Punteros y Arreglos

Profesor Yisheng León

Punteros y arreglos

Puntero al primer elemento del array

```
void main()
{
    int dias[12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

    cout << *dias << endl;
}
```

Importante

Siempre apunta al primer elemento

Siempre apunta al primer elemento (no se puede modificar)
Acceso a los elementos del array:
Por índice o con aritmética de punteros

Punteros y arreglos

Paso de arrays a subprogramas
¡Esto explica por qué no usamos **&** con los parámetros array!

El nombre del array es un puntero: ya es un paso por referencia

```
const int N = ...;
void cuadrado(int arr[N]);
void cuadrado(int arr[], int size); // Array no delimitado
void cuadrado(int *arr, int size); // Puntero
```

Arrays no delimitados y punteros: se necesita la dimensión
Elementos: se acceden con índice (arr[i]) o con puntero (*arr)
Una función sólo puede devolver un array en forma de puntero:

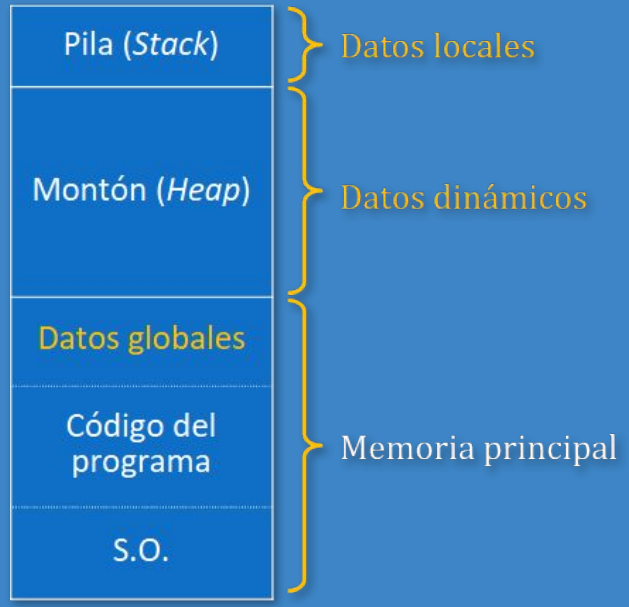
```
intPtr inicializar();
```

Memoria y datos del programa

Memorias y datos del programa

Regiones de la memoria

El sistema operativo distingue varias regiones en la memoria:



Memorias y datos del programa

Memoria principal

Datos globales del programa:
Declarados fuera
de los subprogramas


```
tRegistro;
const int N = 1000;
typedef tRegistro tArray[N];
typedef struct {
    tArray registros;
    int cont;
} tLista;

int main() {
```

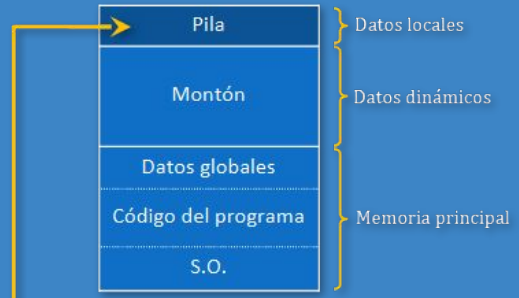
Memorias y datos del programa

La pila (stack)

Datos locales de subprogramas:
Parámetros por valor
y variables locales

```
void func(tLista lista, double &total)
{
  tLista aux;
  int i;
  ...
```

Y los punteros temporales
que apuntan a los argumentos
de los parámetros por referencia



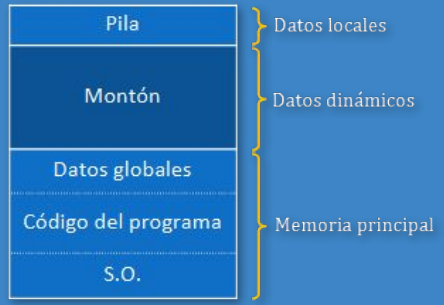

```
                                    &resultado
func(lista, resultado)
```

Memorias y datos del programa

El montón (heap)

Datos dinámicos

Datos que se crean y se destruyen durante la ejecución del programa, a medida que se necesita



Sistema de gestión de memoria dinámica (SGMD)

Cuando se necesita memoria para una variable se solicita

- El SGMD reserva espacio y devuelve la dirección base

Cuando ya no se necesita más la variable, se destruye

- Se libera la memoria y el SGMD cuenta de nuevo con ella

Memoria dinámica

Memoria dinámica

Datos dinámicos

Se crean y se destruyen durante la ejecución del programa

Se les asigna memoria del montón

¿Por qué utilizar memoria dinámica?

