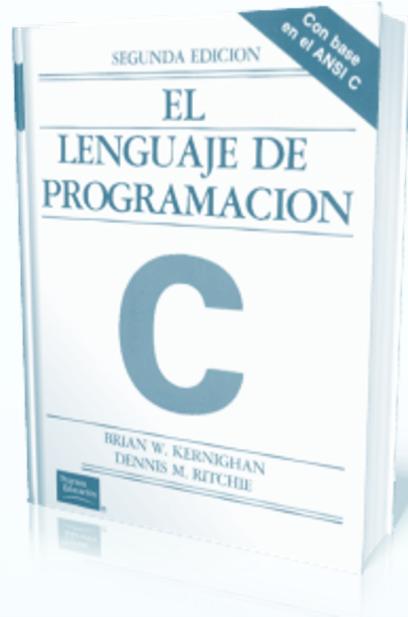


1 Programación avanzada en lenguaje estructurado

Objetivo: Elaborar programas en lenguaje C empleando estructuras de almacenamiento de datos complejas.



Bibliografía



El lenguaje de programación C.
Brian W. Kernighan, Dennis M. Ritchie,
segunda edición, USA, Pearson
Educación 1991.

Antecedentes

Algoritmo

Método para resolver un problema mediante una serie de pasos precisos, definidos y finitos. Debe poseer las siguientes características

- Preciso u ordenado.
- Definido o conciso.
- Finito.
- Mostrar una salida.
- Sencillo y legible.
- Eficaz.
- Eficiente.

Los algoritmos pueden ser de dos tipos:

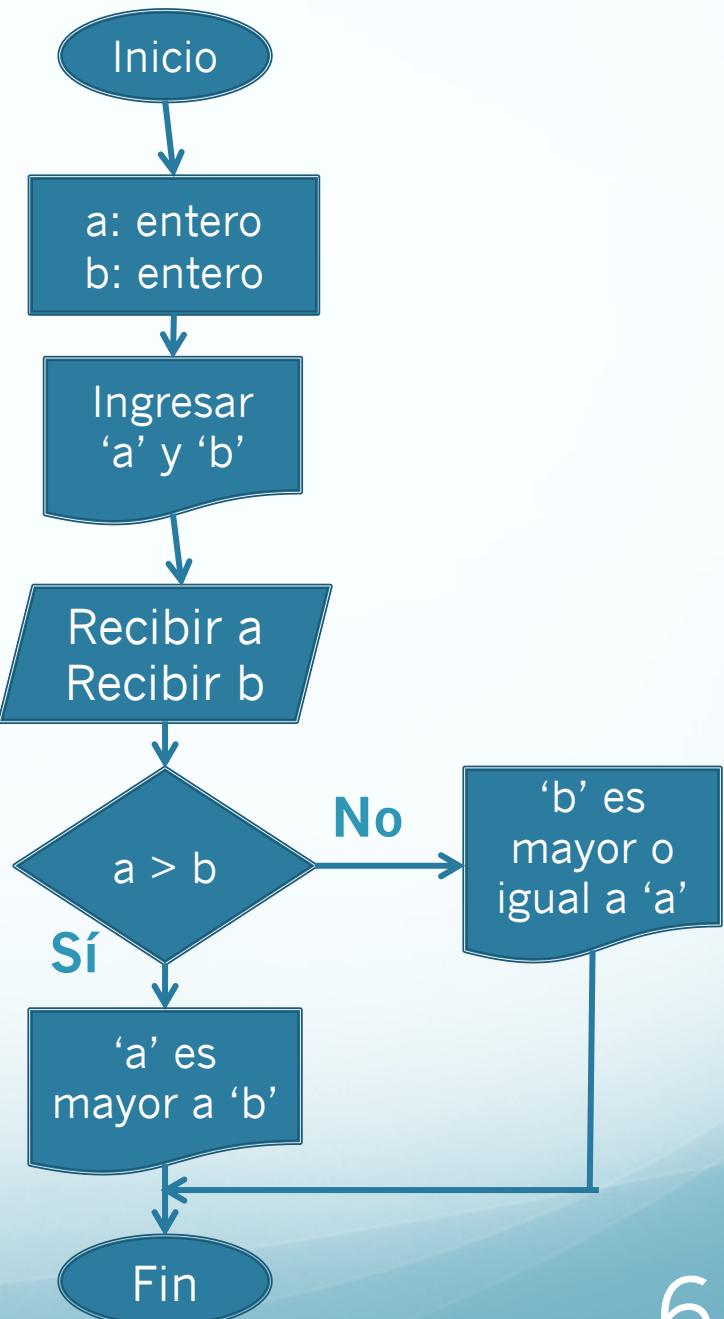
- ✓ **Gráfico:** Es la representación gráfica de las operaciones que realiza un algoritmo (Diagrama de Flujo).
- ✓ **No gráfico:** Representa en forma descriptiva las operaciones que debe realizar un algoritmo (Pseudocódigo).

Las etapas utilizadas para crear un algoritmo son:

- **Análisis del problema:** Datos iniciales, datos de entrada, restricciones y salida esperada.
- **Construcción del algoritmo:** Pasos a seguir para resolver el problema.
- **Verificación del algoritmo:** Prueba de escritorio.

Diagrama de flujo

Problema: Verificar si un número es mayor que otro número dado.



Pseudocódigo

INICIO

a: Entero

b: Entero

ESCRIBIR “Valor del primer número”

LEER a

ESCRIBIR “Valor del segundo número”

LEER b

SI a > b **entonces**

ESCRIBIR “El número a es mayor a b”

FIN DEL SI

EN CASO CONTRARIO

ESCRIBIR “El número b es mayor o igual a a”

FIN DE EN CASO CONTRARIO

FIN

Problema: Verificar si un número a es mayor que otro número b dados.

Estructura de un programa en lenguaje C

Un programa en C consiste en una o más funciones, de las cuales una de ellas debe llamarse main() y es la principal de todas.

