

# Lezione 03: Allocazione della memoria

## Free store(heap)

La vita di un oggetto con nome è determinata dallo scope in cui è stato definito. A volte può però rendersi necessario creare un oggetto la cui esistenza sia indipendente dallo scope in cui è stato creato. Ad esempio, voglio creare un oggetto all'interno di una funzione e poi restituirlo in modo che possa continuare ad essere utilizzato. Per ottenere ciò si utilizzano:

- **new** per creare un oggetto.
- **delete** per distruggerlo.
- **delete[]** se si tratta di un array.

## New

In C++, **new** serve per allocare memoria dinamicamente sul **heap**, invece che sullo **stack**.

Restituisce un puntatore al tipo allocato. In caso di utilizzo con una classe viene chiamato automaticamente il

**costruttore**. Per oggetti complessi, viene chiamato il costruttore per ogni elemento.

```
int* p = new int;    // alloca 1 intero
*p = 10;            // assegna valore

std::string* s = new std::string("ciao");

int* arr = new int[5];    // alloca array di 5 interi (inizializzati con valori casuali)
double* v = new double[3]{1.1, 2.2, 3.3}; // inizializzazione diretta
```

## Delete

In C++, **delete** serve per liberare memoria allocata con **new**.

Se il tipo è una classe, viene chiamato anche il **distruttore**.

```
int* p = new int(42);  
delete p;    // libera memoria  
p = nullptr; // opzionale ma buona pratica
```

## Delete[ ]

In C++ il **delete[ ]** va utilizzato quando si creano oggetti con la **new[ ]**.

Fa la distruzione di ogni elemento e poi libera l'intero blocco di memoria.

```
std::string* arr = new std::string[3];  
arr[0] = "Ciao";  
arr[1] = "Mondo";  
arr[2] = "!";  
  
delete[] arr;    // chiama il distruttore su tutti gli elementi  
arr = nullptr;
```

## Problemi di gestione della memoria

In C++ possiamo avere tre principali problematiche relegate alla gestione della memoria nello heap:

- Oggetti abbandonati(leaked).
- Cancellazione prematura.
- Doppia cancellazione.

Entrando nello specifico, un **oggetto abbandonato(leaked)** è un blocco di memoria che non viene più raggiunto dal programma, ma non è stato deallocato. Se questo succede migliaia di volte, la RAM si riempie ed il programma rallenta o crasha.

```
int* p = new int(42);  
p = new int(100); // ✗ perdita del primo oggetto
```

La **doppia cancellazione** invece avviene quando viene chiamata **delete** più volte sullo stesso puntatore. In questo caso la **delete** avviene su memoria già liberata, portando quindi ad un comportamento indefinito. Questo errore può portare a crash del programma oppure a comportamenti imprevedibili. È quindi buona pratica mettere a **NULL** il puntatore dopo la **delete**.

```
int* p = new int(42);  
delete p;  
delete p; // ✗ doppia cancellazione
```

La **cancellazione prematura** avviene invece quando viene cancellato un oggetto ma all'intero del codice successivo è ancora utilizzato il puntatore che lo riferiva. Il puntatore diventa quindi **darling (penzolante)** e punta a memoria non più valida. Questo errore porta a stampare valori spazzatura o a crash improvvisi del programma. Può essere difficile da individuare in fase di debug e può far perdere ore allo sviluppatore.

```
int* p = new int(42);  
delete p;  
std::cout << *p << "\n"; // ✗ uso di memoria già liberata
```