces con el número resultante, después de 6 o menos rondas de restas, siempre llegaremos al número 495. Al llegar a este punto, cada vez que volvamos a repetir este proceso, siempre obtendremos 495, ya que el número más grande posible que se puede generar con sus cifras es el 954, y el menor es el 459, y al restarlos, da 495 otra vez.

$$954 - 459 = 495$$

Cabe resaltar que, en el caso que después de realizar una resta se obtenga un número menor a 100, se debe completar con ceros a la izquierda hasta tener 3 dígitos. Por ejemplo, si partimos del número 121, el máximo número posible es 211 y el mínimo es 112, y al restar 211 - 112 se obtiene 99, por lo que al momento de generar las nuevas parejas de números, se deberá interpretar ese 99 como si fuera un 099 de modo que el mayor número que se pueda formar con esas cifras sea el 990 y el menor sea el 99.

A este número 495 se le conoce como la Constante de Kaprekar para números de 3 cifras, pues, fue Dattatreya Ramchandra Kaprekar (1905-1986) quien descubrió un comportamiento similar para los números de 4 cifras que siempre llegan a 6174 y lo presentó en la Conferencia Matemática de Madrás en 1949.

Se le pide:
Elaborar un programa en lenguaje C usando el paradigma de programación modular, que permita ingresar un número de 3 cifras, y verifique que se puede llegar a 495 en 6 pasos o menos. En su solución, no será necesario que usted valide que el número ingresado sea de 3 cifras o que no tenga 3 dígitos iguales. Es decir usted deberá asumir que el usuario siempre ingresa números de 3 cifras, y deberá asumir que nunca ingresa números de 3 dígitos iguales (tales como el 111, el 222, el 333, entre otros).

Para cualquier número de 3 cifras ingresado por teclado, su programa deberá mostrar un mensaje indicando el número de pasos que necesitó para llegar a 495, así como también un mensaje indicando si se cumplió la premisa de que se requieren 6 pasos o menos.

A tener en cuenta

Si usted desea analizar el número 121, su análisis deberá consistir en los siguientes pasos:

- Paso 1: 211 - 112 = 99
- Paso 2: 990 - 99 = 891
- Paso 3: 981 - 189 = 792
- Paso 4: 972 - 279 = 693
- Paso 5: 963 – 369 = 594
- Paso 6: 954 – 459 = 495

Se dice que el número 121 necesita 6 pasos para llegar a 495, debido a que a partir del paso 7 en adelante, siempre se obtendrá nuevamente el valor 495 de manera repetida.

Su programa deberá incluir:

- La función main.
- Una función que permita extraer la unidad, decena y centena de un número de 3 cifras (simular el paso de parámetros por referencia enviando las direcciones de las variables de su función main).
- Desde la función main, deberá ordenar las unidades, decenas y centenas de menor a mayor en tres variables nuevas llamadas cifra_menor, cifra_medio y cifra_mayor.
- Una función que genere el máximo número posible y el mínimo número posible a partir de 3 cifras (deberá generar ambos valores en una sola función de tipo void).
- Una función que reciba un número entero de pasos y verifique si es que dicho número es menor o igual a 6.

Casos de prueba para verificación de solución

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

Número	Pasos	Impresión
121	6	El número 121 requiere 6 pasos para llegar a 495. 6 es menor o igual a 6.
279	3	El número 279 requiere 3 pasos para llegar a 495. 3 es menor o igual a 6.

Dado que usted solo sabe usar estructuras iterativas con entrada controlada, controladas por contador, usted deberá siempre hacer 6 iteraciones. Para el caso de números que requieran menos de 6 pasos para llegar a 495, se obtendrán repetidas veces 495 en los pasos adicionales. Por ejemplo, el número 279 requiere solo 3 pasos para llegar a 495, y desde el paso 4 hasta el paso 6 se volverá a obtener el mismo valor:

- Paso 1: 972 - 279 = 693
- Paso 2: 963 – 369 = 594
- Paso 3: 954 – 459 = 495
- Paso 4: 954 – 459 = **495** (valor repetido)
- Paso 5: 954 – 459 = **495** (valor repetido)
- Paso 6: 954 – 459 = **495** (valor repetido)

Su solución deberá contar cuántos pasos se están realizando hasta obtener el número 495 por primera vez. Cuando una iteración genere el 495 por primera vez, usted deberá idear un mecanismo para salvaguardar el valor del contador de pasos.

Tenga en cuenta que en la siguiente iteración se volverá a obtener 495 por lo que su mecanismo no puede consistir en validar si la diferencia del máximo número generado menos el mínimo número generado es igual a 495 (en nuestro ejemplo, esto daría verdadero en los pasos 3, 4, 5 y 6, pero usted solo desea salvar el 3, debido a que el paso 3 es el primer paso en el que se obtuvo 495). Usted deberá implementar un mecanismo que le permita conocer cuál es el número de pasos necesario para llegar a 495 por primera vez.

Artículo tomado de: https://www.bbc.com/mundo/noticias-49426284
El misterioso número 6174 que ha intrigado a matemáticos durante 70 años
Dalia Ventura
BBC News Mundo

2.3.8. Número Lambda

Se conoce como números lambda (λ) a los términos de la sucesión: 0, 1, 2, 5, 12, 29, 70, 169, 408, 985, 2378, 5741, 13860, 33461, 80782, 195025, 470832, 1136689, 3215042, 7761798, 18738638, 45239074, 109216786, 263672646, 636562078, 1536796802, ..., que se obtienen mediante la siguiente relación de recurrencia:

$$\lambda_n = \begin{cases} 0 & \text{si } n = 0; \\ 1 & \text{si } n = 1; \\ 2\lambda_{n-1} + \lambda_{n-2} & \text{otros casos.} \end{cases}$$

Donde, la secuencia de números λ comienza con 0 y 1 si n es 0 o 1 respectivamente, y luego cada número lambda (λ_n) se obtiene a partir de los dos anteriores, sumando dos veces el número lambda anterior (λ_{n-1}) y el número lambda previo a este (λ_{n-2}) .

Un número lambda de n (λ_n) también se puede obtener mediante el redondeo al resultado de la siguiente fórmula:

$$\lambda_n = \frac{\left(1+\sqrt{2}\right)^n - \left(1-\sqrt{2}\right)^n}{2\sqrt{2}}$$

Se le pide a usted que elabore un programa en el lenguaje C que, haciendo uso de la programación modular con estructuras algorítmicas selectivas e iterativas, realice lo siguiente:

- Realice la lectura de un número n ($0 \leq n \leq 25$).
- Calcule el número lambda λ_n utilizando la primera definición recurrente.
- Calcule el número lambda λ_n utilizando la segunda fórmula. Para realizar el redondeo del número, use la función round. En el lenguaje C la declaración de la función round se encuentra en el archivo de cabecera math.h.
- Muestre ambos números obtenidos.
- En el caso que n no pertenezca al rango permitido ($0 \leq n \leq 25$), se debe mostrar en pantalla el mensaje *"n debe ser mayor igual que 0 y menor igual que 25."*

Su solución deberá incluir las siguientes funciones, además de la función `main`:

- `void leer_datos(int *n)` //Función que, simulando el paso de parámetros por referencia, lee del teclado el valor de n.

- `int numero_lambda(int n)` //Función que obtiene el número lambda de n mediante la relación de recurrencia. Retorna -1 si n no es válido.
- `int numero_lambda_con_formula(int n)` //Función que obtiene el número lambda de n aplicando la fórmula. Retorna -1 si n no es válido.

Casos de prueba para verificación de solución

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

n	Impresión
-1	n debe ser mayor igual que 0 y menor que 25.
26	n debe ser mayor igual que 0 y menor que 25.
0	Número lambda (0): 0 El número lambda (0) usando la fórmula es 0
1	Número lambda (1): 1 El número lambda (1) usando la fórmula es 1
2	Número lambda (2): 2 El número lambda (2) usando la fórmula es 2
3	Número lambda (3): 5 El número lambda (3) usando la fórmula es 5
5	Número lambda (5): 29 El número lambda (5) usando la fórmula es 29
10	Número lambda (10): 2378 El número lambda (10) usando la fórmula es 2378
25	Número lambda (25): 1311738121 El número lambda (25) usando la fórmula es 1311738121

2.3.9. Sumatoria de primeros números compañeros de Pell

Se conoce como los *números compañeros de Pell* (Q_n) a los términos de la sucesión: 2, 2, 6, 14, 34, 82, 198, 478, 1154, 2786, 6726, 16238, 39202, 94642, 228486, 551614, 1331714, 3215042, 7761798, 18738638, 45239074, 109216786, 263672646, 636562078, 1536796802, ..., que se obtienen mediante la siguiente relación de recurrencia:

$$Q_n = \begin{cases} 2 & \text{si } n = 0; \\ 2 & \text{si } n = 1; \\ 2Q_{n-1} + Q_{n-2} & \text{en otros casos.} \end{cases}$$

Donde, la secuencia de números compañeros de Pell (Q_n) comienza con 2 y 2 si n es 0 o 1 respectivamente, luego cada número Q_n se obtiene a partir de los dos anteriores, sumando dos veces el número compañero de Pell anterior (Q_{n-1}) y el número compañero de Pell previo a éste (Q_{n-2}) .

Un número compañero de Pell de n (Q_n) también se puede obtener mediante el redondeo al resultado de la siguiente fórmula:

$$Q_n = \left(1 + \sqrt{2}\right)^n + \left(1 - \sqrt{2}\right)^n$$

Se le pide a usted que elabore un programa en el lenguaje C que, haciendo uso de la programación modular con estructuras algorítmicas selectivas e iterativas, muestre la suma de los n primeros números compañeros de Pell. El programa debe realizar lo siguiente:

- Realice la lectura del número n que será la cantidad de los primeros n números compañeros de Pell que se deben sumar $(0 < n < 25)$.

- Calcule la suma de los primeros n números compañeros de Pell utilizando la primera definición recurrente.
- Calcule la suma de los primeros n números compañeros de Pell utilizando la segunda fórmula. Para realizar el redondeo del número, use la función round. En el lenguaje C la declaración de la función round se encuentra en el archivo de cabecera math.h.
- Muestre ambas sumas obtenidas.
- En el caso que n no pertenezca al rango permitido ($0 < n < 25$), se debe mostrar en pantalla el mensaje *"El valor ingresado de n es incorrecto."*

Su solución deberá incluir las siguientes funciones, además de la función `main`:

- `void leer_datos(int *n)` //Función que, simulando el paso de parámetros por referencia, lee del teclado el valor de n.
- `int sumatoria_primeros_numeros_companheros_pell(int n)` //Función que obtiene la suma de los primeros n números compañeros de Pell mediante la relación de recurrencia. Retorna -1 si n no es válido.
- `int sumatoria_primeros_numeros_companheros_pell_aplicando_formula(int n)` //Función que obtiene la suma de los primeros n números compañeros de Pell aplicando la fórmula. Retorna -1 si n no es válido.

Casos de prueba para verificación de solución

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

n	Impresión
0	El valor de n ingresado es incorrecto.
25	El valor de n ingresado es incorrecto.
1	La suma del primer número compañero de Pell es 2 La suma del primer número compañero de Pell usando la fórmula es 2
2	La suma de los primeros 2 números compañeros de Pell es 4 La suma de los primeros 2 números compañeros de Pell usando la fórmula es 4
3	La suma de los primeros 3 números compañeros de Pell es 10 La suma de los primeros 3 números compañeros de Pell usando la fórmula es 10
4	La suma de los primeros 4 números compañeros de Pell es 24 La suma de los primeros 4 números compañeros de Pell usando la fórmula es 24
5	La suma de los primeros 5 números compañeros de Pell es 58 La suma de los primeros 5 números compañeros de Pell usando la fórmula es 58
10	La suma de los primeros 10 números compañeros de Pell es 4756 La suma de los primeros 10 números compañeros de Pell usando la fórmula es 4756
24	La suma de los primeros 24 números compañeros de Pell es 1086679440 La suma de los primeros 24 números compañeros de Pell usando la fórmula es 1086679440

2.3.10. Validación de una contraseña numérica

Cuando se intenta acceder por primera vez al sistema web de un determinado banco, primero debe registrar un usuario y una contraseña (password) numérica de 6 dígitos. Se le ha encomendado desarrollar un programa que valide la contraseña a registrar por el usuario con las siguientes reglas:

- Debe tener solo 6 cifras.
- Las cifras no deben ser repetidas.
- No deben haber ningún cero en la contraseña.

- El valor de la primera cifra no debe ser la menor de todas las demás cifras. Es decir, el menor dígito no puede estar en la primera cifra comenzando por la izquierda.

Si se cumplen todas las condiciones, la contraseña es válida, caso contrario, no la es. Por ejemplo:

- 45024 no es una contraseña válida porque solo tiene 5 cifras.
- 333333 no es una contraseña válida porque todas sus cifras son iguales.
- 435702 no es una contraseña válida porque hay un cero en una de sus cifras.
- 135214 no es una contraseña válida porque el menor dígito se encuentra en dos posiciones, y una de ellas es la 1era posición.
- 465982 es una contraseña válida.

Se le pide a usted desarrollar un programa en el lenguaje C que, haciendo uso de la programación modular con estructuras algorítmicas selectivas e iterativas, valide que la contraseña o password ingresado sea válido. Además debe mostrar si tiene 6 cifras, si son todas sus cifras iguales, si contiene ceros, y presentar al dígito menor con su respectiva posición comenzando por la izquierda. De encontrarse el dígito menor repetido, se muestra la posición de más a la izquierda. El programa debe considerar lo siguiente:

- Realice la lectura de una contraseña numérica de 6 cifras. Tenga en cuenta que si ingresa una contraseña que inicia con cero, dicho cero no será considerado como una cifra de la contraseña leída.
- Muestre si tiene o no 6 cifras.
- Si es de 6 cifras y todas sus cifras son iguales, muestre "Todas sus cifras son iguales".
- Si es de 6 cifras y contiene al cero en alguna de sus cifras, muestre "Contiene cero(s)".
- Solo si es de 6 cifras y el dígito menor se encuentra en la primera cifra, muestre el dígito menor indicando que está en la primera cifra de la contraseña.
- Verifique que la contraseña ingresada sea válida. Si cumple con todas las reglas previamente descritas, se dice que es válida y muestra el mensaje "El password ingresado es válido", caso contrario, se muestra el mensaje "Lo sentimos, el password ingresado no es válido".

Su solución deberá incluir las siguientes funciones, además de la función `main`:

- `void leer_datos(int *password)` //Función que, simulando el paso de parámetro por referencia, lee el valor de la contraseña o *password* ingresado por teclado.
- `int es_de_seis_cifras(int password)` //Función que verifica si el *password* es de 6 cifras.
- `int tiene_ceros(int password)` //Función que verifica si el *password* contiene al 0 en alguna de sus cifras.
- `int todas_cifras_iguales(int password)` //Función que verifica si el *password* tiene todas sus cifras iguales.
- `void menor_digito(int password, int *min_digit, int *pos_min_digit` //Función que, dado el *password*, retorna su dígito menor y la posición en el que se encuentra. Si hay repetidos, retorna la posición del que está más a la izquierda. El *password* ya tiene 6 cifras.

Casos de prueba para verificación de solución

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

Password	Impresión
45024	No tiene 6 cifras. Lo sentimos, el password ingresado no es válido.
333333	Tiene 6 cifras. Todas sus cifras son iguales. El dígito menor es 3 y está en la posición 1. Lo sentimos, el password ingresado no es válido.
435702	Tiene 6 cifras. Contiene cero(s). Lo sentimos, el password ingresado no es válido.
135214	Tiene 6 cifras. El dígito menor es 1 y está en la primera posición. Lo sentimos, el password ingresado no es válido.
465982	Tiene 6 cifras. El password ingresado es válido.

2.3.11. Misteriosos números múltiplos de 9

Para todo número de 2 cifras, se sabe que si al mayor número posible formado por dichas cifras se le resta el menor número posible formado por dichas cifras, siempre se obtiene un múltiplo de 9 (incluyendo el cero).

Se le pide implementar un programa en lenguaje C que solicite un rango de dos números a y b, ambos de 2 cifras, y compruebe que todos los números dentro de dicho rango (incluyendo los límites del rango) cumplan con esta condición. Para cada número procesado, su programa deberá mostrar la decena, la unidad, la cifra mayor, la cifra menor, el máximo número generado, el mínimo número generado, la diferencia del máximo número generado menos el mínimo número generado, y el factor que hace posible que esta diferencia corresponda a un número múltiplo de 9. Por ejemplo, para el número 24, se tiene que la decena es 2, la unidad es 4, la cifra mayor es 4, la cifra menor es 2, el máximo número generado es 42, el menor número generado es 24, la diferencia entre el máximo y el mínimo es 18, y el factor que hace posible que 18 sea múltiplo de 9 es el número 2 (ya que 9 x 2 = 18).

