Principios de Programación

Arreglos

Estructuras de datos básicas, declaración e inicialización de arreglos, acceso por índice y recorridos con `for`.

Recordando la semana anterior

Base previa

- Ya se recorrieron valores con `for`, `while` y `do-while`
- Ya se usaron contadores, acumuladores y comparaciones

Definición

Un arreglo es la primera estructura de datos del curso: permite agrupar valores del mismo tipo bajo un mismo nombre para accederlos y recorrerlos por posición.

- Reúne datos relacionados en una sola colección
- Evita repetir variables casi iguales
- Permite leer, modificar y procesar los datos con bucles

Necesidad de una colección de datos

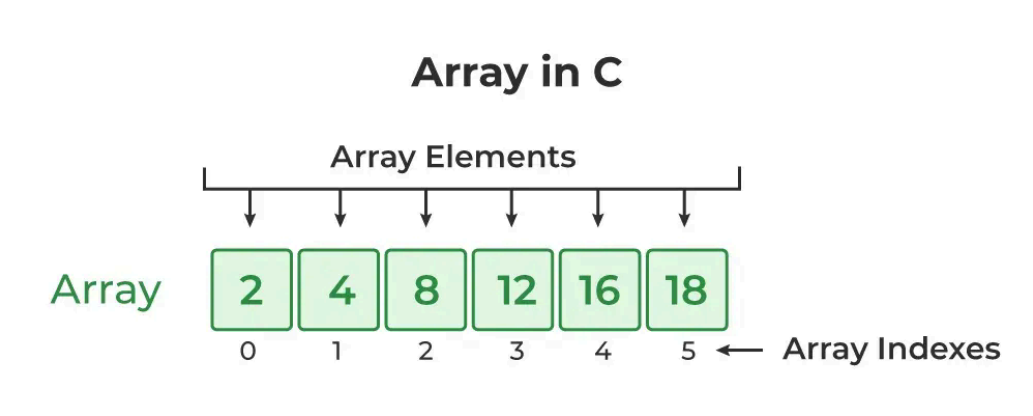
SIN ARREGLOS

```c
int nota1, nota2, nota3, nota4, nota5;

printf("Ingrese nota 1: ");
scanf("%d", &nota1);
printf("Ingrese nota 2: ");
scanf("%d", &nota2);
printf("Ingrese nota 3: ");
scanf("%d", &nota3);
// ...
```

Problema

- El patrón se repite con cambios mínimos
- Si la cantidad de datos aumenta, el código se vuelve largo y rígido
- Calcular sumas, promedios o máximos exige escribir muchas líneas parecidas



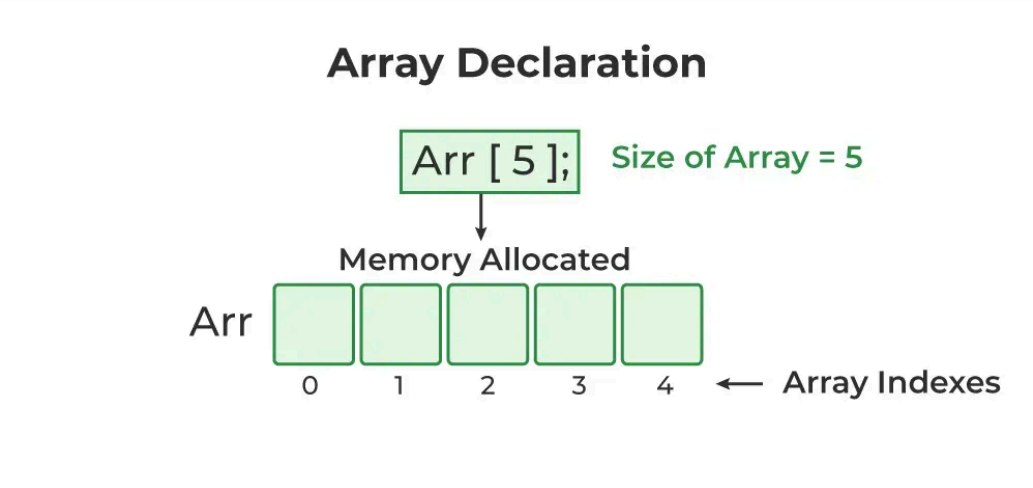
Definición

Una colección de elementos del mismo tipo almacenados en posiciones consecutivas de memoria.

Características

- Homogéneo: todos los elementos comparten el mismo tipo
- Ordenado: cada posición tiene un índice
- Finito: tiene una cantidad determinada de elementos
- De tamaño fijo: el tamaño queda definido al declararlo (**por ahora**)

Declaración de un arreglo



SINTAXIS

```
tipo nombre[tamano];
```

Componentes

- `tipo`: clase de dato de cada elemento
- `nombre`: identificador del arreglo
- `tamano`: cantidad de posiciones disponibles

SINTAXIS

```
int notas[5];
float temperaturas[10];
char apellido[50];
```

Regla

En este curso se asume que el **tamaño queda determinado** al momento de declarar el arreglo.

