




TDL - Mod II

 Age	2023
--	------

Asignacion de memoria dinamica

- **Malloc:** `void * malloc(size)`
 - Retorna un puntero de tipo `void *` el cual es el inicio en memoria de la porcion reservada.
 - Si no puede reservar esa cantidad de memoria la funcion regresa un puntero `NULL`.
 - Para arreglos dinamicos: `= void * malloc (N*sizeof(type))`
 - Se puede acceder con `ptr[x]` o `*(ptr+x)`
- **Calloc:** `void * calloc(N,size)`
 - Retorna un puntero de tipo `void *` el cual es el inicio en memoria de la porcion reservada.
 - Reserva un espacio para un arreglo de N objetos.

- El espacio es inicializado con todos los bits en cero. Por esta razón, consume un poco más de memoria y demora un poco más con respecto a malloc.
- **Realloc:** void * realloc(ptr, size)
 - Cambia el tamaño del objeto apuntado por ptr al tamaño especificado por size.
 - Busca memoria → Reserva memoria → Copia el objeto → Libera la conexión anterior.
 - Si size es cero y ptr no es nulo, el objeto al que apunta es liberado.
- **Free:** void free(ptr)
 - Libera el espacio apuntado por ptr (elimina conexión con la memoria reservada).
 - Si ptr es NULL no se realiza ninguna acción.



Importante antes de hacer cosas con el puntero creado, fijarse que no esté en nulo...

Declaración de una lista o pila

```
#include <stdio.h>
int main(){
    struct nodo {
        int valor;
        struct nodo * ptr;
    }
    struct nodo *Pila = NULL, *aux;
}
```

Matrices dinámicas

```

/* Programa asignar una matriz dinámica */
#include <stdlib.h>
int main(void)
{
    const int f=3; /* número de filas */
    const int c=7; /* número de columnas */
    double **mat; /* puntero de puntero */
    int i; /* contador */

    /* Asigno memoria (falta el control de errores)*/
    mat=(double**)calloc(f,sizeof(double*));

    for(i=0; i<f; i++) {
        mat[i]=(double*)calloc(c,sizeof(double));
    }
    < ... sigue ... >
}

```

```

//Liberar memoria
for(int i=0;i<f;i++){
    free(mat[i]);
}
free(mat); //solo libero la conexion al primer

```

Para la reserva de memoria, siempre es necesario pasar la dirección del puntero ** creado.



Copiar datos de un bloque de memoria a otro:
memcpy(destino,origen,size)

Archivos de Texto

```

FILE * arch; //crea un puntero a la dirección
arch = fopen("nombre.txt","modo_apertura");
if(arch==NULL) printf("El archivo no existe");
fclose (arch); //retorna cero si fue exitoso

```

Operaciones entrada/salida

MODOS DE APERTURA DE ARCHIVO

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.

- **fprintf(arch o stdout, *cadena):** escribe en el archivo lo apuntado por cadena
- **fscanf(arch, tipo de datos a recuperar, dir variables para guardar datos (opcional))**

- **int feof(arch)**

- La función feof retorna 1 si el archivo terminó y 0 en otro caso.
- Note que la función feof indica si ya se realizó una operación fuera del límite del archivo; no si se encuentra posicionado en el límite del archivo.

```
while (! feof( arch )){  
    /* procesamiento de los datos */  
    /* operación de lectura */  
}
```

- **int fgetc(arch):** lee un caracter desde el archivo y lo convierte a int.
 - fgetc(stdin) equivale a getchar().
- **int fputc(int c,arch):** escribe el caracter leído desde el archivo en la variable c.
 - fputc('a', stdout) equivale a putchar('a')
- **fgets(*cadena, n, arch)**
 - Lee n-1 caracteres para agregar un carácter nulo inmediatamente después del último carácter leído en el array.
- **fputs(*cadena, arch)**
 - Escribe lo apuntado por cadena en el archivo.
 - No escribe el \0.

```
while ((caracter = fgetc(fuente)) != EOF) {  
    fputc(caracter, destino);  
}  
//*****//  
while (fgets(linea, sizeof(linea), fuente) != NULL) {  
    fputs(linea, destino);  
}
```

Desplazamiento en el archivo

Al abrir un archivo en modo de acceso "r" o "w" el desplazamiento se inicializa en 0 (comienzo del archivo), en cambio, si se utiliza el modo "a" el desplazamiento comienza al final del archivo.