Su solución deberá indicar si el número evaluado cumple con esta condición o no. Asimismo, deberá contar cuántos números de su rango (incluyendo los límites del rango) cumplieron con la condición, cuántos no, y cuál es el porcentaje de números que cumplieron con esta condición respecto al total de números evaluados. Debe asumir que el usuario siempre ingresa un rango válido (es decir, usted no debe validar que a y b tengan 2 digitos ni tampoco debe validar que b sea mayor o igual que a). Recuerde que para poder imprimir el símbolo «%» como parte de la cadena de salida de un *printf* usted debe utilizar la combinación %% (doble porcentaje).

Su solución debe incluir las siguientes funciones:

- La función main.
- Una función que permita extraer la decena y la unidad de un número de 2 cifras (deberá simular el paso de parámetros por referencia).
- Una función que permita generar el mayor y menor número posible a partir de 2 cifras (deberá simular el paso de parámetros por referencia).
- Una función que valide si un número es múltiplo de 9.
- Una función que retorne el factor que hace posible que un número sea múltiplo de 9 (por ejemplo, el factor que hace que el 18 sea múltiplo de 18 es 2, ya que 2 x 9 = 18).

- Una función que reciba una frecuencia y un total, y calcule el valor del porcentaje que representa dicha frecuencia respecto al total. Por ejemplo, si la frecuencia es 3 y el total es 8, esta fucción retornará 37.5, pues 3 es el 37.5 % de 8 (note que esta función retorna solo el valor 37.5 sin el símbolo de porcentaje).

Casos de prueba para verificación de solución

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

a	b	Impresión
71	74	Número 71: decena = 7, unidad = 1, mayor = 7, menor = 1 máx = 71, mín = 17, dif = 54. Cumple condición (9x6=54) Número 72: decena = 7, unidad = 2, mayor = 7, menor = 2 máx = 72, mín = 27, dif = 45. Cumple condición (9x5=45) Número 73: decena = 7, unidad = 3, mayor = 7, menor = 3 máx = 73, mín = 37, dif = 36. Cumple condición (9x4=36) Número 74: decena = 7, unidad = 4, mayor = 7, menor = 4 máx = 74, mín = 47, dif = 27. Cumple condición (9x3=27) Se encontraron 4 números que cumplen con la condición Se encontraron 0 números que no cumplen la condición Porcentaje de números que cumplen la condición: 100.00 %
18	24	Número 18: decena = 1, unidad = 8, mayor = 8, menor = 1 máx = 81, mín = 18, dif = 63. Cumple condición (9x7=63) Número 19: decena = 1, unidad = 9, mayor = 9, menor = 1 máx = 91, mín = 19, dif = 72. Cumple condición (9x8=72) Número 20: decena = 2, unidad = 0, mayor = 2, menor = 0 máx = 20, mín = 2, dif = 27. Cumple condición (9x2=18) Número 21: decena = 2, unidad = 1, mayor = 2, menor = 1 máx = 21, mín = 12, dif = 9. Cumple condición (9x1=9) Número 22: decena = 2, unidad = 2, mayor = 2, menor = 2 máx = 22, mín = 22, dif = 0. Cumple condición (9x0=0) Número 23: decena = 2, unidad = 3, mayor = 3, menor = 2 máx = 32, mín = 23, dif = 9. Cumple condición (9x1=9) Número 24: decena = 2, unidad = 4, mayor = 4, menor = 2 máx = 42, mín = 24, dif = 18. Cumple condición (9x2=18) Se encontraron 7 números que cumplen con la condición Se encontraron 0 números que no cumplen la condición Porcentaje de números que cumplen la condición: 100.00 %

2.3.12. Número Harshad

Un Número de Harshad es aquel número que es divisible por la suma de sus dígitos. Algunos números que cumplen con esta condición son: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 18, 20, 21,24, 27, 30, 36, 40, 42, 45, 48, 50, 54, 60, 63, 70, 72, 80, 81, 84, 90, 100, 102, 108, 110, 111, 112, 114, 117, 120, 126, 132, 133, 135, 140, 144, 150, 152, 153, 156, 162, 171, 180, 190, 192, 195, 198, 200, 201, 204, entre otros.

Es muy común creer que todos los múltiplos de 9, son también números Harshad, debido a la presencia de números como el 9, el 18, el 27, el 36, el 45, el 54, el 63, el 72 o el 81. Sin embargo esto no es así (por ejemplo, el número 99 no es divisible entre 18, resultado de sumar 9+9).

Se le pide implementar un programa en lenguaje C que solicite al usuario un número entero positivo de máximo 10 dígitos, y evalúe si se trata de un número Harshad o no, y si es múltiplo de 9 o no. Deberá imprimir los mensajes correspondientes ante cada validación (es Harshad, no es Harshad, es múltiplo de 9, no es múltiplo de 9). Para el caso de encontrar algún número múltiplo de 9 que no sea Harshad, su programa además deberá imprimir el mensaje "Sorprendente!".

Finalmente, debe multiplicar todas las cifras del número y luego restar todas sus cifras, dicho resultado también debe mostrarse.

Su solución deberá incluir las siguientes funciones:

- La función main.
- Una función que, simulando el paso de parámetro por referencia, lea el número ingresado por teclado del usuario.
- Una función que valide si un número es Harshad o no.
- Una función que sume todos los dígitos de un número, que funcione para cualquier número de n cifras, donde $0 < n \leq 10$.
- Una función que valide si un número es múltiplo de 9 o no.
- Una función que multiplique y luego reste todas las cifras de cualquier número de n cifras, donde $0 < n \leq 10$.

Casos de prueba para verificación de solución

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

Número ingresado	Impresión
89	89 no es Harshad. No es múltiplo de 9. Si se multiplican todas sus cifras y luego se restan, se obtiene 55.
90	90 es Harshad. Es múltiplo de 9. Si se multiplican todas sus cifras y luego se restan, se obtiene -9.
97	97 no es Harshad. No es múltiplo de 9. Si se multiplican todas sus cifras y luego se restan, se obtiene 47.
99	99 no es Harshad. Es múltiplo de 9. ¡Sorprendente! Si se multiplican todas sus cifras y luego se restan, se obtiene 63.
100	100 es Harshad. No es múltiplo de 9. Si se multiplican todas sus cifras y luego se restan, se obtiene -1.
111	111 es Harshad. No es múltiplo de 9. Y además es capicúa! Si se multiplican todas sus cifras y luego se restan, se obtiene -2.
171	171 es Harshad. No es múltiplo de 9. Y además es capicúa! Si se multiplican todas sus cifras y luego se restan, se obtiene -2.

2.3.13. Trabajo realizado para extraer el líquido de un cilindro (adaptado del laboratorio 3 del ciclo 2020-2)

(Adaptado de [10])

En la figura 2.11 se muestra un recipiente cilíndrico con base circular de radio r m y altura H m que contiene un líquido. Si el líquido en el interior del recipiente tiene una altura de L m, ¿cuál es el trabajo requerido en

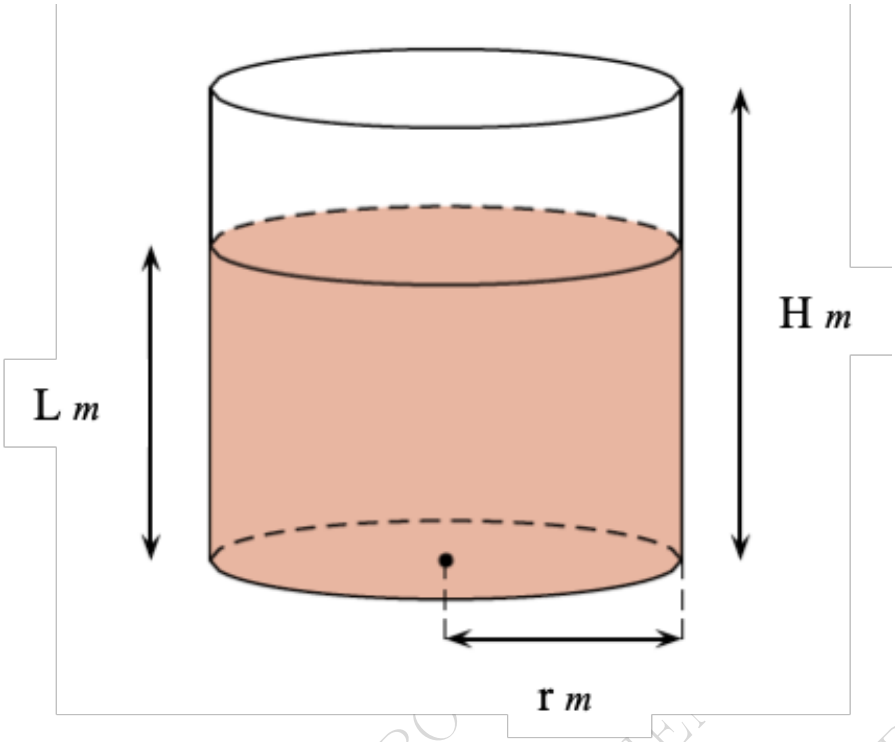


Figura 2.11: Recipiente con un fluido en su interior.
Fuente: http://canek.uam.mx/Integral/Cap03.Aplicaciones/3.5.Trabajo/FTTrabajo.pdf

kJ para bombear todo el líquido hasta la parte superior del recipiente? Considere que la densidad del líquido es $\rho\ kg/m^3$.

Para resolver el problema se deberá tener en cuenta un conjunto de n capas de líquido, cada una con un mismo grosor Δy, como se muestra en la figura 2.12 en la cual se resalta la i-ésima capa de líquido a una distancia y_i^* con respecto de la parte superior del recipiente. El origen del eje vertical será la parte superior del recipiente con el sentido positivo hacia abajo.

Si el número n de capas tiende a ser muy grande, entonces el grosor Δy de cada capa tiende a ser muy pequeño. Para este caso, el trabajo W_i que realiza la fuerza F_i al elevar hasta la parte superior del recipiente la i-ésima capa es

$$W_i = F_i y_i^*$$

La fuerza F_i debe ser igual a la fuerza de atracción gravitacional que actúa sobre la i-ésima capa:

$$F_i = m_i g$$

donde g es la aceleración de la gravedad ($9.8\ m/s^2$) y m_i es la masa de la i-ésima capa en forma de un pequeño cilindro de altura Δy, es decir:

$$m_i = (\rho)(\text{volumen de la } i\text{-ésima capa})$$

Considerando que el radio r de la i-ésima capa es igual al radio r del recipiente, el cálculo del volumen V_i de la i-ésima capa es

$$V_i = \pi r^2 \Delta y$$

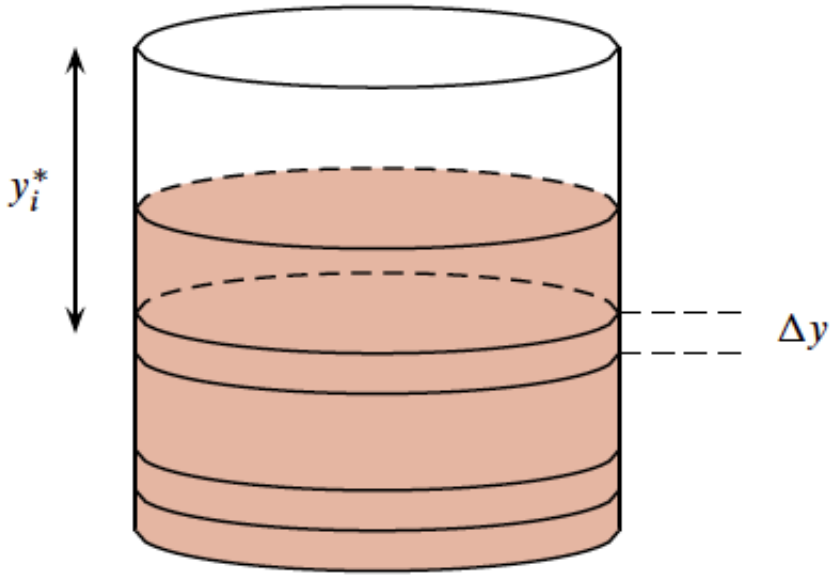


Figura 2.12: Análisis del líquido por capas de grosor Δy.
Fuente: http://canek.uam.mx/Integral/Cap03.Aplicaciones/3.5.Trabajo/FTTrabajo.pdf

Además:

$$W_i = F_i y_i^*$$

El trabajo total que se requiere para elevar, a la parte superior del recipiente, todo el líquido (las n capas) es

$$W = \lim_{n\to\infty} \sum_{i=1}^{n} W_i$$

Para hallar el límite, considere como $\Delta y \to 0$ si $\Delta y = 0.00001$ e $\infty = L/\Delta y$.

Se le pide a usted que elabore un programa en el lenguaje C que, haciendo uso de la programación modular con estructuras algorítmicas selectivas e iterativas, realice lo siguiente:

- Determine el trabajo requerido en kJ para bombear todo el líquido hasta la parte superior del recipiente.

Su solución deberá incluir solo las siguientes funciones, además de la función `main`:

- `void leer_datos(double *altura_cilindro, double *altura_liquido, double *radio_base, double *densidad)`
  /*Función que, simulando el paso de parámetros por referencia, lee los valores de la altura del cilindro, la altura del líquido, el radio de la base del cilindro y la densidad del líquido en kg/m^3. Las longitudes están en m.*/
- `double calcular_volumen_capa_i(double radio_i)`
  /* Función que calcula el volumen de la capa i de acuerdo con su radio r_i. */
- `double calcular_masa_capa_i(double radio_i, double densidad)`
  /* Función que calcula la masa de la capa i de acuerdo con su radio r_i y densidad del líquido. Esta función debe invocar a la función calcular_volumen_capa_i.*/

- `double calcular_fuerza_capa_i(double radio, double densidad)`
  /* Función que calcula la fuerza para bombear la capa i de acuerdo con el radio r y la densidad del líquido. Esta función debe invocar a la función calcular_masa_capa_i. */
- `double calcular_trabajo_capa_i(double radio_base, double densidad, double y_i)`
  /* Función que calcula el trabajo realizado por la fuerza para bombear la capa i de acuerdo con su radio r, la densidad del líquido y la posición y_i en el eje y. Esta función debe invocar a la función calcular_fuerza_capa_i. */
- `double calcular_trabajo_fluido(double altura_cilindro, double altura_liquido, double radio_base, double densidad)`
  /* Función que calcula el trabajo requerido en Joules (J) para bombear todo el líquido hasta la parte superior del recipiente. $1J = 1\ Nm$ y $1N = 1\ kg.m/s^2$ */

No debe usar estructuras algorítmicas selectivas ni iterativas anidadas.

Casos de prueba para verificación de solución

Use los siguientes casos para verificar si su solución es correcta.

H	L	r	ρ	Impresión
3	4	1	1000	La altura del líquido debe ser menor que la altura del cilindro.
3	2	0	1000	Los datos deben ser mayores que cero.
3	2	1	1000	El trabajo total que se requiere para elevar a la parte superior del recipiente todo el líquido es 123.15 kJ.
8	7	4	900	El trabajo total que se requiere para elevar a la parte superior del recipiente todo el líquido es 13965.24 kJ.

2.3.14. Trabajo realizado para extraer el líquido de un cono (adaptado del laboratorio 3 del ciclo 2020-2)

(Adaptado de [10])

En la figura 2.13 se muestra un recipiente de forma cónica con la base circular en la parte superior y cuyo radio es r m, su altura es H m. En su interior se encuentra un líquido con altura de L m, ¿cuál es el trabajo requerido en kJ para bombear todo el líquido hasta la parte superior del recipiente? Considere que la densidad del líquido es $\rho\ kg/m^3$.

Para resolver el problema se deberá tener en cuenta un conjunto de n capas de líquido, cada una con un mismo grosor Δy, como se muestra en la figura 2.14 en la cual se resalta la i-ésima capa de líquido a una distancia y_i^* con respecto de la parte superior del recipiente. El origen del eje vertical será la parte superior del recipiente con el sentido positivo hacia abajo.

