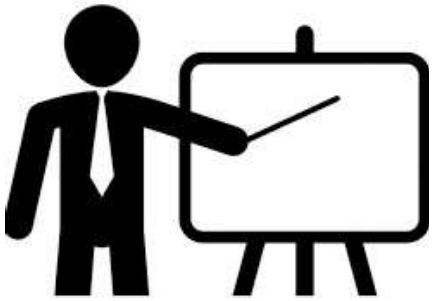


1.4 Apuntadores

1.4.1 Paso por valor / Paso por referencia

```
void principal() {
    int a,b;
    a = 5;
    b = 10;
    funcion(a,b);
    printf("a = %d y b = %d",a,b);
}
void funcion(int a, int b){
    a = 100;
    b = 200;
}
```

1.4.1 Paso por valor / Paso por referencia

- En paso por valor, los valores de los parámetros que se pasan a una función se copian en “nuevas variables” y las modificaciones realizadas en esos valores solo permanecen “vigentes” mientras se esté ejecutando la función.

1.4.1 Paso por valor / Paso por referencia

- Por lo tanto, para que se pueda modificar un valor original es necesario
  - ✓ Devolver el valor en la función
  - ✓ Utilizar variables globales

1.4.1 Paso por valor / Paso por referencia

- En paso por referencia, los parámetros que se pasan a una función son *referencias* a las variables y las modificaciones realizadas en esos valores permanecen “vigentes” una vez finalizada la ejecución de una función.

1.4.1 Paso por valor / Paso por referencia

Ejemplo

```
var i, j : integer;
procedure P(k, m : integer);
begin
    k := k - m;
    m := k + m;
    k := m - k;
end;
i := 2;
j := 3;
P(i, j);
```

1.4.1 Paso por valor / Paso por referencia

Ejemplo

```
program main ()
begin
    integer a, b, c, i
    a = 6
    b = 7
    c = 8
    i = fun(a, b, c)
    print i, a, b, c
end

integer fun (integer x, integer y, integer z)
begin
    if (x > 6) then
        y = 25
    z = x + y
    return y + z
end
```

1.4.2 Definición de apuntador

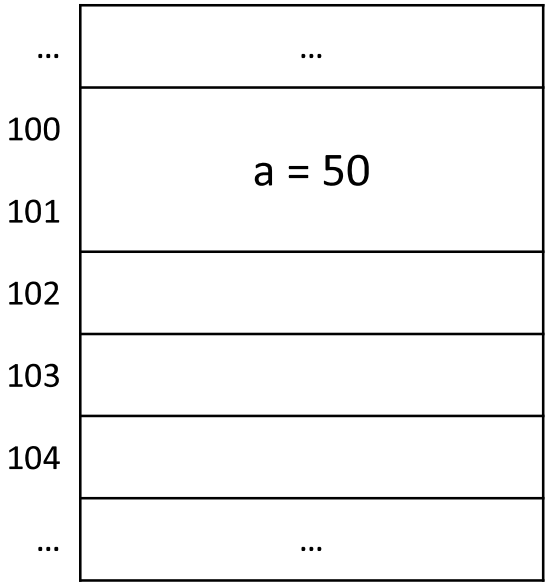
- Un apuntador es un tipo especial de variable cuyo valor es una dirección de memoria.
- Esa dirección de memoria se refiere a otra variable, que a su vez, contiene un valor específico.

1.4.2 Definición de apuntador

- Es una variable especial que hace referencia indirecta a un valor.
- Al proceso de hacer referencia a un valor a través de un apuntador se le conoce comúnmente como indirección.

1.4.3 El operador dirección (&)

- Cuando se antepone este operador al identificador de una variable , se obtiene la dirección de memoria donde se almacena esa variable.



int a = 50;

&a = ?

1.4.4 El operador indirección (*)

- Este operador se utiliza para especificar que una variable será un apuntador.
- De esta forma es posible acceder al espacio de memoria que este direcciona.

```
int *var1;
```

En la declaración de “var1” se usa el operador indirección para especificar que se trata de un apuntador

1.4.5 Uso práctico de apuntadores

- ¿Cómo se usa un apuntador?

