(Strutture e tipi utente -- Allocazione dinamica: gestione di liste)

ESERCIZIO n. 1

Si consideri il problema di rappresentare i dati di un programma semplificato di gestione anagrafica, le informazioni, registrate per ogni cittadino sono:

Cognome, Nome, Età, Sesso, Stato civile.

Si scriva un programma in linguaggio C che utilizzi un'opportuna struttura dati (allocata dinamicamente) per rappresentare le informazioni del problema.

Leggere da tastiera il numero di cittadini (n) con cui riempire l'elenco dell'anagrafe. Occuparsi dell'acquisizione dati e stampare a video (Cognome, Nome) di tutti i cittadini maggiorenni restituendo di questi ultimi anche l'età media e la percentuale di uomini e donne.

Esempio d'esecuzione:

Rossi Mario
Verdi Sara
....
eta' media: 42
Uomini 54%
Donne 36%

```
#include <iostream.h>
#include <stdlib.h>

#define DIM_STRING   30

struct tipo_anagraf {
  char cognome[DIM_STRING];
  char nome[DIM_STRING];
  int eta;
  char sesso;
  char stato_civile[DIM_STRING];
};

int main ( ) {

 tipo_anagraf *elenco; // puntatore usato successivamente per allocare
                       // dinamicamente un vettore di record anagrafici.
 int i, n, maggiorenni, uomini, donne, count = 0;
 float media, percent;


        /* acquisizione dati e memorizzazione nell'array di struct */
 do {
   cout << "\n Inserisci il numero totale di persone per l'anagrafe: ";
   cin >> n; cin.ignore();
 } while (n <=0);

 elenco = new tipo_anagraf[n];

 do {

   cout << "\nPersona ( "<< 1+count <<" ): ";
   cout << "\nCognome:    "; cin >> elenco[count].cognome; cin.ignore();
   cout << "\nNome:       "; cin >> elenco[count].nome; cin.ignore();

   do {
     cout << "\neta':     "; cin >> elenco[count].eta; cin.ignore();
   } while (elenco[count].eta < 0 && elenco[count].eta > 110);

   do {
     cout << "\nsesso(M/F): "; cin >> elenco[count].sesso; cin.ignore();
     if (elenco[count].sesso >='a') elenco[count].sesso -= ('a'-'A');
   } while (elenco[count].sesso != 'M' && elenco[count].sesso != 'F');

   cout << "\nstato civile: ";
   cin >> elenco[count].stato_civile;cin.ignore();

   count++;

 } while ( count < n );
```

```
 maggiorenni = 0; uomini = 0; media = 0; // elaborazione informazioni
 for (i = 0; i < n; i++){

   if (elenco[i].eta >= 18) {

     maggiorenni++;
     media = media + elenco[i].eta;
     cout << "\n\nCognome: “ <<  elenco[i].cognome;
     cout << “\nNome:” << elenco[i].nome;

     if (elenco[i].sesso == 'M')
       uomini += 1;
   }
 }

 media = media / maggiorenni;
 percent = ((float)uomini) / ((float)maggiorenni) * 100;

 cout << "\n\neta' media: “ << media;
 cout << “\nUomini : ” << percent << ” % ”;
 cout << “\nDonne: “ << 100-percent << ” % ”;

 system(“PAUSE”);
 return 0;
} // end main
```

Esercizio n. 2

Si implementi un sottoprogramma che riceve i seguenti parametri:

- una struct contenente un array di interi *a* e la sua dimensione *n;*
- un valore intero *b.*

Il sottoprogramma calcola il numero di occorrenze del valore intero *b* nell'array *a* e restituisce al chiamante il valore trovato.

Per esempio, se l'array contiene i seguenti valori: { 0, 2, 3, 0, 3, 5 } ed il valore intero *b* passato al sottoprogramma è 3, il risultato che il sottoprogramma dovrà produrre e passare al chiamante è: 2.

