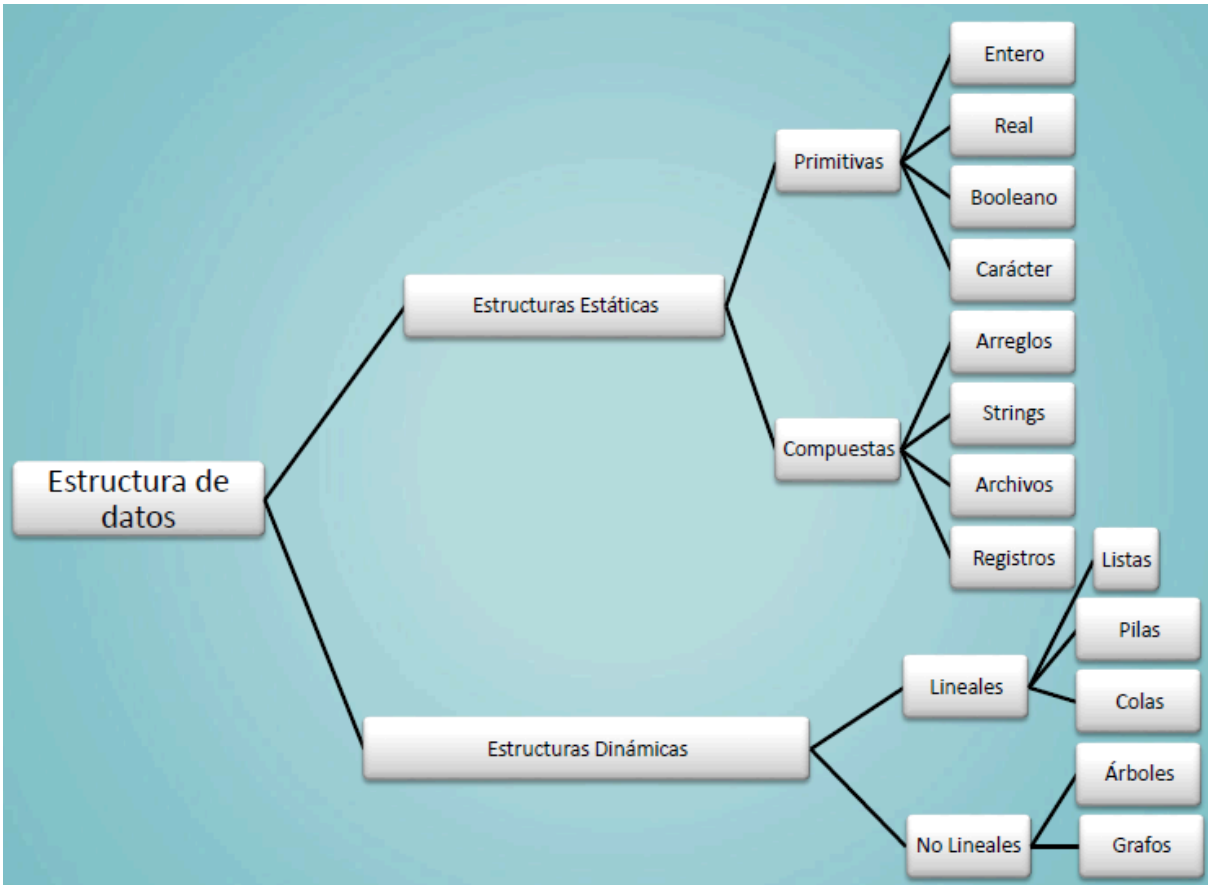


ARREGLOS

Un arreglo (o array) es una colección de variables del mismo tipo almacenadas en posiciones contiguas de memoria.

