ARGOMENTO: Liste

ESERCIZIO n° 1

Un file binario (CABALA.DAT) contiene informazioni (numero e parola associata di al massimo 50 caratteri) su alcuni numeri interi positivi. Lo stesso numero può comparire più volte, con parole associate diverse.

Si scriva un programma in C che:

Legga gli elementi del file CABALA.DAT e li inserisca in una lista L (ordinata sul campo parola);

Stampi a terminale la lista L.

Soluzione:

```
/* PROGRAMMA PRINCIPALE */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct {int numero;
                char parola[50]; }
                              element_type;

typedef struct   list_element
     { element_type value;
       struct list_element  *next;
      } item;

typedef  item* list;
```



```
/* PROTOTIPI */
int isequal(element_type, element_type);
int isless(element_type, element_type);
void showel (element_type);
list ordins(element_type, list);
void showlist (list);


void main(void)
{element_type   e;
 list   L;
 FILE *f1;

 /* DOMANDA a */
 f1 = fopen("CABALA.DAT", "rb");
 L=NULL;
 while
  (fread(&e,sizeof(element_type),1,f1)!=0)
          L=ordins(e, L);
 fclose(f1);

 /* DOMANDA b */
 while (!(L==NULL))
       {showel(L->value));
        L= L->next;
       }
}

 /*fine main*/
```

```
/* FUNZIONI AUSILIARIE */

int isequal(element_type e1,
            element_type e2)
          /*  uguaglianza sulla parola*/
{return   !strcmp(e1.parola, e2.parola); }


int isless(element_type e1, element_type e2)
       /* relazione d'ordine sulla parola */
{return (strcmp(e1.parola, e2.parola)<0);}


void showel (element_type e)
{ printf("%s\t %d\n",e.parola, e.numero);}
```



```
list   ordins(element_type el, list root)
{element_type  e;
 int trovato=0;
 list aux, prec, current; /*inserimento ordinato
                    con possibili duplicazioni */
 aux=(list)malloc(
          sizeof(struct list_element));
  aux->value = el;
  aux->next = NULL;
  if (root==NULL)    /* primo elemento */
       root=aux;
  else
   {current=root;
    while((current!=NULL)&& !trovato)
       {if (isless(current->value,el))
          { prec=current;
            current=current->next; }
        else trovato=1; }

    if (current==NULL)
         {prec->next=aux;}   /*in fondo */
    else
         if (current==root)
             {aux->next=root;
              root=aux;}    /* in testa */
         else
             {prec->next=aux;
                    /* inserisce in mezzo */
              aux->next=current;}
       }
   }
 return root;
}
```

ESERCIZIO n° 2

Due file (binari) (F1.DAT ed F2.DAT) contengono dati rappresentanti i codici e la quantità dei prodotti di due magazzini. Ogni codice è una stringa di 6 caratteri, mentre la quantità è un valore intero. Lo stesso codice può comparire in entrambi i file, ma compare una sola volta all'interno di ciascun file.

Si scriva un programma C che:

Crei due liste, L1 e L2, dei prodotti (codici e quantità) rispettivamente di F1.DAT ed F2.DAT comuni ai due file. Creare le liste in modo che siano ordinate sulla base del valore del campo quantità.

A partire da L1 ed L2 produca un file binario (COMUNI.DAT) in cui ogni prodotto comune è scritto una sola volta insieme alla quantità complessiva dei due magazzini;

Attraverso una procedura **iterativa**, stampi a terminale i codici dei prodotti di L1 presenti in magazzino in quantità maggiore di una quantità letta a terminale.

È possibile utilizzare ***librerie C*** (ad esempio per stringhe). Qualunque ***libreria utente*** utilizzata va riportata nello svolgimento.

Schema della soluzione:

```
/* PROGRAMMA PRINCIPALE */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct {char codice[6];
                int quantita;} element_type;
typedef enum {false, true}  bool;

typedef struct  list_element
     { element_type         value;
       struct list_element  *next;
     } item;

typedef  item* list;

/* PROTOTIPI */
bool isequal(element_type, element_type);
bool isless(element_type, element_type);
void showel (element_type);
list ordins(element_type, list);
element_type member(element_type, list);

void main(void)
{element_type prodotto1, prodotto2,
              prodotto;
 list  L1,L2,L;
 FILE *f1, *f2, *f3;
 bool trovato; int qty;
 L1=NULL;
 L2=NULL;
 f1 = fopen("F1.DAT", "rb");
 f2 = fopen("F2.DAT", "rb");
```

```
/*  creazione liste prodotti  */
while (fread(&prodotto1,
          sizeof(element_type),1,f1)!=0)
  {rewind(f2);
   trovato=false;
   while (fread(&prodotto2,
    sizeof(element_type),1,f2)!=0)&&!trovato)
      {if (isequal(prodotto1,prodotto2))
          { trovato=true;
            L1=ordins(prodotto1,L1);
            L2=ordins(prodotto2,L2);
          }
      }
  }
fclose(f1);
fclose(f2);

L=L1;
f3 = fopen("COMUNI.DAT", "wb");

/* i codici nelle due liste sono gli
   stessi ma non nello stesso ordine */
while (!(L==NULL))
 {strcpy(prodotto.codice,
           (L->value).codice);
  prodotto2=member(L->value,L2);
  prodotto.quantita= (L->value).quantita+
                prodotto2.quantita;
  fwrite(&prodotto,
       sizeof(element_type),1,f3);
  L = L->next;
 }
fclose(f3);
```



```
printf("Quantita minima? ");
scanf("%d",&qty);
while (!(L1==NULL))
  {if ((L1->value).quantita>qty)
      showel(L->value);
   L1 = L1->next;
  }
}


/* FUNZIONI AUSILIARIE */

bool isequal(element_type e1,
             element_type e2)
          /*  uguaglianza sul codice  */
{return (strcmp(e1.codice,e2.codice)==0); }

bool isless(element_type e1,
            element_type e2)
   /* relazione d'ordine sulla quantita  */
{return   (e1.quantita<e2.quantita);}

void showel (element_type e)
{ printf("%s\t%d\n",e.codice,e.quantita);}
```

```
list   ordins(element_type el, list root)
{element_type  e;
 int trovato=0;
 list aux, prec, current;
          /* inserimento ordinato con
             possibili duplicazioni */
 aux=(list)malloc(
          sizeof(struct list_element));
  aux->value = el;
  aux->next = NULL;
  if (root==NULL)    /* primo elemento */
       root=aux;
  else
   {current=root;
    while((current!=NULL)&& !trovato)
       {if (isless(current->value,el))
          { prec=current;
            current=current->next; }
        else trovato=1; }

    if (current==NULL)
         {prec->next=aux;}   /*in fondo */
    else
         if (current==root)
             {aux->next=root;
              root=aux;}    /* in testa */
         else
             {prec->next=aux;
                    /* inserisce in mezzo */
              aux->next=current;}
       }
   }

 return root;
}
```



```
element_type member (element_type el,
                     list l)
 /* ricerca - iterativa
    si suppone che l'elemento esista
    e venga restituito dalla funzione  */
{ while (!(l==NULL))
     { if (isequal(el,l->value))
                      return(l->value);
           else  l=l->next;
     }
}
```