Si el número n de capas tiende a ser muy grande, entonces el grosor Δy de cada capa tiende a ser muy pequeño. Para este caso, el trabajo W_i que realiza la fuerza F_i al elevar hasta la parte superior del recipiente la i-ésima capa es

$$W_i = F_i y_i^*$$

La fuerza F_i es igual a la fuerza de atracción gravitacional que actúa sobre la i-ésima capa $= m_i g$:

$$F_i = m_i g$$

g es la aceleración de la gravedad ($9.8\ m/s^2$).

m_i es la masa de la i-ésima capa:

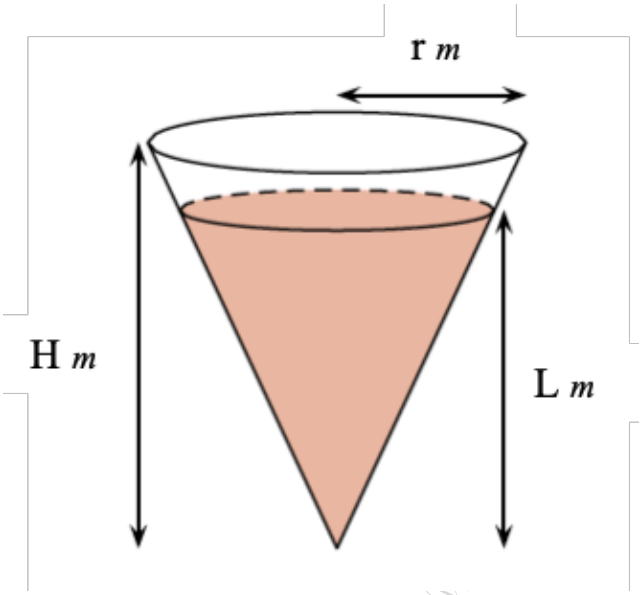


Figura 2.13: Recipiente cónico con un fluido en su interior.
Fuente: http://canek.uam.mx/Integral/Cap03.Aplicaciones/3.5.Trabajo/FTTrabajo.pdf

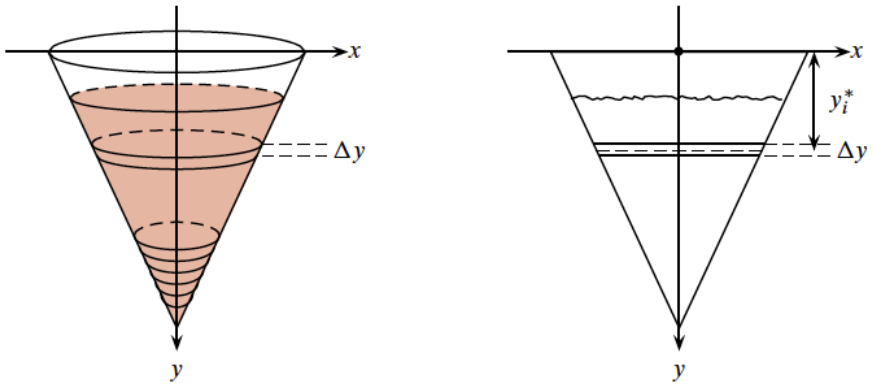


Figura 2.14: Análisis del líquido por capas de grosor Δy.
Fuente: http://canek.uam.mx/Integral/Cap03.Aplicaciones/3.5.Trabajo/FTTrabajo.pdf

$$m_i = (\rho)(\text{volumen de la } i\text{-ésima capa})$$

Considerando que el radio de la i-ésima capa es r_i y el Δy es muy pequeño, el cálculo del volumen V_i de la i-ésima capa sería como si fuera un cilindro circular de radio r_i:

$$V_i = \pi(r_i)^2 \Delta y$$

Aplicando el concepto de triángulos semejantes en la figura 2.15, se puede hallar r_i

Entonces:

$$\frac{r}{H} = \frac{r_i}{H - y_i^*} \Rightarrow r_i = \frac{r(H - y_i^*)}{H}$$

Finalmente, el trabajo total que se requiere para elevar, a la parte superior del recipiente, todo el líquido (las n capas) es :

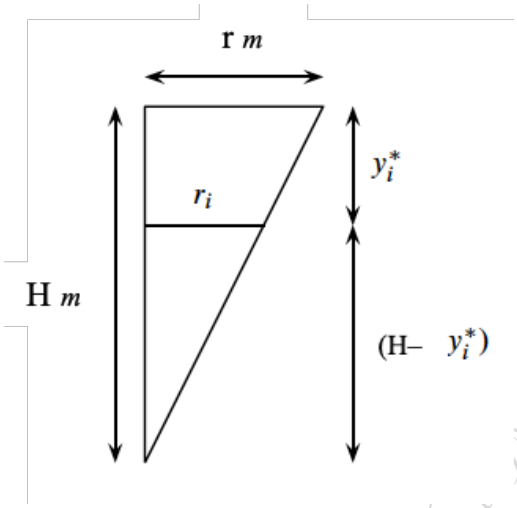


Figura 2.15: Análisis de triángulos semejantes Δy.
Fuente: http://canek.uam.mx/Integral/Cap03.Aplicaciones/3.5.Trabajo/FTTrabajo.pdf

$$W = \lim_{n\to\infty} \sum_{i=1}^{n} W_i$$

Para hallar el límite, considere como $\Delta y \to 0$ si $\Delta y = 0.00001$ e $\infty = L/\Delta y$.

Se le pide a usted que elabore un programa en el lenguaje C que, haciendo uso de la programación modular con estructuras algorítmicas selectivas e iterativas, realice lo siguiente:

- Determine el trabajo requerido en kJ para bombear todo el líquido hasta la parte superior del recipiente.

Su solución deberá incluir solo las siguientes funciones, además de la función `main`:

- `void leer_datos(double *altura_cono, double *altura_liquido, double *radio_cono, double *densidad)`
  /*Función que, simulando el paso de parámetros por referencia, lee los valores de la altura del cono, la altura del líquido, el radio de la base superior del cono y la densidad del líquido en kg/m^3. Las longitudes están en m.*/
- `double calcular_volumen_capa_i(double radio_i)`
  /* Función que calcula el volumen de la capa i de acuerdo con su radio r_i. */
- `double calcular_masa_capa_i(double radio_i, double densidad)`
  /* Función que calcula la masa de la capa i de acuerdo con su radio r_i y densidad del líquido. Esta función debe invocar a la función calcular_volumen_capa_i.*/
- `double calcular_radio_capa_i(double altura_cono, double radio_base_cono, double y_i)`
  /* Función que calcula el radio de la capa i de acuerdo con la altura H del cono, su radio r y la posición y_i en el eje y.*/
- `double calcular_fuerza_capa_i(double altura_cono, double radio_base_cono, double densidad, double y_i)`
  /* Función que calcula la fuerza para bombear la capa i de acuerdo con la altura H del cono, su radio r, la densidad del líquido y la posición y_i en el eje y. Esta función debe invocar a la función calcular_radio_capa_i y calcular_masa_capa_i. */
- `double calcular_trabajo_capa_i(double altura_cono, double radio_base_cono, double densidad, double y_i)`
  /* Función que calcula el trabajo realizado por la fuerza para bombear la capa i de acuerdo con la altura H del cono, su radio r, la densidad del líquido y la posición y_i en el eje y. Esta función debe invocar a la función calcular_fuerza_capa_i. */

- `double calcular_trabajo_fluido(double altura_cono, double altura_liquido, double radio_base, double densidad)`
  /* Función que calcula el trabajo requerido en Joules (J) para bombear todo el líquido hasta la parte superior del recipiente. $1J = 1\ Nm$ y $1N = 1\ kg.m/s^2$ */

No debe usar estructuras algorítmicas selectivas ni iterativas anidadas.

Casos de prueba para verificación de solución

Use los siguientes casos para verificar si su solución es correcta.

H	L	r	ρ	Impresión
8	10	6	1000	La altura del líquido debe ser menor que la altura del cono.
12	10	0	1000	Los datos deben ser mayores que cero.
12	10	6	1000	El trabajo total que se requiere para elevar a la parte superior del recipiente todo el líquido es 11545.36 kJ.
2	1.8	1	900	El trabajo total que se requiere para elevar a la parte superior del recipiente todo el líquido es 8.75 kJ.

2.3.15. Trabajo realizado para comprimir o alargar un resorte (adaptado del laboratorio 3 del ciclo 2020-2)

(Adaptado de [10])

En la figura 2.16 se muestra un resorte en 3 posibles estados (natural, alargado y comprimido). El resorte se alarga una longitud x por la acción de una fuerza F o se comprime una longitud x por la acción de una fuerza F, ¿cuál es el trabajo W en J o Nm que realiza la fuerza F en el extremo derecho para el alargamiento del resorte?

Para alargar un resorte una longitud x sin llegar a su límite de elasticidad, se debe aplicar una fuerza F directamente proporcional a la longitud x:

$$F = kx$$

k en N/m se conoce como la constante del resorte. La fuerza que se requiere para comprimir un resorte es directamente proporcional a la longitud que se comprime o encoge. Tanto para el alargamiento como para la compresión de un resorte, la expresión $F = kx$ representa la fuerza aplicada conocida como la Ley de Hooke.

En la figura 2.17 se muestra un resorte en su estado natural de longitud a sujeto a una pared. Se desea estirar hasta la longitud de b aplicando una fuerza f en el extremo derecho, analicemos como hallar el trabajo que realiza la fuerza f para el alargamiento del resorte.

Se divide el intervalo $[a, b]$ en n-subintervalos, todos ellos con la misma longitud $\Delta x = \frac{b-a}{n}$, considerando $a < b$ (ver figura 2.18). En dicha recta, x_i^* es un punto cualquiera en el intervalo $[x_{i-1}, x_i]$ y la fuerza que se requiere para alargar el resorte hasta x_i^* es $f(x_i^*)$.

Si Δx es muy pequeño, el número n de subintervalos tiende a ser muy grande. Entonces $f(x_i^*)$ en $[x_{i-1}, x_i]$ no varía mucho y se podría decir que $f(x_i^*)$ es casi constante. Por lo tanto, el trabajo W_i que realiza la fuerza $f(x_i^*)$ para alargar el resorte desde x_{i-1} hasta x_i es aproximadamente:

$$W_i \approx f(x_i^*)\Delta x$$

Si se considera una cantidad grande de n subintervalos en $[a,b]$, el trabajo W que realiza la fuerza f al estirar el resorte desde a hasta b sería:

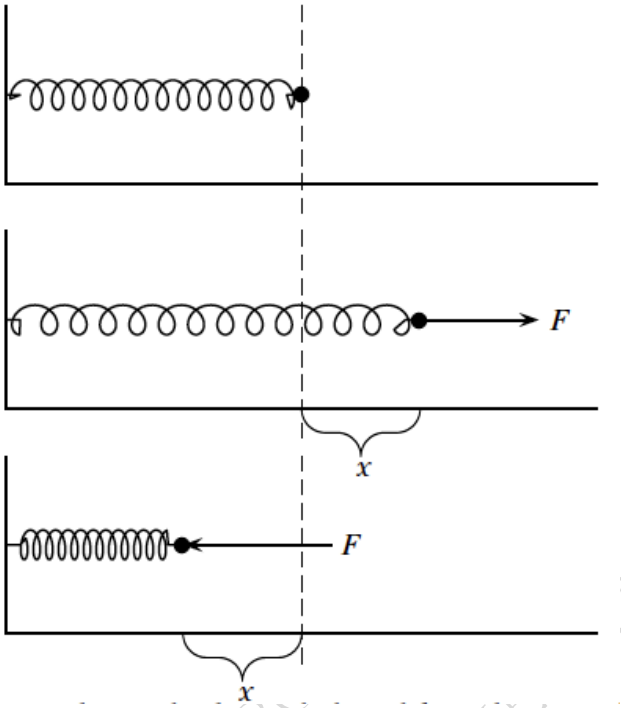


Figura 2.16: Resorte en 3 posibles estados (natural, alargado y comprimido).
Fuente: http://canek.uam.mx/Integral/Cap03.Aplicaciones/3.5.Trabajo/FTTrabajo.pdf

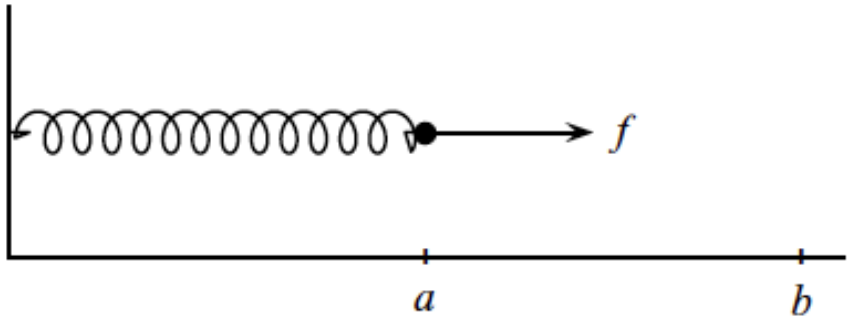


Figura 2.17: Se desea estirar un resorte desde su estado natural con longitud a hasta b, aplicando una fuerza F en el extremo derecho.
Fuente: http://canek.uam.mx/Integral/Cap03.Aplicaciones/3.5.Trabajo/FTTrabajo.pdf

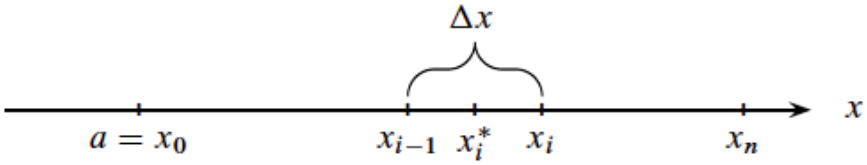


Figura 2.18: Análisis en un punto cualquiera del intervalo $[x_{i-1}, x_i]$.
Fuente: http://canek.uam.mx/Integral/Cap03.Aplicaciones/3.5.Trabajo/FTTrabajo.pdf

$$W = \lim_{n\to\infty} \sum_{i=1}^{n} W_i$$

Para hallar el límite, considere como $\Delta x \to 0$ si $\Delta x = 0.00001$ e $\infty = (b-a)/\Delta x$.

Se le pide a usted que elabore un programa en el lenguaje C que, haciendo uso de la programación modular con estructuras algorítmicas selectivas e iterativas, realice lo siguiente:

- Determine el trabajo en J o Nm que realiza la fuerza f para el alargamiento del resorte desde a hasta b.

Su solución deberá incluir solo las siguientes funciones, además de la función `main`:

- `void leer_datos(double *constante_resorte, double *pos_a, double *pos_b)`
  /*Función que, simulando el paso de parámetros por referencia, lee los valores de la constante k en N/m del resorte, la posición inicial a en el eje X y la posición final b en el eje X. Las longitudes están en m.*/
- `double calcular_f_x_i(double x_i, double cte_resorte)`
  /* Función que calcula la fuerza $f(x_i^*)$ que se requiere para alargar el resorte hasta x_i^*. */
- `double calcular_trabajo_i(double x_i, double cte_resorte, double delta_x)`
  /* Función que calcula el trabajo W_i que realiza la fuerza $f(x_i^*)$ para alargar el resorte desde x_{i-1} hasta x_i. */
- `double calcular_trabajo(double pos_ini, double pos_fin, double cte_resorte)`
  /* Función que calcula el trabajo en Joules (J) que realiza la fuerza f al estirar el resorte desde una posición inicial a hasta una posición final b. $1J = 1\ Nm$ y $1N = 1\ kg.m/s^2$ */

No debe usar estructuras algorítmicas selectivas ni iterativas anidadas.

Casos de prueba para verificación de solución

Use los siguientes casos para verificar si su solución es correcta.

K	a	b	Impresión
0	-1	2	Los datos no son válidos.
8.3	4	2	La posición final (b) debe ser mayor que la posición inicial (a)..
8.3	0	30	El trabajo total que realiza la fuerza F para alargar 30.00 cm es 37.35 Nm.
7.5	1	8	El trabajo total que realiza la fuerza F para alargar 7.00 cm es 2.36 Nm.

2.3.16. Trabajo realizado por una fuerza variable (adaptado del laboratorio 3 del ciclo 2020-2)

(Adaptado de [15])

Considere una partícula que se desplaza a lo largo del eje x bajo la acción de una fuerza que varía con la posición. La figura 2.19 muestra una fuerza variable aplicada en una partícula que se mueve desde la posición inicial x_i a la posición final x_f.

Imagine una partícula que se somete a un desplazamiento muy pequeño Δx, mostrado en la figura. La componente F_x de la fuerza F_x, es aproximadamente constante en este intervalo pequeño; para este pequeño desplazamiento, se puede aproximar el trabajo realizado en la partícula por la fuerza utilizando la siguiente ecuación:

$$W \approx F_x \Delta x$$

que es el área del rectángulo sombreado en la figura 2.19. Si la curva F_x en términos de x se divide en un gran número de tales intervalos, el trabajo total realizado para el desplazamiento de x_i a x_f es aproximadamente igual a la suma de un gran número de tales términos

$$W \approx \sum_{x_i}^{x_f} F_x \Delta x$$

Si se hace que el tamaño de los desplazamientos pequeños se aproxime a cero, el número de términos en la suma aumenta sin límite, pero el valor de la suma se aproxima a un valor definido que es igual al área limitada por la curva Fx y el eje x, expresado como:

$$W = \lim_{\Delta x \to 0} \sum_{x_i}^{x_f} F_x \Delta x$$

Para hallar el límite, considere como $\Delta_x \to 0$ si $\Delta_x = 0.00001$.