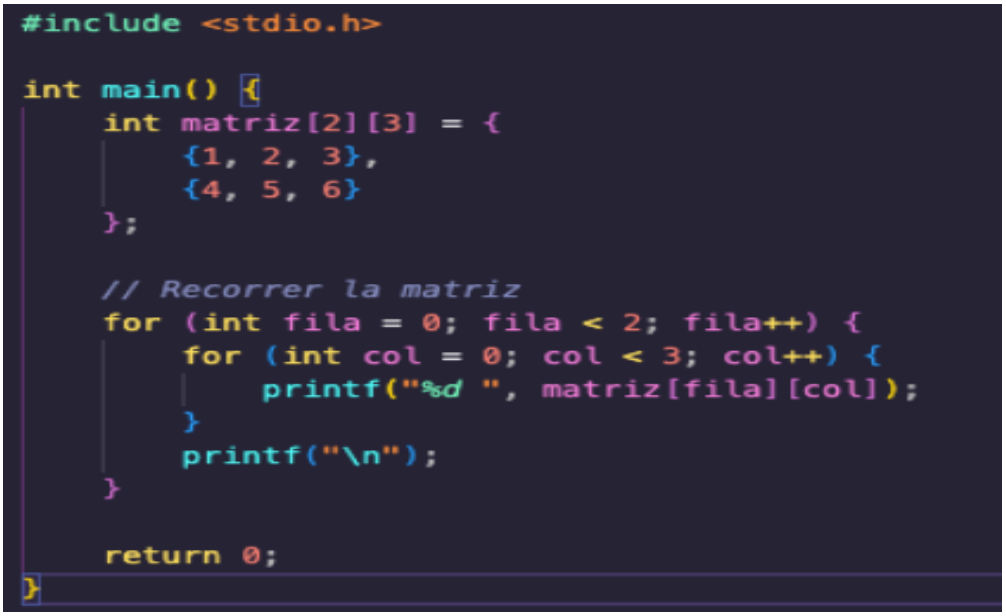
Se puede definir como un conjunto finito ordenado de elementos homogéneos.

Sirve para guardar muchos datos similares bajo un mismo nombre.

Como se define y recorre un arreglo UNIDIMENSIONAL?

```c
#include <stdio.h>

int main() {
    int numeros[5] = {10, 20, 30, 40, 50};

    // Recorrido: mostrar cada número
    for (int i = 0; i < 5; i++) {
        printf("Elemento %d: %d\n", i, numeros[i]);
    }

    return 0;
}
```

Como se define y recorre un arreglo BIDIMENSIONAL?



```c
#include <stdio.h>

int main() {
    int matriz[2][3] = {
        {1, 2, 3},
        {4, 5, 6}
    };

    // Recorrer la matriz
    for (int fila = 0; fila < 2; fila++) {
        for (int col = 0; col < 3; col++) {
            printf("%d ", matriz[fila][col]);
        }
        printf("\n");
    }

    return 0;
}
```

Operaciones con Arreglo: Inserción

En C los arreglos tienen tamaño fijo, así que podemos reemplazar valores, pero no agregar nuevos dinámicamente sin usar estructuras más avanzadas.

1)Verificar que el arreglo tenga posiciones disponibles para insertar un nuevo elemento

2)Si existe posiciones disponibles proceder a realizar la inserción

3)Si el arreglo esta ordenado se debe realizar la inserción en la posición que corresponda para que quede ordenado después de realizar la misma (al inicio, al medio o al final)

4)Si el arreglo no esta ordenado se puede optar por insertar el nuevo elemento al final del arreglo.

5)Para realizar la inserción se debe mover los elementos una posición hacia la derecha.

Es necesario conocer la posición del último elemento del arreglo.

Como calcular cuantos elementos tiene un arreglo?

```c
int size = sizeof(original) / sizeof(original[0]);
```

sizeof(original) devuelve el tamaño total en bytes ocupado por el arreglo.

sizeof(original[0]) devuelve el tamaño en bytes del primer elemento del arreglo (un entero en este caso).

Al dividir el tamaño total del arreglo entre el tamaño de un elemento, se obtiene el número de elementos que contiene el arreglo.

Operaciones con Arreglo: Eliminacion

1)Verificar que el arreglo se encuentre el elemento a eliminar
2)Si existe el elemento en el arreglo proceder a realizar la eliminación
3)La eliminación se realiza moviendo los elementos una posición hacia la izquierda los elementos del arreglo, quedando una posición libre Para esto es necesario identificar la posición en la que se encuentra el elemento a eliminar.
4)Luego de realizar la eliminación el arreglo cuenta con un elemento menos

Operaciones con Arreglo: Busqueda

BUSQUEDA BINARIA

Para realizarla, es necesario contar con un arreglo ordenado.
Se toma un elemento central, normalmente el elemento que se encuentra a la mitad del arreglo, y se lo compara con el elemento buscado Si el elemento buscado es menor, se toma el intervalo que va desde el elemento central al principio, en caso contrario, se toma el intervalo que va desde el elemento central hasta el final del intervalo.