- El programa debe contener, por lo menos una función: la función main().
- Cada función o programa, consta de un cuerpo de función delimitado por llaves de comienzo y fin { }.
- En el cuerpo de la función van incluidas: sentencias, variables, funciones, etc. terminadas cada una de ellas por un punto y coma (;).

```
00010111010001000111111110100000  
0100101100001101011101101011001000  
1011000001010110010001000011100010011  
1001100101101001101001111011110111  
0110100#include <stdio.h>0110100011  
001001100010100011101001111011110111  
0001001int main00010111  
0101001{000110  
11001100printf("Hello World")00011  
100000111return 42;0101011101  
0110100010001110100111010000110  
010011011101011101110000010100011  
001001000101011001001110111010001011  
1010011100110101110001010101000110  
00110000110111110101001111110001
```

Estructura de selección if-else

La estructura completa es if-else y su sintaxis es la siguiente:

```
if (condición){  
    /* Código a ejecutar si la condición es  
    verdadera*/  
} else {  
    /* Código a ejecutar si la condición es falsa*/  
}
```

Estructura de selección switch-case

La estructura switch-case tiene la siguiente sintaxis:

```
switch (opcion){  
    case valor1:  
        /* Código a ejecutar*/  
        break;  
    case valor2:  
        /* Código a ejecutar*/  
        break;  
    ...  
    case valorN:  
        /* Código a ejecutar*/  
        break;  
    default:  
        /* Código a ejecutar*/  
}  
}
```

Estructura de selección condicional (?)

La expresión condicional (también llamado operador ternario) permite realizar una comparación rápida. Su sintaxis es la siguiente:

Condición ? SiSeCumple : SiNoSeCumple

Estructura de repetición while

La estructura del ciclo while es la siguiente:

```
while (condición){  
    /*Código a ejecutar*/  
}
```

Estructura de repetición do-while

La estructura del ciclo do-while es la siguiente:

```
do {  
    /*  
     Código a ejecutar  
    */  
} while (condición);
```

Estructura de repetición for

La estructura de del ciclo (o bucle) for es la siguiente:

```
for (valorInicial; condición; (in/de)-cremento) {  
    /*Código a ejecutar*/  
}
```

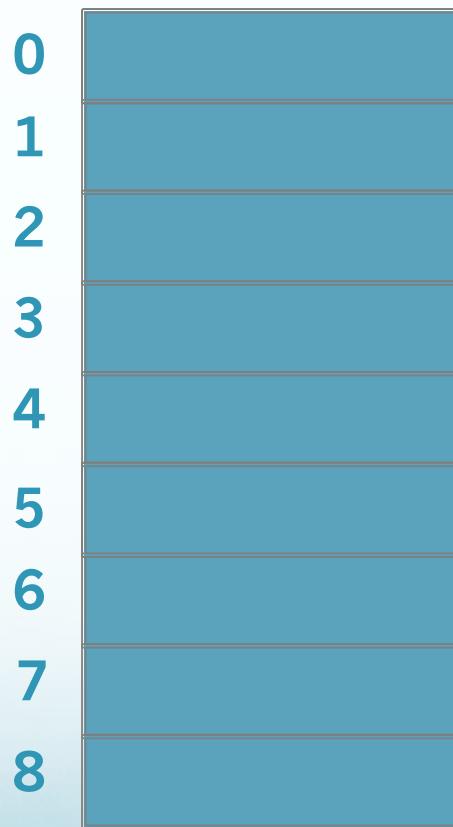
Arreglos unidimensionales

Un arreglo puede tener una o varias dimensiones. Para acceder a un elemento en particular de un arreglo se usa un índice.

El formato para declarar un arreglo unidimensional es:

tipo nombre_arr [tamaño]

La primera localidad siempre corresponde a la localidad 0 y la última corresponde a n-1, en donde n es el número de localidades del arreglo.



Variables de tipo entero:

| Tipo | Bits | Valor Mínimo | Valor Máximo |
|-----------------------------------------------------------------|------|----------------|---------------|
| signed char | 8 | -128 | 127 |
| unsigned char | 8 | 0 | 255 |
| signed short | 16 | -32 768 | 32 767 |
| unsigned short | 16 | 0 | 65 535 |
| signed int | 32 | -2 147 483 648 | 2 147 483 647 |
| unsigned int | 32 | 0 | 4 294 967 295 |
| signed long | 32 | -2 147 483 648 | 2 147 483 647 |
| unsigned long | 32 | 0 | 4 294 967 295 |
| Si se omite el clasificador, por defecto se considera “signed”. | | | |
| enum | 16 | -32 768 | 32 767 |

Variables de tipo punto flotante:

| Tipo | Bits | Valor Mínimo | Valor Máximo |
|-------------|------|--------------|--------------|
| float | 32 | 3.4E-38 | 3.4E38 |
| double | 64 | 1.7E-308 | 1.7E308 |
| long double | 80 | 3.4E-4932 | 3.4E4932 |

Las variables de tipo flotante son siempre con signo.

Ejemplo

```
#include<stdio.h>

main() {
    enum dias {lunes, martes, miercoles, jueves, viernes, sabado, domingo};
    enum dias varEnum;
    varEnum = viernes;
    switch(varEnum) {
        case lunes:
        case martes:
            printf("¡Inicio de semana!\n");
            break;
        case miercoles:
            printf("Ombligo de semana.\n");
            break;
        case jueves:
        case viernes:
        case sabado:
            printf("¡Inicia el fin de semana!\n");
            break;
        case domingo:
            printf("Se acabO la semana.");
            break;
    }
}
```

Ejemplo 1.1

```
#include<stdio.h>

main() {
    char palabra [20];
    int i = 0;
    printf("Introduce una palabra: ");
    scanf("%s", palabra);
    printf("La palabra ingresada es:\n");
    for (i = 0 ; i < palabra[i] ; i++) {
        printf("%c\n", palabra[i]);
    }
}
```

Arreglos con funciones

Una función es un conjunto de declaraciones, definiciones, expresiones y sentencias que realizan una tarea específica.

El formato general de una función en C es:

```
tipoValorRetorno nombreFunción ([Parámetros]){
    //variables locales
    //código de la función
    return tipoValorRetorno
}
```

Ejemplo

```
#include<stdio.h>

void imp_rev(char s[]) {
    int t;
    for( t=strlen(s)-1; t>=0; t--)
        printf("%c",s[t]);
    printf("\n");
}

main() {
    char nombre[]="Facultad";
    imp_rev(nombre);
}
```

1.1 Arreglos de varias dimensiones y arreglos de apuntadores

Los arreglos son una colección de variables del mismo tipo que se referencian utilizando un nombre común.

