Punteros

Algoritmos y Estructuras de Datos II

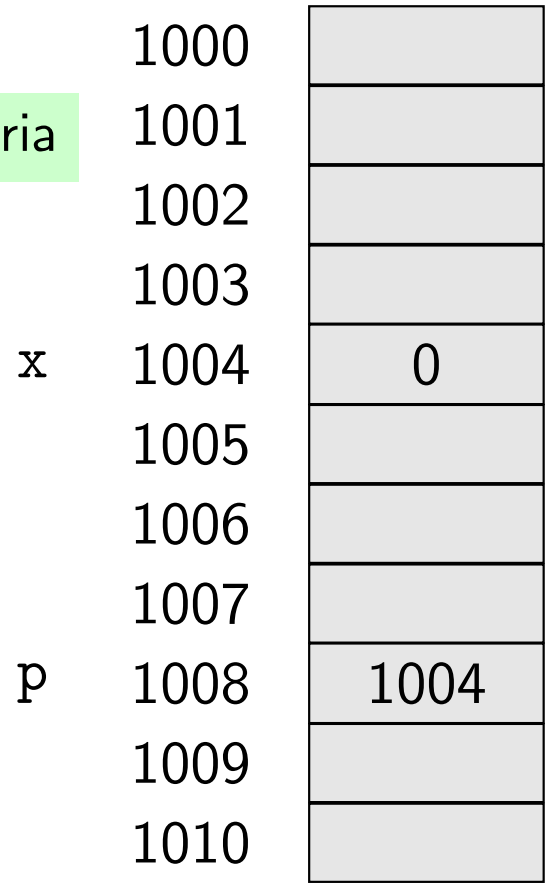
Contenidos

- ✓ Punteros.
- ✓ Parámetros por valor vs. Parámetros por referencia.
- ✓ Memoria Dinámica.

