Laboratorio 8

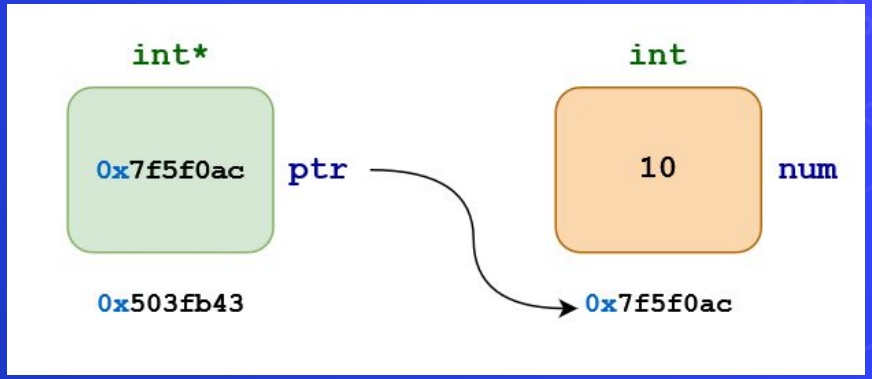
Programación de Computadores (2023-2)

Temas del Laboratorio 8

- Punteros en C
- Uso de punteros en funciones
- Punteros dobles
- Ejercicios

Punteros en lenguaje C

Un puntero es una variable que almacena la dirección de memoria de otra variable. En lugar de almacenar directamente un valor, un puntero almacena la dirección de memoria donde se encuentra el valor



Representación de un puntero en C

Punteros en lenguaje C

Toda variable que declaremos se encuentra en una posición de memoria. Los punteros permiten trabajar sobre esas posiciones de memoria.

Los punteros también son variables y su tamaño es de 8 bytes (en arquitecturas de 64 bits)

```c
#include <stdio.h>

int main(){

    int n;
    printf("La posicion de memoria de la variable n es: %p\n", &n);

    return 0;
}
```

```
leonardo@leonardo-acer:~/ProgramacionDeComputadores/Lab8$ ./pos_memoria
La posicion de memoria de la variable n es: 0x7ffc699ef754
leonardo@leonardo-acer:~/ProgramacionDeComputadores/Lab8$ ./pos_memoria
La posicion de memoria de la variable n es: 0x7fff11659824
leonardo@leonardo-acer:~/ProgramacionDeComputadores/Lab8$ ./pos_memoria
La posicion de memoria de la variable n es: 0x7fffef35c704
leonardo@leonardo-acer:~/ProgramacionDeComputadores/Lab8$ ./pos_memoria
La posicion de memoria de la variable n es: 0x7fffe3e5ec34
leonardo@leonardo-acer:~/ProgramacionDeComputadores/Lab8$
```

Programa de ejemplo para mostrar la posición de memoria de una variable

Operadores de los punteros

Declaración de Puntero

```
// Declaración de un puntero a un entero
int *p;
// Declaración de un puntero a un char
char *c;
// Declaración de un puntero a un float
float *f;
```

Puntero nulo

```
int *ptr = NULL; // Puntero nulo
```

Operador de Dirección (&)

```
int x = 10;
int *p;
// 'p' ahora contiene la dirección de memoria de 'x':
p = &x;
```

Operador de Indirección (*)

```
int y = *p;
// 'y' ahora contiene el valor almacenado en la dirección
de memoria apuntada por 'p'
```

Aritmética de punteros

```
int arr[5] = {1, 2, 3, 4, 5};
int *ptr = arr;

// Acceder a elementos del array usando aritmética de punteros
int segundoElemento = *(ptr + 1);  // Equivalente a arr[1]
```

Uso de punteros

```c
#include <stdio.h>

int main() {

    // Variable 'x' de tipo entero
    int x;

    // Puntero guarda la dirección de memoria de 'x'
    int *p = &x;

    // Se asigna el valor de 10 hacia donde apunta 'p'
    *p = 10;

    printf("El valor de x es: %d\n", *p);
    printf("Su dirección de memoria es: %p\n", p);
    printf("La dirección de memoria del puntero p es: %p\n", &p);

    return 0;
}
```

Punteros y arreglos

Un arreglo en C puede considerarse como una colección de elementos del mismo tipo almacenados en ubicaciones contiguas de memoria.

