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 solucion simple para un caso particular(puede haber mas 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 mas al caso base. Los pasos que sigue el caso recursivo son los siguientes:
   a. El procedimiento se llama a si 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.

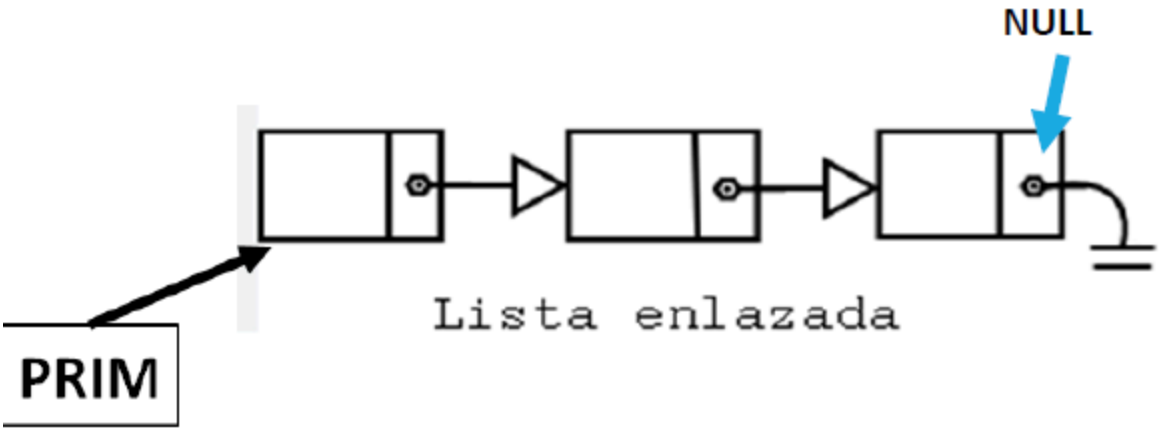
ESTRUCTURA DE DATOS LINEALES

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

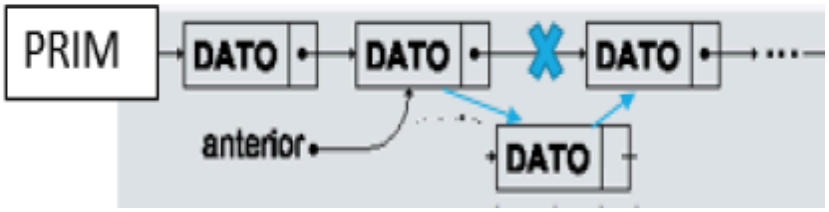
```
nuevo (p);
si (p = null) entonces escribir (“Error”);
sino
t:=prim;
escribir (“Ingrese dato”);
Leer (dato);
p.valor:=dato;
p.proximo:=prim;
prim:= p;
```

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 ) ∧ (*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 arhivos 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

Archivos
Un archivo es un espacio en el disco donde se puede guardar información. En lugar de guardar los datos sólo en la memoria (que se pierde cuando termina el programa), los archivos permiten leer y escribir datos que permanecen guardados.

Tipos de operaciones con archivos:
En C, se pueden hacer principalmente 4 operaciones con archivos:

1. Abrir un archivo
2. Leer un archivo
3. Escribir en un archivo
4. Cerrar el archivo

Todo esto se hace con la ayuda de la biblioteca estándar stdio.h y el tipo FILE.

Abrir un archivo:
Para trabajar con un archivo, primero se declara un puntero de tipo FILE y se usa la función fopen().

Modos de apertura de archivos

Modo	Significado
`"r"`	Leer (el archivo debe existir)
`"w"`	Escribir (crea el archivo o lo sobreescribe si ya existe)
`"a"`	Agregar (append) al final del archivo
`"r+"`	Leer y escribir (el archivo debe existir)
`"w+"`	Leer y escribir (borra el contenido anterior)
`"a+"`	Leer y agregar (no borra el contenido anterior)

Funciones útiles

Función	Descripción
`fprintf(FILE*, ...)`	Escribir texto formateado
`fscanf(FILE*, ...)`	Leer texto formateado
`fgets(cadena, tamaño, archivo)`	Leer línea completa
`fputs(cadena, archivo)`	Escribir línea
`fgetc(archivo)`	Leer un solo carácter
`fputc(caracter, archivo)`	Escribir un solo carácter
`feof(archivo)`	Devuelve verdadero si se llegó al final del archivo
`fclose(archivo)`	Cierra el archivo

1. Persistencia de Datos:

- Los programas tradicionales en C (sin archivos) almacenan datos solo en memoria RAM.
- Estos datos se pierden al finalizar la ejecución del programa, lo que obliga a recargar la información en cada ejecución.
- Solución: Almacenar datos de forma permanente en un medio externo (ej. disco rígido) a través de archivos.