Punteros

Son variables que almacenan direcciones de memoria

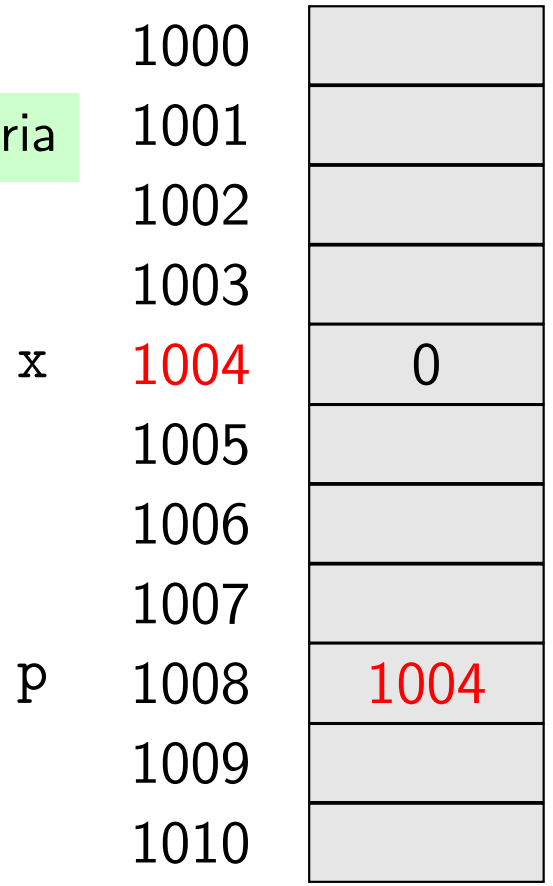
```
int x = 0;
int *p = &x;
```



Punteros

Son variables que almacenan direcciones de memoria

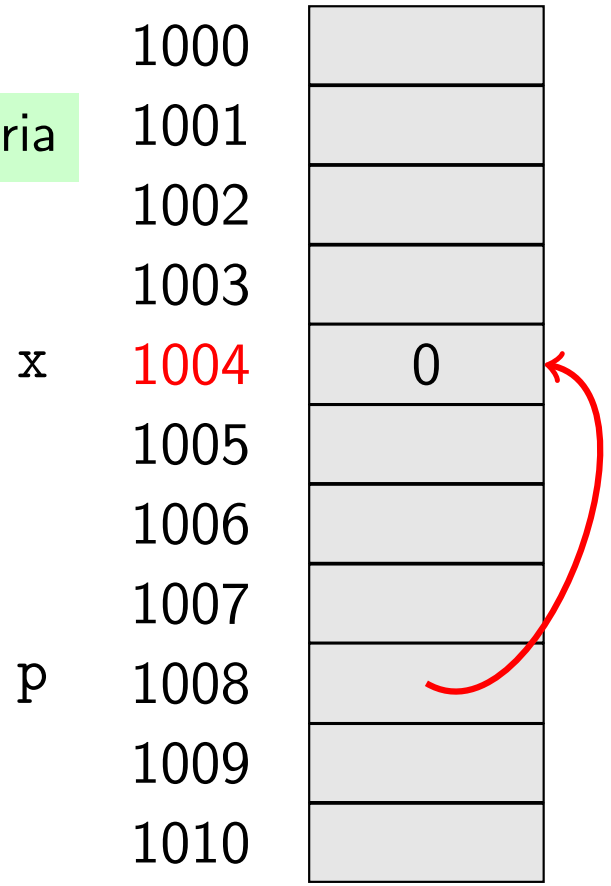
```
int x = 0;
int *p = &x;
```



Punteros

Son variables que almacenan direcciones de memoria

```
int x = 0;
int *p = &x;
```

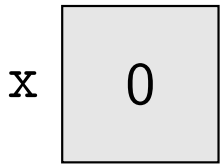


Punteros

```
int main(void) {
    int x = 0 ;
    int *p = NULL;
    int *q = NULL;

    p = &x;
    (*p) = 4;
    printf("%d\n", x);
    q = p;
    printf("%d\n", *q);

    return(0);
}
```

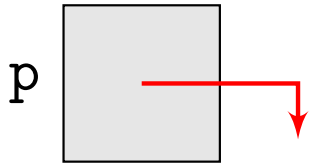
Punteros

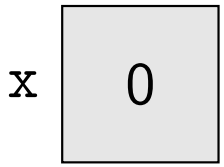
```
int main(void) {
    int x = 0 ;
    int *p = NULL;
    int *q = NULL;

    p = &x;
    (*p) = 4;
    printf("%d\n", x);
    q = p;
    printf("%d\n", *q);

    return(0);
}
```



Punteros

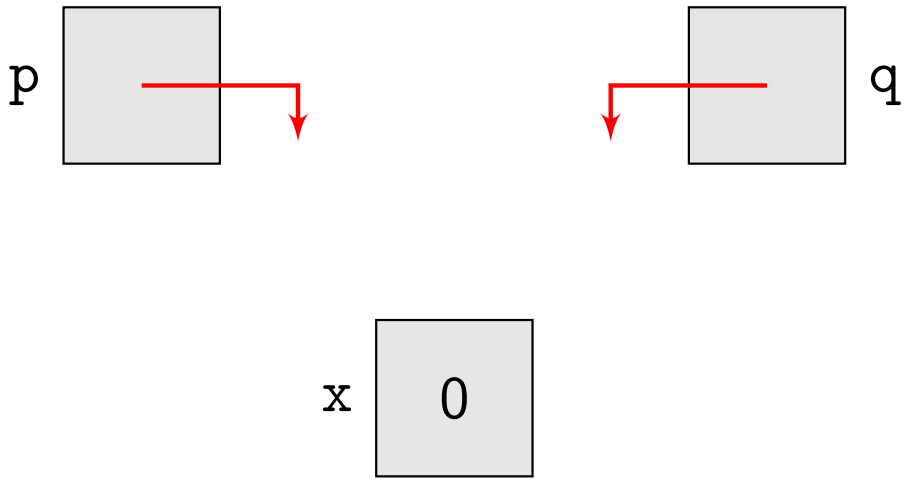
```
int main(void) {
    int x = 0 ;
    int *p = NULL;
    int *q = NULL;

    p = &x;
    (*p) = 4;
    printf("%d\n", x);
    q = p;
    printf("%d\n", *q);

    return(0);
}
```





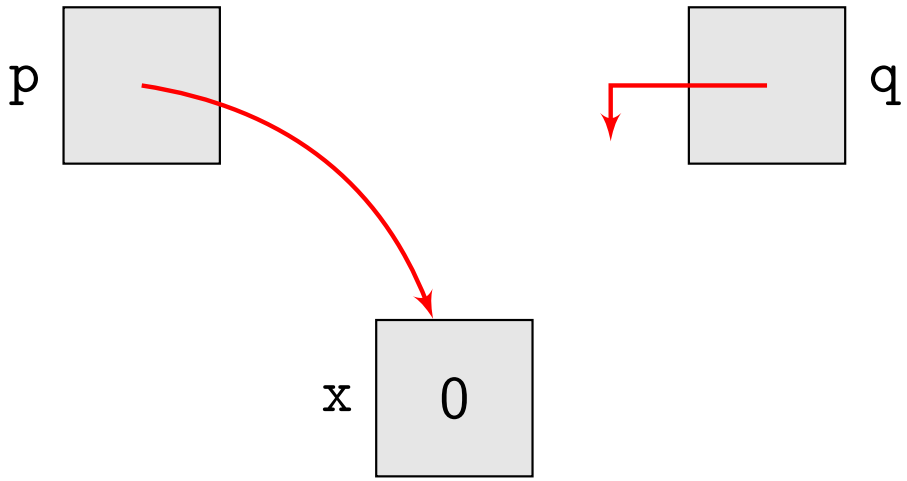
Punteros

```
int main(void) {
    int x = 0 ;
    int *p = NULL;
    int *q = NULL;

    p = &x;
    (*p) = 4;
    printf("%d\n", x);
    q = p;
    printf("%d\n", *q);

    return(0);
}
```



