Politecnico di Milano - Dipartimento di Elettronica e informazione

Prof. Mauro Negri

Fondamenti di Informatica 1 febbraio 2013

II prova in itinere

Matricola _____ Cognome_____ Nome_____

Durata prova: 2 ore

Istruzioni

LE RISPOSTE DEVONO ESSERE SCRITTE SU QUESTO PLICO (anche sul retro dei fogli)

Non separare questi fogli. Si può utilizzare la matita.

L'uso di cellulari, libri, eserciziari, appunti o calcolatrici durante lo svolgimento della prova comporta l'annullamento della prova.

Non è possibile uscire dall'aula durante la prova tranne in casi eccezionali.

Esercizio 1 (6 punti) _____

Esercizio 2.a (9 punti) _____

Esercizio 2.b (9 punti) _____

Esercizio 3 (6 punti) _____

Punteggio totale (30 punti) _____

Esercizio 1. Scrivere un programma C di nome multiplo che chiede al terminale due numeri e li memorizza nelle variabili chiamate N e base (i numeri sono accettati se >0, altrimenti vengono richiesti). Il programma conta quante cifre vanno rimosse da N (a partire da destra) prima di generare un numero che sia multiplo di base. Non appena il programma trova il multiplo visualizza al terminale il numero di cifre rimosse da n prima di generare il multiplo. Inoltre se il numero N ricevuto è un multiplo di base il programma visualizza 0, mentre nel caso in cui non sia stato trovato alcun multiplo di base dall'algoritmo il programma visualizza il numero totale di cifre rimosse.

Esempi

N	base	risultato	motivazione
12333	2	3	12333, 1233 e 123 non sono multipli di 2, ma 12 sì.
12300	2	0	12300 è multiplo di 2
1333	2	4	1333, 133, 13, 1 non sono multipli di 2

```
#include <stdio.h>
int main()
 {int N, base,rimosse=0;
    do
      { printf("introdurre N: "); scanf("%d",&N); printf("introdurre base: ");
        scanf("%d",&base);
      }
   while((N<=0)||(base<=0));

  while ((N!=0) && (N%base !=0)) {N=N/10; rimosse++;}//N!=0 non necessario
  printf("rimosse =%d\n",rimosse);
}
```

Esercizio 2. Si ipotizzi di realizzare una gestione ordini di un'azienda. Siano date le seguenti definizioni di due liste monodirezionali

```
#DEFINE MaxRighe 30;
typedef  struct  { int codice; int qty; int prezzo; } t_rigaO;
struct ordine { int id; t_rigaO V[MaxRighe]; int Nrighe; struct ordine * next; };
struct ordine *listaOrdini=NULL;
struct prodotto { int codice; int totqty; struct prodotto * next; };
struct prodotto *listaProdottiOrdinati=NULL;
```

dove:

- ListaOrdini memorizza gli ordini registrati nel sistema. Un ordine è descritto da un identificatore dell'ordine (id); l'id viene assegnato dal programma all'atto della memorizzazione dell'ordine nella lista – intero progressivo che parte da 1 per il primo ordine memorizzato. L'ordine contiene al massimo 30 righe d'ordine e Nrighe (da 1 a 30) indica quante righe sono le righe realmente presenti nell'ordine (si supponga che non esista mai un ordine vuoto). In ogni riga si memorizzano i dati di acquisto di ogni singolo prodotto, memorizzando il codice del prodotto acquistato (deve essere un valore tra 1 e 150), la quantità acquistata (deve essere maggiore di 0) e il prezzo unitario (deve essere maggiore di 0).
- ListaProdottiOrdinati che memorizza un elemento per ogni prodotto che è stato acquistato almeno in un ordine e per ogni prodotto riporta il suo codice, e la somma delle quantità acquistate complessivamente da tutti gli ordini esistenti nell'altra lista. Se un prodotto offerto dall'azienda non è acquistato da nessun ordine presente nella lista ListaOrdini allora non deve esistere un elemento per tale prodotto nella lista (non esiste quindi un prodotto con totqty=0). Un elemento è creato solo quando viene generato il primo ordine in cui lo si compra.

