PROGRAMACIÓN AVANZADA

Apuntadores

APUNTADORES (O PUNTEROS)

Los **apuntadores** o punteros son variables que contienen la dirección de otras variables.

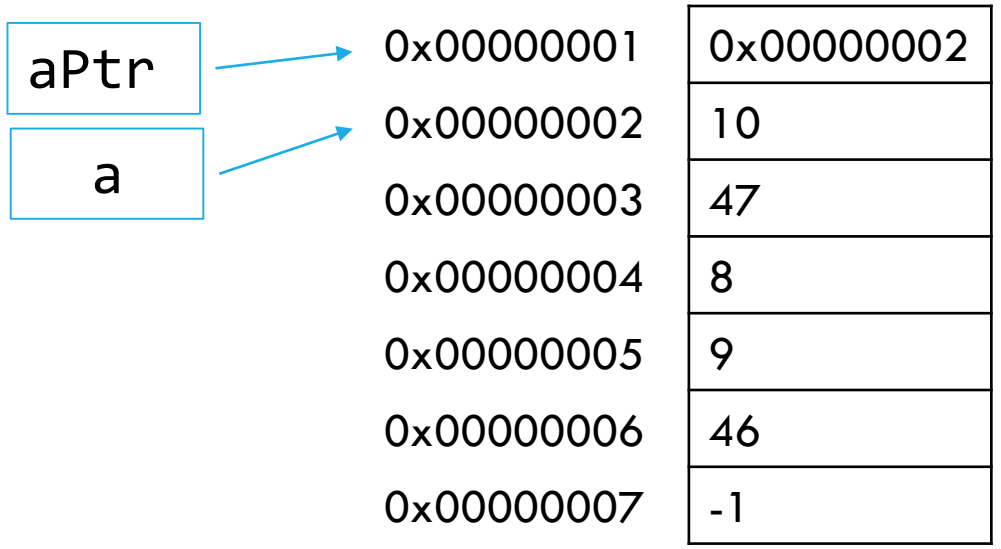
En C son particularmente importantes para que nuestros programas ocupen menos **espacio** y se ejecuten más **rápido**.

PUNTEROS

Como se definió anteriormente, los **valores** de los punteros son **direcciones de memoria**.

¿Qué efecto tiene este código en la RAM?

```
int *aPtr;
int a = 10;
aPtr = &a;
```



CREAR PUNTEROS A TIPOS DE DATOS NUMÉRICOS

```c
#include "stdio.h"

int main(void) {
        int *aPtr;
        int a = 10;
        aPtr = &a;
        printf("%p\n", &a);
        printf("%p\n", aPtr);
        printf("%p\n", &aPtr);
        return 0;
}
```

Desde un puntero, podemos obtener su dirección de memoria (**&aPtr**), la dirección de memoria a donde apunta (**aPtr**) y el valor almacenado en la dirección de memoria a donde apunta (***aPtr**).

De la variable **a** podemos obtener su valor (**a**) y su dirección de memoria (**&a**).

Los operadores * y **&** son complementarios.

Implemente este código e incluya una nueva línea que imprima, sin usar la variable **a**, el valor almacenado en la dirección de memoria que apunta **aPtr**.

¿PARÁMETROS POR VALOR O POR REFERENCIA?

```c
#include "stdio.h"
int cubeByValue(int n) {
    return n*n*n; }
void cubeByReference(int *nPtr) {
    *nPtr = (*nPtr)*(*nPtr)*(*nPtr); }
int main(void) {
    int n = 3;
    printf("n al cubo es: %d\n", cubeByValue(n));
    printf("El valor de n es: %d\n\n", n);
    cubeByReference(&n);
    printf("El valor de n es: %d\n", n); }
```

¿Por qué **&n**? ¿Por qué no **n**?
Implemente y ejecute ambas variantes.

USAR EL CALIFICADOR CONST CON PUNTEROS

Puntero no constante a datos constantes

- `void printCharacters(const char *stringPtr)`
- `stringPtr` no puede modificar el carácter al que apunta.

Puntero constante a datos no constantes

- `int * const agePtr = &x;`
- El valor al que apunta **agePtr** puede modificarse; pero **agePtr** no puede apuntar a una dirección de memoria diferente.

¿Cómo podemos declarar un puntero constante a datos constantes?

PUNTEROS A ARREGLOS

```
#include "stdio.h"
#include "ctype.h"
void stringToUpperCase(char *sPtr) {
    while(*sPtr !=0) {
        *sPtr = toupper(*sPtr);
        ++sPtr; } }
int main(void) {
    char someString[] = "We all like programming in C.";
    printf("This is the original string: %s\n", someString);
    stringToUpperCase(someString);
    printf("This is the modified string: %s\n", someString); }
```

¿Qué hace este incremento? Implemente e intente diferentes incrementos.
LA ARITMÉTICA DE LOS PUNTEROS DEPENDE DEL TAMAÑO DEL TIPO DE DATOS SEÑALADO:
sPtr + = 2; implica un incremento de 2 * sizeof(char)

ASIGNACIÓN DE MEMORIA DINÁMICA

```c
#include "stdio.h"
#include "stdlib.h"
typedef struct{
    void* data;
    struct ListNode* next;
} ListNode;
int main(void) {
    ListNode* nodePtr = malloc(sizeof(ListNode));
    printf ("%p\n", nodePtr->next);
    printf ("%p\n", (*nodePtr).next);
    free(nodePtr); }
```

Puntero a cualquier tipo de dato.

Puntero a otro nodo de tipo “struct ListNode”.

Asignación dinámica de memoria para un solo nodo.

Liberar la memoria apuntada por nodePtr.

PUNTEROS A FUNCIONES

Un puntero a una función contiene la dirección de la función en la memoria.

Se puede modificar una función para ordenar matrices de valores int, para ello debe recibir el siguiente parámetro adicional:

- `int (*compare)(int a, int b)`

El prototipo de la función anterior también se puede escribir de la siguiente manera:

- `int (*)(int, int)`

IMPORTANTE: Observe que el siguiente código no es un puntero a una función sino una función que devuelve un puntero:

- `int *compare(int a, int b)`

ACTIVIDAD

Implemente las funcionalidades básicas de la estructura de dato **Queue** usando arreglos dinámicos y asignación dinámica de memoria.

MUCHAS GRACIAS