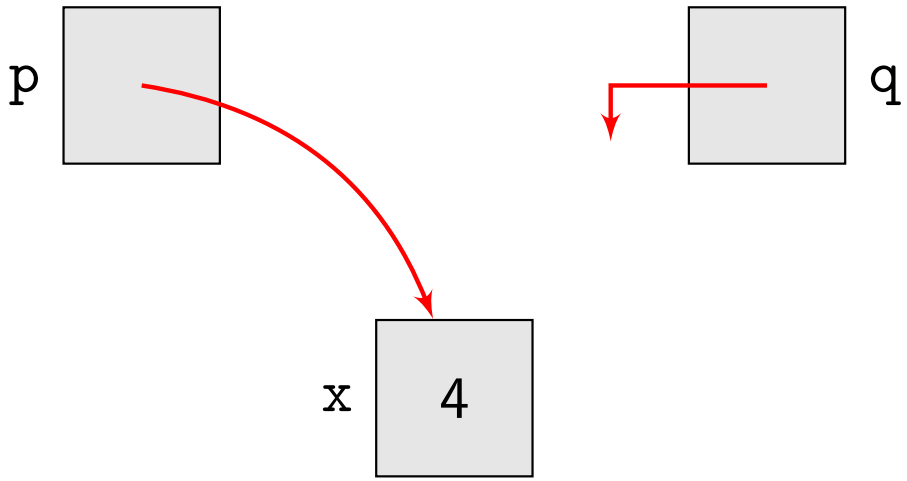
Punteros

```
int main(void) {
    int x = 0 ;
    int *p = NULL;
    int *q = NULL;

    p = &x;
    (*p) = 4;
    printf("%d\n", x);
    q = p;
    printf("%d\n", *q);

    return(0);
}
```



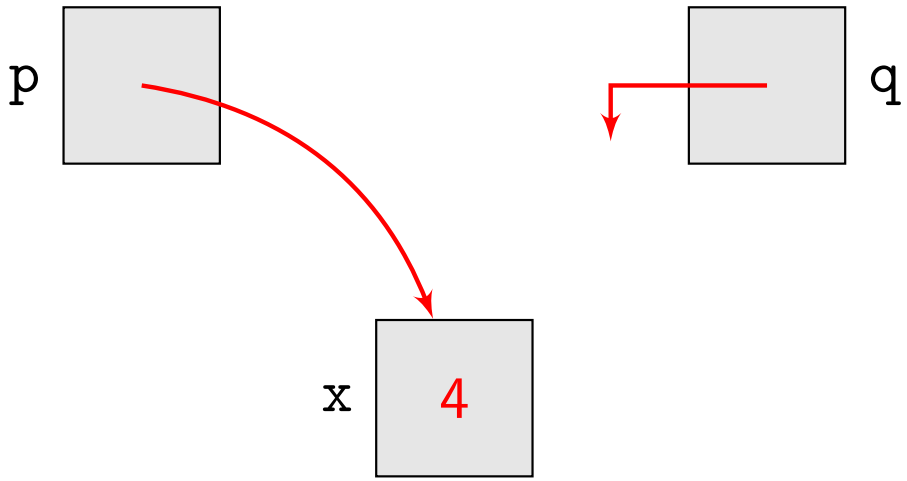
Punteros

```
int main(void) {
    int x = 0 ;
    int *p = NULL;
    int *q = NULL;

    p = &x;
    (*p) = 4;
    printf("%d\n", x);
    q = p;
    printf("%d\n", *q);

    return(0);
}
```



