



GUÍA DE FUNDAMENTOS DE PROGRAMACIÓN

ETS 2026_2

Rocio Reséndiz M.
rresendizm@ipn.mx

Herramientas de programación

Algoritmos

Los algoritmos son elementos clave en el aprendizaje de la programación, ya que ofrecen las bases necesarias para resolver problemas mediante la creación de secuencias lógicas de pasos bien definidos.

Un algoritmo es una secuencia finita de pasos bien definidos que llevan a la resolución de un problema específico. La estructura básica:

- *Entrada:* Especifica qué datos necesita el algoritmo para funcionar y un ejemplo podría ser: Leer dos números
- *Proceso:* Describe los pasos necesarios para procesar los datos de entrada, por ejemplo, operaciones como $\text{suma} \leftarrow \text{número1} + \text{número2}$
- *Salida:* Especifica qué información generada debe ser presentada al usuario, por ejemplo, Escribir El resultado es: suma.

ALGORITMO

Entrada: dos números, **numero1**, **numero2**

Proceso:

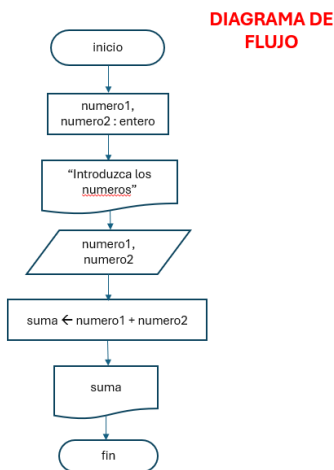
1. Sumar el **numero1** con el **numero2** y guardar el resultado en **suma**

$\text{suma} \leftarrow \text{numero1} + \text{numero2}$

Salida: Mostrar la suma, **suma**

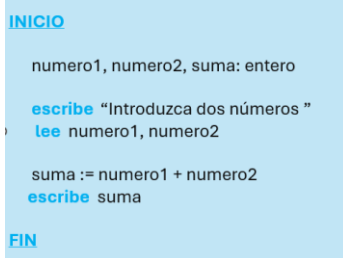
Diagrama de flujo

Los diagramas de flujo son herramientas visuales que, a través de símbolos gráficos, ilustran cómo se va desarrollando el control en un proceso o algoritmo. Utilizan símbolos gráficos para ilustrar las diferentes etapas, decisiones y secuencias de pasos necesarios para llegar a un resultado específico.



Pseudocódigo

Se puede decir que el pseudocódigo es una representación intermedia entre el lenguaje natural y el lenguaje de programación. Se utiliza para diseñar y visualizar algoritmos de manera más comprensible antes de implementarlos en un lenguaje de programación específico. El pseudocódigo ayuda a los programadores a planificar y estructurar sus soluciones sin preocuparse por la sintaxis específica del lenguaje de programación.



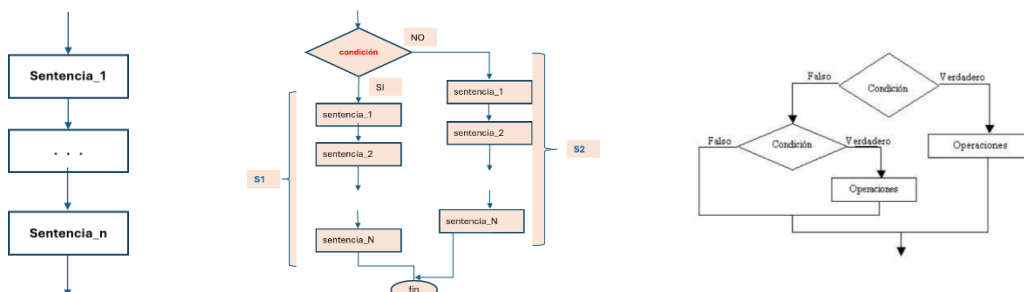
Ejercicios Para cada uno de los siguientes problemas realice el **algoritmo**, **diagrama de flujo** y el **Pseudocódigo**. Recuerde que la solución de dichos problemas con lleva a la escritura de un *algoritmo* el cual se representa mediante un *diagrama de flujo* (aplicar correctamente las reglas y simbología estándar para dicho diseño) después construir el *pseudocódigo* correspondiente y finalmente llevarse a la *codificación*.