Un arreglo consta de posiciones de memoria contigua. La dirección más baja corresponde al primer elemento y la más alta al último.

Cada localidad está asociada con un número, de tal manera que para identificar una localidad específica del arreglo es necesario escribir su número.

Arreglos de varias dimensiones

Lenguaje C permite crear arreglos de varias dimensiones (multidimensionales).

El formato para declarar un arreglo multidimensional es:

tipo nombre_arr [tamaño][tamaño]... [tamaño]

Ejemplo 1.2

```
#include<stdio.h>

int main(){
    int matriz [3][3] = {{1,2,3},{4,5,6},{7,8,9}};
    int i, j;
    printf("Arreglo multidimensional:\n");
    for (i = 0 ; i < 3 ; i++){
        for (j = 0 ; j < 3 ; j++){
            printf("%d, ",matriz[i][j]);
        }
        printf("\n");
    }
    return 42;
}
```

Arreglos de apuntadores

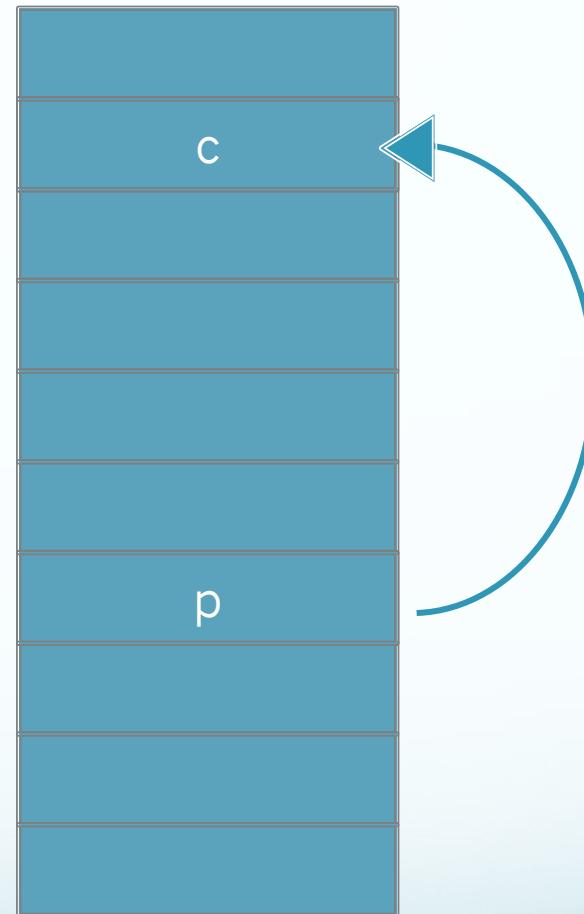
Un apuntador es una variable que contiene la dirección de una variable.

Los apuntadores se utilizan debido a que algunas veces son la única forma de expresar una operación, y debido a que, generalmente, llevan un código compacto y eficiente.

Los apuntadores también pueden emplearse para obtener claridad y simplicidad.

```
#include <stdio.h>

int main () {
    char c;
    char *p;
    p = &c;
    return 5;
}
```



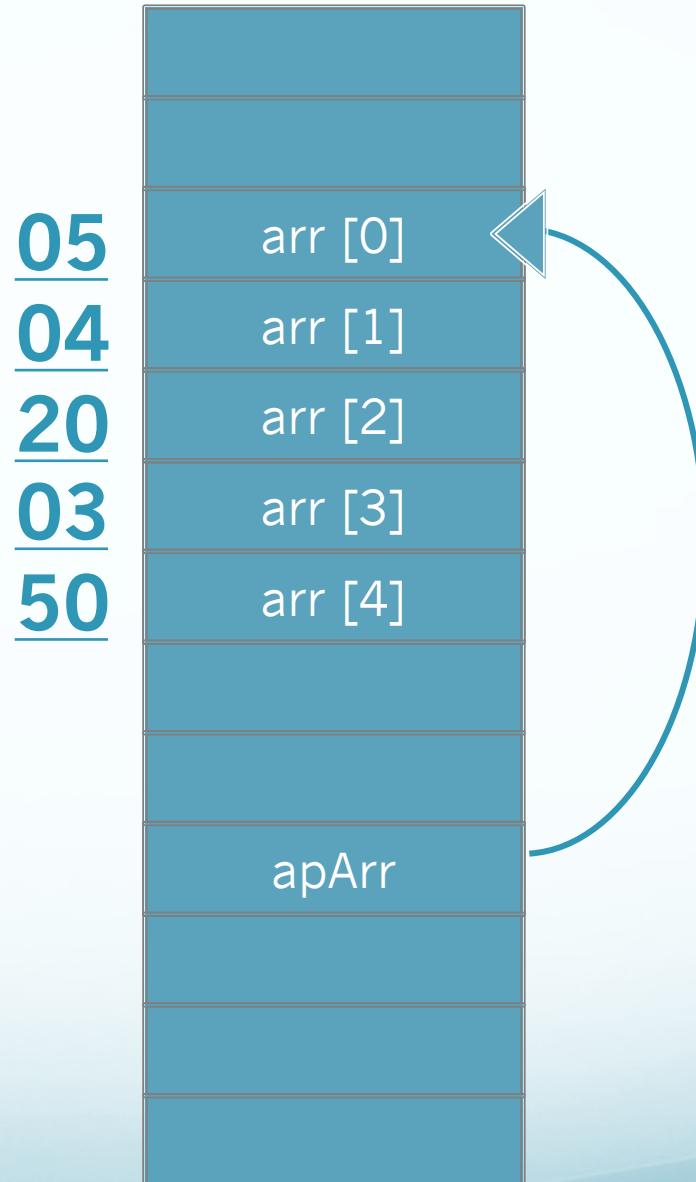
Ejemplo 1.3

```
#include<stdio.h>
```

```
Int main () {
    int a = 5, b = 10, c[10];
    int *apEnt;      // apuntador a entero
    apEnt = &a;      // apEnt -> a
    b = *apEnt;     // b = 5
    b = *apEnt +1; // b = 6
    *apEnt = 0;     // a = 0
    apEnt = &c[0]; // apEnt -> c[0]
    return 34;
}
```