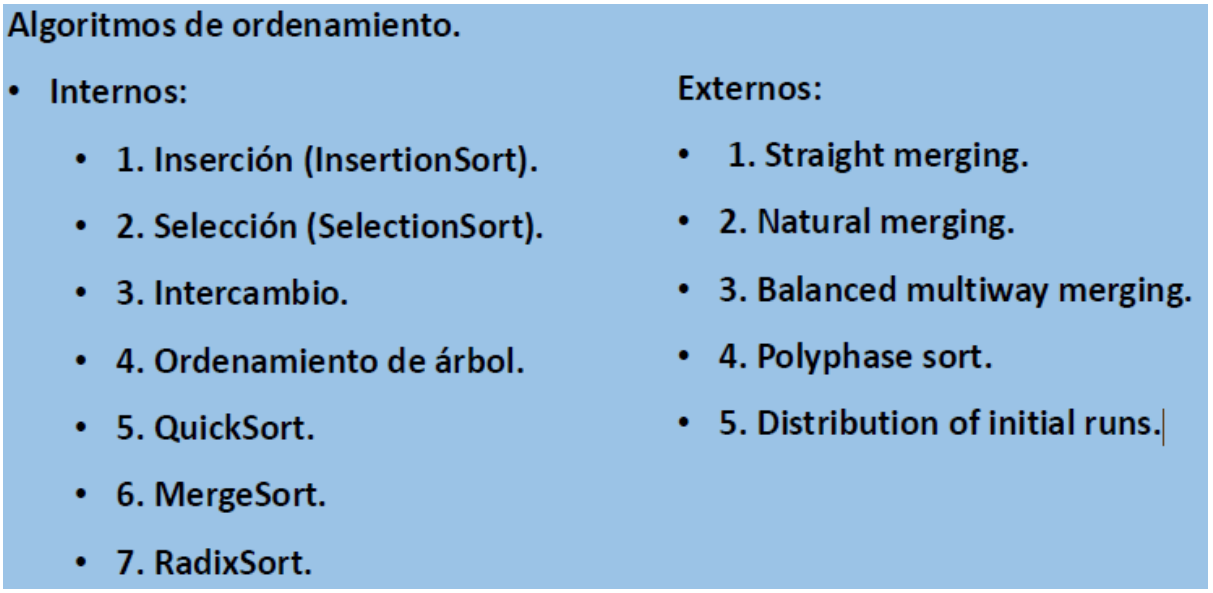
```c
int main() {
    int arreglo[] = {3, 7, 15, 20, 25, 31, 40}; // Arreglo ORDENADO
    int n = sizeof(arreglo) / sizeof(arreglo[0]);
    int buscar, inicio = 0, fin = n - 1, medio;
    int encontrado = 0;
    printf("Ingrese el numero que desea buscar: ");
    scanf("%d", &buscar);
    while (inicio <= fin) {
        medio = (inicio + fin) / 2;

        if (arreglo[medio] == buscar) {
            printf("Numero encontrado en la posicion %d.\n", medio);
            encontrado = 1;
            break;
        } else if (buscar < arreglo[medio]) {
            fin = medio - 1;
        } else {
            inicio = medio + 1;
        }
    }
    if (!encontrado) {
        printf("El numero no se encuentra en el arreglo.\n");
    }
    return 0;
}
```

Es más rápida que la búsqueda lineal, especialmente en arreglos grandes.

Menos comparaciones que la busqueda lineal.

- Se comparan los extremos: izquierda (`inicio`) y derecha (`fin`).
- Se calcula el **índice del medio**:
  ```
  medio = (inicio + fin) / 2
  ```
- Se compara el valor en esa posición con el valor buscado:
  - Si son iguales → **¡Listo! Elemento encontrado.**
  - Si el buscado es menor → Buscar en la **mitad izquierda**.
  - Si es mayor → Buscar en la **mitad derecha**.
- Se repite el proceso hasta encontrarlo o que no queden elementos.