- ✓ Almacén de memoria muy grande: datos o listas de datos que no caben en memoria principal pueden caber en el montón
- ✓ El programa ajusta el uso de la memoria a las necesidades de cada momento: ni le falta ni la desperdicia

Datos y asignación de Memoria

¿Cuándo se asigna memoria a los datos?

Datos globales

- En memoria principal al comenzar la ejecución del programa
- Existen durante toda la ejecución del programa

Datos locales de un subprograma

- En la pila al ejecutarse el subprograma
- Existen sólo durante la ejecución de su subprograma

Datos dinámicos

- En el montón cuando el programa lo solicita
- Existen a voluntad del programa

Datos estáticos frente a datos dinámicos

Datos estáticos

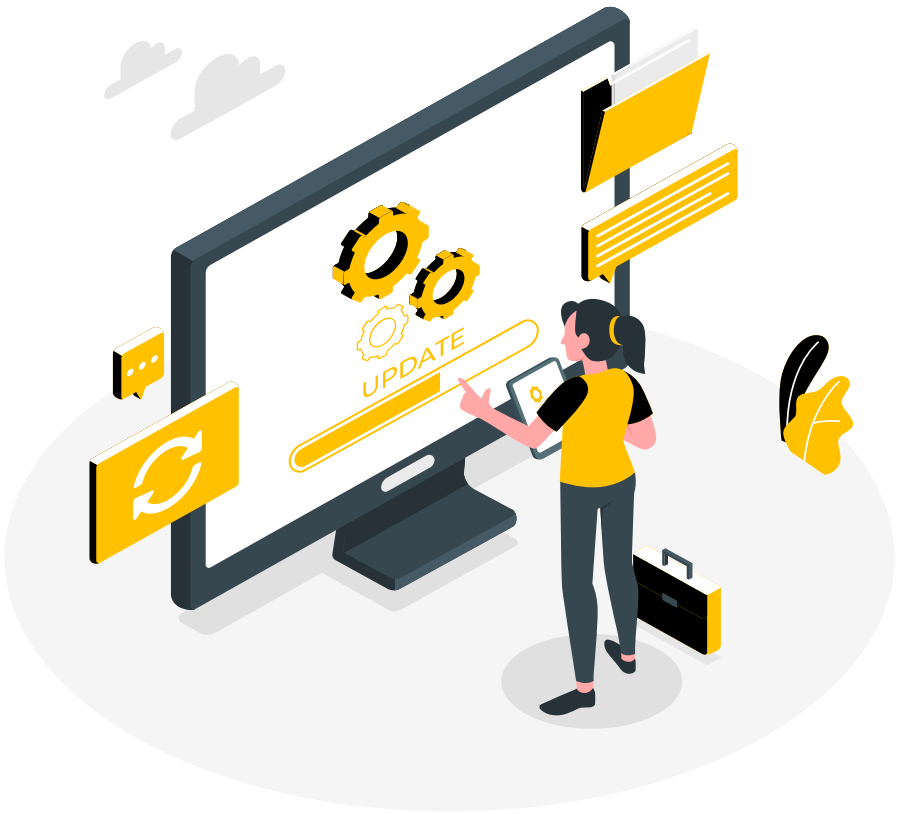
- Datos declarados como de un tipo concreto:
  int i;
- Se acceden directamente a través del identificador:
  cout << i;

Datos dinámicos

- Datos accedidos a través de su dirección de memoria
  Esa dirección de memoria debe estar algún puntero
  Los punteros son la base del SGMD

Los datos estáticos también se pueden acceder a través de punteros

```
int *p = &i;
```



Punteros dinámicos

Creación de datos dinámicos

El operador `new`

`new` *`tipo`* Reserva memoria del montón para una variable del *`tipo`* y devuelve la primera dirección de memoria utilizada, que debe ser asignada a un puntero

```
int *p;       // Todavía sin una dirección válida
p = new int; // Ya tiene una dirección válida
*p = 12;
```

La variable dinámica se accede exclusivamente por punteros

No tiene identificador asociado

```
int i;          // i es una variable estática
int *p1, *p2;
p1 = &i;        // Puntero que da acceso a la variable
                // estática i (accesible con i o con *p1)
p2 = new int; // Puntero que da acceso a una variable
                // dinámica (accesible sólo a través de p2)
```

[redacted] Devuelve `NULL` si no queda memoria suficiente

Inicialización de datos dinámicos

Inicialización con el operador `new`

El operador `new` admite un valor inicial para el dato creado:

```
int *p;
p = new int(12);
```