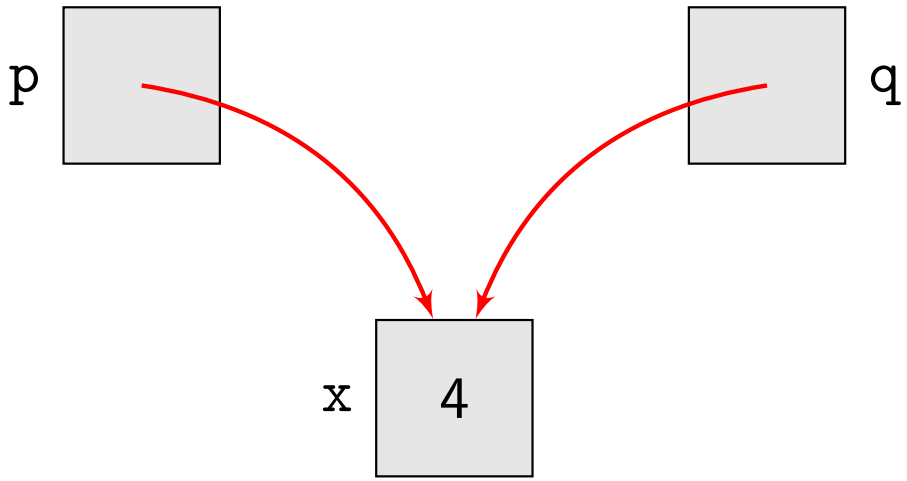
Punteros

```
int main(void) {
    int x = 0 ;
    int *p = NULL;
    int *q = NULL;

    p = &x;
    (*p) = 4;
    printf("%d\n", x);
    q = p;
    printf("%d\n", *q);

    return(0);
}
```



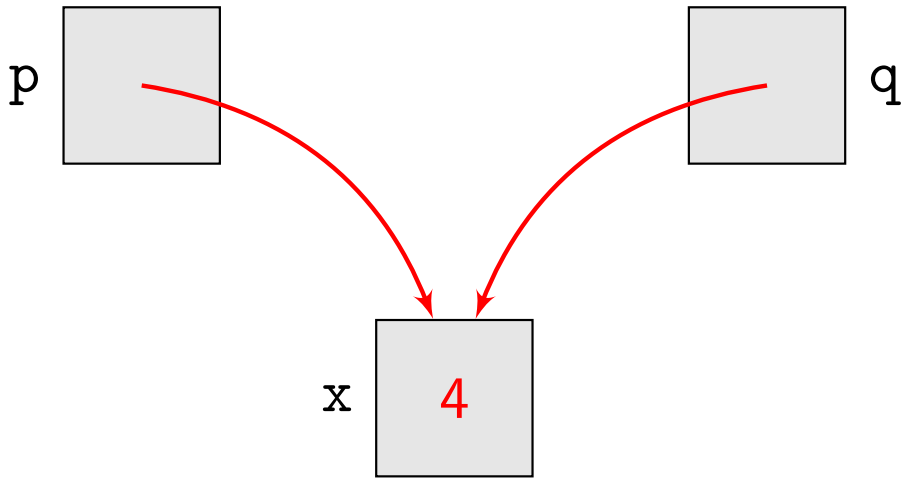
Punteros

```
int main(void) {
    int x = 0 ;
    int *p = NULL;
    int *q = NULL;

    p = &x;
    (*p) = 4;
    printf("%d\n", x);
    q = p;
    printf("%d\n", *q);

    return(0);
}
```



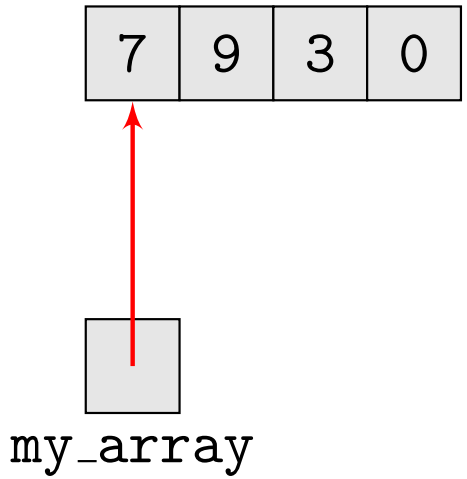
Punteros

```
int main(void) {
    int x = 0 ;
    int *p = NULL;
    int *q = NULL;

    p = &x;
    (*p) = 4;
    printf("%d\n", x);
    q = p;
    printf("%d\n", *q);

    return(0);
}
```



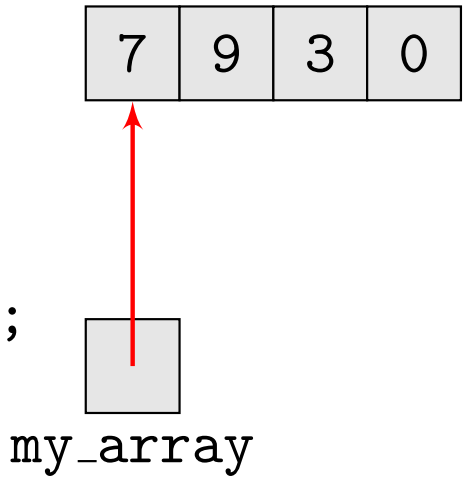
Punteros y Arreglos

```
int main(void) {
    int i = 0;
    int my_array[4] = {7,9,3,0};

    printf("%d\n", my_array[0]);

    for (i = 0; i < 4; i++) {
        printf("%d ", my_array[i]);
    }

    return(0);
}
```



