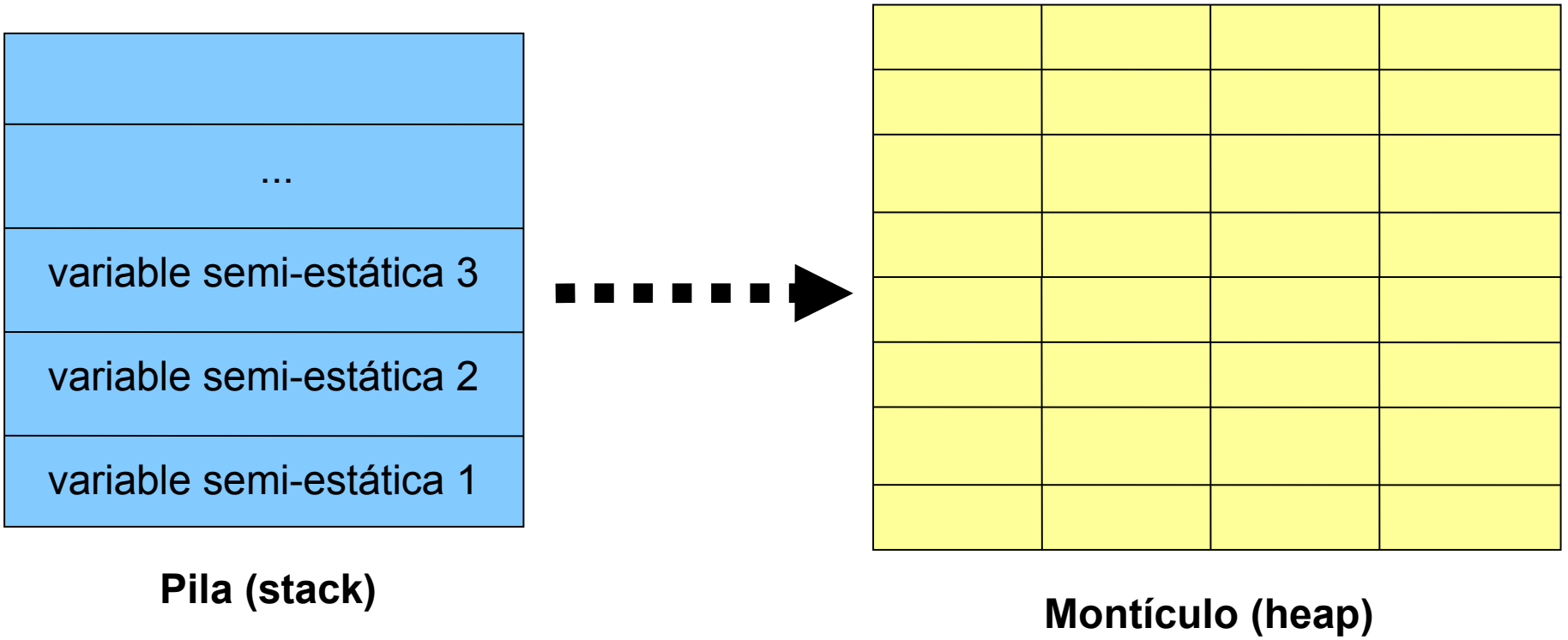
Administración de memoria

- Áreas de memoria
- Punteros
- Aritmética de punteros
- Arreglos y punteros
- Estructuras dinámicas

Áreas de memoria



Punteros (1)

Declaración:

```
<tipo> * <etiqueta_puntero>;
```

Operadores relacionados con punteros:

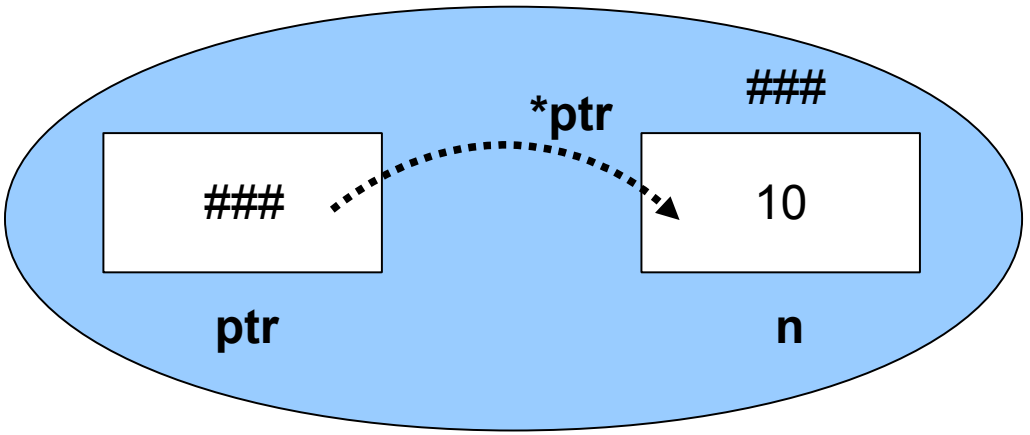
***** : es el operador de indirección. Nos permite acceder al contenido del objeto apuntado (para leerlo o modificarlo).

