Programmazione I
Prova di Programmazione – 11 luglio 2016 – 2 ore

Partendo dal frammento di codice fornito, realizzare un programma per la gestione di un insieme di *blocchi* di memoria. Un *blocco* è una sequenza di byte contigui, a cui è associato un *nome*. Il nome è definito a sua volta da una parola (sequenza di caratteri senza spazi bianchi). Tutti i blocchi hanno la stessa dimensione, definita a tempo di scrittura del programma. Anche la dimensione massima dei nomi dei blocchi, nonché il numero massimo di blocchi disponibili sono definiti a tempo di scrittura del programma. Se un blocco è correntemente in uso da parte di almeno un utente, allora il suo stato è *occupato*, altrimenti lo stato è *libero*. La sequenza di byte di un blocco deve occupare spazio in memoria solo mentre il blocco è occupato. Al contrario non è richiesto che la struttura dati contenente i blocchi occupi sempre meno memoria possibile. All'avvio del programma tutti i blocchi sono *liberi*. Il programma deve fornire le seguenti funzionalità.

1. **ottieni_blocco(nome)** Se non vi è nessun blocco occupato con nome **nome**, allora prende un blocco libero e lo inizializza con tale nome. Fa passare il blocco dallo stato libero allo stato occupato. Ritorna infine l'indirizzo della sequenza di byte del blocco. Se invece c'è un blocco occupato con nome **nome**, memorizza in qualche modo l'informazione che il blocco è in uso da parte di un ulteriore utente, e ritorna direttamente l'indirizzo della sequenza di byte del blocco.
2. **stampa_stato** Per ciascun blocco occupato, stampa il nome del blocco, il numero di utenti che lo stanno utilizzando, e l'indirizzo della sequenza di byte del blocco (per stampare l'indirizzo contenuto in un puntatore a carattere tramite l'operatore di uscita, è necessario convertirlo ad un tipo opportuno, ad esempio mediante **`reinterpret_cast<void *>(puntatore)`**). Ad esempio
   `blocco-A 3 0x2125010`
   `blocco-D 1 0x21250a0`
3. **rilascia_blocco(nome)** Se vi è un blocco occupato di nome **nome**, decrementa il numero di utenti che lo stanno utilizzando. Se il numero raggiunge lo zero, allora fa passare il blocco dallo stato occupato allo stato libero.
4. **salva_stato** Salva in un file di testo, dal nome definito a tempo di scrittura del programma, lo stato dei blocchi, tranne il contenuto delle sequenze di byte.
5. **carica_stato** Carica lo stato dei blocchi dal file, senza preoccuparsi di quale sarà il contenuto delle sequenze di byte dei blocchi occupati. Lo stato precedente è perso.
6. **rilascia_blocco2(nome)** Effettua le stesse operazioni della funzionalità rilascia_blocco, ma, se necessario, modifica anche la struttura dati contenente i blocchi affinché nella funzionalità **ottieni_blocco** e nella funzionalità **salva_stato** non sia mai necessario scorrere anche dei blocchi liberi. Si ottiene il puntegigo massimo se la riorganizzazione della struttura dati è effettuata a costo costante rispetto al numero di blocchi disponibili. Se lo si ritiene opportuno in termini di efficienza e chiarezza del codice, si può realizzare solo questa funzionalità al posto della **rilascia_blocco**.

I parametri di ingresso delle funzionalità sono solo indicativi. Gestire opportunamente le situazioni di errore, tranne l'inserimento di dati in formato errato e di messaggi troppo lunghi da *stdin*.

REGOLE

- Si può utilizzare ogni genere di manuale o di materiale didattico di altra natura
- Per superare la prova, il programma deve essere perfettamente funzionante nelle parti 1 e 2. Il voto ottenuto in questo caso è 18.
- Ciascuna funzionalità DEVE essere implementata mediante almeno una funzione.
- Il voto massimo (almeno 30) si ottiene se
  a) il programma è perfettamente funzionante in ogni sua parte
  b) tutti i principi di ingegneria del codice visti nel corso sono stati applicati