Punteros y Arreglos

```
int main(void) {
    int i = 0;
    int my_array[4] = {7,9,3,0};

    printf("%d\n", *my_array);

    for (i = 0; i < 4; i++) {
        printf("%d ", *(my_array + i));
    }

    /* my_array = NULL error! */

    return(0);
}
```



Notar: `my_array` es un puntero *constante* al primer elemento del arreglo.

Punteros y Parámetros por Referencia

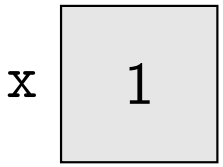
Parámetros por valor

```
int main(void) {
    int x = 1;
    pivote(x);
    printf("%d", x);

    return(0);
}

void pivote(int piv) {
    piv = 5;
}
```

Punteros y Parámetros por Referencia

Parámetros por valor

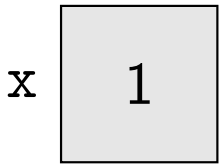
```
int main(void) {
    int x = 1;
    pivote(x);
    printf("%d", x);

    return(0);
}

void pivote(int piv) {
    piv = 5;
}
```

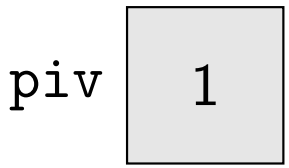


Punteros y Parámetros por Referencia

Parámetros por valor

```
int main(void) {
    int x = 1;
    pivote(x);
    printf("%d", x);

    return(0);
}

void pivote(int piv) {
    piv = 5;
}
```

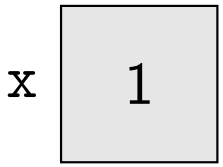


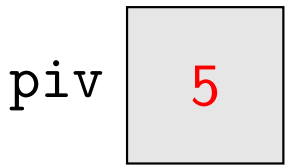


Punteros y Parámetros por Referencia

Parámetros por valor

```
int main(void) {
    int x = 1;
    pivote(x);
    printf("%d", x);

    return(0);
}

void pivote(int piv) {
    piv = 5;
}
```

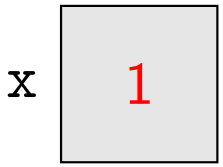




Punteros y Parámetros por Referencia

Parámetros por valor

```
int main(void) {
    int x = 1;
    pivote(x);
    printf("%d", x);

    return(0);
}

void pivote(int piv) {
    piv = 5;
}
```



Punteros y Parámetros por Referencia

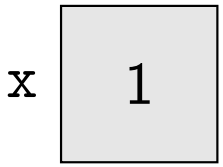
Parámetros por referencia

```
int main(void) {
    int x = 1;
    pivote(&x);
    printf("%d", x);

    return(0);
}

void pivote(int *piv) {
    (*piv) = 5;
}
```

Punteros y Parámetros por Referencia

Parámetros por referencia

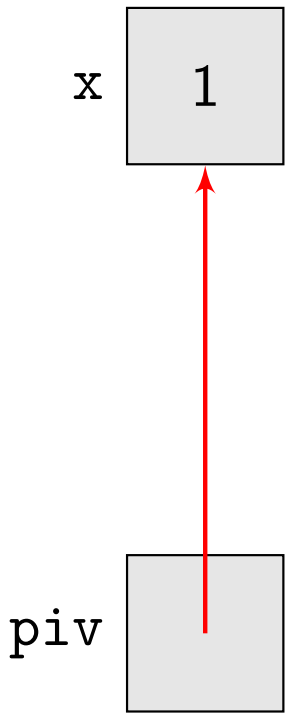
```
int main(void) {
    int x = 1;
    pivote(&x);
    printf("%d", x);

    return(0);
}

void pivote(int *piv) {
    (*piv) = 5;
}
```