-> : es el operador de acceso a miembros. Se escribe `puntero->miembro`, lo cual es equivalente a escribir `(*puntero).miembro`.

& : es el operador de referencia. Nos permite obtener la dirección en memoria de un objeto.

Por ejemplo:

```
int n;
int * ptr = NULL;
ptr = &n;
*ptr = 10;
ptr = &10; //error
```



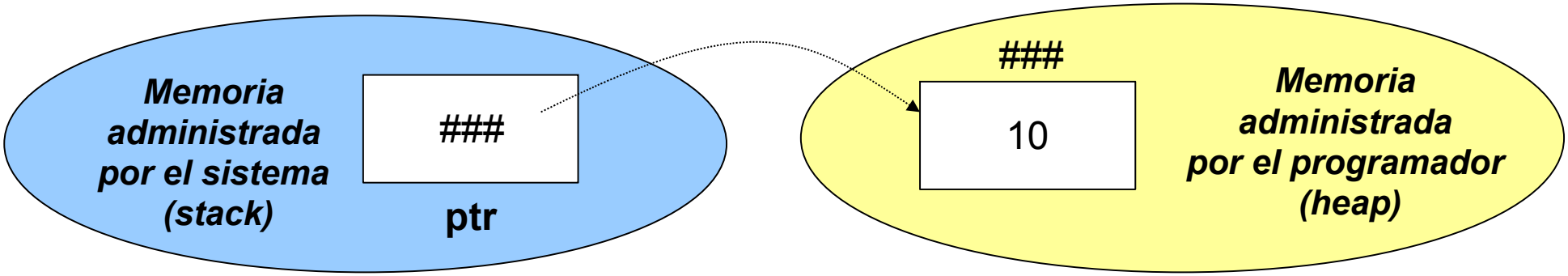
Punteros (2)

Operaciones para obtener y liberar memoria:

```
<tipo_objeto> * <etiqueta_puntero> = new <tipo_objeto>;
```

Por ejemplo:

```
int * ptr = new int;
*ptr = 10;
```




```
delete <etiqueta_puntero>
```

Por ejemplo:

```
delete ptr;
```

Punteros (3)

Recordar que es nuestra tarea liberar la memoria que administramos

Ventajas de `new` respecto a `malloc`:

`new` es una operación de tipado seguro. Es decir, `new` devuelve punteros de un tipo determinado; malloc en cambio, devuelve un puntero a un tipo sin valor (`void*`).

Por ejemplo:

```
int * ptr_new = new int;
int * ptr_malloc = (int*) malloc(sizeof(int));
```

`new` llama al constructor de los objetos. Del mismo modo, `delete` invoca al destructor.

No se deben mezclar `new` y `free`. Lo mismo para `malloc` y `delete`.

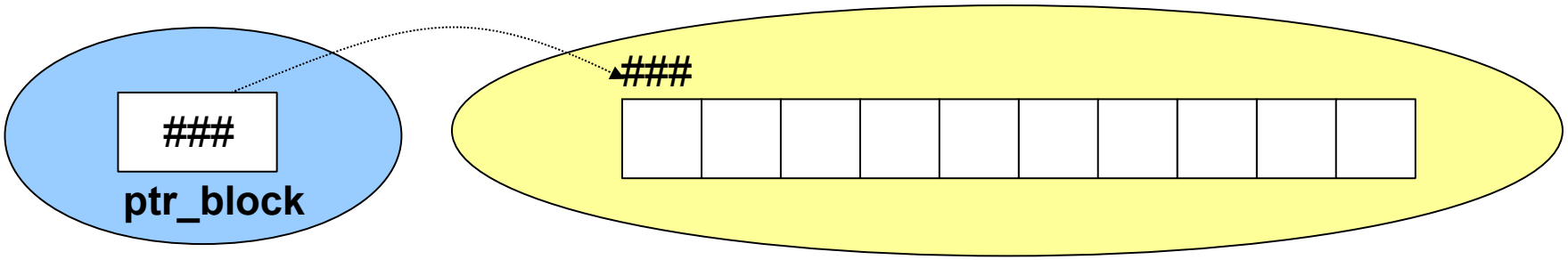
Punteros (4)

Obteniendo bloques de memoria contigua:

```
<tipo_objeto> * <etiqueta_puntero> = new <tipo_objeto>
  [<tamaño_bloque>];
```

Por ejemplo:

```
int * ptr_block = new int[10];
```



Liberando bloques de memoria:

```
delete [] <etiqueta_puntero>;
```

Por ejemplo:

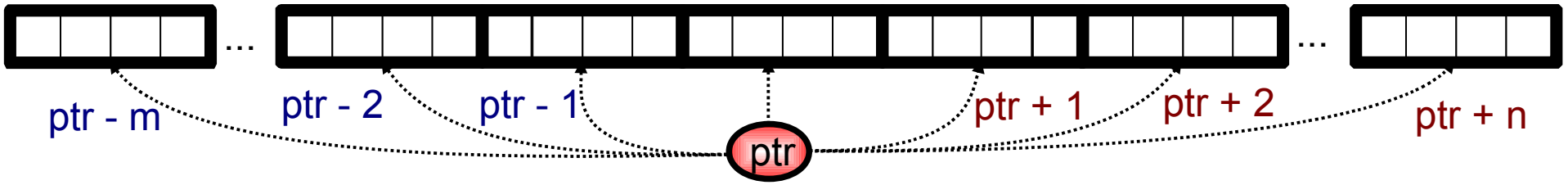
```
delete [] ptr_block;
```

Aritmética de punteros (1)

Es posible obtener un desplazamiento hacia otras posiciones de memoria partiendo de la posición indicada por un puntero.

Una forma es utilizando las operaciones de sumas y restas:

```
<puntero> + <desplazamiento>
<puntero> - <desplazamiento>
```



Otra forma es utilizando la notación de subíndice:

```
<puntero>[<desplazamiento>]
```

Esta operación implica un desplazamiento y una desreferencia:

```
*(<puntero> + <desplazamiento>)
```

Aritmética de punteros (2)

```
// versión 1
int  arr[5] = {1,2,3,4,5};
for  (int i=0; i<5; i++)
    arr[i] = arr[i]*2;
for  (int i=0; i<5; i++)
    cout << arr[i];
```

```
// versión 2
int  arr[5] = {1,2,3,4,5};
    int  * ptr = arr;
 for  (int i=0; i<5; i++)
      arr[i] = ptr[i]*2;
 for  (int i=0; i<5; i++
       cout << ptr[i];
```

```
// versión 3
int  arr[5] = {1,2,3,4,5};
    int  * ptr = arr;
 for  (int i=0; i<5; i++)
      ptr[i] = arr[i]*2;
 for  (int i=0; i<5; i++)
       cout << arr[i];
```

Arreglos y punteros (1)

Lo más importante a tener en cuenta es lo siguiente:

Siempre que un arreglo aparece en una expresión el compilador genera implícitamente un puntero al primer elemento.

Es decir, si el nombre de un arreglo es "`arr`", cuando escribimos arr en una expresión es como si hubiéramos escrito "`&arr[0]`".

Por ejemplo:

```
int  arr[10];
cout << arr << endl;
cout << &arr[0];
```

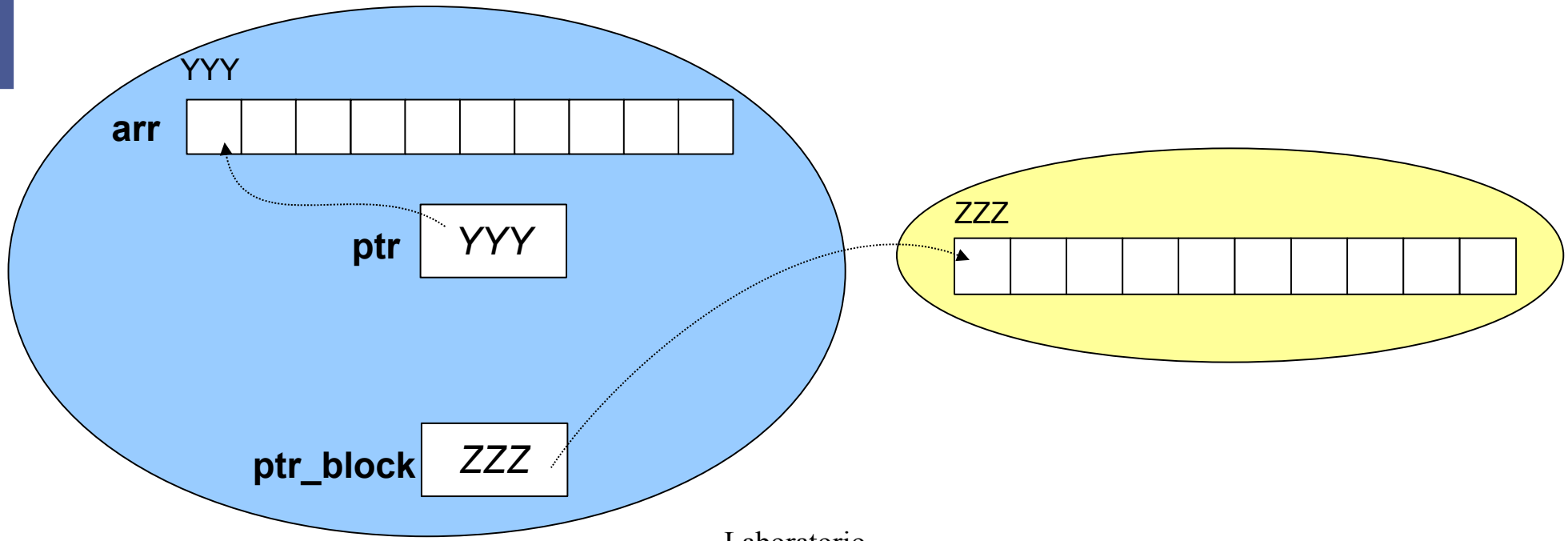
La salida correspondiente, por ejemplo, sería:

```
0xbfb792a8
0xbfb792a8
```