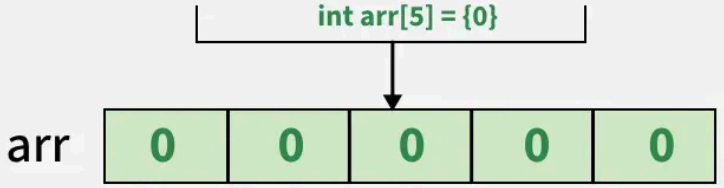
Inicialización de arreglos

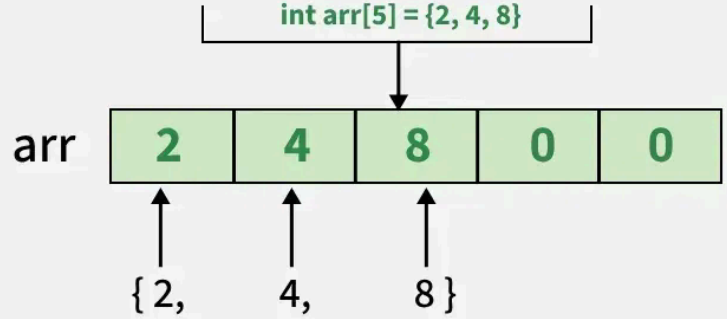
DECLARACIÓN EN CERO

```
int notas[5] = {0};
```

DECLARACIÓN CON VALORES

```
int notas[5] = {2, 4, 8};
```





Inicialización con tamaño inferido

EJEMPLO

```
int notas[] = {85, 92, 78, 90, 88};
```

Lectura

- El compilador cuenta cuántos valores hay en la lista
- En este caso, el arreglo queda con tamaño 5
- La cantidad de elementos iniciales determina el tamaño final

Criterio

Si el tamaño ya se conoce y es parte del diseño del problema, conviene declararlo de forma explícita para que el código resulte más claro.

Acceso a elementos individuales

LECTURA Y ESCRITURA

```
int notas[5] = {85, 92, 78, 90, 88};

int primera = notas[0];
notas[2] = 95;
int suma = notas[0] + notas[1];
```

Reglas

- `arreglo[indice]` sirve para **leer** y también **para modificar**
- El índice puede ser una constante, una variable o una expresión
- El valor del índice debe quedar dentro del rango válido

Acceder fuera del rango del arreglo produce comportamiento indefinido.

Ejemplo: leer, mostrar y modificar

CÓDIGO EN C

```c
#include <stdio.h>

int main() {
    int notas[5];
    int i;

    for (i = 0; i < 5; i++) {
        printf("Ingrese nota %d: ", i + 1);
        scanf("%d", &notas[i]);
    }

    printf("Primera nota: %d\n", notas[0]);
    printf("Ultima nota: %d\n", notas[4]);

    notas[2] = 100;
    printf("Tercera nota (modificada): %d\n", notas[2]);

    return 0;
}
```

Lectura

El bucle carga cada posición del arreglo usando `notas[i]`.

Accesos puntuales

Después de la carga se consultan dos posiciones y se modifica una tercera.

Recorrer un arreglo con `for`

PATRÓN GENERAL

```c
for (int i = 0; i < tamano; i++) {
    // Trabajar con arreglo[i]
}

int notas[5] = {85, 92, 78, 90, 88};

for (int i = 0; i < 5; i++) {
    printf("Nota %d: %d\n", i + 1, notas[i]);
}
```

Lectura

- `i` recorre las posiciones del arreglo
- La condición `i < tamano` evita salir del rango
- `i + 1` solo se usa para mostrar una numeración humana

Ejemplo: calcular el promedio

CÓDIGO EN C

```c
#include <stdio.h>

int main() {
    int notas[5];
    int suma = 0;
    float promedio;
    int i;

    for (i = 0; i < 5; i++) {
        printf("Ingrese nota %d: ", i + 1);
        scanf("%d", &notas[i]);
    }

    for (i = 0; i < 5; i++) {
        suma = suma + notas[i];
    }

    promedio = suma / 5.0;
    printf("El promedio es: %.2f\n", promedio);
    return 0;
}
```

Método

- Primer recorrido: cargar el arreglo
- Segundo recorrido: acumular los valores
- Cálculo final: dividir la suma entre la cantidad de elementos

Observación

Se usa `5.0` para que la división produzca un resultado de punto flotante.