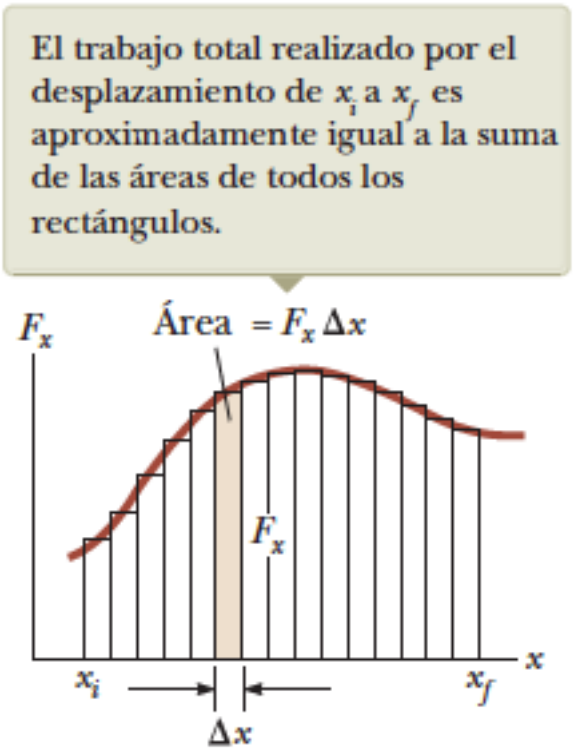


Figura 2.19: El trabajo realizado en una partícula por la componente de fuerza F_x para el pequeño desplazamiento Δx es $F_x \Delta x$, que es igual al área de rectángulo sombreado. Fuente: [15]

Si la fuerza que se aplica a una partícula varía con la posición de acuerdo con la expresión $F_x = f(x) = ax^3 + bx^2 + cx + d$, donde F_x está en $N = mkgs^{-2}$. ¿Cuánto trabajo se realiza en la partícula por la componente de fuerza F_x para moverla desde x_i hasta x_f? El trabajo (W) se expresa en newton metro o joule $(J = Nm = m^2kgs^-2)$.

Se le pide a usted que elabore un programa en el lenguaje C que, haciendo uso de la programación modular con estructuras algorítmicas selectivas e iterativas, realice lo siguiente:

- Determine el trabajo realizado por la fuerza sobre la partícula en el desplazamiento desde x_i cm hasta x_f cm.
- Determine en qué punto a lo largo del eje x se presentó la menor Fuerza en la componente F_x.
- Determine en qué punto a lo largo del eje x se presentó la mayor Fuerza en la componente F_x.

Su solución deberá incluir solo las siguientes funciones, además de la función `main`:

- `void leer_datos(double *a, double *b, double *c, double *d, double *xi, double *xf)` /*Función que, simulando el paso de parámetros por referencia, lee los valores de los coeficientes a, b, c y d, así

como las posiciones de inicio (xi) y de fin (xf) en cm para determinar el trabajo realizado por la fuerza variable.*/

- `double calcular_fuerza(double x, double a, double b, double c, double d)`
  /* Función que calcula la fuerza que se aplica a una partícula en la posición x. */

No debe usar estructuras algorítmicas selectivas ni iterativas anidadas.

Casos de prueba para verificación de solución

Use los siguientes casos para verificar si su solución es correcta.

a	b	c	d	x_i	x_f	Impresión
1	0	0	1	3	2	x_f debe ser mayor que x_i.
1	0	0	1	0	500	El trabajo realizado por la fuerza variable sobre la partícula desde x_i=0.00 cm hasta x_f=500.00 cm es 161.25 J. La menor fuerza en la componente Fx fue de 1.00 N y se presentó en x=0.00 cm. La mayor fuerza en la componente Fx fue de 126.00 N y se presentó en x=500.00 cm.
6	0	0	0	100	200	El trabajo realizado por la fuerza variable sobre la partícula desde x_i=100.00 cm hasta x_f=200.00 cm es 22.55 J. La menor fuerza en la componente Fx fue de 6.00 N y se presentó en x=100.00 cm. La mayor fuerza en la componente Fx fue de 48.07 N y se presentó en x=200.10 cm.
2	-4	-1	1	300	600	El trabajo realizado por la fuerza variable sobre la partícula desde x_i=300.00 cm hasta x_f=600.00 cm es 0.29 J. La menor fuerza en la componente Fx fue de 16 N y se presentó en x=300.00 cm. La mayor fuerza en la componente Fx fue de 283.00 N y se presentó en x=600.00 cm.

2.3.17. Desplazamiento, velocidad promedio y velocidad instantánea (adaptado del laboratorio 3 del ciclo 2020-2)

Una partícula se mueve a lo largo del eje x. Su posición varía con el tiempo de acuerdo con la expresión $x = f(t) = at^2 + bt + c$, donde x está en metros y t está en segundos. Como la posición de la partícula está dada por una función matemática, el movimiento de la partícula es completamente conocido.

Se le pide a usted que elabore un programa en el lenguaje C que, haciendo uso de la programación modular con estructuras algorítmicas selectivas e iterativas, realice lo siguiente:

- Determine el desplazamiento y la velocidad promedio de la partícula en una cantidad (n>0) de intervalos de tiempo t.
- Encuentre la velocidad instantánea de la partícula en el tiempo t medio de cada intervalo de tiempo.
- Determine la mayor de las velocidades promedio de todos los intervalos de tiempo.

El desplazamiento Δx de una partícula se define como su cambio en posición en algún intervalo de tiempo. Conforme la partícula se mueve desde una posición inicial x_i a una posición final x_f, su desplazamiento está dado por

$$\Delta x = x_f - x_i$$

La velocidad promedio $v_x, prom$ de una partícula se define como el desplazamiento Δx de la partícula dividido entre el intervalo de tiempo Δt durante el que ocurre dicho desplazamiento:

$$v_{x,prom} = \frac{\Delta_x}{\Delta_t}$$

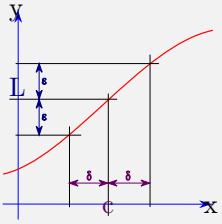
La velocidad instantánea v_x es igual al valor límite de la proporción Δ_x/Δ_t conforme Δ_t tiende a cero:

$$v_x = \lim_{\Delta_t \to 0} \frac{\Delta_x}{\Delta_t}$$

Para hallar el límite, considere como $\Delta_t \to 0$ si $\Delta_t = 0.001$.

Recuerde que:

El límite de una función f(x), cuando x tiende a c es L si y sólo si para todo $\varepsilon > 0$, existe un $\delta > 0$ tal que para todo número real x en el dominio de la función, si $0 < |x - c| < \delta$ entonces $|f(x) - L| < \varepsilon$.
Esto es:
$\lim\limits_{x \to c} f(x) = L \iff \forall \varepsilon > 0,\ \exists \delta > 0 \,|\, \forall x \in \text{Dom}(f),\ 0 < |x - c| < \delta \longrightarrow |f(x) - L| < \varepsilon$



Definición de límite.
Fuente: https://upload.wikimedia.org/wikipedia/commons/d/d1/Límite_01.svg
Se puede hallar el límite para cualquier función, como por ejemplo: Si $\Delta_t = t_2 - t_1 = 0.001$ y $\Delta_x = x_2 - x_1$.

$$\lim_{\Delta_t \to 0} \frac{\Delta_x}{\Delta_t} = \lim_{(t_2 - t_1) \to 0} \frac{x_2 - x_1}{t_2 - t_1} = \lim_{(t_2 - t_1) \to 0} \frac{f(t_2) - f(t_1)}{t_2 - t_1}$$

Su solución deberá incluir solo las siguientes funciones, además de la función `main`:

- `void leer_datos(double *a, double *b, double *c, int *n)` /*Función que, simulando el paso de parámetros por referencia, lee los valores de los coeficientes a, b y c, así como el número de intervalos. */
- `double calcular_posicion(double t, double a, double b, double c)`
  /*Función que calcula la posición en x de la partícula en un determinado tiempo t de acuerdo con la función $f(t) = at^2 + bt + c$. */
- `double calcular_desplazamiento(double t1, double t2, double a, double b, double c)`
  /*Función que calcula el desplazamiento de la partícula en un determinado intervalo de tiempo $\Delta t = t2 - t1$. Debe invocar a la función calcular_posicion.*/
- `double calcular_velocidad_promedio(double t1, double t2, double delta_x)`
  /* Función que calcula la velocidad promedio de la partícula en un determinado intervalo de tiempo $\Delta t = t2 - t1$. */
- `double calcular_velocidad_instantanea(double t, double a, double b, double c)`
  /* Función que calcula la velocidad instantánea de la partícula en un determinado instante de tiempo t. Debe invocar a la función calcular_posicion. Considere $\Delta_t = 0.001$. */

No debe usar estructuras algorítmicas selectivas ni iterativas anidadas.

Casos de prueba para verificación de solución

Use los siguientes casos para verificar si su solución es correcta.

a	b	c	n	c/iteración t1	c/iteración t2	Impresión
2	-4	0	-3			La cantidad de intervalos debe ser mayor que 0.
2	-4	0	2	0	1	El desplazamiento para el intervalo 1 es: -2.00 m. La vel. promedio para el intervalo 1 es: -2.00 m/s. La vel. instantánea de la partícula para el t = 0.50 s es: -2.00 m/s.
				1	3	El desplazamiento para el intervalo 2 es: 8.00 m. La vel. promedio para el intervalo 2 es: 4.00 m/s. La vel. instantánea de la partícula para el t = 2.00 s es: 4.00 m/s. La mayor de las velocidades promedio es 4.00 m/s.
-1	6	-3	3	1	2	El desplazamiento para el intervalo 1 es: 3.00 m. La vel. promedio para el intervalo 1 es: 3.00 m/s. La vel. instantánea de la partícula para el t = 1.50 s es: 3.00 m/s.
				2.5	3.5	El desplazamiento para el intervalo 2 es: 0.00 m. La vel. promedio para el intervalo 2 es: 0.00 m/s. La vel. instantánea de la partícula para el t = 3.00 s es: 0.00 m/s.
				4	5	El desplazamiento para el intervalo 3 es: -3.00 m. La vel. promedio para el intervalo 3 es: -3.00 m/s. La vel. instantánea de la partícula para el t = 4.50 s es: -3.00 m/s. La mayor de las velocidades promedio es 3.00 m/s.

2.3.18. Aceleración promedio e instantánea (adaptado del laboratorio 3 del ciclo 2020-2)

La velocidad de una partícula que se mueve a lo largo del eje x varía de acuerdo con la expresión $v_x = f(t) = at^2 + bt + c$, donde v_x está en m/s, y t está en segundos.

Se le pide a usted que elabore un programa en el lenguaje C que, haciendo uso de la programación modular con estructuras algorítmicas selectivas e iterativas, realice lo siguiente:

- Determine la aceleración promedio de la partícula en una cantidad (n>0) de intervalos de tiempo t.
- Determine la aceleración instantánea de la partícula para cada tiempo t extremo de todos los intervalos de tiempo.
- Determine la menor de las aceleraciones promedio de todos los intervalos de tiempo.

La **aceleración promedio** $a_{x,prom}$ de la partícula se define como el *cambio* en velocidad Δv_x dividido por el intervalo de tiempo Δt durante el que ocurre el cambio:

$$a_{x,prom} = \frac{\Delta v_x}{\Delta t} = \frac{v_{xf} - v_{xi}}{t_f - t_i}$$

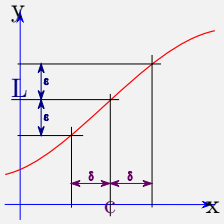
La **aceleración instantánea** a_x es igual al valor límite de la aceleración promedio conforme Δ_t tiende a cero:

$$a_x = \lim_{\Delta_t \to 0} \frac{\Delta v_x}{\Delta t} = \frac{dv_x}{dt}$$

Para hallar el límite, considere como $\Delta_t \to 0$ si $\Delta_t = 0.001$.

Recuerde que:

El límite de una función f(x), cuando x tiende a c es L si y sólo si para todo $\varepsilon > 0$, existe un $\delta > 0$ tal que para todo número real x en el dominio de la función, si $0 < |x - c| < \delta$ entonces $|f(x) - L| < \varepsilon$.
Esto es:
$\lim\limits_{x \to c} f(x) = L \iff \forall \varepsilon > 0,\ \exists \delta > 0 \,|\, \forall x \in \text{Dom}(f),\ 0 < |x - c| < \delta \longrightarrow |f(x) - L| < \varepsilon$



Definición de límite.
Fuente: https://upload.wikimedia.org/wikipedia/commons/d/d1/Límite_01.svg
Se puede hallar el límite para cualquier función, como por ejemplo: Si $\Delta_t = t_2 - t_1 = 0.001$ y $\Delta v_x = v_{x2} - v_{x1}$.

$$\lim_{\Delta_t \to 0} \frac{\Delta v_x}{\Delta_t} = \lim_{(t_2 - t_1) \to 0} \frac{v_{x2} - v_{x1}}{t_2 - t_1} = \lim_{(t_2 - t_1) \to 0} \frac{f(t_2) - f(t_1)}{t_2 - t_1}$$

Su solución deberá incluir solo las siguientes funciones, además de la función `main`:

- `void leer_datos(double *a, double *b, double *c, int *n)` /*Función que, simulando el paso de parámetros por referencia, lee los valores de los coeficientes a, b y c, así como el número de intervalos. */
- `double calcular_velocidad(double t, double a, double b, double c)`
  /*Función que calcula la velocidad en x de la partícula en un determinado tiempo t de acuerdo con la función $f(t) = at^2 + bt + c$. */
- `double calcular_aceleracion_promedio(double t1, double t2, double v1, double v2)`
  /* Función que calcula la aceleración promedio de la partícula en un determinado intervalo de tiempo $\Delta t = t2 - t1$ y un $\Delta v_x = v2 - v1$. */
- `double calcular_aceleracion_instantanea(double t, double a, double b, double c)`
  /* Función que calcula la aceleración instantánea de la partícula en un determinado instante de tiempo t. Debe invocar a la función calcular_velocidad. Considere $\Delta_t = 0.001$. */

No debe usar estructuras algorítmicas selectivas ni iterativas anidadas.

Casos de prueba para verificación de solución

Use los siguientes casos para verificar si su solución es correcta.

a	b	c	n	c/iteración t1	c/iteración t2	Impresión
-5	0	40	-3			La cantidad de intervalos debe ser mayor que 0.
-5	0	40	2			
				0	2	La aceleración promedio para el intervalo 1 es -10.00 m/sˆ2. La acel. instantánea de la partícula para el t=0.00 s es 0.00 m/sˆ2. La acel. instantánea de la partícula para el t=2.00 s es -20.00 m/sˆ2.
				1	3	La aceleración promedio para el intervalo 2 es -20.00 m/sˆ2. La acel. instantánea de la partícula para el t=1.00 s es -10.00 m/sˆ2. La acel. instantánea de la partícula para el t=3.00 s es -30.00 m/sˆ2. La mín. acel. promedio de todos los intervalos es -20.00 m/sˆ2.
-1	6	-3	3			
				1	2	La aceleración promedio para el intervalo 1 es 3.00 m/sˆ2. La acel. instantánea de la partícula para el t=1.00 s es 4.00 m/sˆ2. La acel. instantánea de la partícula para el t=2.00 s es 3.00 m/sˆ2.
				2.5	3.5	La aceleración promedio para el intervalo 2 es 0.00 m/sˆ2. La acel. instantánea de la partícula para el t=2.50 s es 1.00 m/sˆ2. La acel. instantánea de la partícula para el t=3.50 s es -1.00 m/sˆ2.
				4	5	La aceleración promedio para el intervalo 3 es -3.00 m/sˆ2. La acel. instantánea de la partícula para el t=4.00 s es -2.00 m/sˆ2. La acel. instantánea de la partícula para el t=5.00 s es -4.00 m/sˆ2. La mín. acel. promedio de todos los intervalos es -3.00 m/sˆ2.

