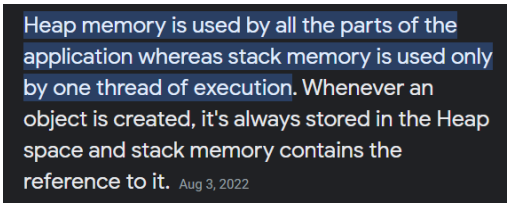
2️⃣ Modulo 2

$$MemoriaDinamica$$

✅ `return ≠ 0` es sinonimo de error (por convencion)

Funciones a utilizar en `<stdlib.h>`:

- **malloc( ):** *intenta* alocación contigua de bytes en la memoria **HEAP**



  - Retorna un puntero del tipo void * el cual es el inicio en memoria de la porción reservada de tamaño size.
  - Si no puede reservar esa cantidad de memoria la función regresa un puntero nulo o NULL.
- **free( ):** libera memoria reservada
  - Si el puntero no es válido o si el espacio ha sido liberado previamente, el comportamiento de la función no está definido.

```
void * malloc( size_t size );
palabra = (char *) malloc(15);

void free( void * ptr );
free(palabra);

// Arreglos Dinamicos
// Para acceder al 4to. elemento se puede utilizar ptr[3] o *(ptr+3)
int *ptr;
ptr = (int *) malloc(100 * sizeof(int) );
```

✅ Es usual usar la función `sizeof()` para indicar el número de bytes a alocar para hacer código independiente del dispositivo, favoreciendo la alta cohesión y el bajo acoplamiento de nuestro programa.

- **calloc( )**: *contiguous/cleared* allocation
  - Reserva espacio para un arreglo de N objetos, cada cual tiene un tamaño size de bytes. El espacio es inicializado a cero todos los bits.

    - Retorna un puntero del tipo void * con la dirección inicial del bloque de memoria reservado.
    - Si no puede reservar esa cantidad de memoria la función regresa un puntero nulo o NULL.
- **realloc( )**:
    - Cambia el tamaño del objeto apuntado por ptr al tamaño especificado por size.
    - El contenido del objeto no cambiará hasta el menor de los tamaños nuevo y viejo.
    - Si el tamaño nuevo es mayor, el valor de la porción nuevamente adjudicada del objeto es indeterminado.
    - Si size es cero y ptr no es nulo, el objeto al que apunta es liberado.

```
void *calloc(size_t N, size_t size);
void *realloc(void *ptr, size_t size);

// Se introducen los conceptos de lista y pila de memoria dinamica.
struct nodo {
  int valor;
  struct nodo *ptr;
}
struct nodo *pila = NULL, *aux;
```

En C, todos los parametros son por **valor**.

Ambas notaciones nos permiten hacer un scanf de los elementos de un arreglo: *&(a[i])* y *&a[i]*

Tipos de arreglos

ANSI C90 define **3 tipos** de arreglos.

ANSI C99 incorpora un **nuevo** tipo → arreglos de **longitud variable**

- la longitud no es una expresión constante, **no se puede determinar en compilación**
- alocación: cuando el programa alcanza la declaración.
- desalocación: cuando finaliza el bloque de código donde se encuentra.
- comparación con **arreglos automáticos:**

  = una vez asignada la memoria, no es realocable

  = el programador no puede liberar la memoria explícitamente

  + la alocación se demora hasta que la ejecución alcance la declaración del arreglo
- comparación con **arreglos estáticos**:

  + la desalocación de memoría se realiza cuando termina el programa (estáticos) y cuando se desactiva el bloque de código donde se encuentra (longitud variable)
- comparación con **arreglos dinámicos**:

  = no es necesario conocer el tamaño en tiempo de compilación

  + los arreglos dinámicos puede cambiar su tamaño en ejecución

+ el programador puede liberar explícitamente la memoria

+ arreglos dinámicos se alocan en la **HEAP** (permite mayor tamaño) mientras que los demás se alocan en la pila de ejecución STACK

```
// AUTOMÁTICOS
int a[10];

// ESTÁTICOS
static int a[10];

// DINÁMICOS
int * a = (int *) malloc(10 * sizeof(int));

// LONGITUD VARIABLE
int n;
scanf("%d", &n);
int a[n];
```

Una clase de almacenamiento define el **alcance** (**visibilidad**) y el **tiempo de vida** de una variable dentro de un programa.