Punteros y Parámetros por Referencia

Parámetros por referencia

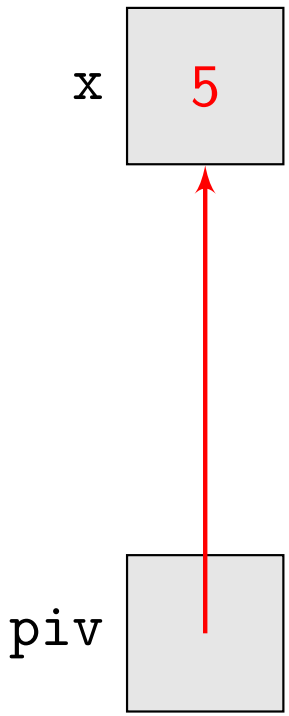
```
int main(void) {
    int x = 1;
    pivote(&x);
    printf("%d", x);

    return(0);
}

void pivote(int *piv) {
    (*piv) = 5;
}
```



Punteros y Parámetros por Referencia

Parámetros por referencia

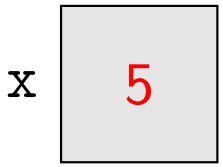
```
int main(void) {
    int x = 1;
    pivote(&x);
    printf("%d", x);

    return(0);
}

void pivote(int *piv) {
    (*piv) = 5;
}
```



Punteros y Parámetros por Referencia

Parámetros por referencia

```
int main(void) {
    int x = 1;
    pivote(&x);
    printf("%d", x);

    return(0);
}

void pivote(int *piv) {
    (*piv) = 5;
}
```



Memoria Dinámica

Hasta ahora hemos usado memoria estática

```
int main(void) {
    char names[1000][100];
    ⋮
}
```

Reservada en tiempo de compilación

Liberada automáticamente al final de la ejecución

Memoria Dinámica

Hasta ahora hemos usado memoria estática

```
int main(void) {
    char names[1000][100];
    ⋮
}
```

¿Y si queremos reservar memoria por demanda?

Memoria Dinámica

Hasta ahora hemos usado memoria estática

```
int main(void) {
    char names[1000][100];
    ⋮
}
```

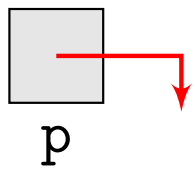
¿Y si queremos liberarla cuando ya no haga falta?

Memoria Dinámica

```
int main(void) {
    int i = 0;
    int *p = NULL;
    p = calloc(5, sizeof(int));
    if (p != NULL) {
        for (i = 0; i < 5 ; i++) {
            printf("%d ", p[i]);
        }
        free(p);
        p = NULL;
    }
    return(0);
}
```

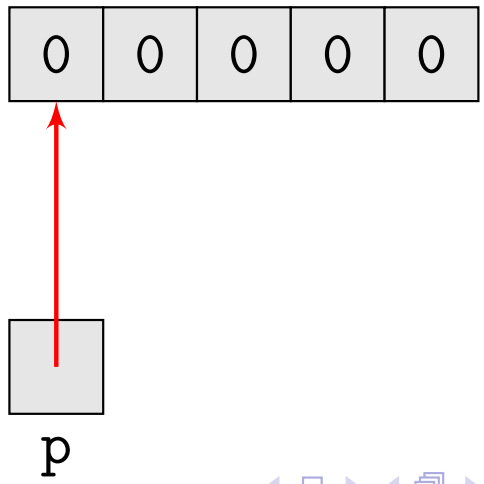
Memoria Dinámica

```
int main(void) {
    int i = 0;
    int *p = NULL;
    p = calloc(5, sizeof(int));
    if (p != NULL) {
        for (i = 0; i < 5 ; i++) {
            printf("%d ", p[i]);
        }
        free(p);
        p = NULL;
    }
    return(0);
}
```



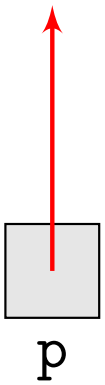
Memoria Dinámica

```
int main(void) {
    int i = 0;
    int *p = NULL;
    p = calloc(5, sizeof(int));
    if (p != NULL) {
        for (i = 0; i < 5 ; i++) {
            printf("%d ", p[i]);
        }
        free(p);
        p = NULL;
    }
    return(0);
}
```



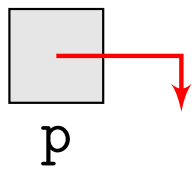
Memoria Dinámica

```
int main(void) {
    int i = 0;
    int *p = NULL;
    p = calloc(5, sizeof(int));
    if (p != NULL) {
        for (i = 0; i < 5 ; i++) {
            printf("%d ", p[i]);
        }
        free(p);
        p = NULL;
    }
    return(0);
}
```



Memoria Dinámica

```
int main(void) {
    int i = 0;
    int *p = NULL;
    p = calloc(5, sizeof(int));
    if (p != NULL) {
        for (i = 0; i < 5 ; i++) {
            printf("%d ", p[i]);
        }
        free(p);
        p = NULL;
    }
    return(0);
}
```