Soluzione

```
#include <iostream.h>
#define MAX_LEN 100


struct archivio
{
   int a[MAX_LEN];
   int n;
};


int conta(archivio arch, int b)
{
   int i, cont;
   cont = 0;
   for (i=0; i< arch.n; i++)
    if (arch.a[i] == b)
      cont++;

  return cont;
}
```

```
/* Sintassi alternativa in C non
in C++ */
struct vettore
{
   int a[MAX_LEN];
   int n;
}
typedef struct vettore archivio;
```

```
/* Sintassi alternativa in C non
in C++ */

typedef struct
{
  int a[MAX_LEN];
  int n;

} archivio;
```

```
/* Sintassi alternativa in C non
in C++ */

typedef struct vettore
{
   int a[MAX_LEN];
   int n;

} archivio;
```

ESERCIZIO n. 3

Scrivere un programma in linguaggio C che risolva il seguente problema.
Leggere dallo standard input una sequenza di informazioni relative a N pezzi meccanici.

Ogni pezzo è descritto dalle seguenti informazioni:

- Codice (alfanumerico con 5 caratteri)
- Quantità

Acquisire dallo standard input un elenco di pezzi lungo a piacere (allocando il vettore dinamicamente o staticamente).
L'elenco ricevuto può contenere delle ripetizioni dello stesso pezzo.
Si chiede di progettare un sottoprogramma che crei un secondo elenco che non contenga ripetizioni e che per ogni pezzo indichi la quantità complessiva.
Nel programma principale stampare a video il codice di ogni elemento e il numero di pezzi complessivo.

P.S.: Se necessario si faccia uso della funzione messa a disposizione nella libreria standard per confrontare due stringhe:

int strcmp(<stringa1>, <stringa2>);

includendo la libreria adatta con: #include <string.h>

Esempio:

dati in ingresso:	dati in uscita:
n = 4	
AX354 20	AX354 25
AW101 12	AW101 12
BY367 4	BY367 4
AX354 5	

```
#include <iostream.h>
#include <string.h>
#define DIM_ELENCO 100

struct pezzo {
       char codice[5];
       int quantita;
};
// Nel main allochero' degli array statici di lunghezza massima
// DIM_ELENCO dove ogni cella è di tipo: pezzo.
void eliminaDuplicati(pezzo vecchio[], int len_vecchio,
                      pezzo nuovo[], int& len_nuovo);

int main () {

     pezzo elenco[DIM_ELENCO];
     pezzo nuovo[DIM_ELENCO];
     int n_elenco,n_nuovo;
     int i;

     do {
          cout << "\ninserire il numero di pezzi, n = ";
          cin >> n; cin.ignore();
     } while ((n_elenco <= 0)||(n_elenco >= DIM_ELENCO));

     i = 0;
     do {
       cout << "\nInserire descrizione pezzo num. " << i+1;
       cout << "\nCodice:";
       cin >> elenco[i].codice; cin.ignore();
       do {
           cout << "\nQuantita':";
           cin >> elenco[i].quantita; cin.ignore();
       } while (elenco[i].quantita <= 0);
       i++;
     } while (i < n_elenco);

     // crea nuovo elenco privo di duplicati

     eliminaDuplicati(elenco, n_elenco, nuovo, n_nuovo);
     // n_nuovo contiene la lunghezza dell'array nuovo[]

     cout << "\nelenco senza doppioni: ";
     for (i=0; i < n_nuovo; i++) {
       cout << "\n\nCodice: %s",nuovo[i].codice);
       cout << "\nQuantita': %d",nuovo[i].quantita);
     } // end for

     return 0;
} // end main
```

```
void eliminaDuplicati(pezzo vecchio[], int len_vecchio,
                      pezzo nuovo[], int& len_nuovo) {

// Convenzione:
// I parametri -- pezzo vecchio[], int len_vecchio –
// sono considerati parametri in ingresso, quindi si conviene che
// non saranno modificati dal sottoprogramma.
// I parametri -- pezzo nuovo[], int& len_nuovo –
// sono considerati parametri in uscita, quindi si fa' uso del
// fatto che i vettori sono passati tramite il puntatore alla
// prima cella dello stesso, mentre la lunghezza  len_nuovo
// coincidera' con una variabile del programma chiamante passata
// per reference.

// Nota:
// Per marcare i pezzi del vettore vecchio[] gia' copiati nel
// vettore nuovo[], si conviene di assegnargli quantità negativa
// (-1), quindi per la convenzione descritta sopra si rende
// necessario fare una copia locale del vettore vecchio[] e
// lavorare su quest'ultima.

     pezzo elenco[DIM_ELENCO];
     int i, j, k;

     for (i = 0; i < len_vecchio; i++)
       elenco[i] = vecchio[i]; //copia celle che sono delle struct

     k = 0; // variabile che identifica la posizione nel vettore
            // nuovo[] dell'ultima cella copiata
     for (i = 0; i < len_vecchio; i++) {

       if (vecchio[i].quantita >= 0) {

         nuovo[k] =  vecchio[i];
         for (j = i+1; j < n ; j++) {
           cmp =  strcmp(nuovo[k].codice, vecchio[j].codice);
           if (cmp == 0){
             nuovo[k].quantita += vecchio[j].quantita;
             vecchio[j].quantita = -1;
           }
         } // end for

         k++;

       } // end if
     } // end for
   // k, contiene ora il numero di elementi nel vettore nuovo[]
   len_nuovo = k;//scrivo il valore di k nel parametro di ritorno
   return;
} // end elimina_duplicati
```

