Resumen TDL - Mod II

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 razon, consume un poco mas de memoria y demora un poco mas 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 conexion anterior.
  - Si size es cero y ptr no es nulo, el objeto al que apunta es liberado.

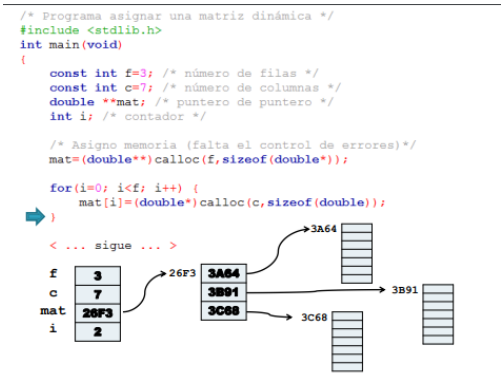
- **Free:** void free(ptr)
  - Libera el espacio apuntado por ptr (elimina conexión con la memoria reservada).
  - Si ptr es NULL no se realiza ninguna accion.

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

Declaracion de una lista o pila

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

Matrices dinamicas



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

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

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

Archivos de Texto

```
FILE * arch; //crea un puntero a la direccion de
arch = fopen("nombre.txt","modo_apertura";
if(arch==NULL) printf("El archivo no abrio correc
fclose (arch); //retorna cero si fue cerrado con
```

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 leido 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.

```
while ((caracter = fgetc(fuente)) != EOF) {
        fputc(caracter, destino);
```

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

Desplazamiento en el archivo

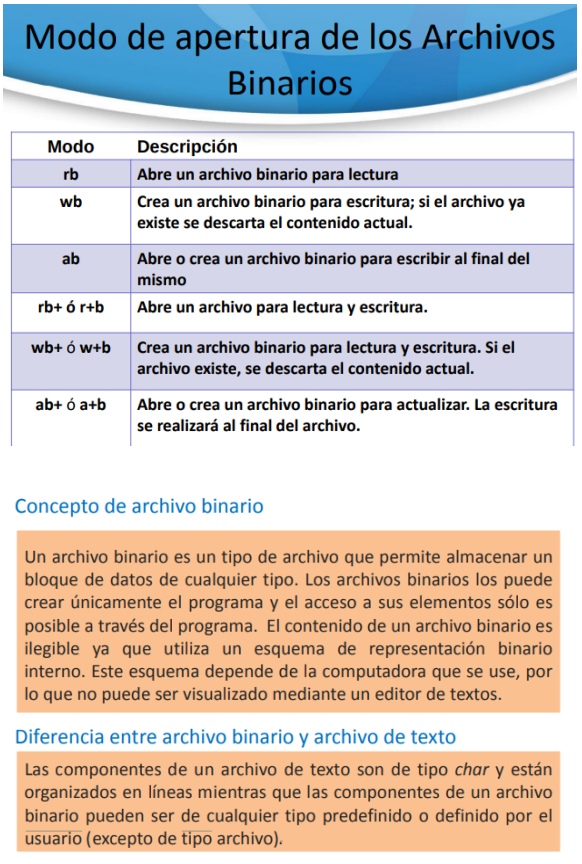
Al abrir un archivo en modo de acceso "r" o "w" el desplazamiento se inicializa en 0 (comienzo del archivo), en cambio, si se utiliza el modo "a" el desplazamiento comienza al final del archivo.

- 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 tamanio, 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 tamanio, desde dispositivo apuntado por stream.



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 codigo, 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**

```c
#include <stdio.h>
#include "cabecera.h"

int main()
{
    printf("Ejecutando...\n");
    VerTexto("Funciona?");
    printf("Terminado.\n");

    return(0);
}
```

Archivo **cabecera.h**

```c
#ifndef CABECERA_H_INCLUDED
#define CABECERA_H_INCLUDED

void VerTexto(char * );

#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**

```c
#include <stdio.h>
#include "cabecera.h"

void VerTexto(char * cadena)
{
  printf("%s\n", cadena);
}
```

- 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

2Operandos1Operador.c

```c
#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;
}
```

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