Memoria Dinámica

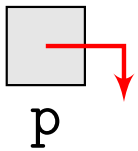
```
struct _info_t {
    char *name;
    int age;
};

typedef struct _info_t *info_t;
```

Memoria Dinámica

```
int main(void) {
    info_t p = NULL;
    p = calloc(1, sizeof(struct _info_t));
    assert(p != NULL);
    p->name = malloc(4 * sizeof(char));
    assert(p->name != NULL);
    p->age = 24;
    p->name = strncpy(p->name, "Leo", 4);

    free(p->name);
    p->name = NULL;
    free(p);
    p = NULL;

    return(0);
}
```

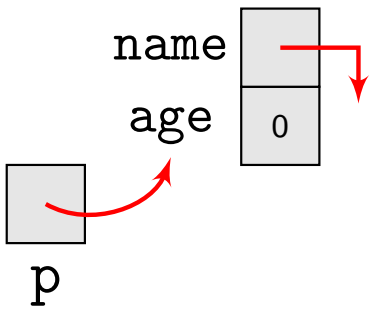
Memoria Dinámica

```
int main(void) {
    info_t p = NULL;
    p = calloc(1, sizeof(struct _info_t));
    assert(p != NULL);
    p->name = malloc(4 * sizeof(char));
    assert(p->name != NULL);
    p->age = 24;
    p->name = strncpy(p->name, "Leo", 4);

    free(p->name);
    p->name = NULL;
    free(p);
    p = NULL;

    return(0);
}
```



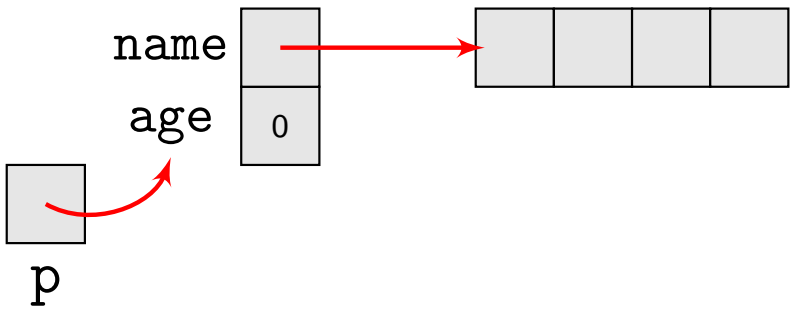
Memoria Dinámica

```
int main(void) {
    info_t p = NULL;
    p = calloc(1, sizeof(struct _info_t));
    assert(p != NULL);
    p->name = malloc(4 * sizeof(char));
    assert(p->name != NULL);
    p->age = 24;
    p->name = strncpy(p->name, "Leo", 4);

    free(p->name);
    p->name = NULL;
    free(p);
    p = NULL;

    return(0);
}
```



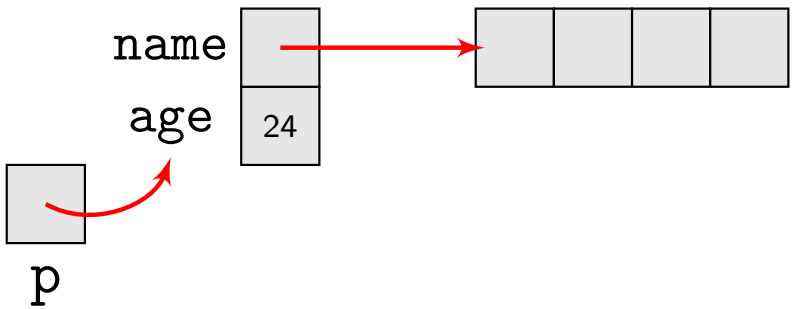
Memoria Dinámica

```
int main(void) {
    info_t p = NULL;
    p = calloc(1, sizeof(struct _info_t));
    assert(p != NULL);
    p->name = malloc(4 * sizeof(char));
    assert(p->name != NULL);
    p->age = 24;
    p->name = strncpy(p->name, "Leo", 4);

    free(p->name);
    p->name = NULL;
    free(p);
    p = NULL;

    return(0);
}
```