ESERCIZIO n. 4 (Gestione liste dinamiche semplicemente concatenate)

Il tipo di dato lista è definito nel modo seguente:

```
struct nodo {
       int dato;
       nodo *next;
};
```

Prevedere la stesura dei seguenti metodi:

```
void  stampaLista(nodo* h);
int   lenLista(nodo* h);
nodo* cercaLista(nodo* h, int x);
// Ritorna il noto contenente la prima occorrenza di x in h

nodo* CreaNodo(int d);

void copiaLista(nodo* h, nodo* &cph);
// Ritorna una copia della lista passata come primo parametro.

nodo* insertHead(nodo* h, nodo* el);
// Inserimento in testa del nodo, ritorna copia del puntatore
// della lista aggiornata.

void  insertOrd(nodo* & h, int x);
// Lista ordinata in senso crescente -- inserimento ordinato

void  cancellaLista(nodo* &h);
```

Successivamente scrivere i sottoprogrammi di manipolazione di liste dinamiche che consentano di:

1. Concatenare due liste di interi h1 e h2 aggiornando h1;

```
void appendLista(nodo* & h1, nodo* & h2);
```

2. Fondere due liste dinamiche di interi (h1 e h2) (ordinate in senso crescente) in un'unica lista anch'essa ordinata (h1) restituendo h2 vuota (h2 == NULL).

```
void MergeS(nodo* & h1, nodo* & h2);
```

3. Fondere due liste dinamiche di interi (h1 e h2) in modo tale che la lista risultante (h1) sia costituita da una successione alternata di nodi della prima e della seconda lista e h2 vuota (h2 == NULL).

```
void MergeInterleaved(nodo* & h1, nodo* & h2);
```

Il programma principale dovrà:

- prevedere l'allocazione di due liste dinamiche semplicemente concatenate che memorizzino due sequenze arbitrariamente lunghe di numeri interi terminate ciascuna con uno 0 (escluso);
- prevedere per ciascuno dei sottoprogrammi implementati, un test di funzionamento.

SOLO PER COMPLETEZZA

P.S.: QUANDO SI UTILIZZA UN COMPILATORE C++, COME QUELLO A DISPOSIZIONE PER IL CORSO (devC++) PER SEMPLICITA' DI COMPRENSIONE SI PREFERISCE QUASI SEMPRE SCRIVERE CODICE CON LA SINTASSI DEL C++,
comunque volendo utilizzare la sintassi C tutti i sottoprogrammi illustrati nell'esercizio dovrebbero avere l' intestazione definita come segue:

```
void  stampaLista(lista h);
int   lenLista(lista h);
nodo* cercaLista(lista h, int x);
nodo* CreaNodo(int d);
void copiaLista(lista h, lista & cph);
lista insertHead(lista h, nodo* el);
void  insertOrd(lista & h, int x);
void  cancellaLista(lista &h);
void appendLista(lista & h1, lista & h2);
void MergeS(lista& h1, lista & h2);
void MergeInterleaved(lista & h1, lista & h2);
```

```
/* Alternativa in C */
typedef struct Nodo {
       int dato;
       struct Nodo *next;
} nodo;
typedef nodo* lista;
```