Arreglos y punteros (2)

Los arreglos y punteros son diferentes. Por ejemplo:

```
int arr[10];
int * ptr = arr;              // Puntero al primer elemento
int * ptr_block = new int[10];  // Puntero a un bloque
```



Arreglos y punteros (3)

Debido a que los nombres de los arreglos se convierten en punteros, cuando pasamos un arreglo a una función, en realidad se pasa un puntero al primer elemento. Es decir:

```
void funcion(int arr[] , int longitud) {…}
```

es lo mismo que:

```
void funcion(int * arr , int longitud) {…}
```

Hay una denominada “equivalencia” entre la aritmética de punteros y la indexación arreglos. Esto nos permite:

acceder a un arreglo a través de un puntero, y

utilizar un puntero para *simular* un arreglo.

Arreglos y punteros (4)

Veamos un pequeño ejemplo:

```
void cargar_arreglo (int * arreglo, int limite) {
    for(int i = 0; i < limite; i++)
        arreglo[i] = i;                 // *(arreglo+i) = i;
}

void mostrar_arreglo (int * arreglo, int limite) {
    for(int i = 0; i < limite; i++)
        cout << arreglo[i] << endl; // cout << *(arreglo+i) << endl;
}

int main() {
    int * arr = new int[10];        // int arr[10];
    cargar_arreglo(arr, 10);        // cargar_arreglo(&arr[0], 10);
    mostrar_arreglo(arr + 5, 5);  // mostrar_arreglo(&arr[5], 5);
    delete [] arr;
    return 0;
}
```

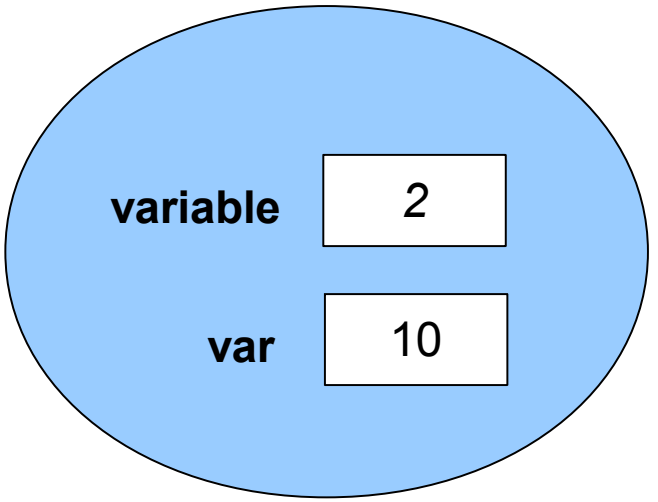
Pasaje de parámetros (1)

- Pasaje por copia:

```
void modificar(int var)
{
   var = 10;
   cout << "modificar - valor variable: "
            << var << endl;
}

int main (int argc, char * argv[])
{
    int variable = 2;
    modificar(variable);
    cout<<"main - valor variable: "
          << variable << endl;

   return 0;
}
```




```
modificar - valor variable: 10
main - valor variable: 2
```

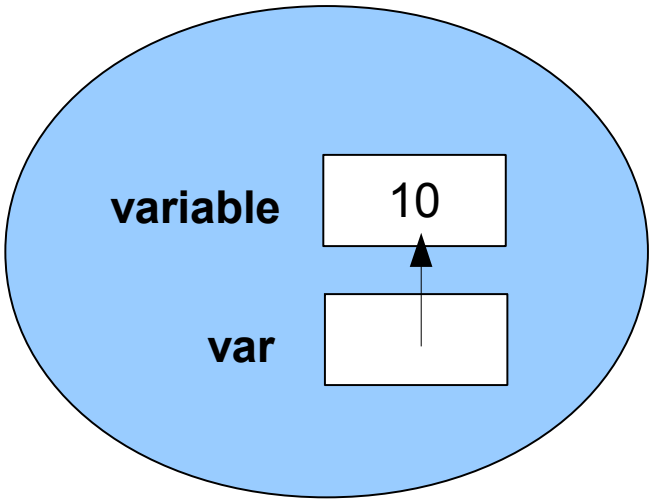
Pasaje de parámetros (2)

- Pasaje por referencia:

```
void modificar(int & var)
{
   var = 10;
   cout << "modificar - valor variable: "
             << var << endl;
}

int main (int argc, char * argv[])
{
    int variable = 2;
    modificar(variable);
    cout<<"main - valor variable: "
          << variable << endl;

   return 0;
}
```




```
modificar - valor variable: 10
main - valor variable: 10
```

