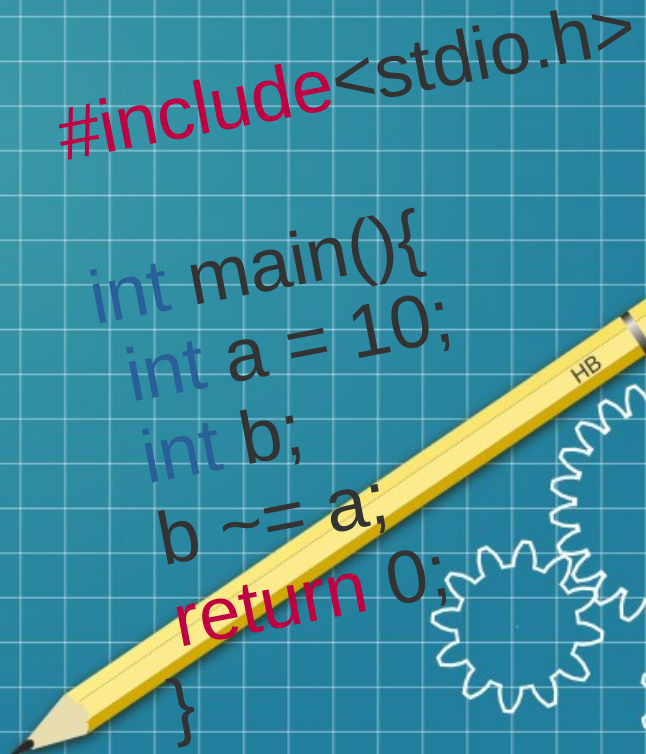
Arreglos y punteros

- Arreglos y punteros
- Aritmética de punteros
- Parámetros de punteros en funciones
- Asignación dinámica en memoria
- Función main con parámetros



Palomeque Nestor Levi

Arreglos y punteros

Un arreglo en C es una colección de elementos del mismo tipo, almacenados en ubicaciones de memoria contiguas, accesibles mediante índices.

Un puntero es una variable que almacena la dirección de memoria de otra variable, permitiendo acceso y manipulación directa de memoria.

Los arreglos y los punteros están estrechamente relacionados. En muchos casos, un nombre de arreglo se comporta como un puntero al primer elemento del arreglo.

La diferencia principal entre ellos radica en cómo se interpretan y se comportan en ciertos contextos.

Cuando utilizamos el nombre de un arreglo en una expresión, generalmente se convierte implícitamente en un puntero al primer elemento del arreglo. Esto significa que podemos realizar operaciones de punteros directamente con el nombre del arreglo, como la aritmética de punteros. Sin embargo, hay casos en los que esta relación no se aplica, especialmente cuando se trata de arreglos multidimensionales o cuando el nombre del arreglo está siendo utilizado en ciertos contextos, como cuando se pasa como argumento a sizeof.

Los arreglos tienen algunas propiedades distintas de los punteros. Por ejemplo, no se pueden reasignar para apuntar a otro lugar en memoria después de su inicialización. Además, el tamaño del arreglo es parte de su tipo, lo que significa que int arr[5] y int arr[10] son tipos de arreglos diferentes.

Aritmética de punteros

La aritmética de punteros en C es una característica muy importante que te permite manipular direcciones de memoria de forma eficiente y acceder a datos de manera flexible.

Es crucial garantizar que las operaciones de punteros se realicen dentro de los límites de la memoria asignada para evitar errores de segmentación y comportamientos indefinidos. Además, debes tener cuidado de no hacer referencia a direcciones de memoria no válidas o no inicializadas.

Incremento y decremento de punteros: puedes incrementar y decrementar un puntero utilizando los operadores **++** y **--**

Suma y resta de enteros a punteros: puedes sumar o restar un entero a un puntero utilizando los operadores aritméticos **+** y **-**

En ambos casos el tamaño del tipo apuntado es crucial. Si ***ptr*** apunta a un entero, ***ptr* + 1** moverá el puntero al siguiente entero en la memoria.