2.3.19. Centro de masa de un sistema bidimensional (adaptado del laboratorio 3 del ciclo 2020-2)

(Adaptado de [10])

Para hallar el centro de masa de una placa con densidad constante δ cuya forma está delimitada en el plano xy por una función $f(x)$, el eje x y las líneas verticales en los extremos desde a hasta $f(a)$ y b hasta $f(b)$ (ver figura 2.20), se divide la superficie de la lámina en n franjas verticales de igual anchura Δx de acuerdo con la figura 2.21.

El área de la franja es $\Delta_i A = f(x_i^*)\Delta x$

La masa de la franja es $\Delta_i m = \delta \Delta_i A = \delta f(x_i^*)\Delta x$

Para un número de n de franjas verticales que tiende al infinito y las Δx tienden a cero, la masa M de la placa es

$$M = \lim_{n\to\infty} \sum_{i=1}^{n} \delta f(x_i^*)\Delta x$$

El momento de la lámina con respecto al eje y es

$$M_y = \lim_{n\to\infty} \sum_{i=1}^{n} x_i^* \delta f(x_i^*)\Delta x$$

El momento de la lámina con respecto al eje x es

$$M_x = \lim_{n\to\infty} \sum_{i=1}^{n} \frac{\delta}{2}[f(x_i^*)]^2 \Delta x$$

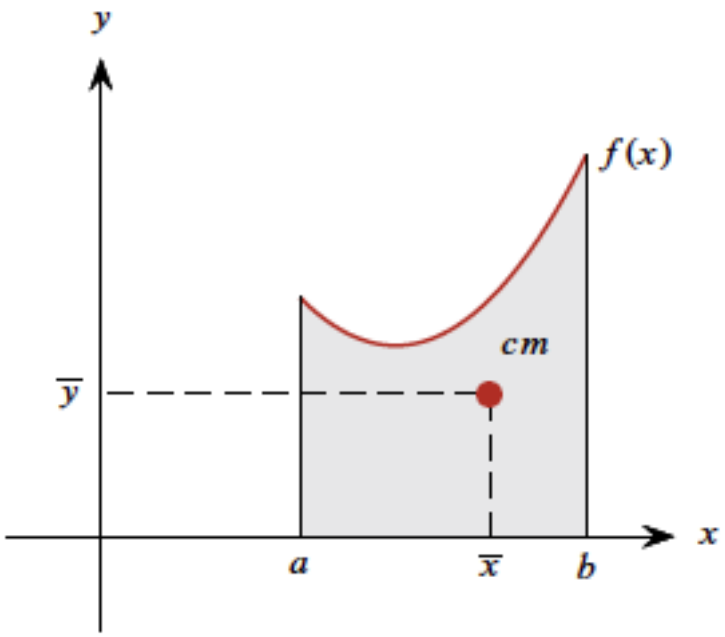


Figura 2.20: Centro de masa de una placa delimitada por una función $f(x)$, el eje x y las líneaas verticales en los extremos desde x_i hasta $f(x_i)$ y x_f hasta $f(x_f)$. Fuente: http://canek.uam.mx/Integral/Cap03.Aplicaciones/3.7.Momento/FTMomento.pdf

.

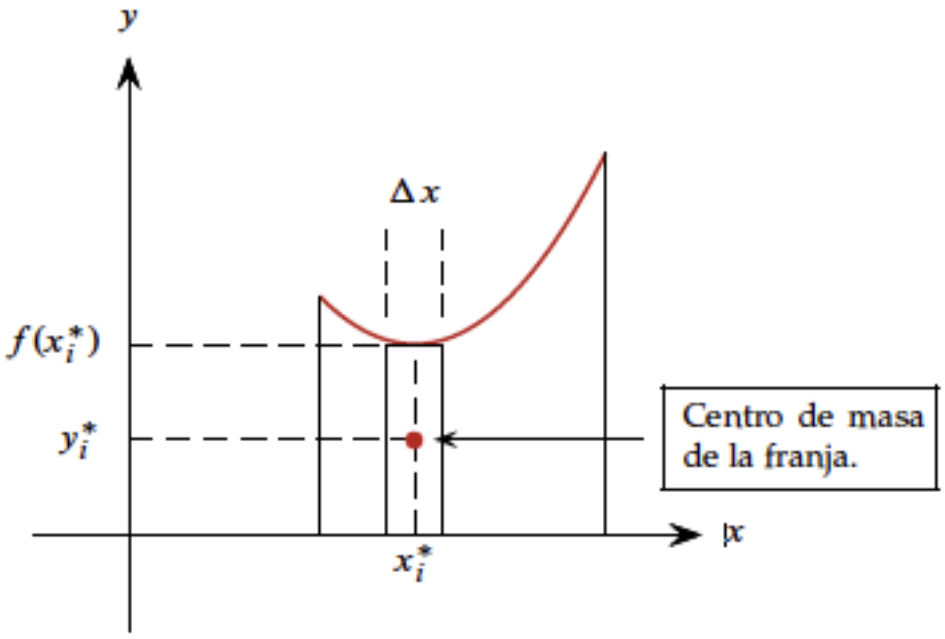


Figura 2.21: Centro de masa de la franja vertical, con coordenadas x_i^* y y_i^*, el ancho de la franja es Δx y su altura es $f(x_i^*)$. Fuente: http://canek.uam.mx/Integral/Cap03.Aplicaciones/3.7.Momento/FTMomento.pdf

Finalmente, las coordenadas $\bar{x}$ e $\bar{x}$ del centro de masa de la lámina con densidad constante δ son:

$$\bar{x} = \frac{M_y}{M}$$

$$\bar{y} = \frac{M_x}{M}$$

Se le pide a usted que elabore un programa en el lenguaje C que, haciendo uso de la programación modular con estructuras algorítmicas selectivas e iterativas, realice lo siguiente:

- Determine el centro de masa de una placa delgada con densidad constante δ, cuya superficie está delimitada por $f(x) = ax^2 + bx - c$, el eje x del plano cartesiano y las líneas verticales en los extremos de a hasta $f(a)$ y b hasta $f(b)$.

Para hallar el límite, considere como $\Delta x = 0.001$, e $\infty = n$ sería la cantidad de franjas de ancho Δx que pueden entrar entre a y b.

Su solución deberá incluir solo las siguientes funciones, además de la función `main`:

- `void leer_datos(double *a, double *b, double *c, double *xi, double *xf, double *densidad)` /*Función que, simulando el paso de parámetros por referencia, lee los valores de los coeficientes a, b y c, así como las posiciones de inicio (xi) y de fin (xf) en m. para determinar el centro de masa, y la densidad de la placa delgada.*/
- `double calcular_fx(double x, double a, double b, double c)` /*Función que calcula $f(x)$ en una determinada posición x de acuerdo con la función $f(x) = ax^2+bx+c$. */
- `double calcular_masa(double a, double b, double c, double xi, double xf, double densidad)` /* Función que calcula la masa de la lámina delimitada por $f(x)$, el eje x, y las líneas verticales de xi hasta $f(xi)$ y de xf hasta $f(xf)$.*/
- `double calcular_momento_y(double a, double b, double c, double xi, double xf, double densidad)` /* Función que calcula el momento de la lámina con respecto al eje y, con densidad constante δ y delimitada por $f(x)$, el eje x, y las líneas verticales de xi hasta $f(xi)$ y de xf hasta $f(xf)$.*/
- `double calcular_momento_x(double a, double b, double c, double xi, double xf, double densidad)` /* Función que calcula el momento de la lámina con respecto al eje x, con densidad constante δ y delimitada por $f(x)$, el eje x, y las líneas verticales de xi hasta $f(xi)$ y de xf hasta $f(xf)$.*/
- `double calcular_centro_masa(double a, double b, double c, double xi, double xf, double densidad, *ptr_x_centro_masa, *ptr_y_centro_masa)` /* Función que calcula las coordenadas $\bar{x}$ e $\bar{y}$ del centro de masa de la lámina con densidad constante δ delimitada por $f(x)$, el eje x, y las líneas verticales de xi hasta $f(xi)$ y de xf hasta $f(xf)$. Tenga en cuenta que en está función las coordenadas del centro de masa se retornan simulando el paso de parámetros por referencia.*/

No debe usar estructuras algorítmicas selectivas ni iterativas anidadas.

Casos de prueba para verificación de solución

Use los siguientes casos para verificar si su solución es correcta.

a	b	c	δ	x_i	x_f	Impresión
-1	6	-5	1.5	5	1	xi debe ser menor que xf.
-1	6	-5	1.5	1	5	Las coordenadas del centro de masa [x, y] son [3.00 m, 1.60 m].
-1	6	-3	1.2	2	4	Las coordenadas del centro de masa [x, y] son [3.00 m, 2.84 m].

2.3.20. Centro de masa de una placa delimitada entres dos curvas (adaptado del laboratorio 3 del ciclo 2020-2)

(Adaptado de [10])

Para hallar el centro de masa de una placa con densidad constante δ en kg/m^3 cuya forma está delimitada en el plano xy por la gráfica de dos funciones $f(x)$ y $g(x)$, se divide la superficie de la placa en n franjas verticales de igual anchura Δx de acuerdo con la figura 2.22.

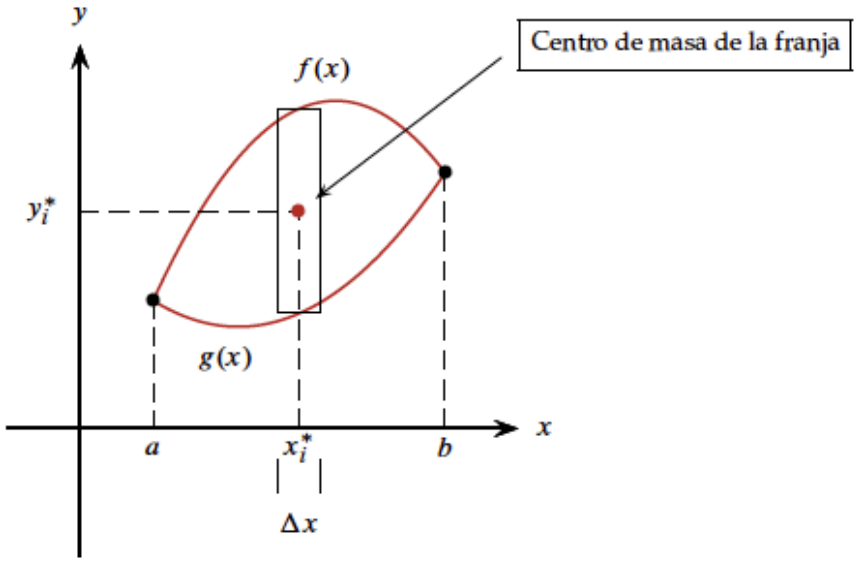


Figura 2.22: Centro de masa de la franja vertical, con coordenadas x_i^* y y_i^*, el ancho de la franja es Δx y su altura es $f(x_i^*) - g(x_i^*)$. Fuente: http://canek.uam.mx/Integral/Cap03.Aplicaciones/3.7.Momento/FTMomento.pdf

El área de la franja es $\Delta_i A = [f(x_i^*) - g(x_i^*)]\Delta x$

La masa de la franja es $\Delta_i m = \delta \Delta_i A = \delta[f(x_i^*) - g(x_i^*)]\Delta x$

Observe que:

$$y_i^* = \frac{f(x_i^*) + g(x_i^*)}{2}$$

Para un número de n de franjas verticales que tiende al infinito y las Δx tienden a cero, la masa M de la placa es

$$M = \lim_{n\to\infty} \sum_{i=1}^{n} \delta[f(x_i^*) - g(x_i^*)]\Delta x$$

El momento de la placa con respecto al eje y es

$$M_y = \lim_{n\to\infty} \sum_{i=1}^{n} x_i^* \delta[f(x_i^*) - g(x_i^*)]\Delta x$$

El momento de la placa con respecto al eje x es

$$M_x = \lim_{n\to\infty} \sum_{i=1}^{n} \frac{\delta}{2}[[f(x_i^*)]^2 - [g(x_i^*)]^2]\Delta x$$

Finalmente, las coordenadas $\bar{x}$ e $\bar{x}$ del centro de masa de la placa con densidad constante δ son:

$$\bar{x} = \frac{M_y}{M}$$
$$\bar{y} = \frac{M_x}{M}$$

Se le pide a usted que elabore un programa en el lenguaje C que, haciendo uso de la programación modular con estructuras algorítmicas selectivas e iterativas, realice lo siguiente:

- Determine el centro de masa de una placa delgada con densidad constante δ, cuya superficie está delimitada por las gráficas de las funciones $f(x) = ax^2 + bx - c$ y $g(x) = mx^2 + nx - p$ desde x_i hasta x_f.

Para hallar el límite, considere como $\Delta x = 0.001$, e $\infty = n$ sería la cantidad de franjas de ancho Δx que pueden entrar entre a y b.

Su solución deberá incluir solo las siguientes funciones, además de la función `main`:

- `void leer_datos(double *a, double *b, double *c, double *m, double *n, double *p, double *xi, double *xf, double *densidad)`
  /*Función que, simulando el paso de parámetros por referencia, lee los valores de los coeficientes a, b y c, m, n, p, así como las posiciones de inicio (x_i) y de fin (x_f) en m. para determinar el centro de masa, y la densidad en kg/m^3 de la placa delgada.*/
- `double calcular_funcion_x(double x, double coef_1, double coef_2, double coef_3)`
  /*Función que calcula el resultado de una función cuadrática con coeficientes $coef_1$, $coef_2$ y $coef_3$ en una determinada posición x de acuerdo con la función $coef_1x^2 + coef_2x + coef_3$. */
- `double calcular_masa(double a, double b, double c, double m, double n, double p, double xi, double xf, double densidad)`
  /* Función que calcula la masa de la placa delimitada por las gráficas de las funciones $f(x)$ y $g(x)$ desde x_i hasta x_f.*/
- `double calcular_momento_y(double a, double b, double c, double m, double n, double p, double xi, double xf, double densidad)`
  /* Función que calcula el momento de la placa con respecto al eje y, con densidad constante δ y delimitada por las gráficas $f(x)$ y $g(x)$ desde x_if hasta x_f.*/
- `double calcular_momento_x(double a, double b, double c, double m, double n, double p, double xi, double xf, double densidad)`
  /* Función que calcula el momento de la lámina con respecto al eje x, con densidad constante δ y delimitada por las gráficas de las funciones $f(x)$ y $g(x)$ desde x_i hasta x_f.*/
- `double calcular_centro_masa(double a, double b, double c, double m, double n, double p, double xi, double xf, double densidad, *ptr_x_centro_masa, *ptr_y_centro_masa)`
  /* Función que calcula las coordenadas $\bar{x}$ e $\bar{y}$ del centro de masa de la lámina con densidad constante δ delimitada por las gráficas de las funciones $f(x)$ y $g(x)$ desde x_i hasta x_f. Tenga en cuenta que las coordenadas del centro de masa se retornan simulando el paso de parámetros por referencia.*/

No debe usar estructuras algorítmicas selectivas ni iterativas anidadas.

Casos de prueba para verificación de solución

Use los siguientes casos para verificar si su solución es correcta.

a	b	c	m	n	p	δ	x_i	x_f	Impresión
1	2	3	-1	6	-5	1.5	5	1	xi debe ser menor que xf.
0	1	0	0.142	0	0	1.5	0	7	Las coordenadas del centro de masa [x, y] son [3.52 m, 2.82 m].
0	1	0	0.142	0	0	1.5	0	7	Las coordenadas del centro de masa [x, y] son [3.52 m, 2.82 m].
-1	6	-3	1	-6	9	19304	1	5	Las coordenadas del centro de masa [x, y] son [3.00 m, 3.00 m].

2.3.21. Fuerza y presión de un fluido (adaptado del laboratorio 3 del ciclo 2020-2)

(Adaptado de [15])

El agua llena una altura H m detrás de un dique de ancho w m (ver figura 2.23). Determine la fuerza resultante en kN y la presión promedio en kPa que el agua ejerce sobre el dique. La densidad del agua es $\rho = 1\,000\ kg/m^3$, $1Pa \equiv 1N/m^2$, $1g \equiv 9.8m/s^2$.

