Reserva Dinámica, Explícita, y Manual de Memoria: Heap

Operadores new & delete

Reserva: Analogía con Hotel o Restaurant

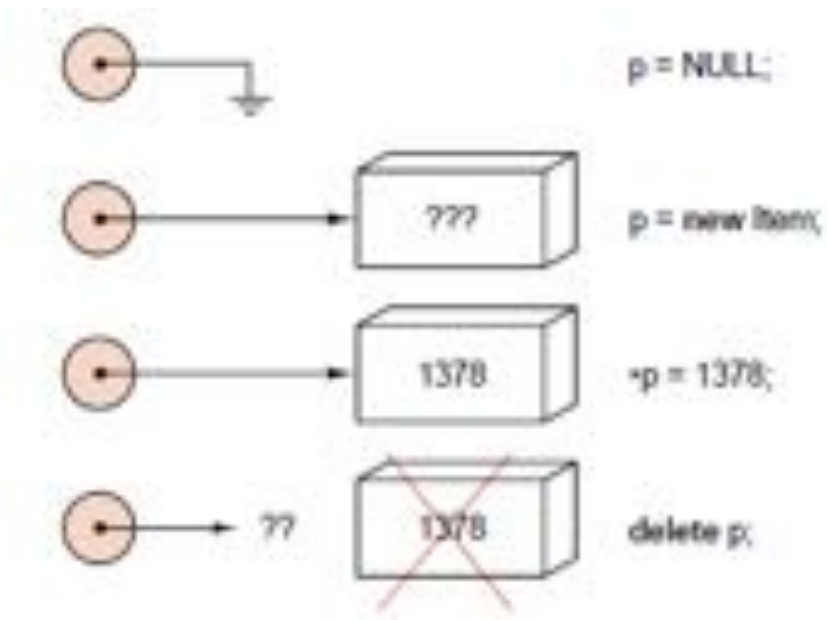
- Llamado para hacer una reserva
  - ¿Qué se pide? ¿Nombre? ¿Cantidad?
  - ¿Qué se registra?
  - ¿Quién lo registras?
  - ¿Qué se retorna?
  - Operador `new`
- Llamado para cancelar o borrar reserva
  - Operador `delete`
- Diferencias con las variables automáticas del stack frame
  - ¿Cómo se reserva?
  - ¿Qué nombre tienen de los objetos?
  - ¿Cuánto duran (lifetiem) los objetos?
  - ¿Cómo se libera la reserva?
  - ¿Quién la lubera?
- ¿Qué aplicación tiene la reserva manual desde el heap?
- Pregunta avanzada: ¿Qué es el Garbage Collector? ¿Tiene C++ GC?

```
// Un int
int* p{new int};
//…
delete p;

// Arreglo de ints
unsigned n;
std::cin >> n;
int* pa{new int[n]};
//…
delete[] pa;
```

Reserva Dinámica Manual de Memoria – Heap

```
int* p;       // ¿valor de p?
p = nullptr;
p = new int; // ¿valor del nuevo int?
p             // ¿valor de p?
*p            // ¿valor de ?
*p = 1378     // ¿se modificó p?
cout << *p;   // ¿qué enviamos?
cout << p;    // ¿qué enviamos?
delete p;
cout << p;    // ¿qué enviamos?
*p = 1378;    // ¿Por qué es incorrecto? ¿Qué pasa?
```



- ¿Crea algo nuevo new? ¿Qué?
- ¿Borra algo delete? ¿Qué?

Preguntas de Repaso Sobre Reserva Dinámica de Memoria

- ¿Qué efecto de lado tiene el operador new? Ejemplo: `auto p{new T};`
- ¿Qué efecto el tiene el operador `delete`? ¿Cómo afecta al puntero? Ejemplo: `delete p;`
- ¿El uso de punteros implica el uso del operador `new`?
- ¿El uso del operador `new` implica el uso punteros?
- ¿Se puede aplicar `delete` a cualquier puntero?
- ¿Qué reserva dinámica es más rápida? ¿La del *heap*? ¿La del *stack*? ¿Por qué?
- ¿Qué estrategias podemos aplicar para hacer más eficiente la reserva?