1. Declaración del apuntador
2. Asociación del apuntador con otra variable
3. Utilizar el apuntador

1.4.5 Uso práctico de apuntadores

Declaración

```
[tipo dato] *[identificador];
```

```
int *a;
long *apuntador;
char *perro;
float *otroApuntador;
```

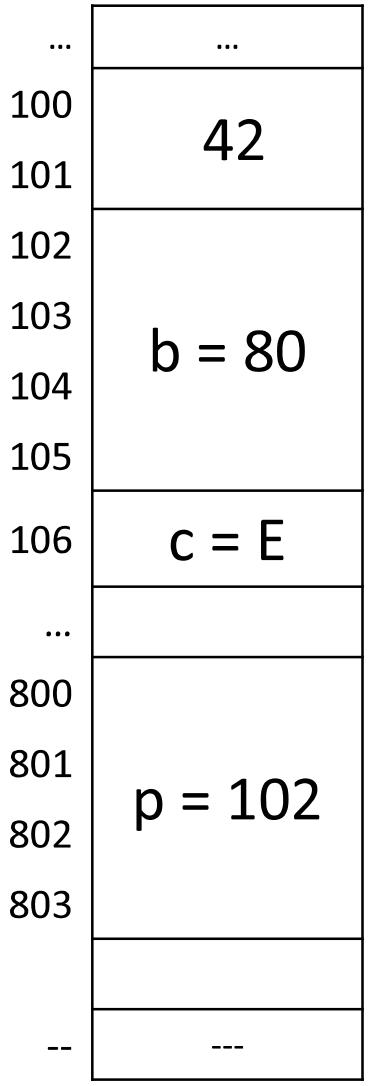
1.4.5 Uso práctico de apuntadores

Asociación

```
int variable1;
char variable2;
float variable3;
int *apuntador1;
apuntador1 = &variable1;
apuntador1 = &variable2;
```

Ubicación en memoria…

```
int a = 42;
long b = 80;
char c = "E";
long *p;
p = &b;
```



1.4.5 Uso práctico de apuntadores

```c
#include <stdio.h>

int main() {
    int a = 5, b = 20, c = 10, d;
    int *p1, *p2;
    p1 = &c;
    p2 = &b;
    d = *p1 + *p2;
    printf("Suma %d\n", d);
    return 0;
}
```

1.4.5 Uso práctico de apuntadores

```c
#include <stdio.h>

void main() {
    int a = 5, b = 20, c = 10, d;
    int *p1, *p2;
    p1 = &c;
    p2 = &b;
    *p1 = 100;
    *p2 = 50;
    printf("Suma %d\n", b+c);
}
```

1.4.5 Uso práctico de apuntadores

```c
#include <stdio.h>

void main() {
    int x,*px;
    px = &x;
    *px = 0;
    printf("%d \n",x);
    *px = *px + 5;
    printf("%d \n",x);
    *px = *px * 2;
    printf("%d \n",x);
    *px = *px + 4;
    printf("%d \n",x);
    *px *= x * 2;
    printf("%d \n",x);
}
```

Lo esencial en apuntadores…

Los 3 elementos necesarios para uso de apuntadores:

- Declaración de la variable (tipo apuntador)
- Asociación con una dirección de memoria (de otra variable)
- Uso del apuntador

Paso por referencia (Apuntadores)

```c
#include<stdio.h>

void mifuncion(int*,int*);

main(void) {
    int i = 2;
    int j = 3;
    mifuncion(&i,&j);
    printf("Después de ... i es %d y j es %d \n ",i,j);
}

void mifuncion(int *k,int *m){
    *k = *k - *m;
    *m = *m + *k;
    *k = *m - *k;
    printf("Después de ... i es %d y j es %d \n",*k,*m);
}
```

Paso por referencia (Apuntadores)

```c
#include<stdio.h>

void permutar(int*, int*);

int main() {
    int a = 5, b = 10;
    permutar(&a, &b);
    printf("Nuevo valor de a = %d y de b = %d",a,b);
}

void permutar(int *x, int *y) {
    int aux;
    aux = *x;
    *x = *y;
    *y = aux;
}
```

1.4.6 Arreglos y Apuntadores

- Un arreglo representa un conjunto de celdas de memoria consecutivas.
- Para poder relacionar apuntadores con arreglos se debe hacer con la dirección del primer elemento del arreglo.

```
int a[10] = {1,2,3,4,5,6,7,8,9,10};
int *p;
p = &a[0];
*p = 10 // a[0] = 10;
```

1.4.6 Arreglos y Apuntadores

- Para modificar los elementos en un arreglo

```
*p = 10 // a[0] = 10;
*(p+3)= 5 // a[3] = 5;
*(p+5)= 8 // a[5] = 8;

*p+3  ??
```

1.4.6 Arreglos y Apuntadores

- El identificador (nombre de la variable) de un arreglo sin subíndice también es la dirección de memoria del primer elemento.

```
int a[5] = {1,2,3};
int *p;
```

`p = a` es equivalente a **`p = &a[0]`**

Ejemplo

```c
#include <stdio.h>

int arreglo[] = {3,25,19,6,5,300};
int *ap;

int main(void) {
    int i;
    ap = arreglo;
    printf("\n\n");
    for (i = 0; i < 6; i++) {
        printf("arreglo[%d] = %d ",i,arreglo[i]);
        printf("ap [%d] = %d \n",i, *(ap + i));
    }
    return 0;
}
```