Realizzare le seguenti funzioni C:

Esercizio 2.a. Funzione **InsertOrdine**. Riceve come parametri i dati di un ordine (ossia i dati delle righe dell'ordine) e le liste necessarie all'operazione. La funzione controlla che l'ordine contenga dati corretti rispetto a quelli imposti nella definizione delle caratteristiche di ListaOrdini; se non sono accettabili la funzione rifiuta l'inserimento dell'ordine. Poi determina il valore di id da assegnare all'ordine (se in un precedente inserimento era stato assegnato l'id 200, ad esempio, si deve generare l'id 201) e crea l'elemento dinamico da caricare con i dati ricevuti dell'ordine; l'elemento viene quindi caricato nella ListaOrdini. Successivamente per ogni prodotto acquistato (in una riga d'ordine) lo si deve individuare nella listaProdottiAcquistati al fine di incrementare totqty con la quantità ordinata nell'ordine che si sta inserendo. Se la funzione è riuscita ad eseguire tutte le operazioni ritorna le liste aggiornate e il valore di ritorno 1, altrimenti ritorna 0.

Esercizio 2.b. Funzione **Cancellaordine** Riceve come parametri l'identificatore id di un ordine e le liste necessarie all'operazione. Ricerca l'ordine e lo elimina da ListaOrdini e poi per ogni prodotto presente nell'ordine cerca l'elemento corrispondente in ListaProdottiAcquistati e decrementa opportunamente il valore dell'attributo totqty. Infine ritorna le liste modificate e il valore 1 se l'operazione ha avuto successo (l'ordine è stato eliminato) e 0 in tutti gli altri casi.

Soluzione esercizio 2.a

```
// Definizioni date
#include <stdio.h>
#include <stdlib.h>
#define MaxRighe 30
typedef  struct  { int codice; int qty; int prezzo; } t_rigaO;
struct ordine { int id; t_rigaO V[MaxRighe]; int Nrighe; struct ordine *next; };
struct ordine *listaOrdini=NULL;
struct prodotto { int codice; int totqty; struct prodotto * next; };
struct prodotto *listaProdottiOrdinati=NULL;

// variabile globale usata per la generazione dell'identificatore progressivo
dell'ordine.

int idOrdine=1;


int insertordine (t_rigaO vet[], int quanti, struct ordine **o,
                  struct prodotto **p)
{// quanti indica quanti elementi di vet sono utili
 int i, trovato;  struct ordine *temp; struct prodotto *scans, *tempP;

 //controlli preliminary sui parametrici
  if ( (quanti <1)|| (quanti >30) ) return 0;
 for (i=0; i<quanti;i++)
    if ( (vet[i].codice <1) || (vet[i].codice > 150) ||
         (vet[i].qty<=0) || (vet[i].prezzo<=0)) return 0;

/* schema seguito: creazione preliminare di tutti gli elementi dinamici
necessari (ordine ed eventuali elementi prodotto non già presenti in
listaProdottiOrdinati. Se una sola creazione fallisce allora si eliminano tutti
gli elementi creati precedentemente (rollback sections). Solo dopo le creazioni
si procede col caricamento*/

//AREA CREAZIONI
//creazione ordine
temp=(struct ordine *) malloc(sizeof(struct ordine));
if (temp==NULL)return 0; //rollback section

// eventuale creazione di prodotti in listaProdottiOrdinati
for (i=0; i<quanti;i++)
 {trovato=0; scans= *p;
   while((scans!=NULL) && (!trovato) )
         if (scans->codice == vet[i].codice) trovato=1; else scans=scans->next;

   if (!trovato)
    {tempP=(struct prodotto *) malloc(sizeof(struct prodotto));
     if (tempP==NULL)
        //rollback section
            { free(temp);
              tempP=*p;




             while (tempP->totqty==0)

attenzione all'inizio con lista vuota
```

```
{*p=(*p)->next; free(tempP); tempP=*p;}
             return 0;
           }

     else /*si crea l'elemento per il prodotto non pre-esistente e si carica il
           codice prodotto e totqty=0 per riconoscerlo nella rollback section
           se necessario e per il successivo incremento di quantità*/

       {tempP->codice=vet[i].codice; tempP->totqty=0;tempP->next=*p; *p=tempP;}
    }//end if !trovato
}// end for

//CARICAMENTO VALORI
//caricamento e inserimento ordine nella lista ordini in testa
temp->id = idOrdine; idOrdine ++;
for (i=0; i<quanti;i++)
    {temp->V[i].codice=vet[i].codice; temp->V[i].qty=vet[i].qty;
     temp->V[i].prezzo=vet[i].prezzo;
    }
temp->Nrighe= quanti;   temp->next= *o; *o=temp;


// incremento delle totqty di ogni prodotto dell'ordine
for (i=0; i<quanti;i++)
   {trovato=0; scans= *p;
    while((scans!=NULL) && (!trovato) )
     if (scans->codice == vet[i].codice)
       {trovato=1; scans->totqty = scans->totqty+ vet[i].qty; }
     else {scans=scans->next;}
   }
return 1;
}
```