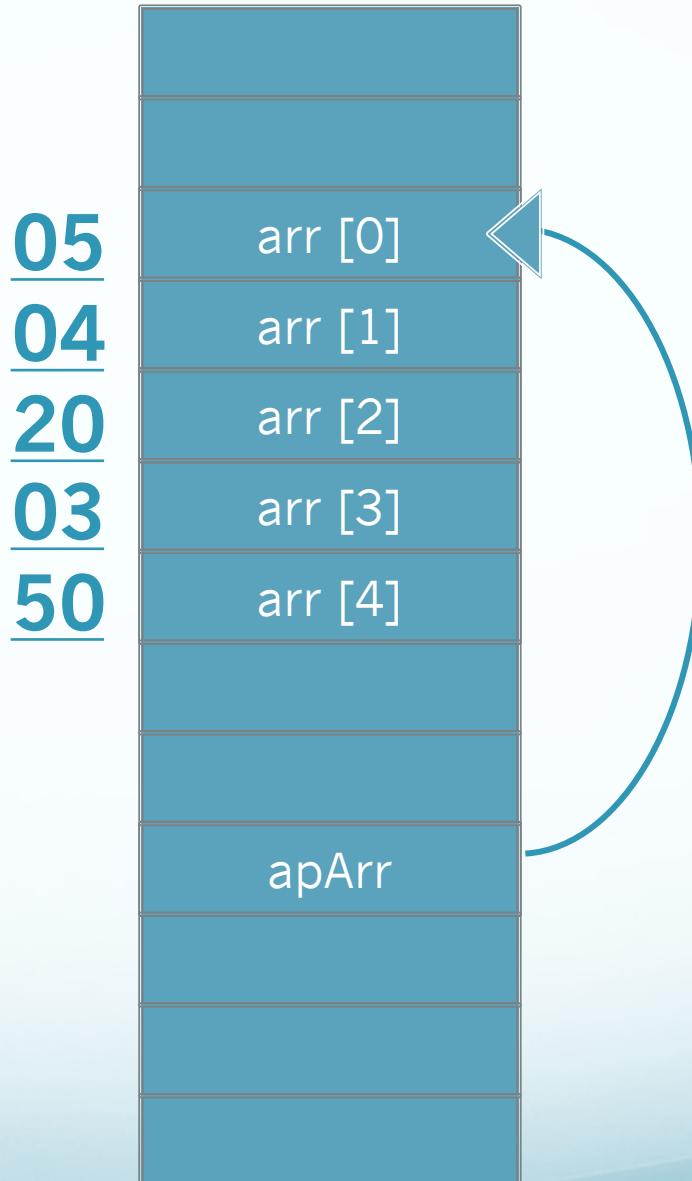
```
#include <stdio.h>

int main () {
    int arr [5];
    int *apArr;
    apArr = &arr[0];
    return 84;
}
```



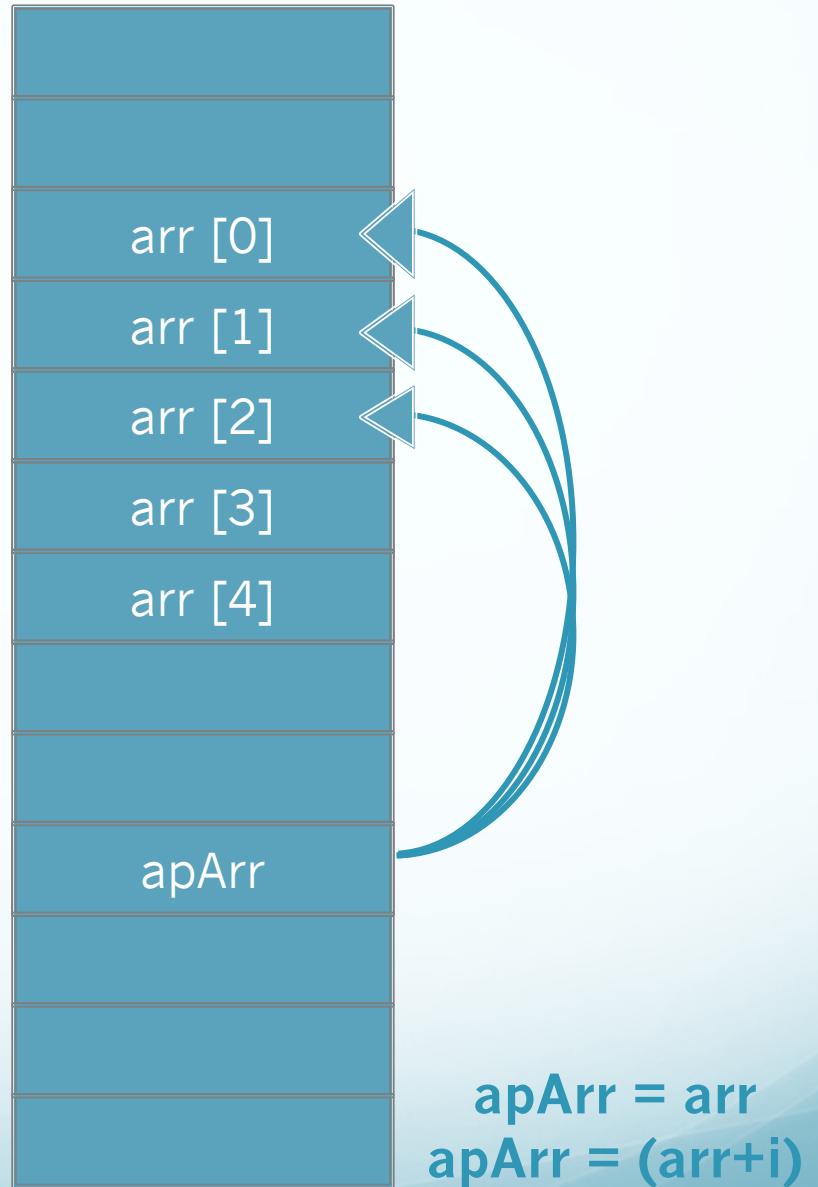
```
#include <stdio.h>

int main () {
    int arr [5];
    int *apArr;
    apArr = &arr[0];
    int x = *apArr;
    return 1;
}
```



```
#include <stdio.h>

int main () {
    int arr [5];
    int *apArr;
    apArr = &arr[0];
    int x = *apArr;
    *(apArr+1);
    *(apArr+2);
    return 8;
}
```