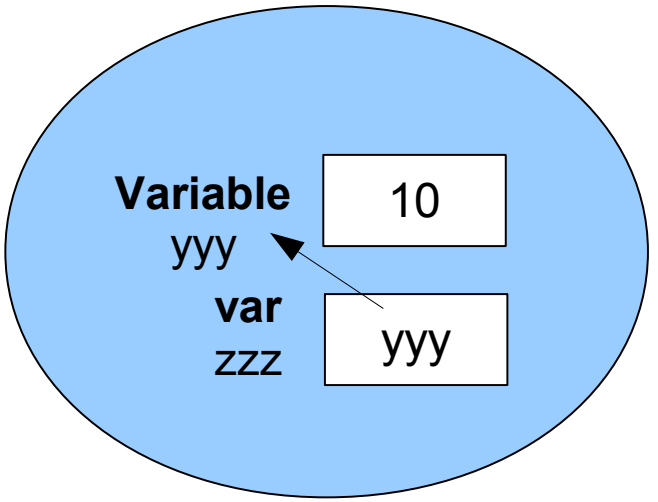
Pasaje de parámetros (3)

- Pasaje “por referencia” al estilo C:

```
void modificar(int * var)
{
   *var = 10;
   cout << "modificar - valor variable: "
            << *var << endl;
}

int main (int argc, char * argv[])
{
    int variable = 2;
    modificar(&variable);
    cout<<"main - valor variable: "
          << variable << endl;

   return 0;
}
```




```
modificar - valor variable: 10
main - valor variable: 10
```

- ¿Qué pasa si modificamos el puntero en lugar del valor que contiene?

Estructuras dinámicas (1)

Veamos un ejemplo:

```
struct nodo_lista {

    string nombre;
    nodo_lista * siguiente;
};


void insertar_nombre(nodo_lista * & lista, string nombre) {
    nodo_lista * nodo = new nodo_lista;
    nodo->nombre = nombre;
    nodo->siguiente = lista;
    lista = nodo;
}

void cargar_nombres(nodo_lista * & lista) {
    string nombre(" ");
    while (nombre != "") {
        cout << "Ingresar nombre (vacío para terminar):" << endl;
        getline(cin, nombre);
        insertar_nombre(lista, nombre);
    }
}
```

Estructuras dinámicas (2)

```
void mostrar_nombres(nodo_lista * lista) {
    while (lista != NULL) {
        cout << lista->nombre << endl;
        lista = lista->siguiente;
    }
}

int main() {
    nodo_lista * nombres = NULL;
    cargar_nombres(nombres);
    mostrar_nombres(nombres);
    return 0;
}
```

¿Este ejemplo está completo?

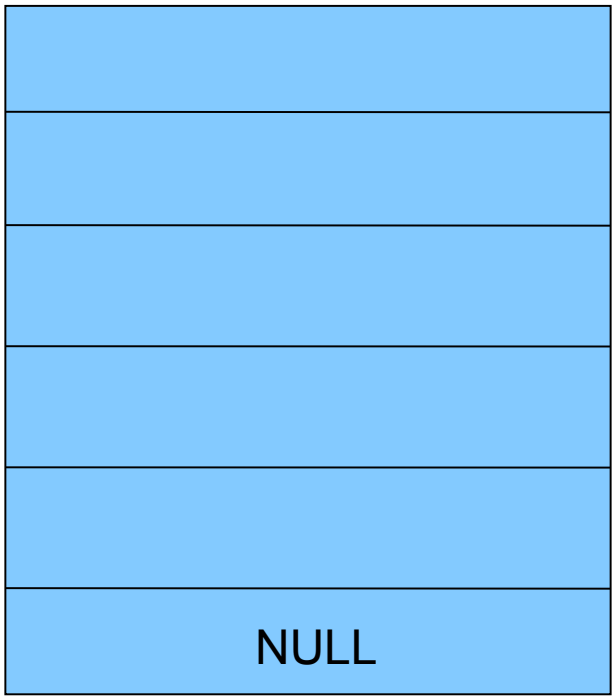
Estructuras dinámicas (3)

```
void vaciar_nombres(nodo_lista * & lista) {
    nodo_lista * actual;
    while (lista != NULL) {
        actual = lista;
        lista = lista->siguiente;
        delete actual;
    }
    lista = NULL;
}

int main() {
    nodo_lista * nombres = NULL;
    cargar_nombres(nombres);
    mostrar_nombres(nombres);
    vaciar_nombres(nombres);
    return 0;
}
```

Estructuras dinámicas (4)

```
int main() {
    nodo_lista * nombres = NULL;
    cargar_nombres(nombres);
    mostrar_nombres(nombres);
    vaciar_nombres(nombres);
    return 0;
}
```