P.P.S.: in C puro non esiste il passaggio parametri per *reference*, ma solo quello per copia; quindi il codice di tutti i sottoprogrammi dovrebbe essere riscritto. Prendendo ad esempio la definizione del sottoprogramma *cancellaLista* potreste scrivere il codice in almeno tre modi diversi.

```
void cancellaLista(nodo* &h) {  // 1mo modo: C++
    nodo *aux;
    while (h!=NULL) { aux = h;
                      h = h->next;
                      delete aux;
    }
}
void cancellaLista(lista &h) { // 2do modo: C++ misto a C
    nodo *aux;
    while (h!=NULL) { aux = h;
                      h = h->next;
                      delete aux;
    }
}
// 3zo modo: C puro
void cancellaLista(lista* h){  /* e' un doppio puntatore a una    */
    nodo *aux;                 /* struct di tipo: nodo;  lista* h;*/
    while ((*h)!=NULL) {       /* e' come scrivere nodo* h;       */
       aux = (*h);
       (*h) = (*h)->next;
       free(aux);
    }
}
```

```
/* Definizione in C puro */
typedef struct Nodo {
       int dato;
       struct Nodo *next;
} nodo;
typedef nodo* lista;
```

```
#include <iostream.h>
#include <stdlib.h>

struct nodo {
       int dato;
       nodo *next;
};

void stampaLista(nodo* h) {
  cout << "\n";
  if (h != NULL) {
    while (h!=NULL) {
        cout << "-" << h->dato;
        h = h->next;
    }
    cout << "\n";
  }
  else cout << "Empty!";
}

int lenLista(nodo* h) {
   int count = 0;
   while (h!=NULL) {
        count++;
        h = h->next;
   }
   return count;
}

nodo *CreaNodo(int d) {

     nodo *p = new nodo;      // Creazione nuovo nodo
     p->dato = d;
     p->next = NULL;
     return p;
}
void copiaLista(nodo* h, nodo* & cph) {
    nodo * aux;
    if (h!=NULL) {
      cph = CreaNodo(h->dato);
      h = h->next;
      aux = cph;
      while (h!=NULL) {
           aux->next = CreaNodo(h->dato);
           h = h->next;
             aux = aux->next;

      }
    }
    else cph = NULL;
}
```

```
nodo* insertHead(nodo* h, nodo *el) {

  if (el != NULL) {
     el->next = h;  // L'elemento viene agganciato in testa
     return el;     // Viene restituito l'indirizzo del nodo
                    // agganciato che sarà quindi la nuova testa
                    // della lista.
  } else return h;
}

nodo* cercaLista(nodo* h, int x) {
// ritorna il punt al nodo che contiene la prima occorrenza di x.
    while (h != NULL) {
         if ((h->dato) == x) return h;
         h=h->next;
    }
    return h;
}

void insertOrd(nodo* & h, int x) {
// Si suppone la lista in ordine crescente
// si inserisce un nuovo nodo rispettando tale ordinamento
// h è passato per reference, per scandire
// la lista devo quindi usare un puntatore ausiliario.

    nodo *curr;
    nodo *pred = NULL;
    nodo *aux = CreaNodo(x);

    if (h == NULL) { h = aux;
                     return;
    }
// N.B.: le condizioni in C vengono valutate da
// sinistra verso destra, quindi nel caso in cui il nuovo nodo
// debba essere aggiunto in coda la 2da condizione del while non
// viene valutata perché in un AND logico dove il primo termine e'
// falso tutta l'espressione è falsa.

   curr = h ;
   while ((curr!=NULL)&&(curr->dato <= x)) {
        pred = curr;
        curr = curr->next ;
   }

   if (pred != NULL) {
       pred->next = aux;
      aux->next = curr;
   } else { // inserimento in testa
        aux->next = h;
        h = aux;
   }
} // end insertOrd
```

```
void appendLista(nodo* & h1, nodo* & h2) {
// restituisce h1 aggiornata con h2 in coda.
// i parametri sono passati per reference per gestire il caso h1
// vuota e h2 piena.
    nodo* aux = h1;
    if (h1 == NULL)  h1 = h2 ;
