Programmazione di sistema

Anno accademico 2017-2018

Esercitazione 2

Per evitare le ambiguità associate all'uso di puntatori nativi, il linguaggio c++ offre il concetto di smart_pointer: oggetti che incapsulano l'indirizzo a cui si punta, che possono essere usati sintatticamente come normali puntatori (tramite gli operatori * e ->) ma che incarnano una particolare semantica in caso di copia e/o movimento.
Includendo il file <memory>, vengono definite le classi (generiche) std::shared_ptr<T> e std::unique_ptr<T>, dove T è il tipo di dato a cui si punta.
La prima classe permette di gestire un puntatore ad una zona di memoria dinamica (allocata tramite la funzione std::make_shared<T>(...) e condivisa in caso di copia) che sarà rilasciata automaticamente quando tutti i suoi riferimenti saranno stati distrutti. La seconda classe, invece, rappresenta un puntatore non copiabile (ma spostabile per movimento) che possiede il riferimento al proprio blocco. Tale riferimento può essere costruito tramite la funzione std::make_unique<T>(...) e viene rilasciato all'atto della distruzione dello smart_pointer.

Utilizzando queste astrazioni, si realizzi un programma che acquisisce, da tastiera, una serie di linee (si usi la funzione std::getline(...) definita nel file <string>) ciascuna delle quali è composta da due parti separate da ",": le due parti rappresentano i nomi di due persone legate da vincolo di amicizia.
La sequenza di ingresso termina quando viene letta una riga vuota.
A questo punto il programma dovrà determinare, per ogni nome, l'elenco degli amici (si consideri l'uso della classe std::set<T> definita nel file <set> e quello della classe std::map<K,V> definita nel file <map>).
Si tenga conto che la relazione di amicizia è bidirezionale (la coppia a,b implica la coppia b,a) e che eventuali duplicati vanno rimossi.
Si faccia in modo di garantire che terminata la lettura di una data sequenza di coppie tutta la memoria acquisita sia liberata, così da poter procedere ad una successiva analisi, senza dover terminare il processo.
Eventuali eccezioni dovranno garantire il corretto rilascio delle risorse acquisite e riportare il programma ad uno stato stabile.

Competenze da acquisire

- Gestione della memoria e smart pointer
- Gestione delle eccezioni
- Uso delle librerie standard (set, string, …)