- ❖ Calcular la suma total e imprimir en pantalla cada término de la siguiente serie: $1/2^1 + 2/2^2 + 3/2^3 + \dots + N/2^N$.
- ❖ Dado N notas de un estudiante calcular:
 - a) Cuantas notas tiene reprobadas.
 - b) Cuantas notas aprobadas.
 - c) El promedio de notas.
 - d) El promedio de notas aprobadas y reprobadas.
- ❖ Realizar las operaciones básicas de una calculadora: suma, resta, multiplicación y división permitiendo seleccionar una de las operaciones anteriores (S o s, R o r, M o m y D o d). Una vez realizada ésta debe permitirse realizar otra operación, en caso de introducir una operación no valida debe especificarse.
- ❖ Al cerrar un expendio de naranjas, 15 clientes que aún no han pagado recibirán un 15% de descuento si compran más de 10 kilos. Determinar cuánto pagará cada cliente y cuanto percibirá la tienda por esas compras.
- ❖ Supóngase que en una reciente elección hubo cuatro candidatos (con identificadores 1, 2, 3, 4). Usted habrá de encontrar el número de votos correspondiente a cada candidato y el porcentaje que obtuvo respecto al total de los votantes. El usuario proporcionará los votos de manera desorganizada y el final de datos está representado por un cero.
- ❖ Una empresa quiere hacer una compra de varias piezas de la misma clase de fábrica de refacciones. La empresa dependiendo del monto de la compra, decidirá qué hacer para pagar al fabricante.
 - Si el total de la compra excede de \$500, 000 la empresa tendrá la capacidad de invertir de su propio dinero un 55% del monto de la compra, pedir prestado al banco un 30% y el resto lo pagará solicitando un crédito al fabricante.
 - Si el monto total de la compra no excede de \$500, 000 la empresa tendrá la capacidad de invertir su propio dinero un 70% y el restante 30% lo pagará solicitando crédito al fabricante.
 - El fabricante cobrará por concepto de intereses un 20% sobre la cantidad que se le pague a crédito.
 Muestre la cantidad a invertir, el préstamo, el crédito y el interés.

Programación estructurada

Las estructuras de control son esenciales en la programación, ya que permiten dirigir el flujo de ejecución del programa. Existen tres tipos principales de estructuras de control:

- **Secuenciales:** Ejecutan un bloque de instrucciones de manera lineal, una tras otra tal como están escritas en el algoritmo.
- **Condicionales o Selectivas:** Permiten ejecutar diferentes conjuntos de instrucciones según se cumpla o no una condición específica.
- **Bucles o Repeticiones:** Habilitan la ejecución repetida de un conjunto de instrucciones, ya sea un número determinado de veces o hasta que se cumpla una condición.



Ejercicios

- ❖ Escribe un programa que pida el año actual y un año cualquiera y muestre un mensaje diciendo cuántos años faltan para llegar a ese año (si es posterior al actual), cuántos han transcurrido desde ese año (si es anterior), o si ese año es el actual.
- ❖ Una fábrica ha sido sometida a un programa de control de contaminación para lo cual se efectúa una revisión de los puntos IMECA generados por la fábrica. El programa de control de contaminación consiste en medir los puntos IMECA que emite la fábrica en cinco días de una semana y si el promedio es superior a los 170 puntos entonces tendrá la sanción de parar su producción por una semana y una multa del 50% de las ganancias diarias cuando no se detiene la producción. Si el promedio obtenido de puntos IMECA es de 170 o menor entonces no tendrá ni sanción ni multa. El dueño de la fábrica desea saber cuánto dinero perderá después de ser sometido a la revisión.
- ❖ Escribe un programa que pida un número y averigüe si es:
 - Cero, mayor o menor que cero.
 - Par o impar (cuando sea mayor que cero)
 - Múltiplo de 8 o no múltiplo de 8 (cuando sea par).
- ❖ Escribe un programa que determine la suma de los M términos de la siguiente serie:
 - a) $\text{suma} = 1! + 2! + 3! + 4! + 5! + \dots$
 - b) $\text{suma} = 5^2 / 3! + 7^4 / 5! + 9^6 / 7! + 11^8 / 9! + \dots$
 - c) $\text{suma} = 1 + 2 + 3 + 5 + 7 + 11 + 13 + \dots$
- ❖ Dado un número entero determinar la suma de sus dígitos. Por ejemplo, si el número entero es 793412 la suma de sus dígitos es 26.
- ❖ Se desea calcular el sueldo de un trabajador, a partir de las horas trabajadas en la semana y la clase a la que pertenece: Trabajadores Clase "A", se les paga \$7 por hora. Trabajadores clase "B", se paga \$5 por hora. Trabajadores clase "C", se les paga \$4 por hora y los de clase "D", \$3.5 por hora. Utilice una estructura selectiva múltiple.
- ❖ En un supermercado, se realizan descuentos por las compras a partir de unas bolitas de colores. Si el cliente saca una bolita color azul, tiene un descuento del 20%, si la bolita es roja, se aplica un descuento del 30% y si saca una bolita color blanca, no se aplica ningún descuento. Diseñe un programa que a partir del importe de la compra y el color de la bolita, muestre lo que debe pagar dicho cliente. Utilice una estructura selectiva múltiple.