Operaciones con Arreglo: Ordenamiento



Algoritmos de intercambio:

En este tipo de algoritmos se toman los elementos de dos en dos, se comparan y se INTERCAMBIAN si no están en el orden adecuado. Este proceso se repite hasta que se ha analizado todo el conjunto de elementos y ya no hay intercambios.

Entre estos algoritmos se encuentran el **Burbuja (BubbleSort) y QuickSort**

```c
int arr[] = {5, 2, 9, 1, 6};
int n = sizeof(arr) / sizeof(arr[0]);
int i, j, temp;

for (i = 0; i < n - 1; i++) {
    for (j = 0; j < n - i - 1; j++) {
        if (arr[j] > arr[j + 1]) {
            // Intercambio
            temp = arr[j];
            arr[j] = arr[j + 1];
            arr[j + 1] = temp;
        }
    }
}

printf("Arreglo ordenado: ");
for (i = 0; i < n; i++) {
    printf("%d ", arr[i]);
}
```

Algoritmos De Inserción:

En este tipo de algoritmo los elementos que van a ser ordenados son considerados no a la vez.

Cada elemento es INSERTADO en la posición apropiada con respecto al resto de los elementos ya ordenados.

Entre estos algoritmos se encuentran el de **Inserción Directa, ShellSort, Inserción Binaria y Hashing**.

```c
void insertionSort(int arr[], int n) {
    int i, key, j;
    for (i = 1; i < n; i++) {
        key = arr[i]; // Toma el elemento actual como la clave
        j = i - 1;   // Comienza a comparar con el elemento anterior

        // Desplaza los elementos mayores a la clave hacia la derecha
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key; // Inserta la clave en la posición correcta
    }
}
```

Algoritmos De Selección:

En este tipo de algoritmos se SELECCIONA o se busca el elemento más pequeño(o más grande) de todo el conjunto de elementos y se coloca en su posición adecuada.Este proceso se repite para el resto de los elementos hasta que todos son analizados.

Entre estos algoritmos se encuentra el de **Selección Directa.**

```c
void selectionSort(int arr[], int n) {
    int i, j, min_idx, temp;
    // Iterar a través de la lista
    for (i = 0; i < n - 1; i++) {
        // Asumir que el primer elemento es el mínimo
        min_idx = i;
        // Buscar el elemento mínimo en el resto de la lista
        for (j = i + 1; j < n; j++) {
            if (arr[j] < arr[min_idx]) {
                min_idx = j;
            }
        }
        // Intercambiar el elemento mínimo con el primer elemento
        if (min_idx != i) {
            temp = arr[i];
            arr[i] = arr[min_idx];
            arr[min_idx] = temp;
        }
    }
}
```

Método	Comparaciones	Fácil de entender	Eficiente
Burbuja	Muchas	✅ Sí	❌ No
Selección	Menos que burbuja	✅ Sí	❌ No
Inserción	Menos si está casi ordenado	✅ Sí	⚠️ Regular

CADENA DE CARACTERES

Una cadena es un conjunto de caracteres, o valores de tipochar, terminados con el carácter nulo, es decir el valor numérico 0.
Declaraciones de cadenas:
char nombre_arr[tam]=”cadena";
char cadena[5]={'h','o','l','a',' 0'};

CADENA DE CARACTERES:

- FUNCIONES:
  - strcopy(destino, origen); // Copia una cadena en otra.
  - strlen(cadena); // Da la longitud(int) de la cadena.
  - strcmp(cad1,cad2); // Compara cadenas caracter x caracter.

```c
if (strcmp(a, b) == 0) {
    printf("a y b son iguales\n");
}

if (strcmp(a, c) != 0) {
    printf("a y c son diferentes\n");
}
```

  - strcat(destino, origen); // concatena dos cadenas
  - strchar(cadena, caracter); // Busca un caracter y devuelve un resultado de puntero.

```c
char palabra[] = "computadora";
char *ptr = strchr(palabra, 'u');

if (ptr != NULL) {
    printf("Encontrado: %c en la posición %ld\n", *ptr, ptr - palabra);
} else {
    printf("No se encontró el carácter\n");
}
```