Se crea la variable, de tipo `int`, y se inicializa con el valor 12

```
#include <iostream>
using namespace std;
#include "registro.h"

int main() {
   tRegistro reg;
   reg = nuevo();
   tRegistro *punt = new tRegistro(reg);
   mostrar(*punt);
   ...
```

Eliminación de datos dinámicos

El operador `delete`

`delete` *`puntero`*`;` Devuelve al montón la memoria usada por la variable dinámica apuntada por *`puntero`*

```
int *p;
p = new int;
*p = 12;
...
delete p; // Ya no se necesita el entero apuntado por
p
```

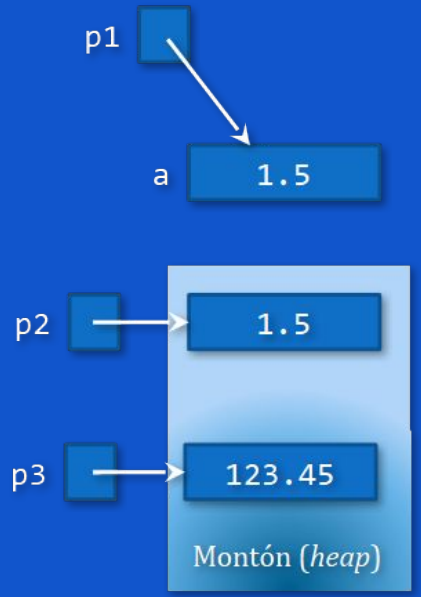
👀 ¡El puntero deja de contener una dirección válida!

Ejemplo de variables dinámicas

```
#include <iostream>
using namespace std;

int main() {
    double a = 1.5;
    double *p1, *p2, *p3;
    p1 = &a;
    p2 = new double;
    *p2 = *p1;
    p3 = new double;
    *p3 = 123.45;
    cout << *p1 << endl;
    cout << *p2 << endl;
    cout << *p3 << endl;
    delete p2;
    delete p3;

    return 0;
}
```



Identificadores:

4

(a, p1, p2, p3)

Variables:

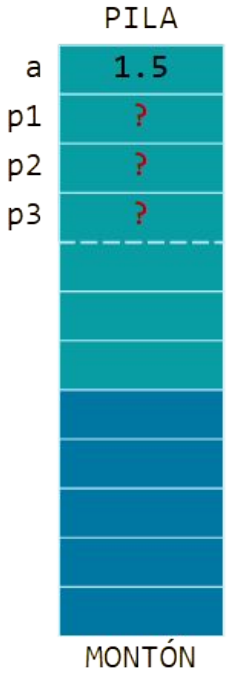
6

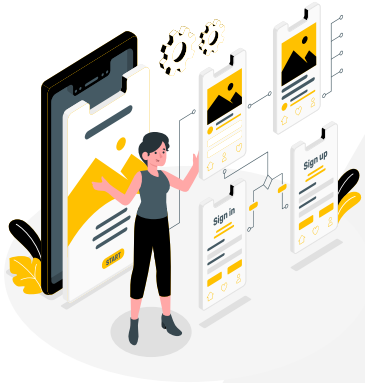
(+ *p2 y *p3)

Ejemplo de variables dinámicas

```
#include <iostream>
using namespace std;

int main() {
    double a = 1.5;
    double *p1, *p2, *p3;
```

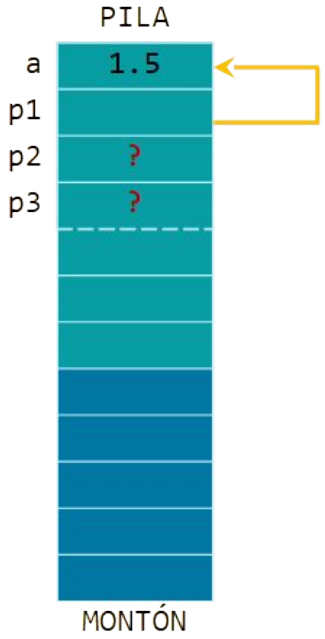


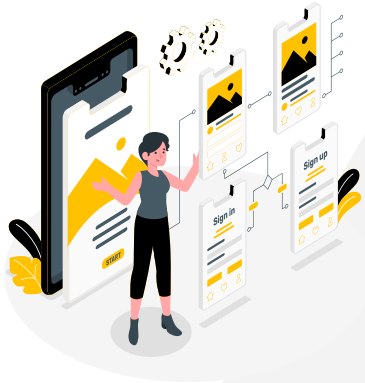


Ejemplo de variables dinámicas

```
#include <iostream>
using namespace std;

int main() {
    double a = 1.5;
    double *p1, *p2, *p3;
    p1 = &a;
```