Ejemplo: encontrar el mayor valor

CÓDIGO EN C

```c
#include <stdio.h>

int main() {
    int numeros[5];
    int mayor, i;

    for (i = 0; i < 5; i++) {
        printf("Ingrese numero %d: ", i + 1);
        scanf("%d", &numeros[i]);
    }

    mayor = numeros[0];

    for (i = 1; i < 5; i++) {
        if (numeros[i] > mayor) {
            mayor = numeros[i];
        }
    }

    printf("El mayor es: %d\n", mayor);
    return 0;
}
```

Estrategia

- Tomar el primer elemento como valor inicial de `mayor`
- Comparar desde la segunda posición en adelante
- Reemplazar `mayor` cuando aparece un valor superior

Ejemplo: llenar un arreglo con valores

CÓDIGO EN C

```c
#include <stdio.h>

int main() {
    int numeros[10];
    int i;

    for (i = 0; i < 10; i++) {
        numeros[i] = i + 1;
    }

    for (i = 0; i < 10; i++) {
        printf("%d ", numeros[i]);
    }
    printf("\n");

    return 0;
}
```

Lectura

- El índice sigue yendo de `0` a `9`
- El valor almacenado usa `i + 1` para generar la secuencia `1` a `10`
- Índice y dato no tienen por qué coincidir

Índices inválidos

CÓDIGO CON ERROR

```c
int notas[5] = {85, 92, 78, 90, 88};

printf("%d\n", notas[5]);
printf("%d\n", notas[-1]);
```

MENSAJE / PROBLEMA

- Los índices válidos son `0`, `1`, `2`, `3` y `4`.
- `notas[5]` y `notas[-1]` quedan fuera del arreglo.
- El programa entra en comportamiento indefinido.

CORRECCIÓN

- Verificar siempre que el índice quede en el rango `0 <= i < tamano`.
- En los recorridos, usar la condición `i < tamano`.

Valores indefinidos

CÓDIGO CON ERROR

```
int notas[5];
int suma = 0;
int i;

for (i = 0; i < 5; i++) {
    suma = suma + notas[i];
}
```

MENSAJE / PROBLEMA

- `notas[i]` no tiene un valor definido si antes no se inicializa o se lee.
- Sumar esos datos produce resultados incorrectos.

CORRECCIÓN

```
int notas[5] = {0};
```

o bien leer cada elemento antes de usarlo.

Condición de recorrido

CÓDIGO CON ERROR

```c
int notas[5] = {85, 92, 78, 90, 88};
int i;

for (i = 0; i <= 5; i++) {
    printf("%d ", notas[i]);
}
```

MENSAJE / PROBLEMA

- La condición `i <= 5` permite que `i` llegue a `5`.
- Esa iteración intenta acceder a `notas[5]`, que no existe.

CORRECCIÓN

```c
for (i = 0; i < 5; i++) {
    printf("%d ", notas[i]);
}
```

Ejemplo: estadísticas de notas

Problema

Leer 5 notas y calcular:

- suma
- promedio
- mayor
- menor

Método

- Cargar el arreglo
- Inicializar `mayor` y `menor` con `notas[0]`
- Recorrer el arreglo para acumular y comparar

Ejemplo: estadísticas de notas

CÓDIGO EN C

```c
#include <stdio.h>

int main() {
    int notas[5];
    int suma = 0;
    int mayor, menor;
    float promedio;
    int i;

    for (i = 0; i < 5; i++) {
        printf("Ingrese nota %d: ", i + 1);
        scanf("%d", &notas[i]);
    }

    mayor = notas[0];
    menor = notas[0];

    for (i = 0; i < 5; i++) {
        suma = suma + notas[i];
        if (notas[i] > mayor) {
            mayor = notas[i];
        }
        if (notas[i] < menor) {
            menor = notas[i];
        }
    }

    promedio = suma / 5.0;

    printf("Suma: %d\n", suma);
    printf("Promedio: %.2f\n", promedio);
    printf("Mayor: %d\n", mayor);
    printf("Menor: %d\n", menor);

    return 0;
}
```

Observaciones

- Un mismo recorrido puede resolver varias tareas a la vez
- La comparación parte del primer elemento, no de un valor arbitrario
- El promedio se calcula al final

Ejemplo: contar elementos mayores

CÓDIGO EN C

```c
#include <stdio.h>

int main() {
    int numeros[10];
    int valor;
    int contador = 0;
    int i;

    for (i = 0; i < 10; i++) {
        printf("Ingrese numero %d: ", i + 1);
        scanf("%d", &numeros[i]);
    }

    printf("Ingrese valor de referencia: ");
    scanf("%d", &valor);

    for (i = 0; i < 10; i++) {
        if (numeros[i] > valor) {
            contador = contador + 1;
        }
    }

    printf("Hay %d numeros mayores que %d\n", contador, valor);
    return 0;
}
```

Lectura

- El arreglo permite separar la fase de carga de la fase de conteo
- Cada comparación usa `numeros[i]`
- `contador` resume cuántos elementos cumplen la condición

Ejercicio 1

Leer `10` enteros, calcular la suma, calcular el promedio y contar cuántos valores son mayores que ese promedio.

Ejercicio 2

Leer `8` enteros, determinar el mayor y el menor, y mostrar la diferencia entre ambos.

Ejercicio 3

Leer `10` enteros, contar cuántos son pares y mostrar el total encontrado.