Funciones

Es un conjunto de instrucciones que resuelven una tarea específica. La estructura de una función esta conformada por las partes ilustrada a continuación:

```

FUNCIÓN

tipo_retorno nombre_función ( lista de parámetros )
{
    declaraciones variables locales;
    sentencias;
    return valor;
}
  
```

Donde:

- **tipo_de_retorno:** es el tipo del valor devuelto por la función, o, en caso de que la función no devuelva valor alguno, la palabra reservada *void*.
- **nombre_de_la_función:** es el nombre o identificador asignado a la función.
- **lista_de_parámetros:** es la lista de declaración de los parámetros que son pasados a la función. Éstos se separan por comas. Debemos tener en cuenta que pueden existir funciones que no utilicen parámetros.
- **cuerpo_de_la_función:** está compuesto por un conjunto de sentencias que llevan a cabo la tarea específica para la cual ha sido creada la función.
- **return expresión:** mediante la palabra reservada *return*, se devuelve el valor de la función.

```

float precio(float base, float impuesto)
{
    float calculo;
    calculo = base + (base * impuesto);
    return calculo;
}
  
```

Ejercicios

- ❖ El número de sonidos emitidos por un grillo en un minuto, es una función de la temperatura (ver eq. 2). Construya un programa que permita calcular la temperatura en grados centígrados °C con base al número de sonidos emitidos por un grillo.

$$FA = \frac{S}{4} + 40 \text{ ----- (2)}$$

Dónde:

FA = representa la temperatura en grados Fahrenheit

S = número de sonidos emitidos por un grillo.

- ❖ Realizar un programa que lea tres números reales N1, N2 y N3 que mediante una función *media3* calcule la media de los tres números y la función mostrar para desplegar el resultado.
- ❖ Hacer un programa que dado dos puntos P(px, py) y Q(qx, qy) obtener la pendiente y la ordenada al origen de la recta $f(x) = mx + b$ que se forma con dichos puntos.
- ❖ Los vértices de un triángulo son P(px, py), Q(qx, qy) y R(rx, ry) realice un programa para obtener las ecuaciones de las rectas $Ax + By + C = 0$ que conforman los lados del triángulo.
- ❖ Realizar un programa que calcule la desviación estándar S, de cinco números reales proporcionados por el usuario. La fórmula es la siguiente:

$$S^2 = \frac{1}{n} \sum_{i=1}^5 (x_i - \bar{x})^2$$

Argumentos y parámetros por valor/referencia

En C todos los argumentos que se pasan a una función se pasan por valor. En otras palabras, se pasa una copia del valor del argumento y no el argumento en sí (por ello, este procedimiento se conoce en algunas ocasiones como **paso por copia**). Al pasar una copia del argumento original a la función, cualquier modificación que se realice sobre esta copia no tendrá efecto sobre el argumento original utilizado en la llamada de la función. Se puede considerar un argumento pasado por valor como una variable local de la función a la que se ha pasado, de tal modo que los cambios que se realicen sobre ésta tendrán efecto sólo dentro de la función.

```
#include <stdio.h>

void modificar(int variable);

void main( void){
    int i = 1;
    printf("ni=%i antes de llamar a la función modificar", i);
    modificar(i);
    printf("ni=%i después de llamar a la función modificar", i);
}
```

```
void modificar(int variable){
    printf("nvariable = %i dentro de modificar", variable);
    variable = 9;
    printf("nvariable = %i dentro de modificar", variable);
}
```

Dado que lo que se pasa a la función *modificar* es una copia de la variable *i*, el valor de ésta en la función *main* no se ve alterado cuando dentro de la función *modificar* se cambia el valor de variable. De ahí, la salida del ejemplo anterior es la siguiente:

```
i=1 antes de llamar a la función modificar
variable = 1 dentro de modificar
variable = 9 dentro de modificar
i=1 después de llamar a la función modificar
```

Cuando se pasa un argumento por valor, realmente se pasa una copia de éste, y si esta copia se modifica el argumento original no se ve alterado.