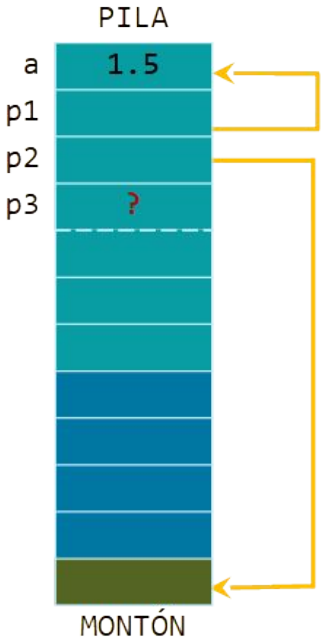


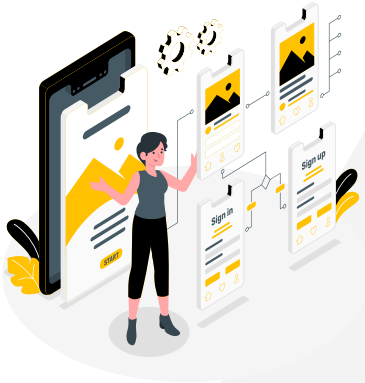


Ejemplo de variables dinámicas

```
#include <iostream>
using namespace std;

int main() {
    double a = 1.5;
    double *p1, *p2, *p3;
    p1 = &a;
    p2 = new double;
```

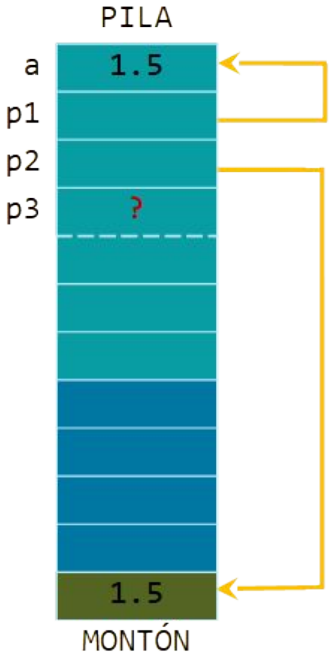


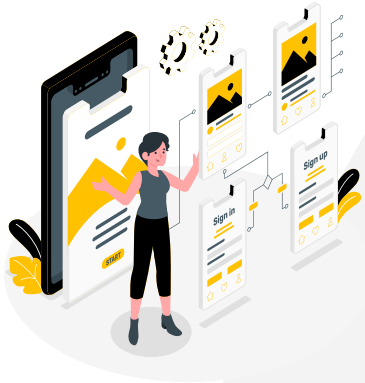


Ejemplo de variables dinámicas

```
#include <iostream>
using namespace std;

int main() {
    double a = 1.5;
    double *p1, *p2, *p3;
    p1 = &a;
    p2 = new double;
    *p2 = *p1;
```

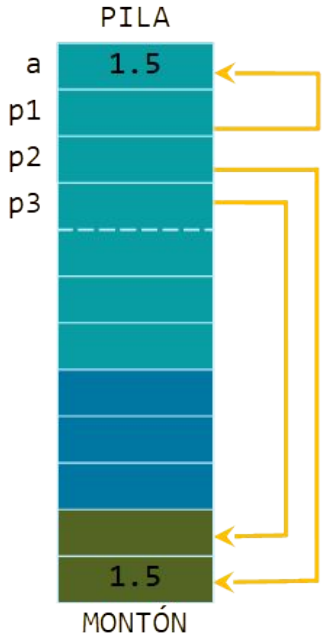


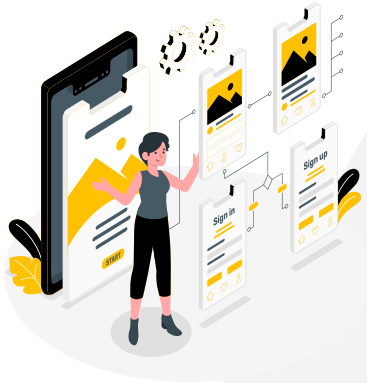


Ejemplo de variables dinámicas

```
#include <iostream>
using namespace std;

int main() {
    double a = 1.5;
    double *p1, *p2, *p3;
    p1 = &a;
    p2 = new double;
    *p2 = *p1;
    p3 = new double;
```

