<ctime>

Storicamente, `<ctime>` arriva dal mondo C e ruota attorno a due rappresentazioni: `std::time_t`, un conteggio dei secondi dall'epoca (di solito 1 gennaio 1970 UTC), e `std::tm`, una struttura scomposta in anno, mese, giorno, ora e così via. Questo modello è semplice da serializzare, interoperabile con librerie di sistema e spesso sufficiente quando dobbiamo solo prendere l'ora corrente e stamparla.

La libreria `<ctime>` (ereditata dal C) fornisce strumenti per lavorare con date e orari.
Due elementi principali:

- **`std::time_t`**: rappresenta il numero di secondi trascorsi dall'**epoca UNIX** (1 gennaio 1970, 00:00:00 UTC).
- **`std::tm`**: struttura che scompone il tempo in campi leggibili (anno, mese, giorno, ora, minuti, secondi).

MEMBRI STATICI

I membri statici in C++ sono elementi che appartengono alla classe e non alle singole istanze. Questo significa che esiste un'unica copia condivisa tra tutti gli oggetti della stessa classe e che possono essere utilizzati anche senza creare un oggetto, tramite la sintassi Classe::membro. Le variabili statiche di classe sono quindi variabili condivise da tutte le istanze e vengono definite al di fuori della classe. Le funzioni statiche invece non ricevono il puntatore nascosto this, quindi non possono accedere ai membri non statici della classe. Possono però essere richiamate senza istanziare oggetti e accedere ad altri membri statici.
Il vantaggio principale dei membri statici è la possibilità di condividere dati tra tutti gli oggetti di una classe, di raggruppare funzioni di utilità legate logicamente alla classe e di evitare l'uso di variabili globali sparse.
Questo significa che esiste **una sola copia condivisa** di quel membro, indipendentemente da quante istanze della classe vengano create.

- Gli **attributi statici** mantengono un unico valore comune a tutti gli oggetti.
- I **metodi statici** possono essere chiamati senza un'istanza, usando il nome della classe.

Una variabile dichiarata come `static` in una classe è **condivisa da tutti gli oggetti**. Non appartiene all'oggetto, ma alla classe stessa.

Un **metodo statico** non ha accesso diretto ai membri non statici della classe, perché non è legato a un oggetto specifico. Può però accedere ai membri **statici** della classe.

Aspetto	Membro normale (non statico)	Membro statico (di classe)
Appartenenza	Appartiene a ogni singolo oggetto	È condiviso da tutta la classe
Memoria	Ogni oggetto ha la sua copia	Esiste un'unica copia globale
Accesso	Tramite istanza (`oggetto.attr`)	Tramite istanza **o** `Classe::attr`
Inizializzazione	Direttamente nel costruttore	Deve essere definito fuori dalla classe
Durata	Limitata allo scope dell'oggetto	Valido finché il programma è in esecuzione

Allocazione Dinamica della Memoria in C++

In C++ l'**allocazione dinamica della memoria** è il meccanismo che permette di riservare spazio in memoria durante l'esecuzione del programma (runtime), invece che in fase di compilazione.
Per gestirla si utilizzano principalmente due operatori:

- **new** → riserva memoria nello heap e restituisce un puntatore all'area allocata.
- **delete / delete[]** → liberano la memoria precedentemente allocata, restituendola al sistema.

Un **oggetto automatico** è una variabile creata in modo implicito dal compilatore quando il flusso di esecuzione entra in un blocco (ad esempio una funzione, un ciclo o uno scope delimitato da `{}`) e distrutta automaticamente quando il blocco termina.

Gli oggetti automatici sono allocati nello **stack**.

- **Durata:** limitata al blocco in cui sono definiti.
- **Gestione memoria:** automatica, non serve new né delete.
- **Esempio tipico:** variabili locali in una funzione.

Un **oggetto dinamico** è una variabile creata **a runtime** con l'operatore `new`.
Non viene distrutta automaticamente alla fine dello scope: rimane in memoria finché non viene liberata esplicitamente con `delete`.
Gli oggetti dinamici sono allocati nello **heap**.

- **Durata:** persiste fino a quando non viene esplicitamente deallocato.
- **Gestione memoria:** manuale, tramite `new` e `delete`.
- **Esempio tipico:** strutture dati che devono crescere/ridursi a runtime.

Caratteristica	Oggetto automatico (stack)	Oggetto dinamico (heap)
Creazione	Avviene automaticamente entrando in un blocco	Avviene con `new` esplicito
Distruzione	Automatica, all'uscita dal blocco	Manuale, con `delete` / `delete[]`
Durata	Limitata allo scope	Persiste fino a `delete`
Allocazione	Stack (veloce, spazio limitato)	Heap (più lento, ma grande)
Uso tipico	Variabili locali, parametri	Array dinamici, oggetti complessi
Pericoli	Nessuno specifico	Memory leak, dangling pointer

Sintassi → Tipo *ptr = new Tipo;

```
int *p = new int;     // Alloca dinamicamente un intero
*p = 42;              // Assegna il valore 42 all'intero puntato
delete p;             // Libera la memoria
```

*allocazione array → Tipo *ptr = new Tipo[dimensione];*

⚠️ È fondamentale usare **`delete[]`** per gli array dinamici, altrimenti si possono verificare **memory leak** o comportamenti indefiniti.

Inizializzazione con `new`

È possibile inizializzare direttamente l'oggetto o l'array durante l'allocazione:

```
int *p = new int(100);        // intero inizializzato a 100
int *array = new int[5]{1,2,3,4,5};  // array inizializzato
```