- long **ftell(arch);**
- int **fseek(arch, desplazamiento, origen);**
 - Reubica la posición del puntero al archivo.
 - La nueva posición, medida en caracteres, es obtenida mediante la suma de desplazamiento y la posición especificada por origen.
 - Los valores para origen son: SEEK_SET (inicio del archivo), SEEK_CUR (actual), SEEK_END (final del archivo).

Archivos de texto binarios

Operacion E/S

- **fwrite(&num,sizeof(int),1,arch);**
 - Envía desde el arreglo apuntado por puntero, la cantidad de elementos indicada en cuantos cuyo tamaño es especificado por tamaño, al dispositivo apuntado por stream.
- **fread(vector, sizeof(int),5, arch);**
 - Recibe en el arreglo apuntado por puntero, la cantidad de elementos indicada en cuantos cuyo tamaño es especificado por tamaño, desde dispositivo apuntado por stream.

Modo de apertura de los Archivos Binarios

Modo	Descripción
rb	Abre un archivo binario para lectura
wb	Crea un archivo binario para escritura; si el archivo ya existe se descarta el contenido actual.
ab	Abre o crea un archivo binario para escribir al final del mismo
rb+ ó r+b	Abre un archivo para lectura y escritura.
wb+ ó w+b	Crea un archivo binario para lectura y escritura. Si el archivo existe, se descarta el contenido actual.
ab+ ó a+b	Abre o crea un archivo binario para actualizar. La escritura se realizará al final del archivo.

Concepto de archivo binario

Un archivo binario es un tipo de archivo que permite almacenar un bloque de datos de cualquier tipo. Los archivos binarios los puede crear únicamente el programa y el acceso a sus elementos sólo es posible a través del programa. El contenido de un archivo binario es ilegible ya que utiliza un esquema de representación binario interno. Este esquema depende de la computadora que se use, por lo que no puede ser visualizado mediante un editor de textos.

Diferencia entre archivo binario y archivo de texto

Las componentes de un archivo de texto son de tipo *char* y están organizados en líneas mientras que las componentes de un archivo binario pueden ser de cualquier tipo predefinido o definido por el usuario (excepto de tipo archivo).

Preprocesador

El preprocesamiento es el primer paso en la etapa de compilación de un programa. Es una característica del compilador de C. Todas las directivas del preprocesador o comandos inician con un #.

Ventajas de usar el preprocesador:

- El código C es más portable entre diferentes arquitecturas de máquinas.
- Programas más fáciles de desarrollar, de leer y de modificar.

Directivas

- **include:** sirve para insertar archivos externos dentro de nuestro archivo de código fuente.
 - `#include <archivo>`
 - Busca el archivo en la librería estándar.
 - Se utiliza para los arch de la librería estándar.
 - `#include "archivo"`
 - Busca primero en el directorio actual y luego en la librería estándar.
 - Se utiliza para archivos definidos por el usuario.



A estos dos comandos se los llama prototipos de cabecera

- **define identificador valor:** Si un valor es provisto, el identificador será reemplazado literalmente por valor (el resto del texto en la línea).
 - Macro: Es una operación definida mediante `#define`
 - Una macro sin argumentos es tratada como una constante simbólica.
 - Una macro con argumentos, al ser expandida, reemplaza sus argumentos con los argumentos reales encontrados en el programa.
 - Realiza una sustitución de texto, sin chequeo de tipos.



Para encapsular un pedazo de código, se debe hacer en un do-while para protegerlo.

