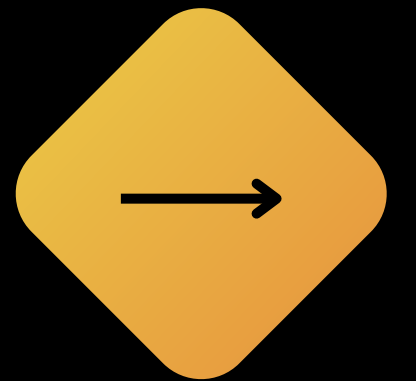


GESTION DE MEMORIA ESTATICA



Estructura de datos

Conceptos fundamentales



Definición:

- **La gestión de memoria estática se refiere al proceso de asignación y liberación de memoria en un programa de computadora durante la fase de compilación. La cantidad de memoria necesaria para variables, estructuras de datos y otros elementos es determinada y reservada antes de que el programa se ejecute. La asignación estática de memoria se realiza generalmente en tiempo de compilación y permanece constante durante la ejecución del programa.**



CARACTERISTICAS

- **Asignación en Tiempo de Compilación:**

La memoria estática se asigna durante la fase de compilación del programa, antes de que se ejecute. Esto contrasta con la asignación dinámica, que ocurre en tiempo de ejecución.

- **Tamaño Fijo:**

El tamaño de la memoria asignada estátamente es fijo y se determina durante la compilación. No puede cambiar dinámicamente durante la ejecución del programa.

- **Eficiencia en Tiempo de Ejecución:**

La gestión de memoria estática tiende a ser más eficiente en tiempo de ejecución en comparación con la gestión dinámica. No hay gastos generales asociados con la asignación y liberación dinámica de memoria.

```
746 }
747 }
748
749 $sort_order = array();
750
751 foreach ($quotes as $key => $value) {
752     $sort_order[$key] = $value['sort_order'];
753 }
754
755 array_multisort($sort_order, SORT_ASC, $quotes);
756
757 $this->session->data['lpa']['shipping_methods'] = $quotes;
758 $this->session->data['lpa']['address'] = $address;
759
760 if (empty($quotes)) {
761     $json['error'] = $this->language->get('
762         error_no_shipping_methods');
763 } else {
764     $json['quotes'] = $quotes;
765 }
766
767 if (isset($this->session->data['lpa']['shipping_method']) && !
768     empty($this->session->data['lpa']['shipping_method']) &&
769     isset($this->session->data['lpa']['shipping_method']['code']
770 )) {
771     $json['selected'] = $this->session->data['lpa']['
772         shipping_method']['code'];
773 } else {
774     $json['selected'] = '';
775 }
776
777 } else {
778     $json['error'] = $this->language->get('error_shipping_methods');
779 }
780
781 $this->response->addHeader('Content-Type: application/json');
```

```
384 {
385     this.$element.find('.next, .prev').length && $.support.transi
386     this.$element.trigger($.support.transition.end)
387     this.cycle(true)
388 }
389
390 this.interval = clearInterval(this.interval)
391
392 return this
393 }
394
395 Carousel.prototype.next = function () {
396     if (this.sliding) return
397     return this.slide('next')
398 }
399
400 Carousel.prototype.prev = function () {
401     if (this.sliding) return
402     return this.slide('prev')
403 }
404
405 Carousel.prototype.slide = function (type, next) {
406     var $active = this.$element.find('.item.active')
407     var $next = next || this.getItemForDirection(type, $active)
408     var isCycling = this.interval
409     var direction = type == 'next' ? 'left' : 'right'
410     var fallback = type == 'next' ? 'first' : 'last'
411     var that = this
412
413     if (!$next.length) {
414         if (!this.options.wrap) return
415         $next = this.$element.find('.item')[fallback]()
416     }
417
418     if ($next.hasClass('active')) return (this.sliding = false)
419
420     var relatedTarget = $next[0]
421     var slideEvent = $.Event('slide.bs.carousel', {
422         relatedTarget: relatedTarget,
423         direction: direction
424     })
425     this.$element.trigger(slideEvent)
```

- **Acceso Rápido:**

Debido a que la asignación de memoria estática se realiza durante la compilación, el acceso a los datos almacenados estátamente es rápido y predecible. No hay necesidad de cálculos adicionales para encontrar la posición en memoria.

Ejemplo:

```
[*]main.cpp ×
1 //Gestion de memoria estatica
2
3 #include <iostream>
4
5 using namespace std;
6
7 int main() {
8     // Variable global
9     int contadorGlobal = 0;
10
11     // Función que utiliza memoria estática
12     auto incrementarContadorLocal = [&]() {
13         int contadorLocal = 0;
14         ++contadorLocal;
15     };
16
17     cout << "Valor actual del contador global: " << contadorGlobal << endl;
18
19     contadorGlobal += 5;
20
21     // Llamando a la función que utiliza la variable local
22     incrementarContadorLocal();
23
24     // Utilizando nuevamente la variable global modificada
25     cout << "Nuevo valor del contador global: " << contadorGlobal << endl;
26
27     return 0;
28 }
29
```

(globals)

C:\Users\Andres\Desktop\main.exe

```
1 Valor actual del contador global: 0
2 Nuevo valor del contador global: 5
3
4 -----
5 Process exited after 0.07437 seconds with return value 0
6 Presione una tecla para continuar . . .
7
8
```