`apArr = arr`
`apArr = (arr+i)`

Aritmética de direcciones:

Si apArr es un apuntador a algún elemento de un arreglo, entonces:

- apArr++; // Incrementa apArr para apuntar al siguiente elemento**
- apArr += i; // Incrementa apArr para apuntar al i elementos adelante.**

Apuntador a función

Declaración de apuntador a función:
int (*af)();

Declaración de función y apuntador a función:
int funcion(int);
int (*af) (int) = &funcion;

Ejemplo:
ans = funcion(5);
ans = *af(5);

Ejemplo: “Asignar memoria dinámica”

Se cuenta con dos rutinas:

almacenar: Regresa un apuntador a ‘n’ posiciones consecutivas para almacenar caracteres.

liberar: libera los caracteres almacenados para poder reutilizar el espacio.



Ejemplo 1.4

```
#include <stdio.h>

#define TAMANO 1000          // tamaño disponible

static char buffer[TAMANO]; // Se crea el espacio
static char *apBuffer = buffer; // Siguiente posición libre

char *almacenar(int caracteres) {
    if ( buffer + TAMANO - apBuffer >= caracteres ) {
        apBuffer += caracteres;
        return apBuffer - caracteres;
    } else {
        return 0;
    }
}
```

Ejemplo 1.4

```
/* Función que libera el número de caracteres enviados */
void liberar(int caracteres) {
    if (buffer + TAMANO - apBuffer - caracteres >= buffer) {
        apBuffer -= caracteres;
    }
}

int main () {
    return 42;
}
```

Ejemplo 1.5

Longitud de una cadena de caracteres

```
#include <stdio.h>
```

```
int longCad (char *cad) {
    char *ap = cad;
    while (*ap != '\0') {
        ap++;
    }
    return ap - cad;
}

int main () {
    char palabra [] = "xochimilco", *ap;
    ap = palabra;
    int longitud = longCad(ap);
    printf ("Longitud de la palabra : %d", longitud);
    return 99;
}
```

Ejemplo 1.6

Apuntador a void

```
#include <stdio.h>
```

```
//datos {carácter = 0,real = 1,entero = 2,cadena = 3};  
void ver(void *, int);
```

```
int main() {  
    char a='b';  
    int x=3;  
    double y=4.5;  
    char *cad="hola";  
    ver(&a, 0);  
    ver(&x, 2);  
    ver(&y, 1);  
    ver(cad, 3);  
    getchar();  
    return 0;  
}
```

Ejemplo 1.6

```
void ver( void *p, int d) {  
    switch(d) {  
        case 0:  
            printf("%c\n",*(char *)p);  
            break;  
        case 1:  
            printf("%d\n",*(double *)p);  
            break;  
        case 2:  
            printf("%ld\n",*(int *)p);  
            break;  
        case 3:  
            printf("%s\n",(char *)p);  
            break;  
        default:  
            printf("Error ");  
    }  
}
```

Arreglo de apuntadores

Ejemplo 1.7

```
#include <stdio.h>
```

```
char *nombre_mes(int);
```

```
int main(void){
```

```
    char *mes = NULL;
```

```
    int n;
```

```
    puts("Introduzca el numero del mes deseado");
```

```
    if (scanf("%d", &n) != 1){
```

```
        printf("Caracter invAlido\n");
```

```
        return;
```

```
}
```

```
mes = nombre_mes(n);
```

```
printf("El mes seleccionado es:\n%s\n", mes);
```

```
return 4;
```

```
}
```

Ejemplo 1.7

```
/* Método que genera un arreglo de caracteres */
char *nombre_mes(int numero){
    static char *mes[] = {
        "Mes invAlido",
        "Enero", "Febrero", "Marzo",
        "Abril", "Mayo", "Junio",
        "Julio", "Agosto", "Septiembre",
        "Octubre", "Noviembre", "Diciembre"
    };

    return (numero < 1 || numero > 12) ? mes[0] : mes[numero];
}
```

Tarea 1.1

Para el ejemplo 1.4, ingresar una rutina en la función principal (main) para que agregue y elimine datos del buffer. Para saber la longitud de los datos insertados, utilizar el ejemplo 1.5.

Cada vez que se ingrese un dato al buffer mandar el resultado a la salida estándar (imprimir el resultado).

1.2 Estructuras

Una estructura es una colección de una o más variables, de iguales o diferentes tipos, agrupadas bajo un solo nombre.

Una estructura es un tipo de dato compuesto que permite almacenar un conjunto de datos de diferente tipo, es decir, permiten agrupar un grupo de variables relacionadas entre sí, y pueden ser tratadas como una unidad.

Los datos que pueden contener una estructura son:

Simples:

- Caracteres
- Números
- Enteros
- Punto flotante

Compuestos:

- Arreglos
- Estructuras



**int x = a;
Int y = b;**

Palabra reservada → **struct**

Nombre o rótulo de la estructura → **punto**

Miembros de la estructura → **{ int x; int y; }**

```
struct punto {  
    int x;  
    int y;  
};
```

```
struct punto p = { 5, 3 };
```

```
printf("x = %d, y = %d", p.x, p.y);
```

```
struct {  
    int x;  
    int y;  
} punto;
```

```
struct {  
    int x, y;  
} punto;
```

```
struct { int x, y; } punto;
```

Ejemplo 1.8

```
#include<stdio.h>

struct punto {
    int x, y;
};

void main() {
    struct punto p = {5, 3};
    printf("x = %d, y = %d", p.x, p.y);
}
```

Ejemplo 1.9

```
#include<stdio.h>

struct alumno{
    int num_cuenta;
    char nombre [50];
};

void main() {
    struct alumno a;
    printf("Introduce tu nombre: ");
    scanf("%s", a.nombre);
    printf("Introduce tu número de cuenta: ");
    scanf("%d", &a.num_cuenta);
    printf("Alumno: %s\n Número de cuenta: %d\n",
        a.nombre, a.num_cuenta);
}
```