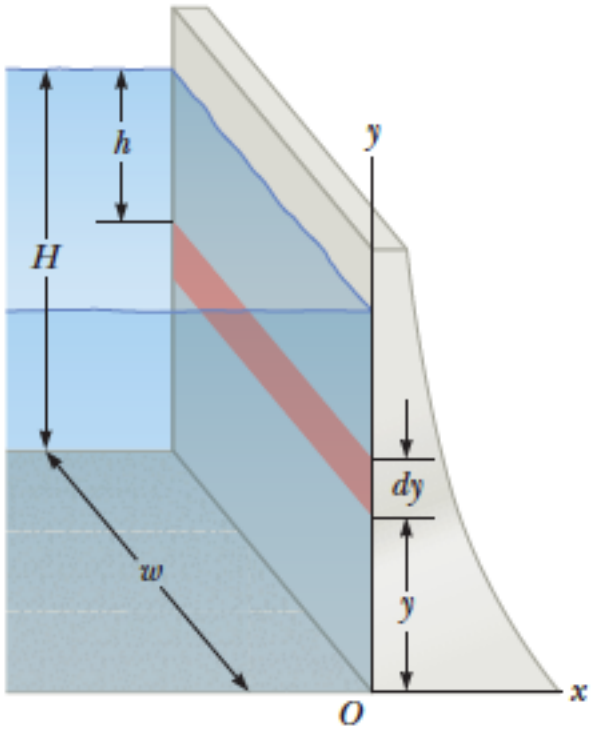


Figura 2.23: El agua ejerce una fuerza sobre un dique. Fuente: [15].

La presión varía con la profundidad, entonces no se puede calcular la fuerza simplemente al multiplicar el área por la presión. Como la presión en el agua se incrementa con la profundidad, entonces también aumenta la fuerza sobre la parte adyacente del dique.

Imagine un eje vertical y, con y = 0 en el fondo del dique. Divida la cara del dique en estrechas tiras horizontales a una distancia y sobre el fondo, como la tira roja de la figura 2.23. La presión sobre cada tira sólo se debe al agua; la presión atmosférica actúa sobre ambos lados del dique.

La presión debida al agua a la profundidad h es

$$P = \rho gh = \rho g(H - y)$$

El área A_i de esta i-ésima franja es $A_i = wd_y$. Si d_y es muy pequeña, entonces la presión P_i sobre dicha i-ésima franja es casi constante y se obtiene con la expresión:

$$P_i \approx \rho gh_i = \rho g(H - y_i)$$

Por otra parte, la fuerza total F_i del fluido sobre esta franja es, aproximadamente:

$$F_i = P_i Ai \approx \rho gh_i(d_y.w) \Rightarrow$$
$$\Rightarrow F_i \approx \rho gwh_i d_y$$

Un conjunto de n franjas cubre la pared vertical del dique por lo que la fuerza total F del fluido sobre esta superficie vertical es, aproximadamente:

$$F \approx \sum_{i=1}^{n} F_i \Rightarrow$$
$$\Rightarrow F \approx \sum_{i=1}^{n} P_i A_i \Rightarrow$$
$$\Rightarrow F \approx \sum_{i=1}^{n} \rho gwh_i d_y$$

En esta última expresión, si el número n de franjas sobre la superficie vertical crece tanto como se quiera, la estimación de la fuerza F será más certera.

Por otra parte, si el número n de franjas tiende a infinito:

$$F = \lim_{n\to\infty} \sum_{i=1}^{n} \rho gwh_i d_y \tag{2.1}$$

La presión promedio debida al agua sobre la cara del dique es:

$$P_{prom} = \frac{F}{A} = \frac{F}{wH} \tag{2.2}$$

Se le pide a usted que elabore un programa en el lenguaje C que, haciendo uso de la programación modular con estructuras algorítmicas selectivas e iterativas, realice lo siguiente:

- Determine la fuerza resultante que el agua ejerce sobre el dique.
- Determine la presión promedio debida al agua sobre la cara del dique.

Para hallar el límite, considere como $d_y = 0.00001$, e $\infty = n$ sería la cantidad de franjas de altura d_y que pueden entrar entre 0 y la altura H del agua.

Su solución deberá incluir solo las siguientes funciones, además de la función `main`:

- `void leer_datos(double *altura_agua, double *altura_dique, double *ancho_dique)`
  /*Función que, simulando el paso de parámetros por referencia, lee los valores de la altura del agua en m, la altura del dique en m y el ancho del dique en m.*/
- `double calcular_fuerza(double altura_agua, double ancho_dique)`
  /* Función que recibe la altura del agua en m, el ancho del dique en m y calcula la fuerza resultante en N que el agua ejerce sobre el dique. Debe resolverlo iterativamente usando la ecuación 2.1.*/
- `double calcular_presion(double fuerza, double altura_agua, double ancho_dique)`
  /* Función que recibe la fuerza resultante en N que el agua ejerce sobre el dique, la altura del agua en m, el ancho del dique en m y calcula la presión promedio en Pa debida al agua sobre la cara del dique. */

No debe usar estructuras algorítmicas selectivas ni iterativas anidadas.

Casos de prueba para verificación de solución

Use los siguientes casos para verificar si su solución es correcta.

Alt. agua en m	Alt. dique en m	Ancho dique en m	Impresión
90	80	80	La altura del agua no puede estar por encima de la altura del dique.
70	80	0	El ancho del dique tiene que ser mayor que 0.
70	80	80	La fuerza resultante que el agua ejerce sobre el dique es de 1920800.27 kN. La presión promedio del agua sobre el dique es de 343.00 kPa.
8	14	30	La fuerza resultante que el agua ejerce sobre el dique es de 9408.01 kN. La presión promedio del agua sobre el dique es de 39.20 kPa.

2.3.22. Fuerza y presión de un fluido sobre una cara vertical de un hexaedro regular (adaptado del laboratorio 3 del ciclo 2020-2)

(Adaptado de [10])

Se tiene un hexaedro regular (cubo) sumergido en un tanque con un líquido de densidad ρ en kg/m^3, como se muestra en la figura 2.24. El nivel del líquido es de H m y la longitud de cada una de las aristas del cubo mide ℓ m. Tenga en cuenta que ℓ puede superar a H. Se busca calcular la fuerza total del fluido (fuerza hidrostática) en kN y la presión promedio en kPa sobre una de las superficies verticales del cubo. $1Pa \equiv 1N/m^2$

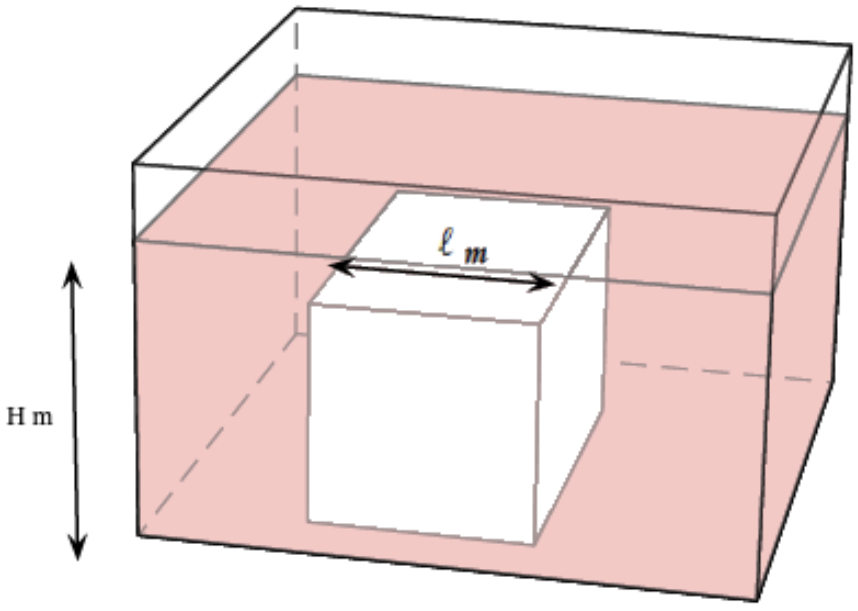


Figura 2.24: El líquido en un contenedor ejerce una fuerza sobre una de las caras verticales del cubo. Fuente: http://canek.uam.mx/Integral/Cap03.Aplicaciones/3.6.Presion/FTPresion.pdf.

La presión varía con la profundidad, entonces no se puede calcular la fuerza simplemente al multiplicar el área por la presión.

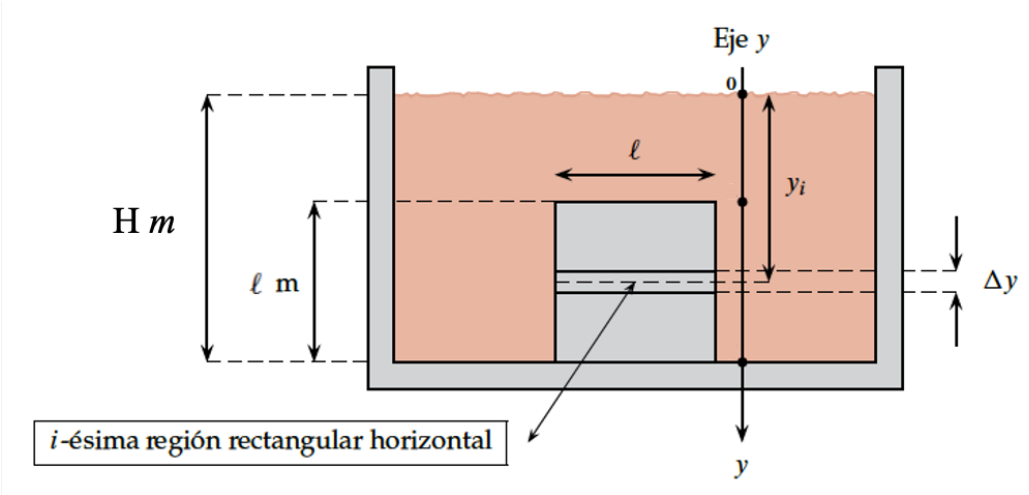


Figura 2.25: Análisis de la i-ésima región rectangular horizontal en una de las caras verticales del cubo sumergido en un fluido. Fuente: http://canek.uam.mx/Integral/Cap03.Aplicaciones/3.6.Presion/FTPresion.pdf.

Se representa una de las caras verticales del cubo en el plano cartesiano que se muestra en la figura 2.25, donde el eje vertical tiene su origen en la parte superior del fluido y el sentido positivo hacia abajo.

De acuerdo con la figura 2.25, si se considera una i-ésima región rectangular horizontal (una franja) con altura Δy m y largo ℓ m, sobre una de las caras verticales del cubo, el área y el Δy de la i-ésima región señalada es muy pequeña, entonces la presión P_i sobre dicha región es casi constante y se obtiene con la expresión

$$P_i = \rho g y_i$$

Donde $1g \equiv 9.8m/s^2$.

La fuerza total F_i del fluido sobre esta región es, aproximadamente:

$$F_i \approx P_i A_i \approx \rho g y_i (\Delta y.\ell) \Rightarrow$$

$$\Rightarrow F_i \approx \rho g \ell y_i \Delta y$$

Un conjunto de n franjas cubre la pared vertical del cubo por lo que la fuerza total F del fluido sobre esta superficie vertical es, aproximadamente:

$$F \approx \sum_{i=1}^{n} F_i \Rightarrow$$

$$\Rightarrow F \approx \sum_{i=1}^{n} P_i A_i \Rightarrow$$

$$\Rightarrow F \approx \sum_{i=1}^{n} \rho g \ell y_i \Delta y$$

Si el número n de franjas tiende a infinito, se tiene una mejor estimación de la fuerza:

$$F = \lim_{n\to\infty} \sum_{i=1}^{n} \rho g \ell y_i \Delta y \tag{2.3}$$

La presión promedio debida al líquido sobre una cara vertical del cubo es:

$$P_{prom} = \frac{F}{A} \tag{2.4}$$

Se le pide a usted que elabore un programa en el lenguaje C que, haciendo uso de la programación modular con estructuras algorítmicas selectivas e iterativas, realice lo siguiente:

- Determine la fuerza hidrostática total sobre una de las paredes verticales del hexaedro regular.
- Determine la presión promedio debida al líquido sobre una de las paredes verticales del hexaedro regular.

Para hallar el límite, considere como $\Delta y = 0.00001$, e $\infty = n$ sería la cantidad de franjas de altura Δy que pueden entrar entre el fondo del contenedor hasta la altura del cubo.

Su solución deberá incluir solo las siguientes funciones, además de la función `main`:

- `void leer_datos(double *alt_liquido, double *arista_cubo, double *densidad)`
  /*Función que, simulando el paso de parámetros por referencia, lee los valores de la altura del líquido en m, la longitud de la arista del cubo en m y la densidad del líquido en kg/m^3. */
- `double calcular_fuerza_resultante(double alt_liquido, double arista_cubo, double densidad)`
  /* Función que recibe la altura del líquido en m, la longitud de la arista del cubo en m y la densidad en kg/m^3 del líquido, y calcula la fuerza resultante en N que el líquido ejerce sobre una cara vertical del cubo. Considere también que la longitud de la arista ℓ del cubo podría ser mayor que la altura H del líquido. Debe resolverlo iterativamente usando la ecuación 2.3.*/
- `double calcular_presion(double fuerza, double altura_liquido, double arista_cubo)`
  /* Función que recibe la fuerza resultante en N que el líquido ejerce sobre una cara vertical del cubo, la altura en m del líquido, la longitud en m de la arista de una cara vertical del cubo, y calcula la presión promedio en Pa debida al líquido sobre una de las caras verticales del cubo. Considere también que la longitud de la arista ℓ del cubo podría ser mayor que la altura H del líquido.*/

No debe usar estructuras algorítmicas selectivas ni iterativas anidadas.

Casos de prueba para verificación de solución

Use los siguientes casos para verificar si su solución es correcta.

Alt. líqu. m	**Arista cubo** m	**Densidad** Kg/m^3	**Impresión**
0	5	1000	Todos los datos deben ser mayores que 0.
5	0	1000	Todos los datos deben ser mayores que 0.
5	3	1000	La fuerza resultante que el líquido ejerce sobre una cara vertical del cubo es de 308.70 kN. La presión promedio del líquido sobre una cara vertical del cubo es de 34.30 kPa..
3	5	1000	La fuerza resultante que el líquido ejerce sobre una cara vertical del cubo es de 220.50 kN. La presión promedio del líquido sobre una cara vertical del cubo es de 14.70 kPa.

2.3.23. Fuerza y presión de un fluido sobre una pared vertical de un semicilindro horizontal (adaptado del laboratorio 3 del ciclo 2020-2)

(Adaptado de [10])

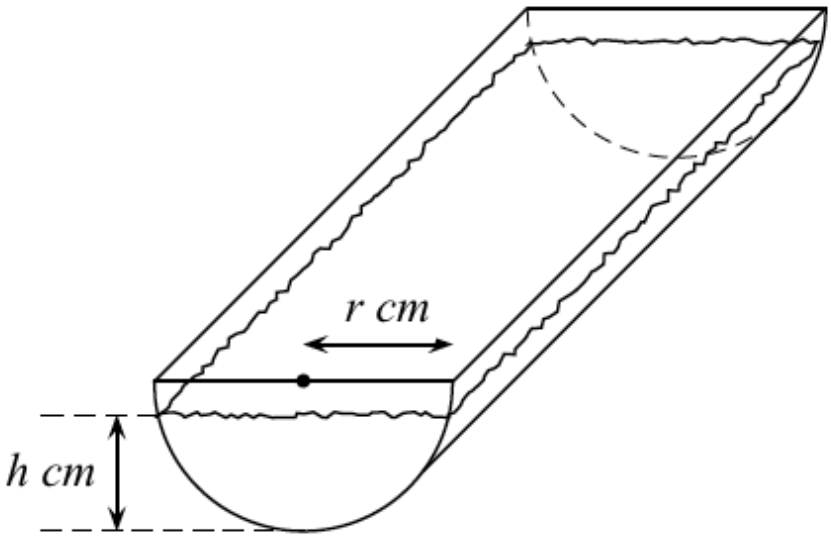


Figura 2.26: El líquido en un contenedor de forma semicilíndrica ejerce una fuerza sobre una de las paredes verticales del contenedor. Fuente: http://canek.uam.mx/Integral/Cap03.Aplicaciones/3.6.Presion/FTPresion.pdf.

En un contenedor de forma de medio cilindro horizontal de radio r cm se encuentra un fluido de densidad ρ km/m^3 cuya altura es de h cm (ver figura 2.26). ¿Cuál es la fuerza en kN y la presión promedio en kPa que ejerce el líquido sobre una de las paredes verticales del contenedor? $1Pa \equiv 1N/m^2$, $1g \equiv 9.8m/s^2$.

La presión varía con la profundidad, entonces no se puede calcular la fuerza simplemente al multiplicar el área por la presión.

Se representa una de las paredes verticales del contenedor en el plano cartesiano que se muestra en la figura 2.27, donde el eje vertical tiene su origen en la parte superior del recipiente y el sentido positivo hacia abajo. Observe que parte del contorno de la pared vertical es la curva $x^2 + y^2 = r^2$ por lo que la abscisa del punto A mostrado en la figura 2.27 es $x = \sqrt{r^2 - y^2}$.