En muchas ocasiones lo que queremos es que una función cambie los valores de los argumentos que le pasamos. Para lograrlo se utiliza lo que se conoce como **paso de argumentos por referencia**. En estos casos, no se pasa una copia del argumento, sino el argumento mismo. Cuando realizamos un paso de argumentos por referencia en C, realmente lo que estamos pasando son direcciones de memoria.

```
#include <stdio.h>

void modificar(int *variable);

void main(void){
    int i = 1;
    printf("i= %i antes de llamar a la función modificar", i);
    modificar(&i);
    printf("i= %i después de llamar a la función modificar", i);
}
```

```
void modificar(int *variable){
    printf("variable = %i dentro de modificar", *variable);
    *variable = 9;
    printf("variable = %i dentro de modificar", *variable);
}
```

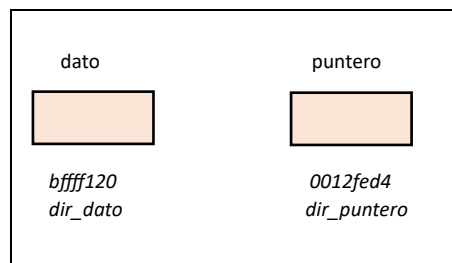
La salida de este ejemplo sería:

```
i=1 antes de llamar a la función modificar
variable = 1 dentro de modificar
variable = 9 dentro de modificar
i=9 después de llamar a la función modificar
```

Como se puede observar, el valor de *i* ha cambiado puesto que la función *modificar* ha utilizado la dirección de memoria de esta variable en la sentencia de asignación **variable = 9*. Dado que el paso de argumentos por referencia es común en C, conviene que en este punto recordemos el concepto de puntero. Consideremos las siguientes declaraciones:

```
int dato;
int * puntero;
```

La primera de las declaraciones reserva memoria para almacenar una variable de tipo entero (*int*) mientras que la segunda declaración reserva memoria para almacenar una dirección. A pesar de que apunta a una variable de tipo entero, lo que se va a almacenar es una dirección. Supongamos que el compilador reserva la dirección en hexadecimal *bffff120* para la variable *dato* y la dirección en hexadecimal *0012fed4* para puntero. A continuación, se muestra gráficamente la representación de la declaración de las variables anteriores.

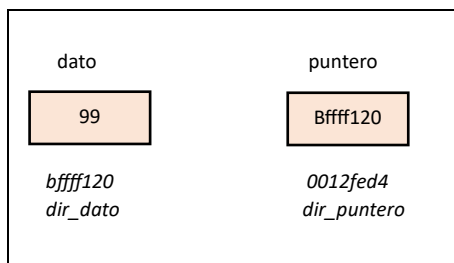


Supongamos que realizamos las siguientes asignaciones:

`dato = 99;`

`puntero = &dato;`

El resultado en la memoria se muestra a continuación:



Ejercicios Para cada inciso realice en su libreta de manera clara y entendible las cajas que ilustren la ejecución de los códigos e indique los valores de cada variable y los resultados obtenidos como salida.

A) <pre> int M(int *x){ int y = 2; -- (*x); *x %= 3; *x *= 8; return(*x * y); } void main(){ int a=5, b, c=9; b = M(&c); c = a*b; b = M(&a); printf("%i, %i, %i", a, b, c); } </pre>	B) <pre> void PV (char p){ p = 's'; } void PR (char * p){ *p = 's'; } void main(){ char pal = 'W'; printf("%c\n", pal); PV(pal); printf("%c\n", pal); PR(&pal); printf("%c\n", pal); } </pre>	C) <pre> void Mas (int, int); void main(){ int z=10, t=5; printf("%d, %d \n", z, t); Mas(z, t); printf("%d, %d \n", z, t); Mas(z, 4*t+20); printf("%d, %d \n", z, t); Mas(32+t, 20); printf("%d, %d \n", z, t); } void Mas(int x, int y){ x= x *y; } </pre>	D) <pre> int A(int x){ x *= x; return x; } int B(int x, int y){ y*=y; x*= y; return (x); } void main(){ int x=2, y=3, a, b; a = A(x) + 2; b = B(y+x, a); printf("%i, %i, %i, %i", x, y, a ,b); } </pre>
---	--	---	---

