

PROGRAMACION 3ER PARCIAL TEORIA

RECURSIVIDAD

La recursividad es una técnica de programación en la que **una función se llama a sí misma para resolver un problema más pequeño** del mismo tipo.

Una función recursiva tiene dos partes importantes:

1. **Caso base (condición de parada):** Una solución simple para un caso particular (puede haber más de una) evita que la función se llame infinitamente.
2. **Llamada recursiva o Caso Recursivo:** una solución que involucra utilizar la función original, con parámetros que se acercan más al caso base. Los pasos que sigue el caso recursivo son los siguientes:
 - a. El procedimiento se llama a sí mismo.
 - b. El problema se resuelve, resolviendo el mismo problema pero de tamaño menor.
 - c. La manera en la cual el tamaño del problema disminuye asegura que el caso base eventualmente se alcanzará.

¿Qué pasa si NO hay caso base?

La función se llamaría a sí misma para siempre → se produce un desbordamiento de pila (stack overflow) y el programa se cae.

¿Cuándo usar recursividad?

Cuando:

- El problema se puede dividir en subproblemas más pequeños.
- No se necesita guardar muchos resultados intermedios (aunque esto puede optimizarse con técnicas como "memoización").
- Te conviene escribir menos código y más elegante (por ejemplo, para árboles, fractales, estructuras jerárquicas).

Cuidados con la recursividad

1. Siempre debe tener un caso base que detenga las llamadas.
2. Puede ser más lenta que las soluciones iterativas si no se optimiza.
3. Consume más memoria por las llamadas anidadas en la pila.

Ejemplo:

```
#include <stdio.h>

int factorial(int n) {
    // Caso base: si n es 0, el factorial es 1
    if (n == 0) {
        return 1;
    }
    // Caso recursivo: n! = n * (n-1)!
    else {
        return n * factorial(n - 1);
    }
}

int main() {
    int numero = 5;
    int resultado = factorial(numero);
    printf("El factorial de %d es %d\n", numero, resultado);
    return 0;
}
```

LISTAS

Las ESTRUCTURAS DE DATOS DINÁMICAS son aquellas en las que el tamaño podrá modificarse durante la ejecución del programa.

PUNTERO

Un puntero es una variable que contiene la dirección de memoria de otra variable, es decir apunta o referencia a una ubicación de memoria en la cual hay datos.

Es un tipo de dato que “apunta” a otro valor almacenado en memoria.

ESTRUCTURA DE DATOS LINEALES

Las estructuras de datos lineales son aquellas en las que los elementos ocupan lugares sucesivos en la estructura y cada uno de ellos tiene un único sucesor y un único predecesor, es decir, sus elementos están ubicados uno al lado del otro relacionados en forma lineal.

Hay tres tipos de estructuras de datos lineales:

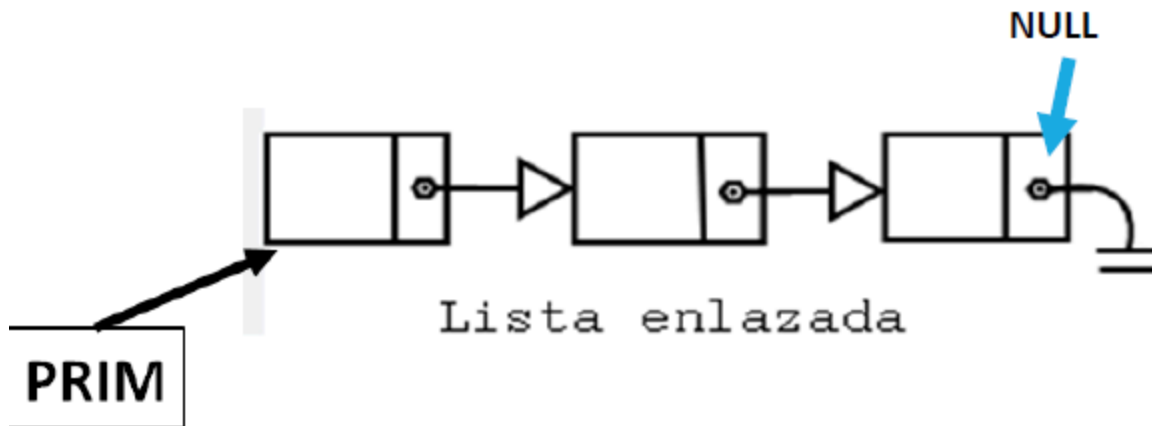
- **Listas enlazadas**

- **Pilas**

- **Colas**

LISTAS ENLAZADAS

Una Lista simplemente enlazada (también llamada lista lineal o lista enlazada simple) es una estructura de datos lineal y dinámica que utiliza un conjunto de nodos para almacenar datos en secuencia, enlazados mediante un apuntador o referencia.



- *La lista simple tiene un apuntador inicial*
- *El último nodo de la lista apunta a nulo*

Almacenamiento de los datos en una lista simple

•Ordenados:

Se recorren lógicamente los nodos de la lista simple

Se ubica la posición definitiva de un nuevo nodo. Se mantiene el orden lógico de los datos.

•Desordenados:

El nuevo dato se agrega al final de la lista simple

Operación Insertar al principio de la lista

nuevo (p);

si (p = null) entonces escribir ("Error");

sino

t:=prim;

escribir ("Ingrese dato");

Leer (dato);

p.valor:=dato;

p.proximo:=prim;

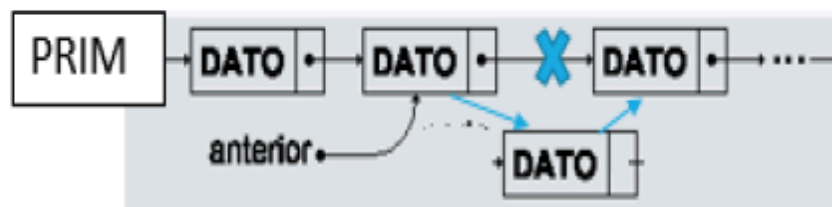
prim:= p;

Operacion Insertar al final de la lista

```
nuevo (p);  
si (p = null) entonces escribir ("Error");  
sino  
    t:=prim;  
    mientras (t<> Null) hacer  
        aux:= t;  
        t:=*t.proximo;  
    fin mientras  
  
    escribir ("Ingrese dato");  
    Leer (dato);  
    *p.dato:=dato;  
    *p.proximo:=null;  
    *aux.proximo:= p;
```

Operacion insertar al medio de la lista

```
nuevo (p);  
si (p = null) entonces escribir ("Error");  
sino  
    escribir ("Ingrese dato a insertar");  
    Leer (dato);  
    t:=prim;  
  
    mientras (t<> Null )  $\wedge$  (*t.valor < dato) )hacer  
        aux:= t;  
        t:=*t.proximo;  
    fin mientras  
  
    si (t=null) entonces escribir ("No se encontró un valor menor al ingresado");  
    sino  
        *p.valor:=dato;  
        *p.proximo:=t;  
        *aux.proximo:= p;  
    fin si
```



Eliminar elemento al principio de la lista

```
t:=prim;  
prim:=*t.proximo;  
disponer (t);
```

Eliminar ultimo elemento de la lista

```
escribir (“Encontrar el ultimo elemento de la lista”);  
t:=prim;  
mientras (*t.proximo<> Null) hacer  
    aux:= t;  
    t:=*t.proximo;  
fin mientras  
  
*aux.proximo:= null;  
disponer(t);
```

Eliminar elemento en el medio de la lista

```
escribir (“Ingrese dato a eliminar”);  
leer (dato);  
t:=prim;  
escribir (“Buscar elemento en la lista”);  
mientras (t<> Null ) ^ (*t.valor <> dato) )hacer  
    aux:= t;  
    t:=*t.proximo;  
fin mientras
```

```
si (t=null) entonces escribir (“No se encontró el elemento en la lista”);  
sino  
    *aux.proximo:= *t.proximo;  
    disponer(t);  
fin si
```

MODULARIZACION

La **modularización** es dividir un programa grande en partes más pequeñas y organizadas, llamadas módulos. Cada módulo se escribe en archivos diferentes, y normalmente se separan en:

- .h → archivos header (declaraciones).
- .c → archivos de implementación (código de funciones).
- main.c → archivo principal que usa los módulos.