- Alcance: donde puede ser referenciada (global vs. local)
- Tiempo de vida: persistencia (estatica vs. automatica)

Archivos

- La entrada/salida se maneja por **flujos**/**streams** (secuencias de bytes)
- Flujos automaticos al inicio del programa:
  - flujo estandar de entrada al teclado
  - flujo estandar de salida a la pantalla
  - flujo estandar de error a la pantalla
- Son flujos **redireccionables**
- Identificadores especiales de tipo `FILE *` → stdin, stdout, stderr

Funciones de manejo de archivos en `<stdio.h>` (trata a dispositivos como archivos)

- **fopen( )**:
  - abre archivo indicado por la cadena *nombre*
  - asigna un flujo al puntero
  - si falla retrona `NULL`
- **fclose( )**:
  - se desasocia identificador pasado como parametro del archivo en cuestion
  - retorna cero si el archivo fue cerrado con éxito
  - si se detectaron errores, entonces retorna EOF

- **fprintf( )**:
  - envía datos al stream apuntado, bajo el control que especifica cómo los argumentos posteriores son convertidos para la salida
  - comportamiento no definido si no hay suficientes argumentos
  - si el formato termina y restan argumentos, estos son evaluados y posteriormente ignorados
  - retorna el control cuando el final de la cadena de formato es encontrado
  - retorna el numero de caracteres transmitidos, o un valor negativo en caso de error

```
FILE * miArchivo = fopen(const char *nombre, const char *modo);

if (miArchivo == NULL) {
  fprintf(stdout, "Error al abrir el archivo!\n");
  return 1;
}

int resultado = fclose(FILE * miArchivo);
int caracteresTransmitidos = fprintf(FILE * arch, const char *formato, ...);
```

✅ Buena practica verificar si hubo error al abrir el archivo y cuando se termina de utilizar el archivo hay que cerrarlo.

Modos de apertura de archivos

Modo	Descripción
r	Abrir un archivo para lectura.
w	Crear un archivo para escritura. Si el archivo ya existe, se descarta el contenido actual.
a	Abrir o crear un archivo para escribir al final del mismo
r+	Abrir un archivo para lectura y escritura.
w+	Genera un archivo para lectura y escritura. Si el archivo ya existe, se descarta el contenido actual.
a+	Abrir o crear un archivo para actualizar. La escritura se efectuará al final del archivo.

- **fscanf( )**:
  - equivalente a `scanf` pero toma informacion de un archivo en vez de teclado
  - recibe datos del stream apuntado y los muestra por pantalla
  - bajo el control indicado (especifica secuencias de entrada permitidas y como han de ser convertidas para la asignacion)
  - retorna cuando el final de la cadena de formato es encontrado

> 💡 No necesariamente es el caracter especial `\n`

  - retorna el numero de datos de entrada asignados (0 en caso de error)
  - si un error de entrada ocurre antes de cualquier conversion, la funcion `fscanf` retorna `EOF`

- **feof( )**:
  - indica si la última operación realizada sobre el archivo excedió el final de este
  - retorna un número ≠ 0 (TRUE) si el archivo terminó y 0 (FALSE) en otro caso

> 💡 Indica si ya se realizó una operación fuera del límite del archivo; no si se encuentra posicionado en el límite del archivo

