Process Memory

- La memoria associata a un processo è divisa in quattro aree
- Text/Code – per il codice eseguibile, potrebbe essere memoria read-only
- Data Segment – per le variabili globali e statiche, divisa in due parti
  - Inizializzata seguendo le richieste del codice
  - (BSS) Non esplicitamente inizializzata (messa a 0)
- Stack – per le variabili locali, gestita automaticamente
- **Free memory** (heap) per le variabili locali, gestita via new / delete
  - Singolo oggetto: new / delete
  - Array di oggetti: new [] / delete []
  - È possibile (ma non consigliato) usare le funzioni C: malloc / calloc / realloc e free
    - Con le dovute cautele, e non mescolando blocchi di memoria gestiti in modo diverso

Smart pointer

- Gli smart pointer, per default, operano sulla memoria heap
  - Semplificano e rendono più sicura la sua gestione
  - Possono anche essere utilizzati per altre risorse
    - Specificando via “Deleter”, passato al costruttore, come effettuare il cleanup
- Per risorsa si intende “qualcosa” che deve essere
  - Acquisito prima di poter essere utilizzato
  - Rilasciato quando se ne è terminato l’uso
- Gli smart pointer sono un esempio della tecnica RAII
  - Resource Acquisition Is Initialization
  - Il rilascio di una risorsa è effettuato dal distruttore del suo gestore

unique_ptr

- Scelta naturale per oggetti polimorfici
- Gestisce via puntatore un oggetto allocato sullo heap
- L’eliminazione dell’oggetto avviene
  - Quando lo smart pointer esce dal suo scope
  - O quando il puntatore a un altro oggetto è assegnato allo smart pointer via reset()
- Per la sua creazione è preferito l’uso di **make_unique**()
  - Template function factory

shared_ptr

- Scelta naturale per oggetti condivisi
  - Di cui non sia chiaro chi sia il proprietario
- Più shared_ptr possono gestire un unico oggetto
- Un contatore, accessibile via use_count(), tiene traccia del loro numero
- L'oggetto è eliminato quando use_count va a zero
  - Ovvero l'ultimo shared_ptr sull'oggetto esce di scope
- Per la sua creazione è preferito l’uso di **make_shared**()
  - Template function factory

weak_ptr

- Accesso temporaneo all’oggetto posseduto da uno shared_ptr
  - Usati anche per rompere dipendenze cicliche su shared_ptr
- Può essere creato vuoto o legato a uno shared_ptr
- Per poter operare sull’oggetto si ottiene uno shared_ptr via lock()
  - Se la risorsa è stata nel frattempo rilasciata, lock() ritorna nullptr
- Se la risorsa associata è stata rilasciata, expired() ritorna true