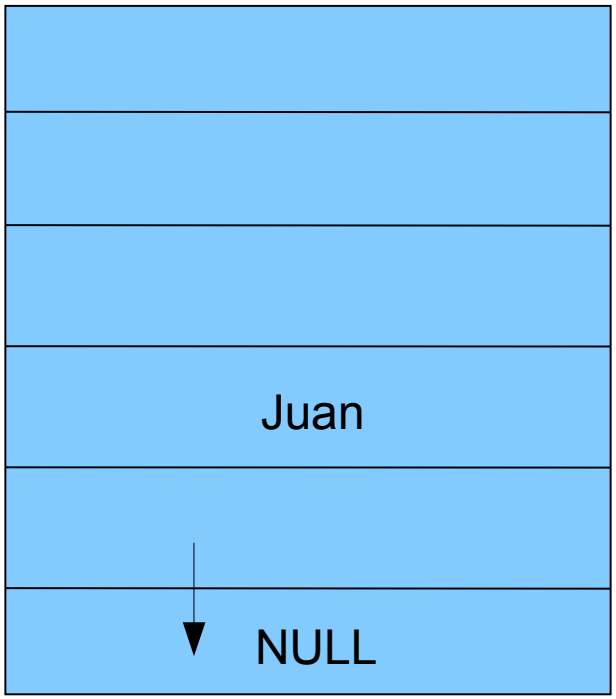


nombres

nodo_lista

Estructuras dinámicas (6)

```
int main() {
    nodo_lista * nombres = NULL;
    cargar_nombres(nombres);
    mostrar_nombres(nombres);
    vaciar_nombres(nombres);
    return 0;
}
void cargar_nombres(nodo_lista * &lista) {
    string nombre(" ");
    while (nombre != "") {
        cout << "Ingresar nombre (vacío para
terminar):\n";
        getline(cin, nombre);
        insertar_nombre(lista, nombre);
    }
}
```



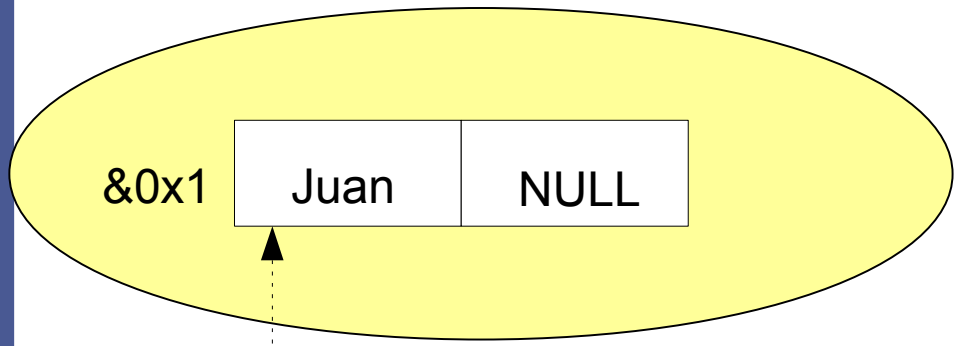
nombre

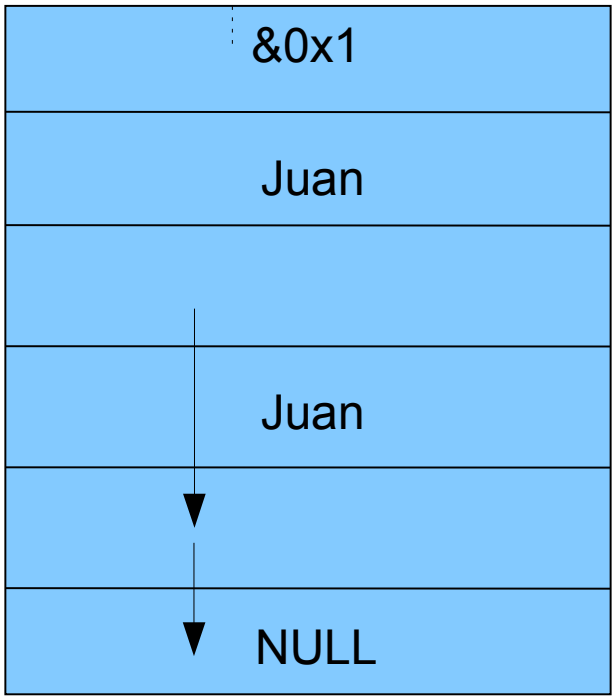
lista (cargar_nombres)

nombres

nodo_lista

Estructuras dinámicas (7)





nodo

nombre
(insertar_nombres)

lista
(insertar_nombres)

nombre
(cargar_nombres)