Asignación: puedo asignar el valor de un puntero a otro utilizando **=**

Comparación de punteros: Puedes comparar punteros utilizando los operadores de comparación **==**, **!=**, **<**, **>**, **<=** y **>=**

La comparación de punteros tiene sentido si los punteros apuntan a elementos de un mismo arreglo o si son NULL.

Ejemplo

En C, los arreglos y los punteros están estrechamente relacionados. En muchos casos, un nombre de arreglo se comporta como un puntero al primer elemento del arreglo.

```c
#include <stdio.h>
int main() {
    int arreglo[] = {10, 20, 30, 40, 50};
    int *ptr = arreglo;     // Declaración de un puntero a enteros e inicialización con la dirección del primer elemento del arreglo
    printf("Contenido del arreglo usando notación de corchetes:\n");
    for (int i = 0; i < 5; i++) {
        printf("%d ", arreglo[i]);
    }
    printf("\n");

    printf("Contenido del arreglo usando aritmética de punteros:\n");
    for (int i = 0; i < 5; i++) {
        printf("%d ", *(ptr + i)); // Equivalente a ptr[i]
    }
    printf("\n");
    return 0;
}
```

Parámetros de punteros en funciones

En C, puedes pasar punteros como parámetros a funciones para permitir que la función modifique los valores de variables fuera de ella.

Al pasar un puntero como parámetro, puedes evitar copiar grandes cantidades de datos en la pila de llamadas, lo que puede mejorar la eficiencia y el rendimiento del programa.

Ejemplo

Por ejemplo, emplear una función que sume todos los elementos de un arreglo, previamente inicializado:

```c
#include <stdio.h>
#define TAM 5
void sumarElementos(int *arr, int tamano) {
    int suma = 0;
    for (int i = 0; i < tamano; i++) {
        suma += arr[i]; // Acceso al elemento del arreglo utilizando corchetes. También puede usar: *(arr+i)
    }
    printf("La suma de los elementos es: %d\n", suma);
}
int main() {
    int arreglo[TAM] = {1, 2, 3, 4, 5};
    sumarElementos(arreglo, TAM); // Pasando el arreglo como puntero a la función
    return 0;
}
```

Asignación dinámica de memoria

La asignación de memoria dinámica permite reservar y liberar memoria durante la ejecución de un programa. Esto proporciona flexibilidad para manejar datos cuya cantidad o tamaño no se conoce en tiempo de compilación.

En C, hay dos funciones principales para la gestión de memoria dinámica: malloc() y free().

- **malloc(size_t size)**: se utiliza para asignar un bloque de memoria de un tamaño específico en bytes. La memoria asignada no se inicializa, lo que significa que puede contener valores residuales. Toma como argumento el número de bytes de memoria que se deben asignar y devuelve un puntero al inicio de esa memoria asignada. Si la asignación de memoria tiene éxito, **malloc()** devuelve un puntero al inicio del bloque de memoria, de lo contrario devuelve **NULL** si no puede asignar la memoria solicitada.
- **calloc(size_t num, size_t size)**: se utiliza para asignar memoria para un número específico de elementos de un tamaño dado y además inicializa toda la memoria asignada a cero. Posee dos argumentos: el número de elementos y el tamaño de cada elemento.
- **free(void* ptr)**: Después de haber utilizado la memoria asignada dinámicamente, es importante liberarla. La función **free()** se utiliza para liberar el bloque de memoria previamente asignado. Se debe pasar como argumento el puntero del bloque de memoria asignado con **malloc o calloc**. La falta de liberación de memoria puede provocar fugas de memoria, lo que puede hacer que el programa utilice más y más memoria a medida que se ejecuta y, eventualmente, puede llevar a un fallo del programa debido a la falta de memoria disponible.

