Esercitazioni di Informatica B

Riepilogo C

Stefano Cereda
stefano.cereda@polimi.it

27/10/2020

Politecnico Milano

TODO

Finire slide lezione >= 25

Un'organizzazione che si occupa di archeologia sottomarina vuole sviluppare un software in linguaggio C che permetta di catalogare tutti i reperti archeologici che sono stati scoperti. Ogni reperto è caratterizzato dalle seguenti informazioni:

- coordinate geografiche in cui si trova
- profondità del reperto rispetto al livello del mare
- peso del reperto
- tipo di reperto, questo può essere: manufatto in terracotta, manufatto in marmo, manufatto in metallo

Tutti i reperti vengono memorizzati in un array di dimensione massima 1000.

Archivio reperti da TdE 26 Giugno 2017 - vedere pdf esame II

1. Si definiscano in linguaggio C le strutture dati e le variabili necessarie per rappresentare l'insieme dei reperti nella variabile *insiemeReperti*.
2. Si supponga che la variabile *insiemeReperti* sia stata precedentemente riempita con le informazioni di 1000 reperti e si sviluppi in C il ciclo che, per ogni reperto, stampa a video il suo peso. Si dichiarino anche tutte le variabili necessarie allo scopo che non sono state dichiarate nel punto precedente. Periodicamente, l'organizzazione progetta spedizioni per raccogliere i reperti scoperti. L'operazione non è sempre possibile perché il veicolo che viene usato per raccogliere i reperti ha una portata massima (un peso massimo che può portare) che non può essere superato. D'altra parte, l'organizzazione vuole fare in modo che il veicolo venga utilizzato nel modo più efficiente

possibile e quindi la spedizione verrà pianificata in modo tale che il veicolo raccolga il maggior numero possibile di reperti nell'area prescelta. La spedizione viene pianificata come segue:
Si parte con un veicolo scarico, di portata *pMax*, da una coppia di coordinate geografiche (*latitudineI* e *longitudineI*) e si esplorano le vicinanze di quelle coordinate geografiche in modo da rimanere nell'ambito della zona di ricerca, i cui confini sono definiti dai punti $latitudineI \pm soglia$, $longitudineI \pm soglia$. Per ogni reperto che si trova nei confini definiti, si pianifica di caricare il reperto a bordo del veicolo, a patto che il suo peso, sommato al peso del carico corrente portato dal veicolo, sia minore di *pMax*. In caso contrario, quel reperto viene scartato e si procede a cercarne un

altro, sempre rimanendo nei confini dell'area di ricerca. La ricerca termina quando il peso dei reperti da trasportare avrà raggiunto il valore *pMax* oppure quando tutti i reperti nella zona di ricerca saranno stati esaminati. L'algoritmo di pianificazione del viaggio, durante l'analisi sopra descritta, crea un array di reperti chiamato *caricoVeicolo* che contiene, alla fine dell'analisi, tutti i reperti che dovranno essere caricati sul veicolo durante l'esecuzione della spedizione.

3. Dato un reperto che assumiamo si trovi nella cella *i*-esima (con *i* che ha un valore qualsiasi compreso tra 0 e 999, estremi inclusi) dell'array *insiemeReperti*, si scriva la condizione che, se vera, ci permette di dire che il reperto si trova nell'area di ricerca definita dai confini indicati sopra.

4. Si sviluppi il frammento di codice in C necessario per acquisire da tastiera: la portata massima del veicolo, le coordinate geografiche di partenza della spedizione, il valore soglia che definisce i confini di ricerca. Si dichiarino tutte le variabili necessarie allo scopo.
5. Si sviluppi in linguaggio C l'algoritmo di pianificazione del carico descritto in precedenza. Oltre alle dovute inizializzazioni di variabili (non è necessario inizializzare *insiemeReperti* che si assume essere già riempito), la parte principale di questo algoritmo sarà costituita a un ciclo che analizza tutti i reperti in *insiemeReperti* e, per ogni reperto:
   - Valuta se si trova nell'area di ricerca (si faccia riferimento alla soluzione fornita per il punto 3)
   - In caso positivo, valuta se può essere caricato sul veicolo

- In caso positivo, aggiorna la variabile *caricoVeicolo* in modo che contenga i dati del reperto in questione

Si dichiarino anche tutte le variabili necessarie allo scopo che non sono state dichiarate nei punti precedenti.

Ordinamento a bolle

Si scriva un programma che riceva in ingresso un vettore di numeri interi. Il programma deve ordinare il vettore usando l'algoritmo dell'ordinamento a bolle e stampare il risultato.