Ejemplo 1.10

```
#include<stdio.h>

struct alumno{
    int num_cuenta;
    char nombre [50];
};

struct alumno a;

void main() {
    printf("Introduce tu nombre: ");
    scanf("%s", a.nombre);
    printf("Introduce tu número de cuenta: ");
    scanf("%d", &a.num_cuenta);
    printf("Alumno: %s\n Número de cuenta: %d\n",
        a.nombre, a.num_cuenta);
}
```

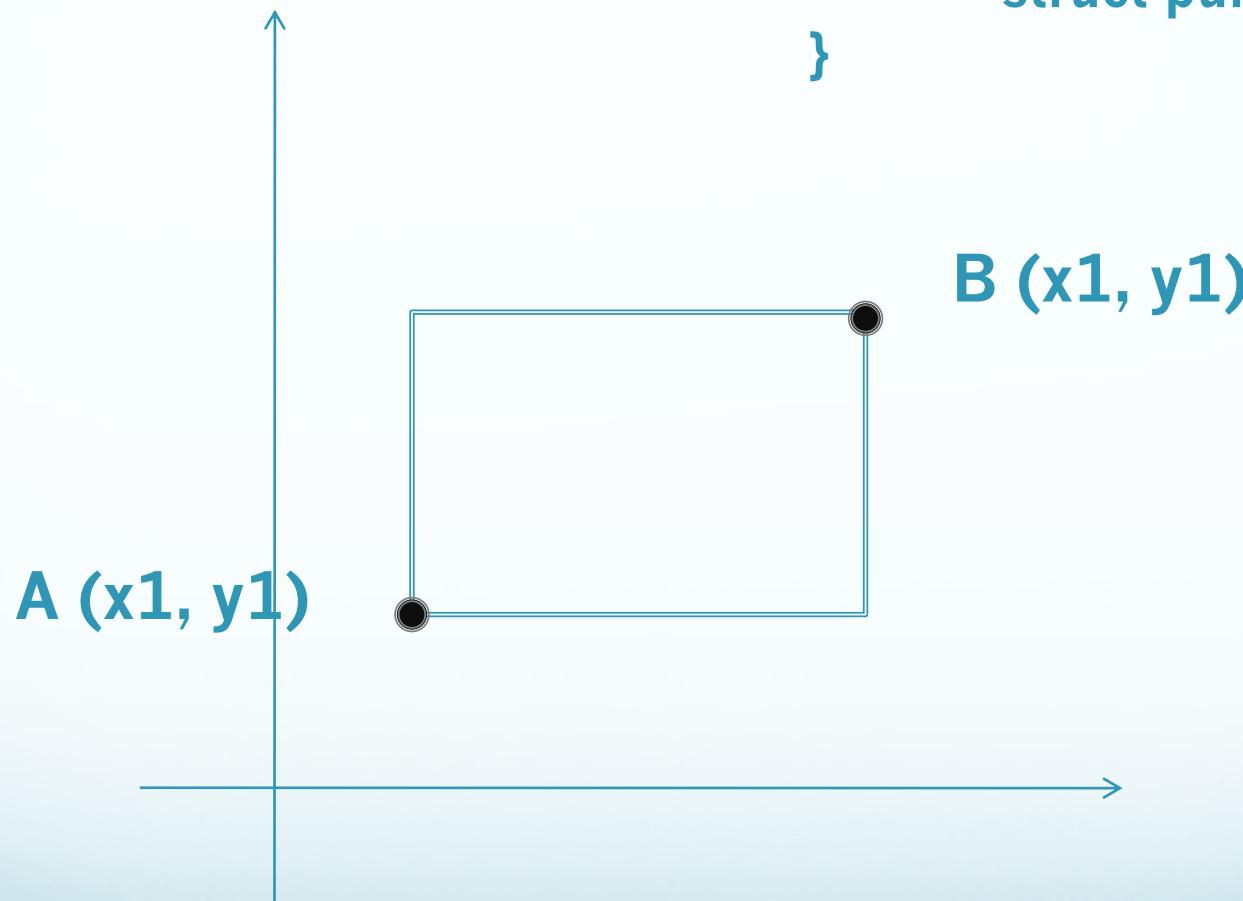
Ejemplo 1.11

```
#include<stdio.h>

struct alumno{
    int num_cuenta;
    char nombre [50];
} a;

void main() {
    printf("Introduce tu nombre: ");
    scanf("%s", a.nombre);
    printf("Introduce tu número de cuenta: ");
    scanf("%d", &a.num_cuenta);
    printf("Alumno: %s\n Número de cuenta: %d\n",
           a.nombre, a.num_cuenta);
}
```

Estructuras anidadas



```
struct rectangulo {  
    struct punto a;  
    struct punto b;  
}
```

Ejemplo 1.12

```
#include<stdio.h>

struct punto {
    int x, y;
};

struct rectangulo {
    struct punto a, b;
};

int main() {
    struct rectangulo r;
    struct punto uno = {5,3}, dos = {8,6};
    r.a = uno;
    r.b = dos;
    printf("Punto a: x = %d, y = %d\n", r.a.x, r.a.y);
    printf("Punto b: x = %d, y = %d\n", r.b.x, r.b.y);
    return 4;
}
```

Una función puede recibir como parámetros estructuras de datos.

```
int ptoEnRect (struct punto a, struct rectangulo y) {  
    if ((y.a.x < a.x && a.x < y.b.x) &&  
        (y.a.y < a.y && a.y < y.b.y) {  
            return 1;  
    } else {  
        return 0;  
    }  
}
```

Una función puede regresar como valor de retorno una estructura de datos.

```
struct punto crearPunto (int x, int y) {  
    struct punto temporal;  
  
    temporal.x = x;  
    temporal.y = y;  
  
    return temporal;  
}
```

Una función también puede recibir como parámetros y regresar como valor de retorno estructuras de datos.

```
struct punto sumarPuntos (struct punto a, struct punto b) {  
    a.x += b.x;  
    a.y += b.y;  
    return a;  
}
```

Cuando se van a pasar estructuras de datos grandes hacia una función, es mejor realizarlo mediante apuntadores.