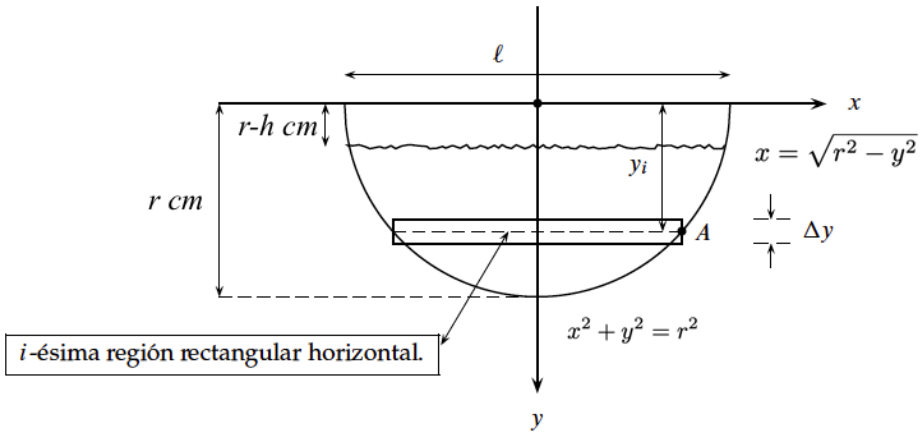


Figura 2.27: Análisis de la i-ésima región rectangular horizontal en una de las paredes verticales del contenedor de forma semicilíndrica. Fuente: http://canek.uam.mx/Integral/Cap03.Aplicaciones/3.6.Presion/FTPresion.pdf.

Si Δy de la i-ésima región señalada es muy pequeña, entonces la presión P_i sobre dicha región es casi constante y se obtiene con la expresión

$$P_i = \rho g y_i$$

La fuerza total F_i del fluido sobre esta región es, aproximadamente:

$$F_i \approx P_i A_i$$

donde el área de la i-ésima región A_i es

$$A_i = \ell \Delta y = 2\sqrt{r^2 - y^2}\Delta y$$

Entonces:

$$F_i \approx P_i A_i \approx [\rho g y_i][2\sqrt{r^2 - y^2}]\Delta y$$

Un conjunto de n franjas cubre la pared vertical del contenedor, por lo que la fuerza total F del fluido sobre dicha pared es, aproximadamente:

$$F \approx \sum_{i=1}^{n} F_i \Rightarrow$$

$$\Rightarrow F \approx \sum_{i=1}^{n} P_i A_i \Rightarrow$$

$$\Rightarrow F \approx \sum_{i=1}^{n} [\rho g y_i][2\sqrt{r^2 - y_i^2}]\Delta y$$

Si el número n de franjas tiende a infinito, se tiene una mejor estimación de la fuerza:

$$F = \lim_{n\to\infty} \sum_{i=1}^{n} \rho g y_i 2\sqrt{r^2 - y_i^2}\Delta y \tag{2.5}$$

El área de una cara vertical del contenedor que cubre el líquido se puede calcular mediante la ecuación 2.6:

$$A = \lim_{n\to\infty} \sum_{i=1}^{n} [2\sqrt{r^2 - y_i^2}]\Delta y \tag{2.6}$$

La presión promedio debida al líquido sobre una cara vertical del contenedor es:

$$P_{prom} = \frac{F}{A} \tag{2.7}$$

Se le pide a usted que elabore un programa en el lenguaje C que, haciendo uso de la programación modular con estructuras algorítmicas selectivas e iterativas, realice lo siguiente:

- Determine la fuerza resultante que el líquido ejerce sobre una de las paredes verticales del contenedor.
- Determine la presión promedio debida al líquido sobre una de las paredes verticales del contenedor.

Para hallar el límite, considere como $\Delta y = 0.00001$, e $\infty = n$ sería la cantidad de franjas de altura Δy que pueden entrar entre el fondo del contenedor hasta la altura del líquido.

Su solución deberá incluir solo las siguientes funciones, además de la función `main`:

- `void leer_datos(double *altura_liquido, double *altura_contenedor, double *densidad)` /*Función que, simulando el paso de parámetros por referencia, lee los valores de la alura del líquido en cm, la altura del contenedor en cm y la densidad del líquido en kg/m^3. */

- `double calcular_fuerza(double alt_liquido, double radio_pared_contenedor, double densidad)`
  /* Función que recibe la altura del líquido en m, el radio de una pared vertical del contenedor en m y la densidad en kg/m^3, y calcula la fuerza resultante en N que el líquido ejerce sobre una pared vertical del contenedor. Debe resolverlo iterativamente usando la ecuación 2.5.*/
- `double calcular_area(double alt_liquido, double radio_pared_contenedor)`
  /* Función que recibe la altura del líquido en m y el radio de una pared vertical del contenedor en m, y calcula el área en m^2 que el líquido abarca sobre una pared vertical del contenedor. Debe resolverlo iterativamente usando la ecuación 2.6.*/
- `double calcular_presion(double fuerza, double altura_liquido, double radio_pared_contenedor)`
  /* Función que recibe la fuerza resultante en N que el líquido ejerce sobre una pared vertical del contenedor, la altura en m del líguido, el radio en m de una pared vertical del contenedor, y calcula la presión promedio en Pa debida al líquido sobre una de las paredes verticales del contenedor. */

No debe usar estructuras algorítmicas selectivas ni iterativas anidadas.

Casos de prueba para verificación de solución

Use los siguientes casos para verificar si su solución es correcta.

Alt. líqu. *cm*	**Radio conten.** *cm*	**Densidad** Kg/m^3	**Impresión**
90	80	890	La alt. del líquido no puede estar por encima de la alt. del contenedor.
0	100	890	Todos los datos deben ser mayores que 0.
80	90	0	Todos los datos deben ser mayores que 0.
80	100	890	La fuerza res. que el líquido ejerce sobre una pared vertical es 5.47 kN. La presión promedio del líquido sobre una pared vertical es de 4.66 kPa.
100	100	890	La fuerza res. que el líquido ejerce sobre una pared vertical es 5.81 kN. La presión promedio del líquido sobre una pared vertical es de 3.70 kPa.

2.3.24. Desplazamiento en base a la velocidad y el tiempo (adaptado del laboratorio 3 del ciclo 2020-2)

(Adaptado de [15])

Una partícula se mueve a lo largo del eje x. Su velocidad varía con el tiempo de acuerdo con la expresión $v_x = f(t) = at^2 + bt + c$, donde v_x está en m/s y t está en segundos.

Suponga que la gráfica v_x vs t para una partícula que se mueve a lo largo del eje x es como se muestra en la figura 2.28. Divida el intervalo de tiempo $t_f - t_i$ en muchos pequeños intervalos, cada uno de duración Δt_n. El desplazamiento de la partícula durante cualquier intervalo pequeño, como el sombreado en la figura 2.28, está dado por $\Delta x_n = v_{xn,prom}\Delta t_n$, donde $v_{xn,prom}$ es la velocidad promedio en dicho intervalo. En consecuencia, el desplazamiento durante este pequeño intervalo simplemente es el área del rectángulo sombreado. El desplazamiento total para el intervalo $t_f - t_i$ es la suma de las áreas de todos los rectángulos desde t_i hasta t_f:

$$\Delta x = \sum_{i=1}^{n} v_{xi,prom}\Delta t_n$$

Ahora, conforme los intervalos se hacen cada vez más pequeños, el número de términos en la suma aumenta y la suma tiende a un valor igual al área bajo la gráfica velocidad-tiempo. Debido a esto, en el límite n $\to \infty$, o $\Delta t_n \to 0$, el desplazamiento es

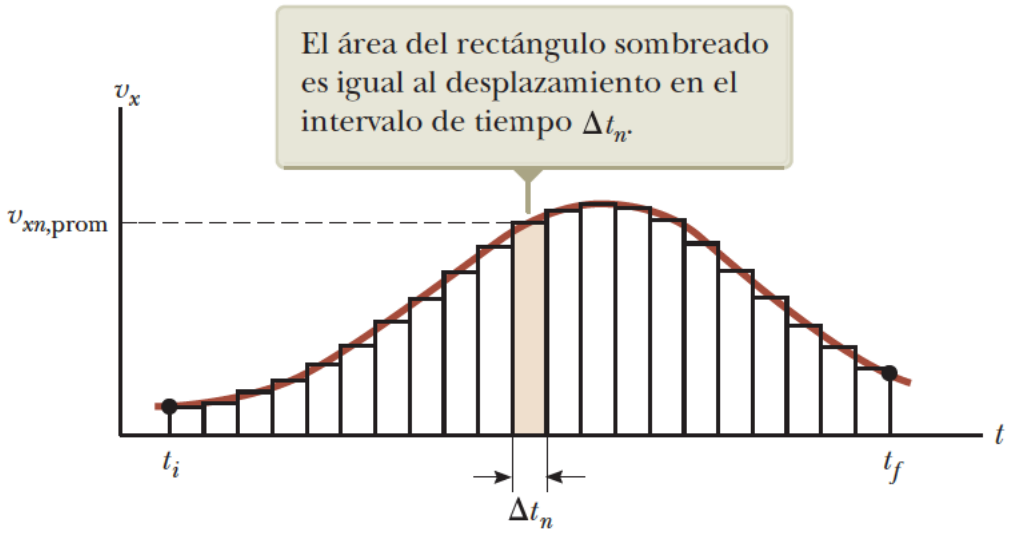


Figura 2.28: Velocidad en función del tiempo para cualquier partícula que se mueve a lo largo del eje x. El área total bajo la curva es el desplazamiento total de la partícula. Fuente: [15]

$$\Delta x = \lim_{\Delta t_n \to 0} \sum_{i=1}^{n} v_{xi,prom} \Delta t_n$$

Para hallar el límite, considere como $\Delta t_n \to 0$ si $\Delta t_n = 0.001$.

La aceleración promedio $a_x, prom$ de una partícula se define como la relación de cambio en su velocidad Δv_x dividida por el intervalo de tiempo Δt durante el que ocurre dicho cambio:

$$a_{x,prom} = \frac{\Delta v_x}{\Delta t} = \frac{v_{xf} - v_{xi}}{t_f - t_i}$$

Se le pide a usted que elabore un programa en el lenguaje C que, haciendo uso de la programación modular con estructuras algorítmicas selectivas e iterativas, realice lo siguiente:

- Determine el desplazamiento de la partícula en un intervalo de tiempo definido por t_i hasta t_f.
- Determine la mínima aceleración promedio en el intervalo de tiempo, así como el instante de tiempo en que ocurre ello.
- Determine la máxima aceleración promedio en el intervalo de tiempo, así como el instante de tiempo en que ocurre ello.

Su solución deberá incluir solo las siguientes funciones, además de la función `main`:

- `void leer_datos(double *a, double *b, double *c, double *ti, double *tf)` /*Función que, simulando el paso de parámetros por referencia, lee los valores de los coeficientes a, b y c, así como los tiempos de inicio (ti) y fin (tf).*/
- `double calcular_velocidad(double t, double a, double b, double c)`
  /* Función que calcula la velocidad de la partícula en un determinado instante de tiempo t. */
- `double calcular_aceleracion_promedio(double t1, double t2, double v1, double v2)`
  /* Función que calcula la aceleración promedio de la partícula en un determinado intervalo de tiempo entre t1 y t2. */

No debe usar estructuras algorítmicas selectivas ni iterativas anidadas.

Casos de prueba para verificación de solución

Use los siguientes casos para verificar si su solución es correcta.

a	b	c	t_i	t_f	Impresión
-5	0	40	3	2	tf debe ser mayor que ti.
-5	0	40	0	2	El desplazamiento en el intervalo [0.00 s, 2.00 s] es: 66.69 m. La mín. acel. promedio es -20.00 m/s^2 para t=2.00 s. La máx. acel. promedio es 0.00 m/s^2 para t=0.00 s.
-1	6	-3	1	5	El desplazamiento en el intervalo [1.00 s, 5.00 s] es: 18.67 m. La mín. acel. promedio es -4.00 m/s^2 para t=5.00 s. La máx. acel. promedio es 4.00 m/s^2 para t=1.00 s.

2.3.25. Longitud de arco (adaptado del laboratorio 3 del ciclo 2020-2)

La longitud de arco es la distancia o camino recorrido a lo largo de una curva generada por la función $f(x)$. Para calcular la longitud de un arco contenido en el plano XY, se puede dividirla en segmentos muy pequeños, escogiendo una cantidad finita de puntos para luego aproximar la longitud mediante la longitud de la poligonal que pasa por dichos puntos. Cuanto más cerca estén los puntos, el valor obtenido tendrá una mayor aproximación a la longitud del arco.

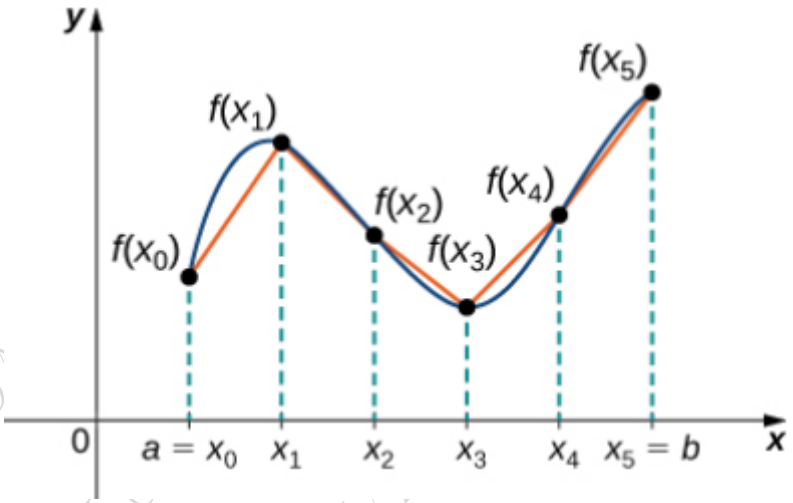


Figura 2.29: Aproximación de la longitud de arco de una curva mediante segmentos de recta. Fuente: https://calculo21.com/wp-content/uploads/2019/12/6.1_40.png

Para aproximar el valor de la longitud del arco de una curva, se puede considerar una serie de puntos en el intervalo $[a, b]$ como en la figura 2.29. Luego, si se traza una línea poligonal con los puntos $[x_0,f(x_0)]$, $[x_1,f(x_1)]$, ..., $[x_n, f(x_n)]$, la suma de todas las longitudes de segmentos sería:

$$\sum_{i=1}^{n} \sqrt{(x_i - x_{i-1})^2 + [f(x_i) - f(x_{i-1})]^2}$$

En la figura 2.30 se muestra un segmento de recta representativo que aproxima la curva sobre el intervalo cerrado $[x_{i-1},x_i]$, y si $\Delta x = x_i - x_{i-1}$ es muy pequeño, entonces:

$$L(f,[a,b]) = \lim_{n\to\infty} \sum_{i=1}^{n} \sqrt{(\Delta x)^2 + [f(x_{i-1} + \Delta x) - f(x_{i-1})]^2}$$

Se le pide a usted que elabore un programa en el lenguaje C que, haciendo uso de la programación modular con estructuras algorítmicas selectivas e iterativas, realice lo siguiente:

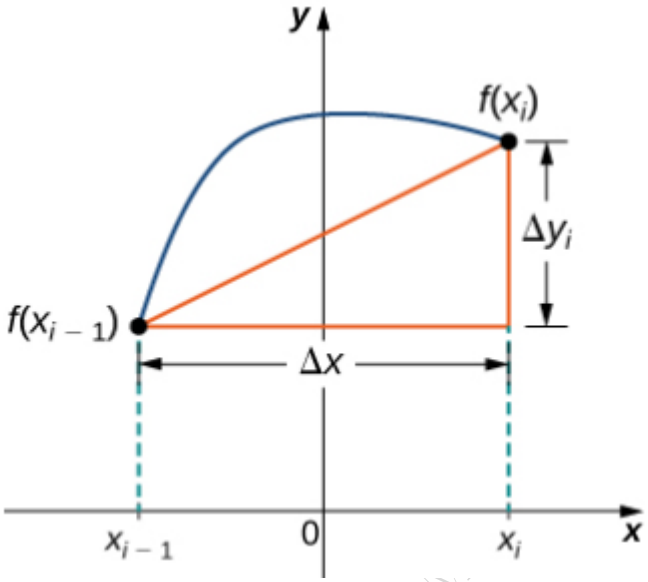


Figura 2.30: Segmento de recta que aproxima la curva sobre el intervalo cerrado $[x_{i-1},x_i]$.
Fuente: https://calculo21.com/wp-content/uploads/2019/12/6.1_41.png

- Determine la longitud de arco generado por la función $f(x)$ desde $x_0 = a$ hasta $x_n = b$.