2. Concepto de Archivo

- Un archivo de datos es un conjunto de registros relacionados, tratados como una unidad y almacenados en un dispositivo externo.
- Los archivos están estructurados como una colección de registros, todos del mismo tipo.
- Cada registro se compone de campos, que son entidades de nivel más bajo (datos individuales).

3. Tipos de Archivos (Según Método de Acceso)

- Archivos Secuenciales:
    - Acceso a los datos en el orden en que fueron grabados (desde el principio al fin).
    - Para acceder a un registro específico, se deben leer todos los registros anteriores.
    - No se puede insertar o borrar un registro sin reescribir todo el archivo.
    - Útiles para lectura completa o procesamiento lineal.
- Archivos de Acceso Directo (o Aleatorio):
    - Permiten acceder directamente a cualquier registro sin leer los anteriores.
    - Ideal para actualizar, eliminar o agregar registros de forma individual.
    - Mayor velocidad de acceso para registros específicos.

4. Operaciones Básicas con Archivos en C Para manipular archivos en C, se necesitan las siguientes funciones:

- Apertura de un Archivo (fopen()):
    - Establece una conexión entre el programa y el archivo físico.
    - Devuelve un puntero a FILE (tipo de dato FILE *), que es esencial para todas las operaciones posteriores.
    - Especifica el modo de apertura:
        - "r": Abrir para lectura (el archivo debe existir).
        - "w": Abrir para escritura (crea el archivo o lo sobrescribe si existe).
        - "a": Abrir para añadir (añade al final, crea si no existe).
        - "r+": Abrir para lectura y escritura (el archivo debe existir).
        - "w+": Abrir para lectura y escritura (crea o sobrescribe).
        - "a+": Abrir para lectura y añadir (añade al final, crea si no existe).
    - Manejo de errores: Si fopen() falla (ej. archivo no encontrado en modo "r"), devuelve NULL. Es crucial verificar esto.

- Cierre de un Archivo (fclose()):
    - Libera el puntero FILE y asegura que todos los datos en búfer se escriban en el archivo físico.

    - Fundamental para evitar corrupción de datos y liberar recursos.

5. Lectura y Escritura de Caracteres
- fputc(caracter, puntero_a_archivo): Escribe un solo carácter en el archivo.
- fgetc(puntero_a_archivo): Lee un solo carácter del archivo. Devuelve EOF (End Of File) al final del archivo.

6. Lectura y Escritura de Cadenas
- fputs(cadena, puntero_a_archivo): Escribe una cadena de caracteres en el archivo.
- fgets(buffer, tamaño_max, puntero_a_archivo): Lee una cadena del archivo, hasta tamaño_max - 1 caracteres o hasta un salto de línea.

7. Lectura y Escritura Formateada
- fprintf(puntero_a_archivo, formato, ...): Escribe datos formateados en el archivo (similar a printf).
- fscanf(puntero_a_archivo, formato, ...): Lee datos formateados del archivo (similar a scanf).

8. Lectura y Escritura de Bloques de Datos (Binarios)
- fwrite(buffer, tamaño_elemento, num_elementos, puntero_a_archivo): Escribe bloques de datos binarios (ej. estructuras completas).
    - buffer: Puntero al inicio de los datos a escribir.
    - tamaño_elemento: Tamaño de un elemento en bytes (ej. sizeof(struct MiStruct)).
    - num_elementos: Número de elementos a escribir.
- fread(buffer, tamaño_elemento, num_elementos, puntero_a_archivo): Lee bloques de datos binarios.

9. Posicionamiento en Archivos (Acceso Aleatorio)
- fseek(puntero_a_archivo, desplazamiento, origen): Mueve el puntero de posición dentro del archivo.
    - desplazamiento: Número de bytes a mover.
    - origen: SEEK_SET (desde el inicio), SEEK_CUR (desde la posición actual), SEEK_END (desde el final).
- ftell(puntero_a_archivo): Devuelve la posición actual del puntero en bytes desde el inicio del archivo.
- rewind(puntero_a_archivo): Mueve el puntero de posición al inicio del archivo.

10. Detección de Fin de Archivo (EOF)
- feof(puntero_a_archivo): Devuelve un valor distinto de cero si se ha alcanzado el fin del archivo.

11. Punteros en el contexto de archivos son cruciales porque:
- FILE * es un puntero a la estructura que C usa para gestionar el archivo.
- Las funciones de lectura/escritura de bloques (fread, fwrite) operan con punteros a las estructuras o arrays en memoria.