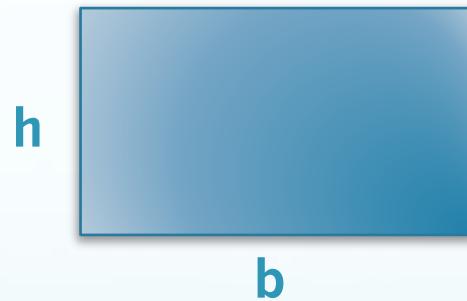
```
void main() {  
    struct rectangulo r, *apRect;  
    apRect = &r;  
    printf("Punto a = (%d, %d)",  
          (*apRect).a.x, (*apRect).a.y);  
}
```

```
printf("Punto a = (%d, %d)", (*apRect)->a.x, (*apRect)->a.y);
```

Tarea 1.2

Realizar un programa que permita calcular el área de un rectángulo a partir de puntos dados. Es necesario utilizar estructuras tanto para los puntos como para el rectángulo.

Si los puntos no dibujan un rectángulo, mandar un mensaje de error.



1.3 Archivos y bancos de datos

Un archivo es un conjunto de datos estructurados en una colección de entidades elementales o básicas denominadas registros que son de igual tipo.

Se puede conseguir la entrada y la salida de datos a un archivo a través del uso de la biblioteca de funciones de la librería STUDIO.H

Un banco de datos es un conjunto de datos pertenecientes al un mismo contexto y almacenados sistemáticamente para su posterior uso.

En este sentido, una biblioteca puede considerarse una base de datos compuesta en su mayoría por documentos y textos impresos en papel e indexados para su consulta.

En la actualidad, y debido al desarrollo tecnológico, la mayoría de los bancos de datos están en formato digital (electrónico), lo que ofrece un amplio rango de soluciones al problema de almacenar datos.

Apuntador a archivo

El apuntador (o puntero) a un archivo es el hilo común que unifica el sistema de E/S con un buffer.

Un apuntador a un archivo es un puntero a la información que define ciertas características sobre él, incluyendo el nombre, el estado y la posición actual del archivo.

Un apuntador identifica un archivo específico y utiliza la secuencia asociada para dirigir el funcionamiento de las funciones de E/S con buffer.

Un apuntador a un archivo es una variable de tipo puntero al tipo FILE que se define en STDIO.H.

Un programa necesita utilizar punteros a archivos para leer o escribir en los mismos.

Para obtener una variable de este tipo se utiliza una secuencia como esta:

FILE *F;

Abrir archivo

La función fopen() abre una secuencia para que pueda ser utilizada y la asocia a un archivo. Su estructura es la siguiente:

```
*FILE fopen(char *nombre_archivo, char *modo);
```

Donde nombre_archivo es un puntero a una cadena de caracteres que representan un nombre valido del archivo y puede incluir una especificación del directorio. La cadena a la que apunta modo determina como se abre el archivo.

Modo de abrir archivo

- **r: Abre un archivo de texto para lectura.**
- **w: Crea un archivo de texto para escritura.**
- **a: Abre un archivo de texto para añadir.**
- **r+: Abre un archivo de texto para lectura / escritura.**
- **w+: Crea un archivo de texto para lectura / escritura.**
- **a+: Añade o crea un archivo de texto para lectura / escritura.**

Cerrar archivo

La función fclose() cierra una secuencia que fue abierta mediante una llamada a fopen(). Escribe la información que se encuentre en el buffer al disco y realiza un cierre formal del archivo a nivel del sistema operativo.

Un error en el cierre de una secuencia puede generar todo tipo de problemas, incluyendo la pérdida de datos, destrucción de archivos y posibles errores intermitentes en el programa.

La estructura de esta función es:

```
int fclose(FILE *apArch);
```

Donde apArch es el apuntador al archivo devuelto por la llamada a fopen(). Si se devuelve un valor cero significa que la operación de cierre ha tenido éxito. Generalmente, esta función solo falla cuando un disco se ha retirado antes de tiempo o cuando no queda espacio libre en el mismo.

Ejemplo 1.13**fopen y fclose**

```
#include<stdio.h>
```

```
int main() {
    FILE *archivo;
    archivo = fopen("ej13.txt", "r");
    if (archivo != NULL) {
        printf("El archivo se abrió correctamente.");
    } else {
        printf("Error al abrir el archivo.");
    }
    fclose(archivo);
    return 30;
}
```

Leer y escribir en archivo

Las funciones fgets() y fputs() pueden leer y escribir cadenas sobre los archivos. Las estructuras de estas funciones son:

```
char *fputs(char *buffer, FILE *apArch);  
char *fgets(char *buffer, int tamaño, FILE *apArch);
```

La función puts() escribe la cadena a un archivo específico. La función fgets() lee una cadena desde el archivo especificado hasta que lee un carácter de nueva línea o longitud-1 caracteres.

fgets**Ejemplo 1.14****#include<stdio.h>**

```
int main() {
    FILE *archivo;
    char caracteres [50];
    archivo = fopen("ej14.txt", "r");
    if (archivo != NULL) {
        printf("El archivo se abrió correctamente.");
        printf("Contenido del archivo\n");
        while (feof(archivo) == 0) {
            fgets (caracteres, 50, archivo);
            printf("%s", caracteres);
        }
    }
    fclose(archivo);
    return 32;
}
```

Ejemplo 1.15**fputs**

```
#include<stdio.h>

int main() {
    FILE *archivo;
    char escribir [] = “Ejemplo 15. \nEscribir cadena en
                      un archivo mediante fputs.”;
    archivo = fopen(“archivo.txt”, “r+”);
    if (archivo != NULL) {
        printf(“El archivo se abrió correctamente.”);
        fputs (escribir, archivo);
    }
    fclose(archivo);
    return 0;
}
```

Las funciones `fprintf()` y `fscanf()` se comportan exactamente como `printf()` y `scanf()`, excepto que operan sobre archivo. Sus estructuras son:

```
int fprintf(FILE *apArch, char *formato, ...);  
int fscanf(FILE *apArch, char *formato, ...);
```

Donde `apArch` es un puntero al archivo devuelto por una llamada a `fopen()`. `fprintf()` y `fscanf()` dirigen sus operaciones de E/S al archivo al que apunta `apArch`.