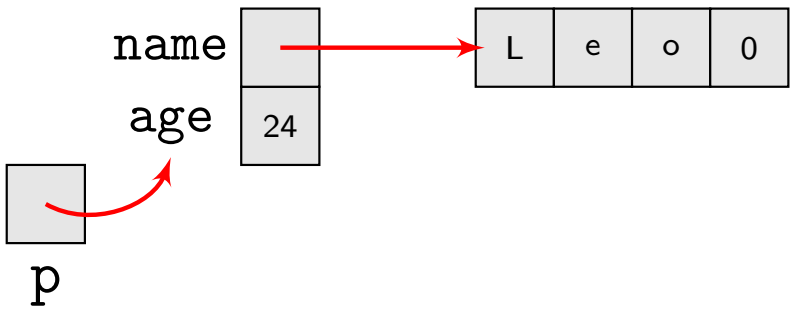
Memoria Dinámica

```
int main(void) {
    info_t p = NULL;
    p = calloc(1, sizeof(struct _info_t));
    assert(p != NULL);
    p->name = malloc(4 * sizeof(char));
    assert(p->name != NULL);
    p->age = 24;
    p->name = strncpy(p->name, "Leo", 4);

    free(p->name);
    p->name = NULL;
    free(p);
    p = NULL;

    return(0);
}
```



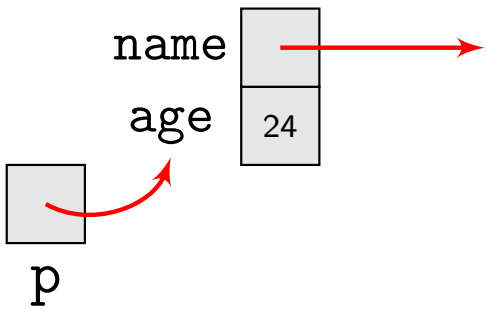
Memoria Dinámica

```
int main(void) {
    info_t p = NULL;
    p = calloc(1, sizeof(struct _info_t));
    assert(p != NULL);
    p->name = malloc(4 * sizeof(char));
    assert(p->name != NULL);
    p->age = 24;
    p->name = strncpy(p->name, "Leo", 4);

    free(p->name);
    p->name = NULL;
    free(p);
    p = NULL;

    return(0);
}
```



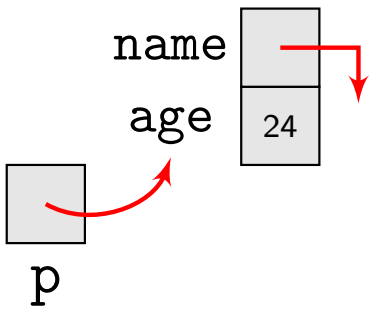
Memoria Dinámica

```
int main(void) {
    info_t p = NULL;
    p = calloc(1, sizeof(struct _info_t));
    assert(p != NULL);
    p->name = malloc(4 * sizeof(char));
    assert(p->name != NULL);
    p->age = 24;
    p->name = strncpy(p->name, "Leo", 4);

    free(p->name);
    p->name = NULL;
    free(p);
    p = NULL;

    return(0);
}
```



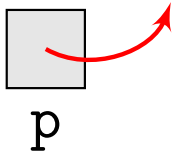
Memoria Dinámica

```
int main(void) {
    info_t p = NULL;
    p = calloc(1, sizeof(struct _info_t));
    assert(p != NULL);
    p->name = malloc(4 * sizeof(char));
    assert(p->name != NULL);
    p->age = 24;
    p->name = strncpy(p->name, "Leo", 4);

    free(p->name);
    p->name = NULL;
    free(p);
    p = NULL;

    return(0);
}
```



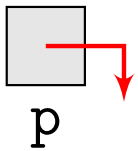
Memoria Dinámica

```
int main(void) {
    info_t p = NULL;
    p = calloc(1, sizeof(struct _info_t));
    assert(p != NULL);
    p->name = malloc(4 * sizeof(char));
    assert(p->name != NULL);
    p->age = 24;
    p->name = strncpy(p->name, "Leo", 4);

    free(p->name);
    p->name = NULL;
    free(p);
    p = NULL;

    return(0);
}
```



Memoria Dinámica

```
int main(void) {
    info_t p = NULL;
    p = calloc(1, sizeof(struct _info_t));
    assert(p != NULL);
    p->name = malloc(4 * sizeof(char));
    assert(p->name != NULL);
    p->age = 24;
    p->name = strncpy(p->name, "Leo", 4);

    free(p->name);
    p->name = NULL;
    free(p);
    p = NULL;

    return(0);
}
```



Verificando el uso de la memoria

```
int main(void) {
    int *p = calloc(1, sizeof(int));

    return(0);
}
```

```
$ valgrind --leak-check=full --show-reachable=yes ./mi_programa
```

```
==17387== LEAK SUMMARY:
==17387==    definitely lost: 4 bytes in 1 blocks
==17387==    indirectly lost: 0 bytes in 0 blocks
==17387==      possibly lost: 0 bytes in 0 blocks
==17387==    still reachable: 0 bytes in 0 blocks
==17387==         suppressed: 0 bytes in 0 blocks
==17387==
==17387== ERROR SUMMARY: 1 errors
```

¿Preguntas?