Arreglos unidimensionales y bidimensionales estáticos

Los arreglos son estructuras datos que permiten almacenar un conjunto de elementos del mismo tipo. Estas estructuras pueden ser:

Unidimensionales

Declaración

```
tipo nombre_variable [TAM];
```

tipo: determina el tipo de dato de cada elemento.
TAM: define cuantos elementos se guardarán.

Un ejemplo de declaración:

```
int numeros[6];
```

Bidimensionales

Declaración

```
tipo_dato nombre [FILAS][COLUMNAS];
```

donde :

tipo_dato: indica el tipo de elemento a almacenar.
nombre: identificador que hace referencia a la tabla.
FILAS: indica el numero de filas que contiene la tabla.
COLUMNAS: indica el número de columnas que contiene la tabla.

Operaciones

- Inserción de elementos
- Copiar arreglos
- Mostrar elementos de un arreglo
- Búsqueda secuencial
- Ordenamiento de un arreglo (inserción, selección, burbuja, Quicksort y Mergesort)
- Búsqueda binaria

Ejercicios

- ❖ Determinar el promedio de **n** notas, indicando cuantas calificaciones son aprobatorias y reprobatorias.
- ❖ Dado un arreglo **A** de **n** valores enteros que va a introducir, se quiere generar otro arreglo **B** que contenga las posiciones de los elementos del arreglo dado que sean iguales a un valor **x** indicado por el usuario. Ejemplo:
- ❖ Entrada: **A** = [4, 6, 8, 2, 6, 9, 6, 1] Salida: **B** = [2,5,7]
 $x=6$
- ❖ Elaborar un programa que solicite al usuario un arreglo **A** de **n** elementos reales para realizar los que se pide:
 - Calcule e imprima la media de los datos dada por la fórmula: $\bar{x} = \frac{1}{n} \sum_{i=0}^n A[i]$
 - Calcular la desviación estándar dada por la fórmula: $S = \sqrt{\frac{\sum_{i=0}^n (A[i] - \bar{x})^2}{n}}$
- ❖ Dadas dos matrices **A** y **B** de tamaño **nxm** realice un programa que sume los elementos de cada fila y cada columna y los deje en dos vectores.
- ❖ Realizar un programa de dadas las matrices **A** y **B** de **nxm** calcular: **A+B**, **A-B** y **A*B** dejando los resultados en una tercera matriz **C**.
 Nota: para la multiplicación debe validar el número de filas de **A** y columnas de **B**.

Datos definidos por el usuario

Las *estructuras* conocidas generalmente con el nombre de registros representan un tipo de dato estructurado. Se utilizan para resolver problemas que involucren tipo de datos heterogéneos.

```
struct nom_estructura{
    tipo_dato miembro_1;
    tipo_dato miembro_2;
    . . .
    tipo_dato miembro_n;
};
```

```
struct Libro{
    char titulo[35];
    char autor[35];
    char editorial[35];
    char DNI[30];
};
```

```
struct CD{
    char artista[30];
    int numPistas;
    float precio;
};
```

Para el acceso a una estructura se pueden realizar de dos formas: operador punto (.) y operador puntero (→)

Ejercicios

- ❖ Se tiene la siguiente definición de la estructura Punto:
- ```
struct Punto{
 float x;
 float y;
};
```

Realizar un programa aplicando la modularidad y programación estructurada que por medio de un **menú** permita:

- Solicite al usuario los valores del punto A y punto B.
- Obtenga el punto medio entre A y B.
- Obtenga la distancia entre el punto A y B

Después de realizar un cálculo el programa debe regresar al menú y el usuario podrá elegir otra operación con los mismos puntos A y B hasta que decida terminar el programa.

