Politecnico di Milano - Dipartimento di Elettronica e informazione

Prof. Mauro Negri

Fondamenti di Informatica **6 settembre 2013**

III appello

Matricola_____ Cognome _____ Nome_____

Durata prova: 1h 30'

Istruzioni

LE RISPOSTE DEVONO ESSERE SCRITTE preferibilmente 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 (13 punti) _____

Esercizio 2 (10 punti) _____

Esercizio 3 (7 punti) _____

Punteggio totale (30 punti) _____

Esercizio 1.
Scrivere una funzione C chiamata Sposta che riceve come parametro la lista LIST così definita

struct elem {char str[21]; struct elem *next;}; struct elem *LIST=NULL;

La lista conterrà un numero arbitrario non conosciuto di elementi e dopo l'esecuzione dovrà essere ritornata, eventualmente modificata, al programma chiamante come parametro.
La funzione legge da terminale una stringa carattere per carattere e termina la lettura quando legge il carattere '\n'; la stringa letta deve essere caricata nella variabile char TEST[21] e se la dimensione della stringa è superiore a quella della variabile essa deve essere troncata opportunamente.
La funzione poi cerca tutti gli elementi della lista che hanno nel campo str il valore contenuto nella stringa TEST e li sposta in fondo alla lista. Infine la funziona ritorna, sempre come parametro, il numero di elementi che sono stati spostati.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
struct elem {char str[20];struct elem *next;};

//funzione non richiesta dall'esercizio - vedi main
struct elem * instesta(struct elem * l, char txt[21]) {
     struct elem * tmp;
     tmp = (struct elem *) malloc(sizeof (struct elem));
     if (tmp == NULL) {printf("Memoria esaurita\n"); return l;  }
     tmp -> next = l; strcpy(tmp->str, txt);  return tmp;
}
//funzione non richiesta dall'esercizio
void stampa(struct elem * l) {
     while (l != NULL) {    printf("%s -> ", l->str);  l = l->next;}
     printf("NULL\n");
}

struct elem * Sposta(struct elem * l, int *spostati) {
     char TEST[21], c; int i, fine;
     struct elem *ultimo = NULL, *tmp = NULL, *coda = NULL, *prec = NULL;

          //se NULL o un solo elemento non si deve spostare nulla
       *spostati = 0;
     if (l == NULL || l -> next == NULL) {    printf("Lista identica"); return l;
     }

          //caricamento stringa
     fine = 0; i=0;
     do {printf("Carattere (invio per terminare): ");
            scanf("%c", &c);
            if (c != '\n') {TEST[i] = c; i++; scanf("%c", &c);  }
            else fine = 1;
            }
     while (i < 21 && !fine);
     TEST[i] = '\0';

          //Ricerca e spostamento

/*posizionamento sull'ultimo elemento della lista iniziale per determinare dove
fermare la scansione. Alternative: contare gli elementi esistenti o gli elementi
esistenti con stringa uguale a quella di test.*/
     ultimo = l;
     while(ultimo -> next != NULL) { ultimo = ultimo -> next; }

/*coda indica l'ultimo elemento della lista durante l'elaborazione; all'inizio
la coda coincide con l'ultimo elemento, ma poi punterà all'ultimo spostato*/
     coda = ultimo;
```

```
/*tmp serve per scandire la coda dal primo elemento sino al penultimo della
lista originale. L'ultimo non è considerato perché comunque non cambierebbe
posizione; nessun problema se considerato*/
       tmp = l;
       while(tmp != ultimo)
         { if (strcmp(tmp->str, TEST) == 0)
            { //elemento intercettato viene spostato logicamente  in coda
              coda -> next = tmp;
              coda = coda -> next; //tmp diventa il nuovo ultimo

        //tmp passa al prossimo. Precedente rimane fermo.
              tmp = tmp -> next;

/*l'elemento individuato va eliminato dalla precedente posizione nella lista. Si
hanno due casi: l'elemento era il primo è il primo della lista e quindi va
cambiata la testa di lista oppure è in mezzo e quindi il precedente che puntava
a lui deve ora puntare al prossimo di quello estratto (attualmente puntato da
tmp)*/

             if (prec == NULL)      l = l -> next;    //prec rimane NULL
            else  prec -> next = tmp;   //prec rimane sull'elemento corrente

//sistemazione campo next dell'elemento spostato in coda
             coda -> next = NULL;     (*spostati)++;
            }//fine spostamento
         else //se elemento non è quello cercato passa al prossimo
             {prec = tmp; tmp = tmp -> next;}
       }//end while
     return l;
}

//funzione non richiesta
int main() {
     struct elem *LIST = NULL;  int spostati = 0;
     LIST = instesta(LIST, "el1");   LIST = instesta(LIST, "el2"); LIST =
instesta(LIST, "el3");
             LIST = instesta(LIST, "el4");

     LIST = Sposta(LIST, &spostati); stampa(LIST); printf("\nSpostamenti: %d\n",
spostati);
}
```

Esercizio 2.

Definire una matrice globale di 20 righe e 10 colonne di interi e scrivere una funzione C chiamata INS che riceve come parametro la matrice e procede a caricarla riga per riga a partire dalla prima con numeri richiesti al terminale. Il numero letto da terminale è inserito nella matrice solo se non è uguale ad uno di quelli già inseriti in precedenza, altrimenti la funzione ne richiede un altro al terminale finchè l'utente non introduce un numero nuovo.

La funzione poi calcola la somma dei valori memorizzati in ogni colonna e determina quale sia la colonna con la somma più bassa. Infine la funzione ritorna al programma chiamante, come parametri, la matrice caricata e l'indice della colonna con la somma più bassa.

Soluzione