¿Por qué usar modularización?

- Hace el código más ordenado y fácil de entender.
- Permite reutilizar funciones en otros proyectos.
- Facilita el trabajo en equipo.
- Es más fácil de probar y mantener.

¿Por qué se usan archivos .h y .c con el mismo nombre?

En C, el archivo .h (header) y el archivo .c (implementación) suelen tener el mismo nombre porque trabajan juntos como un módulo. Pero no es obligatorio que tengan el mismo nombre; simplemente es una buena práctica para mantener tu código organizado y fácil de entender.

¿El orden afecta?

En la mayoría de los casos, NO afecta el orden:

Porque gcc primero compila todos los .c por separado, y luego los enlaza juntos en el ejecutable.

Es decir:

```
gcc main.c input.c operaciones.c -o programa
```

es igual que:

```
gcc operaciones.c input.c main.c -o programa
```

Siempre que todas las dependencias estén satisfechas, el programa va a compilar igual.

PILAS (STACK)

Una pila es una estructura de datos dinámica que **El último que entra es el primero que sale.** (LIFO)

COLAS

Una **cola (queue)** es una estructura de datos donde:

*El primero que entra es el primero que sale(FIFO).***

Se usan punteros y estructuras para hacerla crecer y reducirse dinámicamente.

Operaciones básicas en una cola

1. **Encolar (enqueue)**: agregar un elemento al final.
2. **Desencolar (dequeue)**: sacar el primer elemento (el que está al frente).
3. **Ver frente (peek o front)**: ver quién está primero (sin sacarlo)

PREGUNTAS CUESTIONARIO

- Cuando el buffer se llena o se vacía se actualizan los datos desde y hacia el archivo.

Seleccione una:

Verdadero ☐

Falso

- El hecho de utilizar un buffer significa que se tiene acceso directo al archivo y que cualquier operación que se desee realizar (lectura o escritura) va a ser hecha sobre el buffer .

Seleccione una:

Verdadero

Falso ☐

- En cuanto a los valores permitidos para los bytes, se puede añadir otro carácter a la cadena de modo

- t modo texto Normalmente es el modo por defecto,se suele omitir.

- b modo binario

Seleccione una:

Verdadero ☐

Falso

- Dada la siguiente proposición:

Es un conjunto de datos estructurados en una colección de entidades elementales o básicas denominadas registros que son de igual tipo y constan a su vez de diferentes entidades de nivel más bajos denominadas campos.

Indicar a que concepto corresponde la misma:

La respuesta correcta es: **Archivo**

- Los métodos de acceso a los archivos puede ser directo o indirecto o secuencia.

Seleccione una:

Verdadero

Falso

- Según el tipo de elementos del archivo, este se puede clasificar en:

Archivo binario y de texto.

- En C todas las operaciones que se realizan sobre archivos son hechas a través de funciones ,existen 2 categorías de funciones para trabajar con archivos y son las que usan “ y las que acceden directamente al archivo

Seleccione una:

Verdadero ☐

Falso

- Un problema que pueda ser definido en función de su tamaño, sea este N, pueda ser dividido en instancias más pequeñas ($< N$) del mismo problema y se conozca la solución explícita a las instancias más simples, lo que se conoce como casos base, se puede aplicar inducción sobre las llamadas más pequeñas y suponer que estas quedan resueltas.

Verdadero