- ❖ Se necesita un programa para **gestión de una agencia matrimonial**. Por cada cliente se almacenarán los siguientes datos: **Nombre, ap\_paterno, edad, sexo (M, F), aficiones (Lectura, Viajes, Deportes, Cine, Gastronomía, Juegos de mesa y Perros)**. Una persona puede tener ninguna, una o más aficiones. Se pide en forma de menú las siguientes funciones:

- Función para introducir los datos de una persona.
- Buscar una persona (por el nombre).
- Eliminar una persona (se localiza por el nombre).

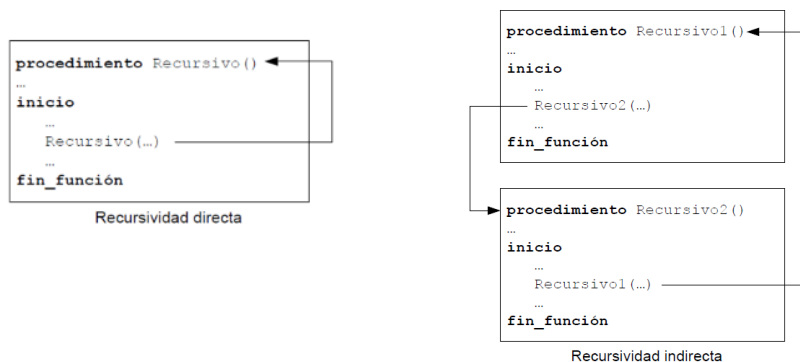
- ❖ Se tiene la siguiente definición de la estructura Estudiante  
Nombre (cadena de caracteres tamaño 25)  
Boleta (entero)  
Semestre (cadena de caracteres tamaño 15)  
Sexo (caracter F/M)  
Calificaciones (arreglo de decimales tamaño 5)

Realizar un programa aplicando la modularidad y programación estructurada que:

- Solicite al usuario los datos correspondientes al estudiante.
- Calcular el promedio obtenido de sus calificaciones.
- Obtener la mayor calificación considerando el caso cuando se tienen notas iguales.

## Recursión

Las funciones en C pueden ser recursivas, en otras palabras, pueden llamarse a sí mismas directa o indirectamente. La recursividad directa es el proceso mediante el que una función se llama a sí misma desde el propio cuerpo de la función, mientras que la recursividad indirecta implica más de una función. Un proceso recursivo debe tener una condición de finalización (caso base), ya que de lo contrario podría continuar infinitamente.



Un ejemplo típico de aplicación de la recursividad es el cálculo del factorial de un número entero. Recordemos que el factorial de un número entero (n!) se calcula de la siguiente manera:  $n! = n * (n-1) * (n-2) * \dots * 2 * 1$

En principio, la solución a este problema podría realizarse sin tener que utilizar la recursividad con el siguiente programa:

```
#include <stdio.h>
int factorial(int numero);
void mostrar (int resultado);

void main(void){
 int resultado, valor = 4;
 resultado = factorial(valor);
 mostrar(resultado);
}
```

```
int factorial(int numero){
 int i, devuelve = 1;

 for(i = 1; i <= numero; i++){
 devuelve = devuelve * i;
 }
 return devuelve;
}

void mostrar (int resultado){
 printf("El factorial de %i es %i\n", valor, resultado);
}
```

Sin embargo, resulta más intuitivo dada la definición de número factorial utilizar una función recursiva como la siguiente:

```
int factorial(int numero){
 if(numero == 1) // Caso base
 return 1;
 else
 return (numero * factorial(numero-1)); // Caso general
}
```

En la función anterior, en el caso de que el argumento utilizado en la llamada sea  $1$ , ésta devuelve  $1$ , y en caso contrario se calcula un producto que involucra a la variable *numero* y una nueva llamada a la función cuyo argumento es menor en una unidad (*numero - 1*).

En muchas ocasiones, la resolución de un problema mediante una función recursiva resulta conceptualmente más clara que la resolución mediante una función iterativa. Sin embargo, la función iterativa resulta algo más compleja. Es evidente que hay tareas que se pueden resolver mediante funciones recursivas o funciones iterativas, aunque es el programador el que tiene que optar por una solución u otra.