  - strstr(cadena, subcadena); // Busca una subcadena dentro de una cadena.

```c
char frase[] = "Me gusta programar en C";
char *ptr = strstr(frase, "programar");

if (ptr != NULL) {
    printf("Subcadena encontrada: %s\n", ptr);
} else {
    printf("Subcadena no encontrada\n");
}
```

ESTRUCTURAS DE REGISTRO

Una **estructura** en C (llamada `struct`) es una forma de **crear un tipo de dato propio**, que

agrupa varios datos diferentes en una sola unidad.

Una estructura o registro es una estructura de datos que agrupa variables que pueden tener tipos diferentes.

Cada componente del registro se conoce como campo o miembro.

Definir una estructura

```
struct Persona {
    char nombre[30];
    int edad;
    float altura;
};
```

Crear variables de tipo estructura

```
struct Persona p1;
```

Ahora `p1` tiene:

- `p1.nombre`
- `p1.edad`
- `p1.altura`

Asignar valores

Para copiar una cadena, usamos `strcpy()` del `<string.h>`.

```
strcpy(p1.nombre, "Lucía");
p1.edad = 25;
p1.altura = 1.65;
```

¿Cuándo usar estructuras?

Cuando tenés **varios datos relacionados entre sí**, como:

- Los datos de un estudiante (nombre, legajo, nota)
- Una fecha (día, mes, año)
- Un producto (nombre, precio, stock)

FUNCIONES Y PROCEDIMIENTOS

Funciones

Una FUNCIÓN es un subprograma que realiza una tarea determinada y bien acotada a la cual le pasamos datos y nos devuelve datos.

Esta función se ejecuta cuando se lo llama (llamada a la función).

Ventajas:

- El programa es mas simple de comprender ya que cada módulo se dedica a realizar una tarea en particular.
- •La depuración queda acotada a cada módulo.
- •Las modificaciones al programa se reducen a modificar determinados módulos.
- •Cuando cada módulo esta bien probado se lo puede usar las veces que sea necesario sin volver a revisarlo.
- •Se obtiene una independencia del código en cada módulo.

PASAJE DE PARÁMETROS

1. Pasaje por valor (el más común en C)

La función **recibe una copia** del valor original. Si la función modifica el valor, **no afecta** al original.

2. Pasaje por referencia (usando punteros)

La función **recibe la dirección** de la variable, por lo tanto, **puede modificar el valor original**.

```c
/*Pasaje por Referencia*/
#include <stdio.h>

void duplicar(int *x) {
    *x = *x * 2;
    printf("Dentro de la funcion: %d\n", *x);
}

int main() {
    int numero = 5;
    duplicar(&numero);
    printf("Fuera de la funcion: %d\n", numero);
    return 0;
}
```

Procedimientos

Un **procedimiento** es una función que **NO devuelve valor**. En C, se usa el tipo `void`.

Procedimientos serían, por ejemplo, la función printf no se invoca para calcular valores nuevos, sino para realizar una tarea sobre las variables

EnC se usa una función de tipo void para realizar un procedimiento.

Una función tipo void no necesariamente tendrá la sentencia return. Si Una función de tipo void hace uso de sentencias return, entonces en ningún caso debe seguir a esa palabra valor alguno.

RETURN

Fuerza la salida inmediata del cuerpo de la función y se vuelve a la siguiente sentencia después de la llamada.
La forma general de la sentencia return es: return[expresion];
Si el tipo de dato de la expresión del return no coincide con el tipo de la función entonces, de forma automática, el tipo de dato de la expresión se convierte en el tipo de dato de la función.

Variables Globales y Locales

Todas
las variables que se encuentren definidas dentro de las llaves de una función (recuerde que main también es una función) tienen validez dentro de dicha función y se llaman VARIABLES LOCALES.
Ahora si una variable puede ser usada desde cualquier función y durante el transcurso de todo el programa, esa es una VARIABLE GLOBAL Las variables globales se definen fuera de cualquier función normalmente debajo de los include que se colocan al comienzo del programa.