- **fgetc( )**:
  - lee caracter (si existe) en el stream apuntado
  - retorna un `unsigned char` convertido a `int`
  - si hubo error retorna EOF
  - `fgetc(stdin)` equivalente a `getchar()`
  - **Indicador de posicion** de ficheros *asociado al stream* es **incrementado** una posicion (si esta definido)
- **fputc( )**:
  - escribe el carácter indicado (convertido a un unsigned char) al stream de salida apuntado
  - en la posicion indicada por indicador de posicion de ficheros asociado al stream
  - avanza el indicador
  - retorna el caracter escrito (si hubo error de escritura retorna EOF)
  - `fputc('a', stdout)` equivalente a `putchar('a')`
- **fgets( )**:
  - lee como maximo `n - 1` caracteres desde el stream apuntado al array apuntado
  - no se leen caracteres luego del salto de linea `\n` (el cual es incluido) o luego de un final de fichero EOF
  - un caracter nulo `\0` es escrito inmediatamente despues del ultimo caracter leido en el array
  - si no lee nada retorna NULL

> 💡 Caracter nulo `\0` ≠ Salto de linea `\n`

- **fputs( )**:
  - el caracter nulo **no** es escrito
  - retorna valor positivo si no hubo errores, sino retorna NULL
  - `fputs(s, stdout)` equivale a `puts(s)`

💡 Confusingly, `gets` deletes the terminating `'\n'`, and `puts` adds it.

```
int feof(FILE *stream);
int fscanf(FILE *stream, const char *formato, ...);
int fgetc(FILE *stream);
int fputc(int c, FILE *stream);
char * fgets(char * cadena, int n, FILE *stream);
```

✅ `unsigned int actual;`
`while ((actual = fgetc(f)) != EOF) {}`

We can't use char since c must be big enough to hold EOF in addition to any possible char. Therefore we use int.
Parentheses are a MUST as the presedence of `!=` is stronger than `=`.

Desplazamiento en el archivo

`r` o `w` posicionan al inicio del archivo mientras que `a` al final

- **ftell( )**: obtener desplazamiento actual en un archivo
- **fseek( )**:
  - la nueva posición, medida en caracteres, es obtenida mediante la suma de desplazamiento y la posición especificada por origen
  - valores para origen son: `SEEK_SET` (inicio del archivo), `SEEK_CUR` (actual), `SEEK_END` (final del archivo)

```
long ftell(FILE *stream);
int fseek(FILE *stream, long int desplazamiento, int origen);
// int fseek(FILE *stream, long offset, int whence);
```

📚 int fseek(FILE *stream, long offset, int origin)
fseek sets the file position for stream; a subsequent read or write will access data beginning at the new position.

For a binary file, the position is set to offset characters from origin, which may be SEEK_SET (beginning), SEEK_CUR (current position), or SEEK_END (end of file).

For a text stream, offset must be zero, or a value returned by ftell (in which case origin must be SEEK_SET). fseek returns non-zero on error.
Therefore, it is **recommended** to use **binary mode** when working with `fseek()`.

Archivos Binarios

- Permite almacenar bloque de datos de **cualquier tipo**
- Lo crea un programa y se accede mediante el unicamente
- Utiliza un esquema de representacion binario interno (dependiente del dispositivo)
- Diferencia con **Archivos de Texto**: tipo char almacenado por lineas
- Comparten operaciones de **apertura** y **cierre**
- Los modos son equivalentes, se agrega el prefijo ***b*** de "*binary*"
- Una vez creado un archivo con un tipo determinado (texto o binario) no puede ser modificado
- `rb+`, `wb+` y `ab+` indicador de posicion de ficheros en distintos puntos

> 💡 Utilizar `.dat` para diferenciar el archivo binario.

Funciones para archivos binarios:

> 💡 Son independientes del tipo de dato que se lea o escriba, es decir, no realizan ninguna interpretación del tipo de dato.