Soluzione esercizio 2.b

```
int Cancellaordine (int id, struct ordine **o,
                        struct prodotto **p)
{ struct ordine *current, *prec;int trovato=0, i;
  struct prodotto *scans, *pscans;

   current= *o; prec=NULL;

//estrazione ordine dalla lista
   while((current != NULL) && (!trovato))
    {   if (current->id == id)
            if (current == *o) //testa
                 {*o = current->next;  trovato=1;}
          else //in mezzo o coda
                 {prec->next = current->next;   trovato=1;}
        else  {prec = current; current = current->next;}
    }
if (!trovato) {printf("ordine inesistente");return 0;}
```

```
// ricerca prodotto e decremento totqty. Se totqty diventa 0 sua
estrazione ed eliminazione dalla lista

for (i=0; i<current->Nrighe;i++)
 {trovato=0; scans= *p; pscans=NULL;
  while((scans!=NULL) && (!trovato) )
       if (scans->codice == current->V[i].codice)
          {trovato=1;
           scans->totqty=scans->totqty - current->V[i].qty;
           if (scans->totqty==0)
                if (scans == *p) //testa
                          *p = scans->next;
                 else   //in mezzo o coda
                    {pscans->next = scans->next;free(scans);}
           else {pscans=scans; scans=scans->next; }
          }
      else {pscans=scans; scans=scans->next;  }
  }
free(current);
return 1;
}

//Le seguenti parti servono solo per provare l'esecuzione del programma.

//funzione di supporto per visualizzare il contenuto delle due liste
void show (struct prodotto *L, struct ordine *O)
{ int i;
  printf("\nSlista ordini");
  while (O!=NULL)
   { printf("\n id=%d", O->id);
     for (i=0; i < (O->Nrighe);i++)
       printf("\n codice=%d; qty=%d;prezzo=%d", O->V[i].codice,O->V[i].qty,
              O->V[i].prezzo);
     O= O->next;
   }//end while
  printf("\nSlista prodotti");
  while (L!=NULL)
        {printf("\n codice=%d; totqty=%d", L->codice, L->totqty); L= L->next;}
 }

//main non richiesto e inserito solo per una prova di esecuzione
int main()
{int N=2;t_rigaO VV[MaxRighe]; int id;

VV[0].codice=101; VV[0].qty=100; VV[0].prezzo=10;
VV[1].codice=102; VV[1].qty=200; VV[1].prezzo=20;

insertordine (VV, N, &listaOrdini, &listaProdottiOrdinati);
show (listaProdottiOrdinati, listaOrdini);

VV[0].codice=102; VV[0].qty=101; VV[0].prezzo=11;
VV[1].codice=103; VV[1].qty=201; VV[1].prezzo=21;
insertordine (VV, N, &listaOrdini, &listaProdottiOrdinati);
show ( listaProdottiOrdinati, listaOrdini);

printf("\nid: "); scanf("%d",&id);
printf("\\nesitocancella: %d", Cancellaordine (id, &listaOrdini,
&listaProdottiOrdinati));
show ( listaProdottiOrdinati, listaOrdini);
}
```

Esercizio 3. Si scriva un programma C che definisce una variabile globale A inizializzata a 1. Poi chiede al terminale se l'utente vuole proseguire? In caso positivo:

1. crea un processo figlio;
2. incrementa invece di 1 il valore della propria variabile A e richiede nuovamente al terminale se l'utente vuole proseguire.

In caso affermativo il programma nel padre ripete le operazioni precedenti 1. e 2. Quando l'utente risponde per la prima volta che non intende proseguire nell'esecuzione, allora il programma nel processo padre si mette in wait per ognuno dei figli che ha generato e procede a visualizzare al terminale il valore ricevuto da tutti i figli (si è liberi di stampare il valore dopo che il programma ha ricevuto tutti i valori o dopo la ricezione del singolo valore) prima di terminare l'esecuzione. Il primo processo figlio creato termina la propria esecuzione ritornando il valore 1, mentre tutti gli altri terminano la loro esecuzione restituendo il valore di A+100.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{int A=1,i, status, finito=0; pid_t p;
 while (!finito)
     {printf("\nhai finito (1=si', 0=no)?");
      scanf("%d",&finito);
      if (!finito)
           {p=fork();
            if (p==-1)exit(0);//fork fallita
            else
              if (p==0) //figlio
                   if (A==1) exit(A); else exit(A+100);
              else A++;//padre
           }
      else //finita creazione figli
         { for (i=1;i<A;i++)
             {p=wait(&status);
              printf("\nvalore ricevuto%d\n",status/256);
             }
           finito=1;
         }
      }
}
```