Un cardiopatico ha uno smartwatch con 48 byte di memoria liberi e vorrebbe tenere in memoria l'ECG relativo alla sua attività cardiaca durante la notte. Il singolo dato è rappresentato dalla differenza di potenziale generata dal battito cardiaco (in Volt) ogni ora, e può essere un intero in intervallo [-1000 +2000] (estremi compresi). Si assuma che lo smartwatch utilizzi una codifica in Complemento a 2 e si risponda alle seguenti domande:

1. Quale è il numero B di bit necessari per registrare ogni singolo dato corrispondente a un'ora di attività cardiaca?

2. Quante ore è possibile registrare ancora nella memoria libera, supponendo che ciascuna cella di memoria possa contenere esattamente B bit? Quanta memoria è necessaria per tenere in memoria 128 ore?
3. Supponiamo ora che siano state registrate le differenze di potenziale generate dal battito cardiaco delle prime 2 ore della notte, e che queste siano [-135, -978]. Si codifichino questi numeri in Complemento a 2 con B bit e si calcoli la loro somma (in Complemento a 2), indicando se questa può essere registrata nello smartwatch, ovvero in B bit.

1. $2^{10} = 1024 < 2000 < 2048 = 2^{11}$ Con B = 12 bit si riesce a coprire l'intervallo $[-2^{11}, 2^{11} - 1]$ e quindi si riesce a registrare l'attività cardiaca di un'ora.
2. Essendo la memoria disponibile pari a 48 byte = 48 * 8 bit = 384 bit e poiché per ogni ora sono necessari 12 bit per tenere in memoria la rappresentazione della differenza di potenziale generata dal battito cardiaco, si possono tenere in memoria ancora al massimo 48 * 8 / 12 = 32 ore. Se si quadruplicasse la memoria (i.e. 48 * 4 = 192 byte) otterrei spazio sufficiente per 32 * 4 = 128 ore

3. Tramite l'algoritmo delle divisioni successive $135_{10} = 10000111_2$, quindi il numero decimale -135_{10} in codifica Complemento a 2 con B = 12 bit è $-135_{10} = 111101111001_{CP2}$. Dato che $978_{10} = 1111010010_2$, poichè -978 è negativo, devo cambiare il valore di tutti i bit ottenuti e sommare 1 per trasformare tale codifica in un numero negativo in Complemento a 2 con B = 12 bit, ovvero $-978_{10} = 110000101110_{CP2}$. La somma di tali due valori rimane nell'intervallo di valori ammissibili con 12 bit, dunque è possibile registrare la somma delle differenze di potenziale generate dal battito cardiaco delle prime 2 ore della notte nello smartwatch.

Esercizi non svolti

Anagrafe

Si vuole gestire l'anagrafe della motorizzazione e dei proprietari di auto. Un veicolo è identificato da:

- Targa
- Modello
- Proprietario (si suppone che ogni auto abbia un unico proprietario)

I proprietari di auto sono identificati dai campi:

- Cognome
- Nome
- Un insieme di indirizzi di recapito (max 3).

Un recapito è composto da: via, numero civico e CAP. L'anagrafe contiene al max N veicoli.

Anagrafe

Definire in C la struttura dati che permette di rappresentare la situazione descritta.

Scrivere il programma C che chiede all'utente di inserire i dati degli N veicoli da memorizzare nell'anagrafe. Per ogni veicolo il programma deve acquisire anche le informazioni relative al proprietario, inclusi i suoi recapiti. In una prima versione del programma si inseriscano esattamente tre recapiti per proprietario, in una seconda versione si consenta all'utente di inserire un numero di recapiti minore o uguale a tre.

Finita la fase di inserimento il programma stampa il contenuto della struttura nel formato:
VEICOLO Targa XXXXX
PROPRIETARIO Cognome XXXXXXXXX
RECAPITI CAP XXXXXXXXX ... XXXXXXXXX

Si consideri il sistema di gestione dei treni in una stazione. In ogni stazione ci sono al più *N_BANCHINE* e ogni banchina consente la sosta e il transito di un treno. Ad ogni treno in stazione viene assegnato uno dei seguenti stati:

- *fuoriStazione*: il treno, in attesa di essere assegnato a una banchina, si trova fuori dalla stazione
- *inIngresso*: il treno è stato assegnato a una banchina ed è in marcia a velocità ridotta per entrare in stazione
- *inSosta*: il treno è fermo a una banchina
- *attesaOut*: il treno è fermo a una banchina in attesa di poter partire

- *inUscita*: il treno sta abbandonando la banchina procedendo a velocità ridotta.