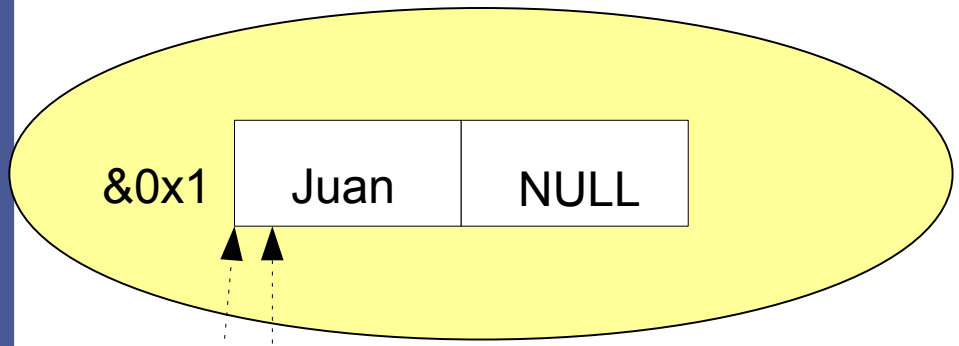
lista
(cargar_nombres)

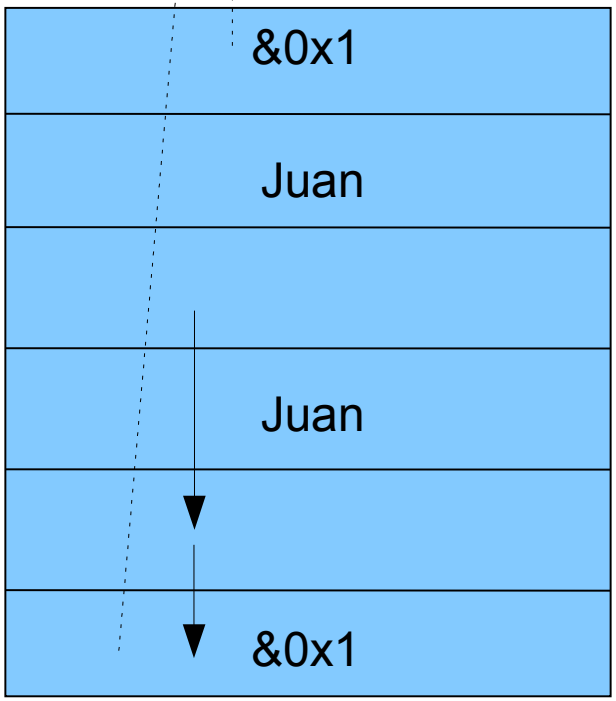
nombres

```
int main() {
    nodo_lista * nombres = NULL;
    cargar_nombres(nombres);
    mostrar_nombres(nombres);
    vaciar_nombres(nombres);
    return 0;
}
void cargar_nombres(nodo_lista * &lista) {
    string nombre(" ");
    while (nombre != "") {
        cout << "Ingresar nombre (vacío para
terminar):\n";
        getline(cin, nombre);
        insertar_nombre(lista, nombre);
    }
}
void insertar_nombre(nodo_lista * & lista,
string nombre) {
    nodo_lista * nodo = new nodo_lista;
    nodo->nombre = nombre;
    nodo->siguiente = lista;
    lista = nodo;
}
```

nodo_lista

Estructuras dinámicas (8)