```
#include <stdio.h>
#define ROWS 20
#define COLS 10

//funzione non richiesta
void stampa(int mat[][COLS]) {  int i, j;
    for (i = 0; i < ROWS; i++)
     {for (j = 0; j < COLS; j++) printf("%d ", mat[i][j]); printf("\n"); }}

int INS (int mat[ROWS][COLS]) {
     int i, j, num, x, y, duplicato, somma, somma_min, j_min;

printf("Inserimento dati nella matrice\n");
for (i = 0; i < ROWS; i++) {
   for (j = 0; j < COLS; j++) //per ogni elemento della matrice

//ciclo di verifica dell'univocità
      do {printf("[%d, %d]: ", i, j);    scanf("%d", &num);
        duplicato = 0;
        for (x = 0; x <= i && !duplicato; x++)
          for (y = 0; (x != i || y < j) && y < COLS && !duplicato; y++)
              if (mat[x][y] == num) duplicato = 1;

        if (!duplicato) mat[i][j] = num;
         else printf("Numero %d non valido... inserirne un altro\n", num);
        } while(duplicato == 1);
     }//fine primo for
 /* il controllo deve avvenire su tutti gli elementi delle righe precedenti
quella nella quale stiamo caricando il dato e per quella corrente ci si deve
invece fermare all'elemento precedente a quello corrente. La condizione di
controllo dei due for risolve questo problema. Tuttavia è accettabile anche una
soluzione che scandisce completamente, con un doppio for, le righe precedenti
alla riga corrente e infine un for che scandisce la parte necessaria della riga
corrente*/

stampa(mat); //non richiesta

/*calcolo la somma della prima colonna e inizializzo il minimo di riferimento.
Inizializzare la somma minima ad un valore come zero è critico perché le somme
potrebbero essere tutte di valore superiore a zero.
somma_min = 0;
for (i = 0; i < ROWS; i++) somma_min = somma_min + mat[i][0];
j_min=0;

for (j = 1; j < COLS; j++) {
     somma = 0;
     for (i = 0; i < ROWS; i++) somma = somma + mat[i][j];
          if ( somma_min > somma) { j_min = j;     somma_min = somma;}
     }
```

```
return j_min;

}
//non richiesta
int main() {int mat[ROWS][COLS]; INS(mat); }
```

Esercizio 3. Rispondere alle seguenti domande, motivando brevemente le risposte alle domande 1, 4 e 5 in modo convincente .

1. #define a 30 è usata per definire una costante al posto di int a=30; le due definizioni hanno un impatto diverso sul programma sebbene "a" possa essere usata come costante in entrambi i casi. Indicare come si comporta il compilatore nei due casi.

   #define viene interpretata dal precompilatore che sostituisce nel programma ogni occorrenza del simbolo a con il valore 30 e quindi il compilatore non vede più tale simbolo. int a=30 viene invece considerata dal compilatore che alloca spazio per una variabile intera che pertanto potrebbe comunque cambiare valore durante l'esecuzione

2. rappresentare la seguente sequenza di bit 1111.1100.1001.1111 in decimale sapendo che è stata codificata in complemento a 2

   -865

3. Data la definizione della variabile double v= 0.52 descrivere la sua rappresentazione in virgola mobile in base allo standard IEEE754 semplice precisione FP32 descritto a lezione;
   a) Definire prima il numero nel formato normalizzato intermedio (TRONCARE parte decimale alla terza cifra decimale):

      $1{,}04*2^{-1}$

   b) Codificare poi il numero normalizzato usando il formato FP32:
      S (1 bit di segno): 0
      M (23 bit per la mantissa – fermarsi ai primi 6 bit): 000010
      E (8 bit per l'esponente): 126 - 01111110

4. Un programma C genera un processo figlio il quale termina la propria esecuzione prima che il padre esegua l'istruzione di wait per attenderne la terminazione. Quali sono le operazioni eseguite dal sistema operativo quando sono eseguite le istruzioni exit() del figlio e wait(…) del padre?

   Il SO mette il processo figlio in stato "zombie" per indicare che è logicamente terminato e rimuove parte delle risorse allocate Il processo padre è in esecuzione nel frattempo).
   Quando poi il padre esegue la wait il SO verifica l'esistenza di un suo figlio in stato zombie, lo trova e quindi conclude la exit del figlio, trasferendo il valore al padre, poi elimina definitivamente il figlio e infine riattiva il processo padre che passa ad eseguire la prossima istruzione.

5. Perché si deve eseguire fopen(…) per accedere ai dati di un file, mentre non esiste un'operazione analoga per accedere ai dati di una variabile (ad es, int var;). Indicare quali operazioni siano eseguite dal sistema operativo all'atto dell'esecuzione della fopen e indicare a cosa serva il parametro di tipo FILE usato in tutte le operazioni di accesso al file?

   Le variabili sono definite dal programma e allocate dal compilatore (globali) o riconosciute nella dimensione per quelle locali, pertanto le globali sono immediatamente disponibili al programma e quelle locali lo saranno all'invocazione del blocco di appartenenza. I file dati viceversa sono oggetti

persistenti disponibili su disco e non agganciati automaticamente ad alcun programma. Per accedere ad un file è quindi necessaria l'operazione fopen che cerca il file e se esiste crea il descrittore del file in memoria (di tipo FILE) nel quale inserisce le informazioni da utilizzare per l'accesso ai dati del file da parte del programma. Per questo dopo l'apertura tutte le istruzioni di accesso al file inclusa l'operazione close richiedono il descrittore del file per l'accesso.