4 - Lifetime

Mentre lo scope descrive la dimensione spaziale (dove) di un nome introdotto da una dichiarazione, il tempo di vita descrive la dimensione temporale, necessario per stabilire quando è legittimo interagire con una determinata entità. Alcune entità, come tipi di dato, funzioni, etichette, possono essere riferite in qualunque momento durante l'esecuzione del programma, mentre un oggetto memorizzato in memoria è utilizzabile solo dopo la sua creazione e soltanto fino a quando viene distrutto.

Creazione di oggetti in memoria

- definizione `int n = 42;` (non basta una dichiarazione pura;
- invocazione dell'espressione `new`, allocando oggetti nell'heap;
- valutazione di una espressione che crea implicitamente un nuovo oggetto (temporaneo e senza nome):

```
std::string str = "Hello ";
std::cout << ( str + "␣world!") << std::endl ;
```

Inizio e fine lifetime

- inizia quando un oggetto *termina* la sua **costruzione**, composta da due fasi:
  - allocazione della memoria "grezza";
  - inizializzazione della memoria quando prevista.
- termina quando un oggetto *inizia* la sua **distruzione**, composta da due fasi:
  - invocazione del distruttore (quando previsto);
  - deallocazione della memoria "grezza".
  Se la costruzione dell'oggetto non termina con successo, non ha iniziato il suo lifetime e non dovrà terminarlo.
  Durante le fasi di creazione e di distruzione dell'oggetto si è fuori dal suo ciclo di vita, perciò le operazioni consentite sono limitate

Tipologie di allocazione

Il lifetime di un oggetto dipende dal tipo di "allocazione" utilizzata per la creazione:

- allocazione statica
- allocazione thread local
- allocazione automatica
- allocazione automatica dei temporanei
- allocazione dinamica

Allocazione Statica

Il lifetime dell'oggetto dura "per tutta l'esecuzione del programma". Sono dotate di memoria ad allocazione statica:

- le variabili definite a namespace scope (dette globali):

    - sono create e inizializzate prima di iniziare l'esecuzione del main, nell'ordine in cui compaiono nell'unità di traduzione in cui sono definite;
    - **Static Inizialization Order Fiasco**: nel caso di variabili globali definite in diverse unità di traduzione, l'ordine di inizializzazione non è specificato causando errori;
    - in caso di terminazione normale del programa le globali sono distrutte dopo la terminazione della funzione main, in ordine inverso rispetto alla creazione.
- i dati membro di classi dichiarati usando `static`:
  stesse regole delle globali, tranne nel caso di template.
- le variabili locali (scope di blocco) dichiarate usando `static`:
  sono allocate prima di iniziare l'esecuzione del main, ma sono inizializzate (solo) la prima volta in cui il controllo di esecuzione incontra la corrispondente definizione, nelle esecuzioni successive l'inizializzazione non viene eseguita
  L'utilizzo dell'allocazione statica quando non strettamente necessario porta a codice poco leggibile e poco mantenibile. Se l'allocazione statica è necessaria, preferire i dati membro statici, se possibile dichiarate private, in modo da confinare il codice problematico.

Allocazione thread local

Un oggetto thread local è simile ad un oggetto globale, ma il suo ciclo di vita non è collegato al programma, bensi ad ogni singolo thread di esecuzione creato dal programma. Il supporto per il multithreading è stato introdotto con c++11.

Allocazione automatica

Una variabile locale ad un blocco di funzione (non dichiarata `static`) è dotata di allocazione automatica. L'oggetto viene creato dinamicamente (sullo stack) ogni volta che il controllo entra nel blocco in cui si trova la dichiarazione. Nel caso di funzioni ricorsive, sullo stack possono esistere contemporaneamente più istanze distinte della stessa variabile locale. L'oggetto viene poi automaticamente distrutto rimuovendolo dallo stack ogni volta che il controllo esce da quel blocco. L'inizio del lifetime è stabilito dal programmatore scrivendo la definizione della variabile locale, mentre la fine è automatica.

Allocazione automatica di temporanei

avviene quando un oggetto viene creato per memorizzare il valore calcolato da un sottoespressione che compare all'interno di una espressione, viene poi distrutto quando termina la valutazione dell'espressione (1). Il lifetime del temporaneo può essere esteso se l'oggetto viene utilizzato per inizializzare un riferimento (2).

1.

```
struct S {
        S (int) { std::cout << "costruzione"; }
        ~S () { std::cout << "distruzione"; }
};
void foo ( S s );
void bar () {
        foo ( S (42)); // allocazione di un temporaneo per S (42)
        // temporaneo distrutto prima di stampare fine
```

```
        std::cout << " fine ";
}
```

2.

```
void bar2 () {
// il temporaneo `e usato per inizializzare il riferimento s
        const S & s = S (42);
        std::cout << "fine";
// il temporaneo `e distrutto quando si esce dal blocco ,
// dopo avere stampato " fine "
}
```

Allocazione dinamica

Un oggetto può essere allocato dinamicamente nella memoria heap con l'espressione `new`, che restituisce l'indirizzo dell'oggetto allocato, che va salvato in un opportuno puntatore. La distruzione non è automatica, ma è responsabilità del programmatore utilizzare l'espressione `delete` sul puntatore che contiene l'indirizzo dell'oggetto.

L'allocazione automatica e quella dei temporanei avviene dinamicamente, ovvero a tempo di esecuzione; la differenza sta nel fatto che la deallocazione è manuale.

Errori frequenti di programmazione:

- *memory leak*: accade quando il programmatore perde l'accesso al puntatore che riferisce l'oggetto allocato dinamicamente: la memoria dell'oggetto è ancora in uso, ma non può più essere letta, scritto o rilasciata
- *use after free*: l'utilizzo di un oggetto dopo che il suo lifetime è concluso, uso di un *puntatore dangling*
- *double free*: uso della delete più volte sullo stesso indirizzo, causando la distruzione di una porzione di memoria che non era più allocata e che potenzialmente è stata riutilizzata per altro.
- *accesso al null pointer*: si prova ad accedere ad un puntatore nulla (deferenziandolo)