- **fwrite( )**:
  - indicador de posición de ficheros para el stream (si está definido) es avanzado por el número de caracteres escritos correctamente.
  - error de escritura ≠ error en pre-escritura
  - retorna cantidad de elementos correctamente escritos
- **fread( )**:
  - un elemento parcialmente leido adquiere un valor indeterminado
- **fseek( )**:
  - para un **archivo binario**, la nueva posición, medida en caracteres desde el principio del fichero, es obtenida mediante la suma de desplaz y la posición especificada por origen
  - para un **archivo de texto**, o bien desplaz será cero, o bien desplaz será un valor retornado por una llamada anterior a la función ftell al mismo archivo y origen será SEEK_SET
  - retorna un valor distinto a cero sólo si una petición no se pudo satisfacer

```
size_t fwrite( const void * puntero, size_t tamanio, size_t cuantos, FILE * stream);
size_t fread( const void * puntero, size_t tamanio, size_t cuantos, FILE * stream);
```

Preprocesador en C

Caracteristica del procesador de C que facilita el desarrollo, y la portabilidad del codigo.

Directivas al preprocesador incian con `#`

Permite:

- incluir archivos (archivos **headers**)
- definir constantes simbolicas y macros
- compilacion condicional del codigo
- ejecucion condicional de las directivas del preprocesador

Operadores `#` y `##` concatenan con comillas y con tokens respectivamente

El alcance de una constante o de una macro cubre desde su definición hasta que se elimina con `#undef` o termina el programa

```
#include <archivo> // directorio predefinido en libreria estandar
#include "archivo" // directorio corriente --(si no lo encuentra)--> directorio estandar

#define MIN(a, b) (a < b) ? a : b
#define saludo(x) "Hola " #x " !\n"
printf("%s", saludo(Ana));
printf("%s", "Hola " "Ana" "!\n"); // printf concatena los strings separados por blancos
```

Compilacion condicional (directivas de preprocesador)

Trabajan como constructores

```
#undef // elimina la definicion de una constante simbolica o macro

#if !defined(NULL)
  #define NULL 0
#endif

#ifdef NULL
  printf("NULL esta definido");
#else
  printf("NULL no se encuentra definido");
#endif
```

💡 `NULL` se encuentra definido en la libreria `<stdio.h>`, por lo tanto si no se incluye la libreria, `#ifdef NULL` seria false.

Programas formados por varios archivos

Utilizacion de archivos cabecera para estandarizar prototipos

Precedencia de los operadores en C

Precedence	Operator	Description	Associativity
1	`++ --`	Suffix/postfix increment and decrement	Left-to-right
	`()`	Function call	
	`[]`	Array subscripting	
	`.`	Structure and union member access	
	`->`	Structure and union member access through pointer	
	`(type){list}`	Compound literal(C99)	
2	`++ --`	Prefix increment and decrement[note 1]	Right-to-left
	`+ -`	Unary plus and minus	
	`! ~`	Logical NOT and bitwise NOT	
	`(type)`	Cast	
	`*`	Indirection (dereference)	
	`&`	Address-of	
	`sizeof`	Size-of[note 2]	
	`_Alignof`	Alignment requirement(C11)	
3	`* / %`	Multiplication, division, and remainder	Left-to-right
4	`+ -`	Addition and subtraction	
5	`<< >>`	Bitwise left shift and right shift	
6	`< <=`	For relational operators < and ≤ respectively	
	`> >=`	For relational operators > and ≥ respectively	
7	`== !=`	For relational = and ≠ respectively	
8	`&`	Bitwise AND	
9	`^`	Bitwise XOR (exclusive or)	
10	`|`	Bitwise OR (inclusive or)	
11	`&&`	Logical AND	
12	`||`	Logical OR	
13	`?:`	Ternary conditional[note 3]	Right-to-left
14[note 4]	`=`	Simple assignment	
	`+= -=`	Assignment by sum and difference	
	`*= /= %=`	Assignment by product, quotient, and remainder	
	`<<= >>=`	Assignment by bitwise left shift and right shift	
	`&= ^= |=`	Assignment by bitwise AND, XOR, and OR	
15	`,`	Comma	Left-to-right