Si assuma che siano già stati introdotti i tipi *Stazione* e *Banchina* (si veda codice qui sotto). In particolare, *Stazione* contiene l'insieme e il numero delle banchine presenti in stazione (*banchine*), la coda dei treni (al più 20) in attesa all'ingresso (*codaTreni*, *nTreniInCoda*) e una variabile Booleana che dice se c'e' qualche treno in manovra in stazione (*bloccata*). *Banchina*, invece, contiene il numero della banchina e i dati del treno in sosta, se la banchina è in stato *occupato*. Ciascun treno, oltre ad avere il proprio stato e il numero della banchina eventualmente assegnata, ha un campo *minutiAttesaOut* che indica da quanti minuti è in attesa in stato *attesaOut*.

Stazione ferroviaria - TdE 29 Gennaio 2018 III

```
1   #define N_BANCHINE 10
2   #define N_TRENI_CODA 20
3
4   typedef char stringa[20];
5   typedef enum { falso, vero } boolean;
6   typedef enum { libero, occupato } statoBanchina;
7   typedef enum
8   {fuoriStazione, inIngresso, inSosta, attesaOut, inUscita}
    statoTreno;
9
10  typedef struct {
11      stringa nome;
12      statoTreno stTreno;
```

```
13          int nBanchinaAssegnata;
14          int minutiAttesaOut;
15      } Treno;
16
17      typedef struct {
18          int numero;
19          statoBanchina stBanchina;
20          Treno trenoSosta;
21      } Banchina;
22
23      typedef struct {
24          Banchina banchine[N_BANCHINE];
```

```
25        int nBanchine; /* indica il numero di banchine presenti
      nella stazione */
26        Treno codaTreni[N_TRENI_CODA];
27        int nTreniInCoda;
28        boolean bloccata;
29     } Stazione;
30
```

Si supponga che sia stata definita la variabile *stazioneMI* di tipo *Stazione* e che questa variabile sia stata riempita in modo opportuno per rappresentare lo stato corrente della stazione di Milano, si risponda alle seguenti richieste:

- Dichiarando tutte le variabili necessarie, si definisca un frammento di programma che determina e stampa l'indice (nell'array di banchine) della banchina in cui è in sosta il treno che è in attesa da più tempo di poter partire.
- Si definisca un frammento di programma che, se la stazione non è bloccata dalle manovre di qualche treno e se vi è almeno un treno in coda per entrare in stazione, blocca la stazione, estrae dalla coda il primo treno che vi era stato inserito, gli assegna lo stato *inIngresso*, e aggiorna la coda in modo tale che poi contenga solo i rimanenti treni in coda. Se per esempio, prima di questa operazione la coda è cosi costituita: stazioneMI.codaTreni[0] = FR2092, stazioneMI.codaTreni[1] = IR097, stazioneMI.codaTreni[2] = FA2673 (per

semplicità abbiamo riportato solo il nome del treno, ma, ovviamente, codaTreni dovrebbe contenere l'intera struct corrispondente), dopo l'operazione si deve avere che stazioneMI.codaTreni[0] = IR097, stazioneMI.codaTreni[1] = FA2673.

Si assuma di avere un sistema operativo con un quanto di tempo di 25 ms e un processo P1 che riesce a decodificare 80 byte di segnale audio per ogni millisecondo di esecuzione.

Supponendo che una corretta riproduzione dell'audio richieda che vengano decodificati almeno 128 kbit/s, si risponda alle seguenti domande **giustificando le risposte**:

1. Se il processo P1 è l'unico attivo, può rispettare la velocità di codifica richiesta?
2. Supponiamo di attivare contemporaneamente a P1 altri 3 processi che abbiano lo stesso quanto di tempo e la stessa priorità di P1 e che ciascun cambio di contesto fra processi implichi un ritardo di 1 ms. Il processo P1 riuscirà a rispettare il limite richiesto per la corretta riproduzione?

1. Se P1 è l'unico attivo, avrà l'intera CPU per se, quindi potrà decodificare 80 * 8 * 1000 bit/s = 640 kbit/s; quindi può rispettare la velocità di codifica richiesta.
2. Se sono attivi altri 3 processi, P1 non avrà l'intero tempo a disposizione, ma solo una parte di esso. In ogni secondo ci sono 1000 ms/(25+1) ms = 38 quanti di tempo circa che devono essere divisi tra i quattro processi. Se tutti hanno la stessa priorità e proseguono la loro esecuzione senza attendere dati dalle periferiche, ogni processo ha a disposizione 38/4=9 quanti di tempo circa. Quindi il nostro processo in un secondo decodificherà: 80 * 8 * 9 *25 bit/s = 144000 bit/s = 144 kbit/s, una quantità maggiore di quella minima richiesta. Di conseguenza, il nostro processo riuscirà a riprodurre correttamente l'audio.