- **undef:** elimina la definición de una constante simbólica o macro.
- **if, elif, else, endif**

#if, #elif, #else y #endif

- Permiten hacer una **compilación condicional** de un conjunto de líneas de código.
- **Sintaxis**

```
#if expresión-constante-1
<sección-1>
#elif <expresión-constante-2>
<sección-2>
.
.
.
#elif <expresión-constante-n>
<sección-n>
<#else>
<sección-final>
#endif
```

- **ifdef y ifndef:** Permiten comprobar si un identificador está o no actualmente definido, es decir, si un #define ha sido previamente procesado para el identificador y si sigue definido.

Headers

Archivo main.c

```
main.c x FuncionVerTexto.c cabecera.h
1 #include <stdio.h>
2 #include "cabecera.h"
3
4 int main()
5 {
6     printf("Ejecutando...\n");
7     VerTexto("Funciona?");
8     printf("Terminado.\n");
9
10    return(0);
11 }
```

Archivo cabecera.h

```
FuncionVerTexto.c cabecera.h x
1 #ifndef CABECERA_H_INCLUDED
2 #define CABECERA_H_INCLUDED
3
4 void VerTexto(char *);
5
6 #endif // CABECERA_H_INCLUDED
```

- Este archivo utiliza «include guards» (guardas include) para evitar múltiples definiciones de la función.

Este archivo no tiene implementaciones, solo prototipos.

Archivo FuncionVerTexto.c

```
FuncionVerTexto.c x cabecera.h
1 #include <stdio.h>
2 #include "cabecera.h"
3
4 void VerTexto(char * cadena)
5 {
6     printf("%s\n", cadena);
7 }
```

- Puede incluirse la cabecera para chequear consistencia y evitar errores.

Compile y verifique que funciona

Argumentos Main

- Argc: cantidad de argumentos recibidos por la funcion main.
- Argv: vector que contiene los argumentos en formato string

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char * argv[])
{
    int n1, n2;
    char oper;
    float result;

    if (argc != 4)
        printf("Número incorrecto de parámetros\n");
    else
    {
        n1 = atoi(argv[1]);
        n2 = atoi(argv[2]);
        oper = *argv[3];

        switch (oper) {
            case '+': result = n1 + n2; break;
            case '-': result = n1 - n2; break;
            case '*': result = n1 * n2; break;
            default : result = (float)n1 / n2;
        }
        printf ("%d %c %d = %.2f", n1, oper, n2, result);
    }
    return 0;
}
```

2Operandos1Operador.c



atoi convierte a int, mientras que atof convierte a float

Compilador GCC

- gcc ejemploMain.c: se creara el exe con nombre por defecto.
- gcc -o ejemplo ejemploMain.c || gcc -Wall ejemplo ejemploMain.c (compila con mas warnings)
- gcc -DLINEWIDTH=80 -o Ej2Modif Ej02_Modif.c (cumple la funcion de define)

OPTIMIZACIÓN DE CÓDIGO CON GCC

Opción	Descripción
0	Optimizar para la velocidad de compilación - sin optimización del código (por defecto).
1,2,3	Optimizar para aumentar la velocidad de ejecución del código (cuanto mayor sea el número, mayor será la velocidad).
s	Optimizar el tamaño del archivo.
funroll-loops	Optimizar activando el desenrollado de bucles. Es independiente de otras opciones de optimización.

Ejemplo: gcc hola.c -o hola -O1

Makefile

all: objetivo (hola)

hola: requisito (hola.o)

gcc hola.o -o hola

hola.o: hola.c

gcc -c hola.c -o hola.o

```
CC = gcc
CFLAGS = -c -Wall
SOURCES = $(wildcard *.c)
OBJS = $(SOURCES:.c=.o)
EXE = hola.exe

all: $(EXE)

$(EXE): $(OBJS)
    $(CC) $(OBJS) -o $@

%.o: %.c
    $(CC) $(CFLAGS) $< -o $@

clean:
    rm -f $(OBJS) $(EXE)
```

Busca todos los archivos .c en el directorio actual

Genera la lista de archivos objeto

\$@ se refiere al objetivo. En este caso "hola.o"

\$< corresponde al 1er. requerimiento. En este caso "hola.c"