Un puntero a **void**, también conocido como **void***, es un tipo de puntero genérico en el lenguaje de programación C. Puede apuntar a cualquier tipo de dato, pero no permite la manipulación directa de los datos a los que apunta sin un tipo de conversión (casting) adecuado. Uno de sus usos es en en funciones que trabajan con datos de varios tipos.

```c
// Asigna memoria para 5 enteros sin inicializar
int *ptr;
ptr = (int*) malloc(5 * sizeof(int));
```

```c
// Asigna memoria para 5 enteros e inicializa a 0
int *ptr;
ptr = (int*) calloc(5, sizeof(int));
```

Ejemplo

Por ejemplo, si queremos sumar todos los elementos de un arreglo:

```c
#include <stdio.h>
#include <stdlib.h>
int sumarElementos(int* arreglo, int longitud) { //Función para calcular la suma de todos los elementos en un arreglo de enteros
    int suma = 0;
    for (int i = 0; i < longitud; ++i) {
        suma += *(arreglo + i);
    }
    return suma;
}
int main() {
    int longitud, *arreglo, suma;
    printf("Ingrese la longitud del arreglo: ");
    scanf("%d", &longitud);

    arreglo = (int*)malloc(longitud * sizeof(int));
    if (arreglo == NULL) {
        printf("Error: No se pudo asignar memoria.\n");
        return 1;
    }
    printf("Ingrese los elementos del arreglo:\n");
    for (int i = 0; i < longitud; ++i) {
        printf("Elemento %d: ", i + 1);
        scanf("%d", arreglo + i);
    }
    suma = sumarElementos(arreglo, longitud);
    printf("La suma de los elementos en el arreglo es: %d\n", suma);

    free(arreglo);    // Liberar la memoria asignada para el arreglo
    return 0;
}
```

Función main con parámetros

La función main en C puede tener parámetros para recibir argumentos pasados al programa desde la línea de comandos.

La forma estándar de main es **int main(int argc, char *argv[])**

- El primer parámetro, **argc**, es un entero que representa el número de argumentos pasados al programa.

- El segundo parámetro, **argv**, es un arreglo de punteros a caracteres que contiene los argumentos como cadenas de caracteres.

Por ejemplo, si ejecutas tu programa desde la línea de comandos como **./programa arg1 arg2**, **argc** sería 3 (incluyendo el nombre del programa) y **argv** contendría el punteros a las cadenas "programa", "arg1" y "arg2".

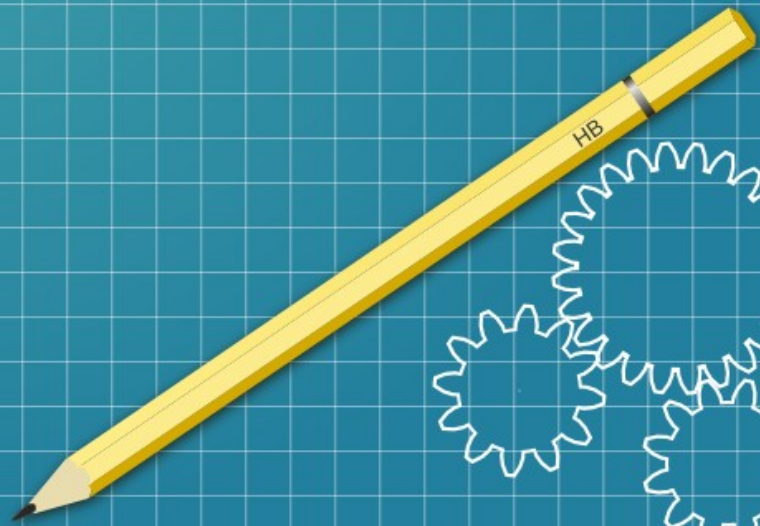
Ejemplo

Por ejemplo, imprimir todos los argumentos pasados al main:

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
    // argc indica la cantidad de argumentos recibidos
    // argv es un arreglo de punteros a cadenas que contiene los argumentos

    printf("Número total de argumentos: %d\n", argc);
    // Imprimir cada argumento
    printf("Argumentos:\n");
    for (int i = 0; i < argc; i++) {
        printf("argv[%d]: %s\n", i, argv[i]);
    }
    return 0;
}
```





HB