Para hallar el límite, considere como $\Delta x \to 0$ si $\Delta x = 0.00001$ e $\infty = (b-a)/\Delta x$.

Su solución deberá incluir solo las siguientes funciones, además de la función `main`:

- `void leer_datos(double *a, double *b, double *c, double *x_ini, double *x_fin)` /*Función que, simulando el paso de parámetros por referencia, lee los valores de los coeficientes a, b y c de la función $f(x) = ax^2 + bx + c$, así como el valor de x_0 de inicio y el x_n final. */
- `double calcular_fx(double a, double b, double c, double x)`
  /*Función que calcula el valor de y de acuerdo con la función $f(x) = ax^2 + bx + c$. */
- `double calcular_distancia(double x_1, double y_1, double x_2, double y_2)`
  /*Función que calcula la distancia entre los puntos [x_1, y_1] y [x_2, y_2].*/
- `double calcular_longitud_arco(double a, double b, double c, double x_ini, double x_fin)`
  /* Función que calcula la longitud del arco de la curva formada por la gráfica de la función $f(x) = ax^2 + bx + c$ en el intervalo [x_ini, x_fin].*/

No debe usar estructuras algorítmicas selectivas ni iterativas anidadas.

Casos de prueba para verificación de solución

Use los siguientes casos para verificar si su solución es correcta.

a	b	c	x_0	x_n	Impresión
-1	6	-3	5.5	2.5	x_0 debe ser menor que x_n.
-1	6	-3	1.5	5.5	La longitud del arco es 9.778204.
0	1.5	-2	0	5	La longitud del arco es 9.013878.
1	-6	9	2	5	La longitud del arco es 6.125727.

2.3.26. Cálculos matemáticos (adaptado del laboratorio 3 del ciclo 2021-1)

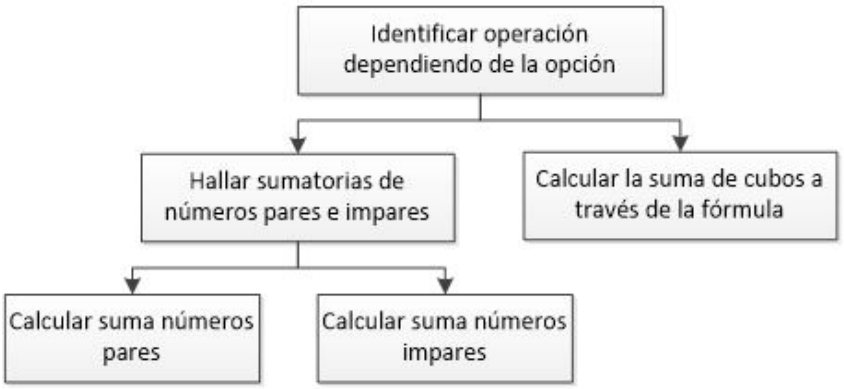
Conforme ha pasado el tiempo, se ha logrado identificar distintos cálculos matemáticos que permiten obtener resultados en base a sumatorias o fórmulas. Estas combinaciones numéricas ayudan a identificar ciertos patrones porque se repiten una y otra vez.

Se le pide elaborar un programa en lenguaje C que permita evaluar determinados cálculos matemáticos dependiendo de lo que el usuario desee. Para ello, el usuario ingresará una opción, y dependiendo de ella, solicitará algunos datos para poder realizar la operación. Asuma que siempre se ingresa una opción correcta.

- **Opción 'P':** Deberá leer la cantidad N de números a evaluar y hallar y mostrar la suma de números pares e impares de los primeros N números naturales. Deberá usar estructuras iterativas.
- **Opción 'S:** Deberá leer dos valores a y b y calcular y mostrar la suma de ambos elevada al cubo utilizando la siguiente fórmula: $a^3 + 3a^2b + 3ab^2 + b^3$.

.

Se le pide implementar su solución que coincida con el siguiente diagrama de módulos:



Nota: al menos un subprograma debe tener dos parámetros por referencia como mínimo.

Caso de prueba

Caso de Prueba 1
Ingrese la opción: S
Ingrese los números para evaluar la suma de cubos: 4 6
El cubo de la sumatoria a través de la fórmula para 4 y 6 es 1000

Caso de Prueba 2
Ingrese la opción: P
Ingrese la cantidad de números a evaluar: 5
La sumatoria de los primeros 5 números pares es 30 y de los primeros 5 números impares es 25

Caso de Prueba 3
Ingrese la opción: P
Ingrese la cantidad de números a evaluar: 20
La sumatoria de los primeros 20 números pares es 420 y de los primeros 20 números impares es 400

Caso de Prueba 4
Ingrese la opción: S
Ingrese los números para evaluar la suma de cubos: 15 22
El cubo de la sumatoria a través de la fórmula para 15 y 22 es 50653

2.3.27. Números capicúas (adaptado del laboratorio 3 del ciclo 2021-1)

Pese a la cantidad de idiomas que existe para lograr la comunicación entre distintos países y culturas, existe un lenguaje que es universal y es a través de las matemáticas, específicamente de los números. A lo largo del tiempo, distintos científicos fueron descubriendo distintos tipos de números, entre ellos se encuentran los capicúas.

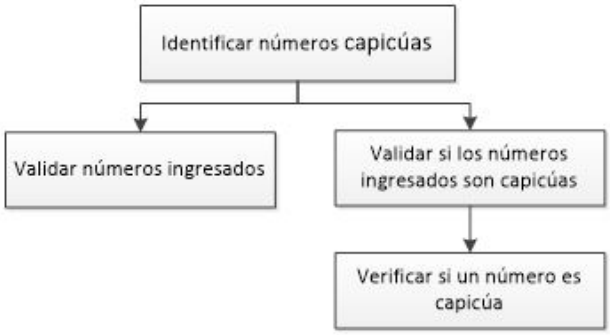
La palabra capicúa proviene del catalán cap i cua y significa "cabeza y cola" y se refiere a cualquier cadena de caracteres o número que se lee de igual manera de izquierda a derecha o de derecha a izquierda.

Si hablamos de número capicúa, se puede afirmar que todos los números de base 10 con un dígito {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} son capicúas.

Se le pide implementar un programa en lenguaje C que, dados tres números ingresados por el usuario, permita evaluar si son capicúas. La implementación deberá validar que los datos ingresados sean de 5 cifras, caso contrario, deberá enviar un mensaje de error: "Los números ingresados no son válidos"

Luego de obtener si los números son o no capicúa, deberá enviar un mensaje si solo alguno de ellos lo es y otro mensaje si es que los tres son a la vez de acuerdo a lo indicado en los casos de prueba.

Se le pide implementar su solución que coincida con el siguiente diagrama de módulos:



Nota: al menos un subprograma debe tener al menos dos parámetros por referencia.

Caso de prueba

Caso de Prueba 1
Ingrese los tres números a evaluar de 5 cifras: 23432 58085 94949
Los tres números son capicúas

Caso de Prueba 2
Ingrese los tres números a evaluar de 5 cifras: 60706 76547 98841
Solo el número 60706 es capicúa

Caso de Prueba 3
Ingrese los tres números a evaluar de 5 cifras: 46564 78187 34202
Los números 46564 y 78187 son capicúas

Caso de Prueba 4
Ingrese los tres números a evaluar de 5 cifras: 89012 54366 38983
Solo el número 38983 es capicúa

Caso de Prueba 5
Ingrese los tres números a evaluar de 5 cifras: 23451 134529 4664
Los números ingresados no son válidos

2.3.28. Números narcisista (adaptado del laboratorio 3 del ciclo 2021-1)

Encontramos número en gran parte de nuestras actividades y hoy en día son fundamentales para explicar distintos comportamientos. Los números más antiguos fueron usados por los chinos y posteriormente fueron adaptados por los japoneses.

Para implementar el sistema de numeración se identificó que existan símbolos para los números del 1 al 9 y para las decenas, centenas y millares. Los chinos escribían verticalmente y leían de arriba abajo y así existen distintas maneras de identificarlos.

Pueden realizarse múltiples combinaciones entre ellos y así identificar distintos tipos de números. Por ejemplo: los números primos, compuestos, narcisista entre otros.

Se le pide implementar un programa en lenguaje C que permita evaluar y mostrar si el número ingresado por el usuario es un número narcisista. Se dice que un número es narcisista cuando se cumple que el número es igual a la suma de los números formados por cada dígito elevado a la cantidad de dígitos del número. Por ejemplo, 153 es un número narcisista de 3 dígitos, ya que $1^3 + 5^3 + 3^3 = 153$

Para ello solicitará el número a evaluar y deberá validar que el número ingresado sea positivo y tenga de tres a seis dígitos, caso contrario, deberá enviar un mensaje de error: "El número ingresado no es positivo y/o no tiene de 3 a 6 dígitos."

Se le pide implementar su solución que coincida con el siguiente diagrama de módulos:

Nota: El subprograma "Validar si un número es narcisista" debe utilizar estructuras iterativas con entrada controlada y por contador.

Caso de prueba

Caso de Prueba 1
Ingrese un número que tenga de tres a seis dígitos: 153
El número 153 es narcisista.

Caso de Prueba 2
Ingrese un número que tenga de tres a seis dígitos: 1547
El número 1547 no es narcisista.

Caso de Prueba 3
Ingrese un número que tenga de tres a seis dígitos: 84
El número ingresado no es positivo y/o no tiene de 3 a 6 dígitos.

Caso de Prueba 4
Ingrese un número que tenga de tres a seis dígitos: 54748
El número 54748 es narcisista.

2.3.29. Números primos y abundantes (adaptado del laboratorio 3 del ciclo 2021-1)

A lo largo de los años, los científicos han ido descubriendo algunas peculiaridades entre los números. En base a ello, se han ido creando tipos de números. Dentro de todos los tipos de números que existen, cada uno tiene características particulares si cumple algunas condiciones.

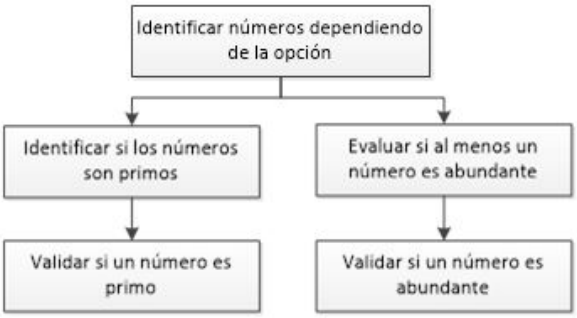
Se le pide elaborar un programa en lenguaje C que permita evaluar si un grupo de números son de un determinado tipo. Para ello, el usuario ingresará una opción y tres números, dependiendo de la acción, su programa realizará alguna acción. Asuma que siempre se ingresa una opción correcta.

- **Opción '#':** Deberá evaluar y mostrar si los números son primos o no.
- **Opción '%':** Deberá evaluar y mostrar si al menos uno de los números es abundante. Un número abundante si cumple que la suma de sus divisores sin contar al mismo número, es mayor que el propio

número. Por ejemplo: el número 12 tiene como divisores {1,2,3,4,6} y se cumple que dicha suma es mayor que 12, por lo tanto, es abundante.

.

Se le pide implementar su solución que coincida con el siguiente diagrama de módulos:



Nota: al menos un subprograma debe tener dos parámetros por referencia como mínimo.

Caso de prueba

Caso de Prueba 1

Ingrese la opción: #
Ingrese los números a evaluar: 45 37 99
Evaluación del número 45 si es primo (1 si es verdadero y 0 si es falso) = 0
Evaluación del número 37 si es primo (1 si es verdadero y 0 si es falso) = 1
Evaluación del número 99 si es primo (1 si es verdadero y 0 si es falso) = 0

Caso de Prueba 2

Ingrese la opción: %
Ingrese los números a evaluar: 9 12 100
Al menos uno de los números ingresados es abundante (1 si es verdadero y 0 si es falso) = 1

Caso de Prueba 3

Ingrese la opción: #
Ingrese los números a evaluar: 31 83 47
Evaluación del número 31 si es primo (1 si es verdadero y 0 si es falso) = 1
Evaluación del número 83 si es primo (1 si es verdadero y 0 si es falso) = 1
Evaluación del número 47 si es primo (1 si es verdadero y 0 si es falso) = 1

Caso de Prueba 4

Ingrese la opción: %
Ingrese los números a evaluar: 99 15 77
Al menos uno de los números ingresados es abundante (1 si es verdadero y 0 si es falso) = 0

2.3.30. Operaciones matemáticas (adaptado del laboratorio 3 del ciclo 2021-1)

A lo largo del tiempo, los científicos han ido descubriendo que los números pueden agruparse de distintas maneras y a través de ellos se pueden realizar muchas operaciones que permiten obtener resultados.

Se le pide elaborar un programa en lenguaje C que permita evaluar determinadas operaciones dependiendo de lo que el usuario desee. Para ello, el usuario ingresará una opción, y dependiendo de ella, solicitará algunos datos para poder realizar la operación. Asuma que siempre se ingresa una opción correcta.

- **Opción 'A':** Deberá leer la cantidad N de números a evaluar y calcular y mostrar la suma de cubos de los N primeros números naturales. Deberá usar una estructura iterativa.
- **Opción 'B:** Deberá leer una cantidad de números que se evaluará. Luego, leerá cada número de tres cifras y finalmente obtendrá el menor y mayor de los números ingresados. Asuma que los números ingresados son de tres cifras.

. Se le pide implementar su solución que coincida con el siguiente diagrama de módulos:

Nota: al menos un subprograma debe tener dos parámetros por referencia como mínimo.

Caso de prueba

Ingrese la opción: B
Ingrese la cantidad de números a evaluar: 7
Ingrese el número 1 a evaluar de tres dígitos: 890
Ingrese el número 2 a evaluar de tres dígitos: 237
Ingrese el número 3 a evaluar de tres dígitos: 100
Ingrese el número 4 a evaluar de tres dígitos: 890
Ingrese el número 5 a evaluar de tres dígitos: 456
Ingrese el número 6 a evaluar de tres dígitos: 999
Ingrese el número 7 a evaluar de tres dígitos: 232
De la lista de 7 números, el número menor es 100 y el número mayor es 999

Ingrese la opción: A
Ingrese la cantidad de números a evaluar: 5
La sumatoria de cubos de los 5 números es: 225

Ingrese la opción: A
Ingrese la cantidad de números a evaluar: 20
La sumatoria de cubos de los 20 números es: 44100

Ingrese la opción: B
Ingrese la cantidad de números a evaluar: 4
Ingrese el número 1 a evaluar de tres dígitos: 456
Ingrese el número 2 a evaluar de tres dígitos: 789
Ingrese el número 3 a evaluar de tres dígitos: 122
Ingrese el número 4 a evaluar de tres dígitos: 390
De la lista de 4 números, el número menor es 122 y el número mayor es 789

Bibliografía

[1] A. Bansal. *Introduction to programming languages*. CRC Press, Boca Raton, FL, 2014.

[2] Y. Chen. *Introduction to Programming Languages Programming in C C++ Scheme Prolog C# and Soa*. Kendall Hunt Pub Co, City, 2016.

[3] P. Deitel and H. Deitel. *C how to program : with an introduction to C++*. Pearson, Boston, 2016.

[4] B. Gottfried. *Schaum's Outline of Programming with C*. McGraw-Hill, 2 edition, 1996.

[5] M. A. Jackson. *Principles of program design*. Academic Press, London New York, 1975.

[6] Y. Kanetkar. *Let Us C*. BPB Publications, 5th edition, 2004.

[7] B. Kernighan and D. Ritchie. *The C programming language*. Prentice Hall, Englewood Cliffs, N.J, 1988.

[8] S. Kochan. *Programming in C*. Developer's Library/Sams Pub, Indianapolis, Ind, 2005.

[9] K. Louden and K. Lambert. *Programming languages : principles and practice*. Course Technology/Cengage Learning, Boston, MA, 2012.

[10] U. A. Metropolitana. Canek: Portal de matemática. urlhttp://canek.uam.mx/.

[11] G. Myers. *Reliable software through composite design*. Petrocelli/Charter, New York, 1975.

[12] H. Schildt. *C, the complete reference*. Osborne/McGraw-Hill, Berkeley, 2000.

[13] T. Schultz. *C and the 8051*. Prentice Hall PTR, Upper Saddle River, N.J, 1998.

[14] R. Sebesta. *Concepts of programming languages*. Pearson Education Limited, Harlow, 2016.

[15] R. Serway. *Física para ciencias e ingeniería 1 (10a. ed)*. CENGAGE Learning, Ciudad de México, 2018.

[16] E. Yourdon. *Modern structured analysis*. Prentice-Hall International, Englewood Cliffs, N.J. London, 1989.