## Ejercicios

- ❖ Escribir una función que solicite al usuario una cadena de caracteres y una función recursiva llamada vocales que devuelva el número de vocales contenidas en la cadena recibida como argumento.
- ❖ Un palíndromo es una palabra que se lee exactamente igual en un sentido o en otro. Escribir un programa que pida al usuario la palabra a analizar y una función recursiva que devuelva 1, si la palabra recibida como argumento es un palíndromo y devuelva 0 en caso contrario.
- ❖ Escribir un programa que tenga como entrada un número entero positivo (mediante una variable entera). El programa debe hallar la suma de los dígitos de cada entero y encontrar cual es el entero cuya suma de dígitos es mayor. La suma de dígitos se ha de realizar con una función recursiva.
- ❖ Haga un programa que solicite al usuario caracteres y los almacene en una un arreglo y una función recursiva para buscar un elemento particular en la lista.
- ❖ Realizar un programa que contenga una función recursiva, que reciba un arreglo de enteros y regrese el último elemento almacenado en él.
- ❖ La expresión matemática  $C(m,n)$  en el mundo de la teoría combinatoria de los números, representa el número de combinaciones de  $m$  elementos tomados de  $n$  en  $n$  elementos.

$$C(m,n) = \frac{m!}{n!(m-n)!}$$

- ❖ Escribir un programa cuyas entradas sean los enteros  $m$ ,  $n$  y calcular  $C(m,n)$  donde  $n!$  es el factorial de  $n$ .

## Manejo de memoria

Para la gestión de la memoria dinámica en C se usan varias funciones cuyos prototipos están declarados en la biblioteca **stdlib.h**, estas funciones son:

```
void * malloc(sizeof(tipo_dato));
void * calloc(size_t numelem, sizeof (tipo_de_dato));
void * realloc(void * ptr, sizeof (tipo_de_dato));
void free(void *ptr);
```

## Ejercicios

- ❖ Realizar un programa que dada una matriz creada dinámicamente **R** de **nxm** con elementos de tipo carácter proporcionados por el usuario. Obtenga la matriz transpuesta también creada dinámicamente **RT** e imprima el contenido de ambas matrices (**R** y **RT**).
- ❖ Hacer un programa que dada una matriz **R** de **nxm** creada dinámicamente con elementos tipo decimales determine:
  - Si es una **matriz nula** (todos los elementos son cero).
  - Obtener  $K \cdot R$  y  $R/k$ , donde  $K$  es una constante proporcionada por el usuario.
  - Si es una **matriz triangular superior** (matriz cuadrada y que todos los elementos de la diagonal principal inferior son cero).
- ❖ Realice un programa que obtenga el producto vectorial de dos vectores  $V_1$ , y  $V_2$  creados dinámicamente cuyos componentes son tres elementos. El programa debe mostrar los dos vectores y el vector resultante.
- ❖ Construya un programa que a través de un menú:

1. **Crear dinámicamente** el vector de **N** componentes
2. Solicite el número de elementos a guardar en el vector.
3. Introduzca los elementos al vector
4. Ordenar los elementos en orden creciente.
5. Ordenar los elementos en orden decreciente.
6. Mostrar los elementos del vector
7. Salir

Después de realizar alguna operación elegida debe regresar al menú hasta que el usuario seleccione salir.

- ❖ Defina la estructura **Proveedor** que contiene los siguientes elementos:
  - Nombre y apellido (cadena de caracteres)
  - Edad: entero
  - Producto (estructura anidada estática) que contiene: Nombre (cadena de caracteres), Cantidad vendida del artículo (entero) y Precio unitario (decimal).
  - Telefonos (estructura anidada estática )

Realizar un programa aplicando la programación modular y estructurada que por medio de un menú permita:

1. Que solicite la cantidad **N** de proveedores a ingresar.
2. Que permita crear dinámicamente un arreglo de **N** proveedores.
3. Solicite al usuario los datos de los proveedores.
4. Obtenga y muestre los **nombres y teléfono de oficina de los proveedores** cuyo artículo **es el mayor vendido** considerando si hay más productos que cumplan la condición.
5. Salir del programa

Después de realizar alguna acción el programa debe regresar al menú y el usuario podrá elegir otra acción hasta que decida salir del programa.

- ❖ Defina la estructura **Libro** que contiene los siguientes elementos:
  - Código (entero)
  - Título (cadena de caracteres)
  - Estado del libro (cadena de caracteres: “préstamo” o “devolución”)
  - Dirección (estructura anidada estática)

- Alumno (estructura anidada estática) que contiene: Boleta (entero), Nombre y apellidos (cadena de caracteres) y Nivel (entero).