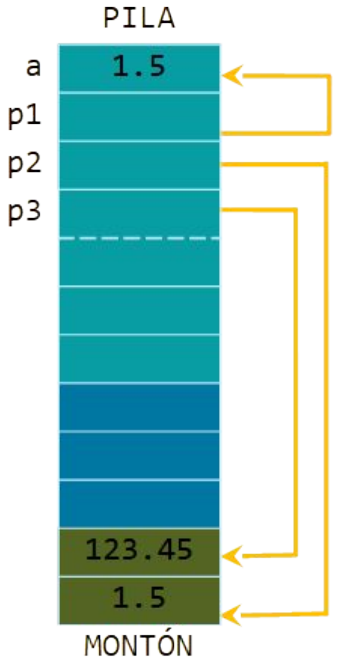


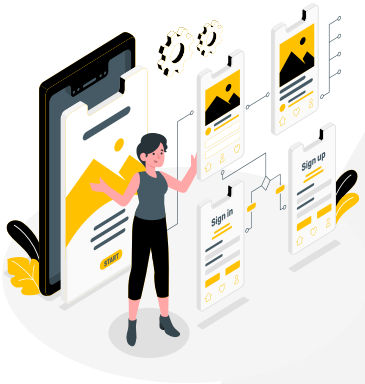


Ejemplo de variables dinámicas

```
#include <iostream>
using namespace std;

int main() {
    double a = 1.5;
    double *p1, *p2, *p3;
    p1 = &a;
    p2 = new double;
    *p2 = *p1;
    p3 = new double;
    *p3 = 123.45;
```

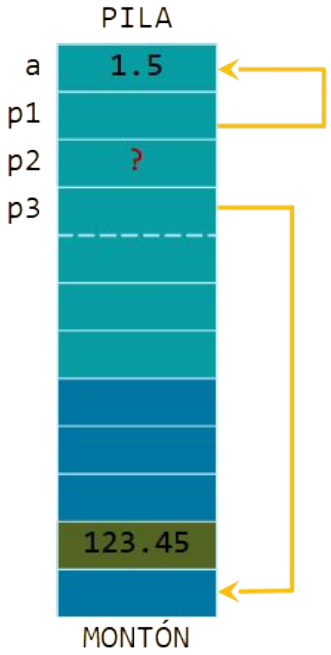


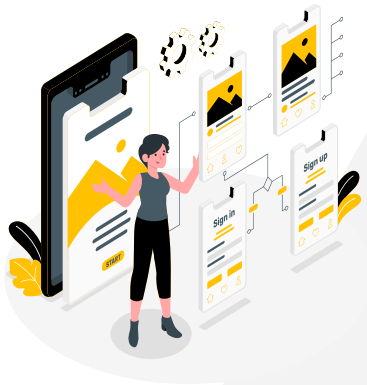


Ejemplo de variables dinámicas

```
#include <iostream>
using namespace std;

int main() {
    double a = 1.5;
    double *p1, *p2, *p3;
    p1 = &a;
    p2 = new double;
    *p2 = *p1;
    p3 = new double;
    *p3 = 123.45;
    cout << *p1 << endl;
    cout << *p2 << endl;
    cout << *p3 << endl;
    delete p2;
```

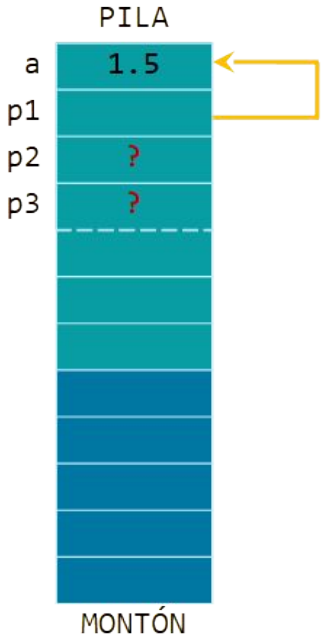


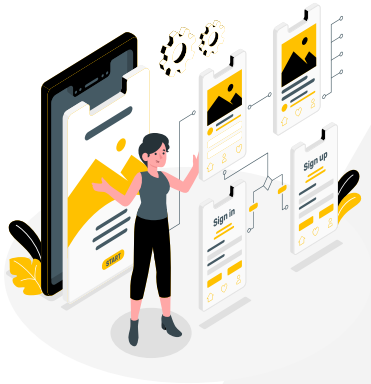


Ejemplo de variables dinámicas

```
#include <iostream>
using namespace std;

int main() {
    double a = 1.5;
    double *p1, *p2, *p3;
    p1 = &a;
    p2 = new double;
    *p2 = *p1;
    p3 = new double;
    *p3 = 123.45;
    cout << *p1 << endl;
    cout << *p2 << endl;
    cout << *p3 << endl;
    delete p2;
    delete p3;
```





Hasta luego