Ejemplo 1.16

fscanf

```
#include<stdio.h>

int main() {
    FILE *archivo;
    char caracteres [50];
    archivo = fopen("archivo.txt", "r");
    if (archivo != NULL) {
        fscanf(archivo, "%s", caracteres);
        printf("%s", caracteres);
    }
    fclose(archivo);
    return 32;
}
```

Ejemplo 1.17**fprintf**

```
#include<stdio.h>

int main() {
    FILE *archivo;
    char escribir [] = “Ejemplo 17.\nEscribe una cadena
                        en un archivo mediante fprintf.”;
    archivo = fopen(“archivo.txt”, “r+”);
    if (archivo != NULL) {
        fprintf(archivo, escribir);
        fprintf(archivo, “%s”, “\nPAMN”);
    }
    fclose(archivo);
    return 0;
}
```

fread

Esta función está pensada para trabajar con registros de longitud conocida. Es capaz de leer uno o varios registros de la misma longitud a partir de una dirección de memoria determinada.

El valor de retorno es el número de registros (bytes) leidos. Su sintaxis es la siguiente:

```
int fread(void *ap, size_t tam, size_t regs, FILE *archivo)
```

fwrite

Esta función está pensada para trabajar con registros de longitud conocida. Es capaz de escribir hacia un fichero uno o varios registros de la misma longitud almacenados a partir de una dirección de memoria determinada.

El valor de retorno es el número de registros escritos. Su sintaxis es la siguiente:

```
int fwrite(void *ap, size_t tam, size_t regs, FILE *archivo)
```

Ejemplo 1.18

```
#include <stdio.h>
```

```
int main(int argc, char **argv) {
    FILE *fe, *fs;
    unsigned char buffer[2048]; // Buffer de 2 Kbytes
    int bytesLeidos;

    // Si no se ejecuta el programa correctamente
    if(argc < 3) {
        printf("El programa se ejecuta de la
               siguiente manera:\n
               \tnombre <archivo_origen>
               <archivo_destino>\n");
        return 1;
    }
```

Ejemplo 1.18

```
// Se abre el archivo de entrada en lectura y binario
fe = fopen(argv[1], "rb");
if(!fe) {
    printf("El archivo %s no existe o
           no se puede abrir", argv[1]);
    return 1;
}

// Crea o sobreescribe el archivo de salida en binario
fs = fopen(argv[2], "wb");
if(!fs) {
    printf("El fichero %s no puede
           ser creado", argv[2]);
    fclose(fe);
    return 1;
}
```

Ejemplo 1.18

```
// Copia los archivos
while((bytesLeidos = fread(buffer, 1, 2048, fe))) {
    fwrite(buffer, 1, bytesLeidos, fs);
}

// Se liberan apuntadores a archivo
fclose(fe);
fclose(fs);
return 0;
}
```

Tarea 1.3

Realizar un programa en lenguaje C que permita llevar el control de calificaciones de un grupo, es decir, que permita almacenar nombre y calificación del alumno en un archivo de texto. Hay que utilizar estructuras.

Al final hay que mostrar la lista de alumnos y sus calificaciones en pantalla.

1.4 Desarrollo de programas simples con estructuras de almacenamiento complejas

- ✓ Arreglos de varias dimensiones y arreglos de apuntadores.
- ✓ Estructuras.
- ✓ Archivos y bancos de datos.

Ejercicio 1.1 Arreglos de varias dimensiones

A=

| | | | |
|----|----|----|----|
| 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 |

Dada la matriz A

Obtener la matriz B:

B=

| | | | |
|----|----|----|----|
| 16 | 15 | 14 | 13 |
| 12 | 11 | 10 | 9 |
| 8 | 7 | 6 | 5 |
| 4 | 3 | 2 | 1 |

La matriz B se obtiene en una función diferente de main.

Ejercicio 1.2 Arreglos de apuntadores

Programar la función strcpy. La función strcpy se encuentra en la biblioteca <string.h> y se utiliza para copiar una cadena de caracteres de un arreglo o apuntador (origen) hacia otro arreglo o apuntador (destino).

La sintaxis de la función es la siguiente:

strcpy(origen, destino)

Ejercicio 1.3 Estructuras

Realizar un programa que permita llevar una lista de alumnos con la calificación final de cada uno.

De igual manera, realizar una función (externa a main) que obtenga el alumno con la calificación más alta.

Ejercicio 1.4 Archivos y bancos de datos

Realizar un programa en C que permita insertar crear, leer y escribir un archivo. Las actividades se seleccionan a partir de un menú.

Cada actividad (crear, leer y escribir) así como el menú se generan en funciones externas a la principal.

Tarea 1.4

A partir de una matriz dada:

| | | | |
|----|----|----|----|
| 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 |

Tarea 1.4

Ordenar los valores de modo inverso, es decir, de mayor a menor:

| | | | |
|----|----|----|----|
| 16 | 15 | 14 | 13 |
| 12 | 11 | 10 | 9 |
| 8 | 7 | 6 | 5 |
| 4 | 3 | 2 | 1 |

Ya que se tenga la matriz resultante, hay que multiplicarla por la estructura tipoDeDatos:

```
struct tipoDeDatos {  
    int i;  
    float f;  
}
```

Tarea 1.4

El reacomodo de la matriz se debe hacer desde otra función pasando como argumento la matriz original. El recorrido de la matriz se debe realizar con un apuntador.

La multiplicación de la matriz por cada uno de los elementos tipoDeDatos se debe realizar desde otra función, pasando como argumentos la matriz y la estructura con los valores proporcionados por el usuario.

Tarea 1.4

El resultado de la multiplicación hay que guardarla en un archivo de texto. El nombre del archivo se debe recibir al momento de ejecutar el programa (como argumento).

Tema 1: Elaborar programas en lenguaje C empleando estructuras de almacenamiento de datos complejas.

- **Subtema 1.1:** Describir el manejo de arreglos de diversos tipos de datos, así como el funcionamiento de un apuntador.
- **Subtema 1.2:** Explicar el concepto de estructura, así como su funcionamiento.
- **Subtema 1.3:** manejar archivos para lectura-escritura.
- **Subtema 1.4:** Elaborar, con base en los conceptos anteriores, programas más completos.