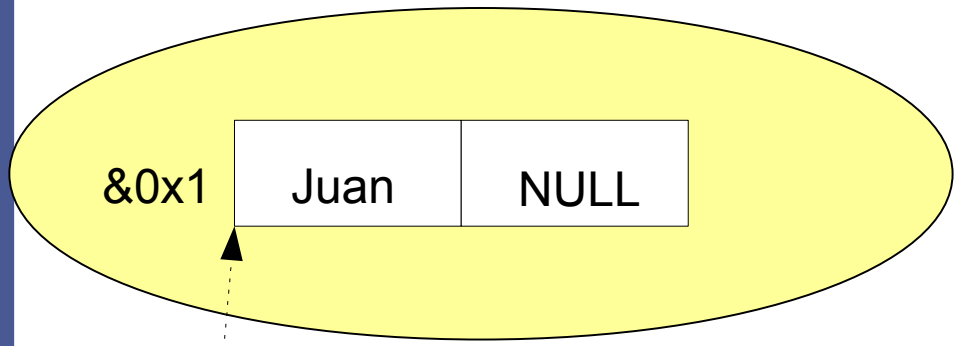


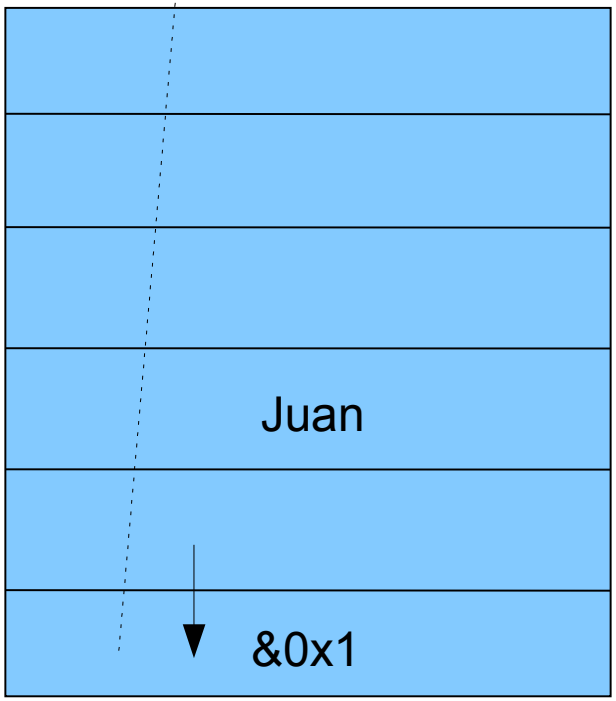



```
int main() {
    nodo_lista * nombres = NULL;
    cargar_nombres(nombres);
    mostrar_nombres(nombres);
    vaciar_nombres(nombres);
    return 0;
}
void cargar_nombres(nodo_lista * &lista) {
    string nombre(" ");
    while (nombre != "") {
        cout << "Ingresar nombre (vacío para
terminar):\n";
        getline(cin, nombre);
        insertar_nombre(lista, nombre);
    }
}
void insertar_nombre(nodo_lista * & lista,
string nombre) {
    nodo_lista * nodo = new nodo_lista;
    nodo->nombre = nombre;
    nodo->siguiente = lista;
    lista = nodo;
}
```

nodo_lista

Estructuras dinámicas (9)





nombre
(cargar_nombres)

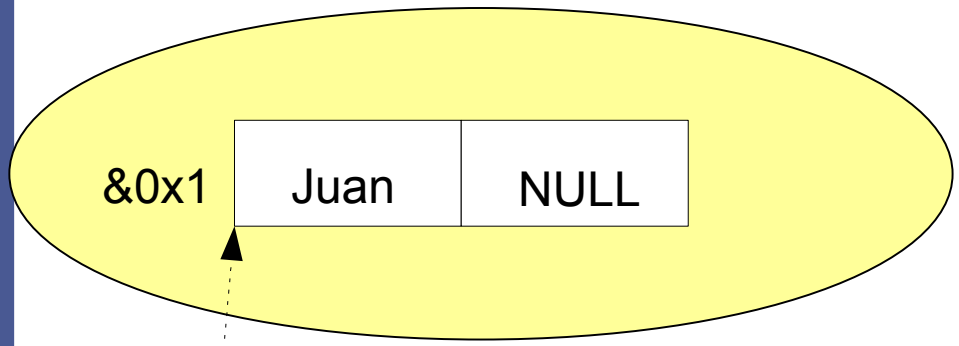
lista
(cargar_nombres)

nombres

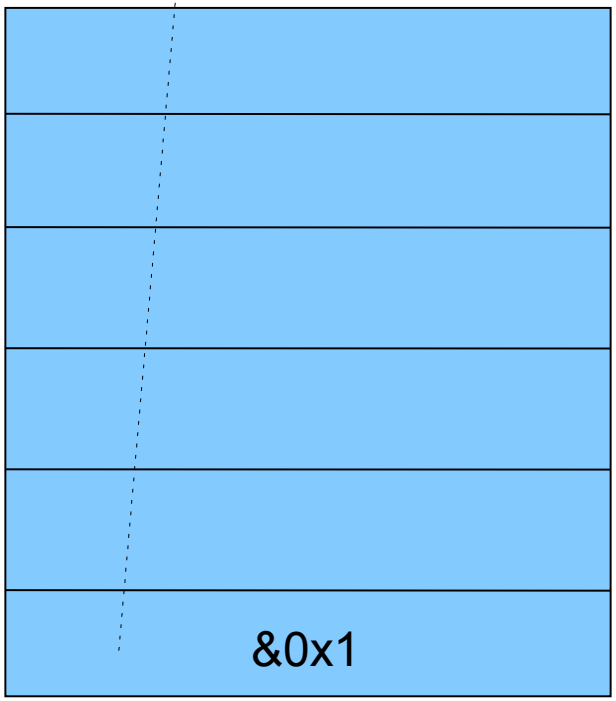
```
int main() {
    nodo_lista * nombres = NULL;
    cargar_nombres(nombres);
    mostrar_nombres(nombres);
    vaciar_nombres(nombres);
    return 0;
}
void cargar_nombres(nodo_lista * &lista) {
    string nombre(" ");
    while (nombre != "") {
        cout << "Ingresar nombre (vacío para
terminar):\n";
        getline(cin, nombre);
        insertar_nombre(lista, nombre);
    }
}
```

nodo_lista

Estructuras dinámicas (10)




```
int main() {
    nodo_lista * nombres = NULL;
    cargar_nombres(nombres);
    mostrar_nombres(nombres);
    vaciar_nombres(nombres);
    return 0;
}
```



nombres

nodo_lista

Consultas: laboratorio.ayda@alumnos.exa.unicen.edu.ar