    else {
           while (aux->next!= NULL) {
                aux = aux->next;
           }
           aux->next = h2;
    }
}
void MergeS (nodo* & h1, nodo* & h2) {
// dopo la chiamata al sottoprogramma h1 punterà alla lista
// risultato della fusione ordinata, h2 a NULL
     nodo* head;
     nodo* last_ins;

     if (h2 == NULL) return;
     if (h1 == NULL) { h1 = h2; h2 = NULL; return; }

     if (h1->dato < h2->dato) {
          head = h1;
          last_ins = h1;
          h1 = h1->next;
     }
     else {
              head = h2;
              last_ins = h2;
              h2 = h2->next;
     }

     while ((h1!=NULL) && (h2!=NULL)){

        if (h1->dato < h2->dato) {
               last_ins->next = h1;
               h1 = h1->next;
               last_ins = last_ins->next;
          }
          else {
                    last_ins->next = h2;
                   h2 = h2->next;
                   last_ins = last_ins->next;
          }
     }
    if (h1!=NULL) { last_ins->next = h1;  }
    if (h2!=NULL) { last_ins->next = h2;  }
    h1 = head;
    h2 = NULL;

} // end MergeS
```

```
void MergeInterleaved(nodo* & h1, nodo* & h2) {

     nodo* head;
     nodo* temp;

     if (h2 == NULL) return;
     if (h1 == NULL) { h1 = h2; h2 = NULL; return; }
     head = h1;
     while ((h1!=NULL) && (h2!=NULL)) {

         temp = h1;
         h1 = h1->next;
         temp->next = h2;

         // scambio puntatori h1 e h2
         temp = h1;
         h1 = h2;
         h2 = temp;
     }

     h1 = head;
     h2 = NULL;

} // end MergeInterleved



void cancellaLista(nodo* &h) {
    nodo *aux;
    while (h!=NULL) {
       aux = h;
       h = h->next;
       delete aux;
    }
}
```

```
int main() {

     nodo* one = NULL;
     nodo* two = NULL;
     nodo* three = NULL;
     nodo* four = NULL;
     nodo *aux;
     int x;

     cout << "\ninserire un intero (0 per terminare): ";
     cin >> x; cin.ignore();
     while (x != 0) {
         insertOrd(one,x);
         cout << "\ninserire un intero (0 per terminare): ";
          cin >> x; cin.ignore();
     }
     cout << "\n\n gli elementi inseriti sono: \n\n";
     stampaLista(one);

     cout << "\nLa lista contiene ";
     cout << lenLista(one)<<" elementi!";

     aux = cercaLista(one,1);
     if (aux)
       cout << "\n 1 e' presente nella lista! ";
     else
      cout << "\n 1 NON e' presente nella lista! ";

     cout << "\ninserire un intero (0 per terminare): ";
     cin >> x; cin.ignore();
     while (x != 0) {
         insertOrd(two,x);
         cout << "\ninserire un intero (0 per terminare): ";
         cin >> x; cin.ignore();
     }
    cout << "\n\ngli elementi inseriti sono: \n\n";
    stampaLista(two);
    cout << "\nLa lista contiene "<< lenLista(two) <<" elementi!";


    cout << "\nCopio le prime due liste....";
    copiaLista(one, three);
    copiaLista(two, four);
```

```
     cout <<"\nFusione Ordinata delle copie delle due liste: \n ";

     MergeS(three, four);
     cout << "\nStampa della prima e della seconda lista”;
     cout << “ dopo la chiamata a sottoprogramma.... \n ";
     stampaLista(three);
     stampaLista(four);

     cout << "\nFusione Interlacciata delle prime due liste: \n ";

     MergeInterleaved(one, two);
     cout << "\nStampa della prima e della seconda lista”;
     cout << “ dopo la chiamata a sottoprogramma.... \n ";
     stampaLista(one);
     stampaLista(two);

     cout << "\nDealloco le 4 liste.... \n ";

     cancellaLista(one);
     cancellaLista(two);
     cancellaLista(three);
     cancellaLista(four);

     system(“PAUSE”);
     return 0;
} // end main
```