Pueden emplearse punteros para almacenar la dirección de memoria del primer elemento del arreglo y utilizar la aritmética de punteros para acceder y/o modificar los elementos del arreglo.

```c
int n = 10;

int arr[n];

for(int i = 0; i < n; i++){
    arr[i] = i+1;
}

int *ptrArr = arr;

// Modifica el valor de arr[0]
*ptrArr = 100;

// Modifica el valor de arr[3]
*(ptrArr+3) = 400;

for(int i = 0; i < n; i++){
    printf("%d ", *(ptrArr+i));
}
```

Uso de punteros en funciones

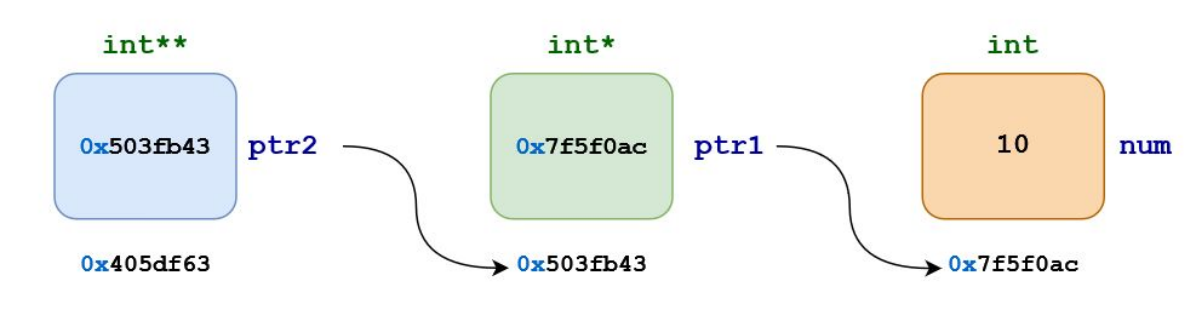
```c
#include <stdio.h>
#include <stdlib.h>

void sumaYResta(int a, int b, int *suma, int *resta) {
    *suma = a + b;
    *resta = a - b;
}

int main(int argc, char *argv[]) {

    if(argc != 3) {
        printf("Uso: %s <primero numero> <segundo numero>\n", argv[0]);
        return 1;
    }

    int x, y, resultadoSuma, resultadoResta;

    x = atoi(argv[1]);
    y = atoi(argv[2]);

    sumaYResta(x, y, &resultadoSuma, &resultadoResta);

    printf("Suma: %d\n", resultadoSuma);
    printf("Resta: %d\n", resultadoResta);

    return 0;
}
```

Punteros dobles (puntero a puntero)

Los punteros dobles son variables que almacenan la dirección de memoria de otro puntero.

Se declara con dos asteriscos (**) y se asignan a la dirección de memoria de un puntero.

Se pueden utilizar para trabajar con matrices.



Representación de puntero doble en C

Punteros dobles (puntero a puntero)

```c
#include <stdio.h>

int main()
{
    int x = 789;

    // Puntero a entero
    int *ptr1;

    // Puntero a puntero a entero
    int **ptr2;

    // Asigna la dirección de 'x' a 'ptr1'
    ptr1 = &x;

    // Asigna la dirección de 'ptr1' a 'ptr2'
    ptr2 = &ptr1;

    printf("Valor de x: %d; direccion de x: %p\n", x, &x);
    printf("Valor de x usando puntero: %d; direccion de x usando puntero: %p\n", *ptr1, ptr1);
    printf("Valor de x usando puntero doble: %d; direccion de x usando puntero doble: %p\n", **ptr2, *ptr2);
    printf("Direccion de puntero ptr1: %p; direccion de puntero doble ptr2; %p \n", ptr2, &ptr2);

    return 0;
}
```

Punteros de tipo void

Son punteros que no tienen un tipo de variable específica, todos los punteros pueden ser transformados a este tipo y viceversa. (Cuidado, puede generar comportamientos inesperados)

```c
int main(){

    int num = 100000;
    void *ptr = &num;
    short *shNum_ptr = ptr;
    short shNum = *shNum_ptr;

    printf("%d  %hd\n", num, shNum);

    return 0;
}
```

Output:

```
$ ./a.out
100000  -31072
```

Ejercicio 1

Crear una función Swap

```c
int main(){
    int a = 2, b = 348;

    printf("antes del swap: a=%d b=%d\n",a,b);
    mi_funcion_swap(&a,&b);
    printf("despues del swap: a=%d b=%d\n",a,b);

    return 0;
}
```

Output:

```
antes del swap: a=2 b=348
despues del swap: a=348 b=2
```

Ejercicio 2

Imprimir el arreglo char[8] “bananas” como si fuera un solo entero tipo long long.

(Usar punteros de tipo void o casting)

```
3247674903064611 4

7089068653200634624
```

El primero o el segundo número dependiendo de la representación de números multibyte que use el computador.

(big o little endian)

Ejercicio 3

Hacer un programa que reciba un string con n palabras separadas por espacios e imprima n líneas del string empezando en la palabra 0,1,2...n.

Ejemplo:

Input:

string de prueba asdf

Output:

string de prueba asdf

de prueba asdf

prueba asdf

asdf