Realizar un programa aplicando la modularidad y programación estructurada que por medio de un menú permita:

1. Que solicite la cantidad N de libros a ingresar.
2. Que permita crear dinámicamente un arreglo de N libros.
3. Solicite al usuario los datos de los libros.
4. Obtenga y muestre los **nombres y código postal de los alumnos** cuyo **título del libro** sea “Algebra Lineal” considerando el caso que no exista desplegar mensaje.
5. Salir del programa

Después de realizar alguna acción el programa debe regresar al menú y el usuario podrá elegir otra acción hasta que decida salir del programa.

## Archivos

Es un conjunto de bits almacenado en un dispositivo de memoria secundaria que puede contener ciertas propiedades y ser recuperado de la misma manera por el sistema operativo para que un programa tenga acceso a este.

Hay dos tipos de archivos:

- **archivos de texto:** es una secuencia de caracteres organizadas en líneas terminadas por un carácter de nueva línea. Estos archivos pueden almacenar fuentes de programas, texto plano, bases de datos simples, etc.
- **archivos binarios:** es una secuencia de bytes que tienen una correspondencia uno a uno con un dispositivo externo. Algunos ejemplos son archivos de fotografías, imágenes de texto, archivos ejecutables, etc.

### Abrir y cerrar un archivo

```
FILE *fopen (const char nombre_archivo, const char modo);
int fclose(FILE *F);
```

### Manejo de archivos en modo texto

```
int fprintf(FILE *F, const char *cadena_de_control,);
int fscanf(FILE *F, const char *cadena_de_control,);
char *fputs(char *str, FILE *F);
char *fgets(char *str, int long, FILE *F);
```

### Manejo de archivos en modo binario

```
size_t fread (void * ptr, size_t size, size_t count, FILE * archivo);
size_t fwrite(void *ptr, size_t tamaño, size_t count, FILE * archivo);
```

## Ejercicios

- ❖ Escribe un programa para leer un archivo de texto carácter por carácter y contar el número total de caracteres que contiene, incluyendo espacios y saltos de línea.
- ❖ Crea un programa para copiar todo el contenido de un archivo de origen (.txt) a un archivo de destino (.txt).
- ❖ Lea la secuencia de números enteros del archivo (.txt) creado previamente, calcule y muestre su suma total.
- ❖ Defina una estructura **Estudiante** que contenga los siguientes campos: nombre(cadena de caracteres), matricula (entero) y nota (flotante). Escriba el registro de un solo estudiante recibido del usuario en un **archivo binario**.

- ❖ Una institución educativa desea almacenar las calificaciones de sus alumnos en un archivo de texto llamado **calificaciones.txt**. Cada línea del archivo tendrá el siguiente formato:
  - 1001 Juan 8.5
  - 1002 Maria 9.7
  - 1003 Pedro 6.8
 Desarrolle un programa que permita:
  1. Registrar N alumnos en el archivo.
  2. Mostrar el contenido completo del archivo.
  3. Calcular y mostrar:
    - Promedio general.
    - Calificación más alta.
    - Calificación más baja.
    - Número de alumnos aprobados (calificación  $\geq 6$ ).
- ❖ Una tienda desea almacenar la información de sus productos en un archivo binario llamado **productos.dat**. Cada producto contiene:
  - Código
  - Nombre
  - Precio
  - Existencias
 El programa deberá mostrar un menú con las siguientes opciones:
  1. Agregar productos.
  2. Mostrar todos los productos.
  3. Buscar un producto por código.
  4. Incrementar en un 10% el precio de todos los productos.
  5. Mostrar el valor total del inventario (precio  $\times$  existencias).

## Bibliografía

- Joyanes L. Fundamentos generales de programación. Mc Graw-Hill Interamericana, 2013
- Reese, R. Understanding and using C pointers. O' Reilly, 2013
- Kernighan, B. & Ritchie, D. Lenguaje de programación. Prentice-Hall, 1991
- Joyanes L. Programación en C, C++, Java y UML. Mc Graw-hill, 2014
- Sznajdleder, P,. Programación estructurada a fondo. Algaomega, 2017
- Ceballos, Francisco j. C/C++ Curso de programación, 2da.Edición. RA-MA, 2002
- Gottfried, Byron. Programación en C. Mc Graw-Hill, 1991
- Cairó, Osvaldo. Fundamentos de programación. Piensa en C. 1ª Edición. Pearson Educación, 2006
- García Bernejo, J. R. Programación estructurada en C. Pearson, 2008.