Seconda soluzione:

Usiamo un approccio a moduli (utilizzando funzioni ricorsive):

el.h	header file associato ad el.c
el.c	funzioni di utilità associate al tipo degli elementi
listp.h	header file associato a listp.c
listp.c	funzioni di libreria per la lista
main.c	funzione main

```
/* LIST ELEMENT TYPE - file el.h*/
typedef struct {char codice[6];
                int quantita;} element_type;

typedef enum {false, true}  bool;

bool isequal(element_type, element_type);
bool isless(element_type, element_type);
void showel (element_type);
```

```
/* LIST ELEMENT TYPE - file el.c*/
#include "el.h"
#include <string.h>

bool isequal(element_type e1,
             element_type e2)
          /*  uguaglianza sul codice  */
{return (strcmp(e1.codice,e2.codice)==0); }

bool isless(element_type e1,
            element_type e2)
   /* relazione d'ordine sulla quantita  */
{return   (e1.quantita<e2.quantita);}

void showel (element_type e)
{ printf("%s\t%d\n",e.codice,e.quantita);}
```



```
/* LIST INTERFACE  - file listp.h*/
#include "el.h"

typedef struct  list_element
     { element_type         value;
       struct list_element  *next;
     } item;

typedef  item* list;

/*  PROTOTIPI DI FUNZIONE (extern) */
list emptylist();
bool empty(list);
element_type head(list);
list tail(list);
list cons(element_type, list);
list ordins(element_type, list);
element_type member(element_type, list);
```

```
/* LIST IMPLEMENTATION - file listp.c */
#include "listp.h"
#include <stdlib.h>

list  emptylist()
{ return NULL; };

bool  empty(list l)
{ return (l==NULL); };

element_type head(list l)
{ if (empty(l)) { abort(); }
  else  return(l->value); }

list  tail(list l)
{ if (empty(l)) { return emptylist(); }
  else         { return (l->next); }
}

list  cons(element_type e, list l)
{ list t;
  t=(list)malloc(sizeof(item));
  t->value=e;
  t->next=l;
  return(t);
}
```



```
list   ordins(element_type el, list l)
{
          /* inserimento ordinato con
             possibili duplicazioni */
 if (empty(l)) return(cons(el,l));
  else
     {
        if (isless(el,head(l)))
                return(cons(el,l));
        else return(cons(head(l),
                  ordins(el,tail(l))) );
     }
}

element_type member (element_type el,
                     list l)
/* cerca nella lista l un elemento
   che abbia lo stesso codice di el -
   si suppone che l'elemento esista
   sicuramente e che venga restituito
   dalla funzione                    */
{ if (!empty(l))
     { if (isequal(el,head(l)))
         return head(l);
       else return member(el,tail(l));
     }
 else abort();
}
```

```
/* PROGRAMMA PRINCIPALE */
#include <stdio.h>
#include <stdlib.h>
#include "listp.h"


void main(void)
{element_type   prodotto1,
              prodotto2,prodotto;
 list  L1,L2,L;
 FILE *f1, *f2, *f3;
 bool trovato; int qty;
 L1=emptylist();
 L2=emptylist();
 f1 = fopen("F1.DAT", "rb");
 f2 = fopen("F2.DAT", "rb");

/*  creazione liste prodotti  */
while (fread(&prodotto1,
          sizeof(element_type),1,f1)!=0)
  {rewind(f2);
   trovato=false;
   while((fread(&prodotto2,
              sizeof(element_type),1,f2)!=0)
        &&!trovato)
     {if (isequal(prodotto1,prodotto2))
          { trovato=true;
            L1=ordins(prodotto1,L1);
            L2=ordins(prodotto2,L2);
          }
     }
  }
fclose(f1);
fclose(f2);
```



```
L=L1;
f3 = fopen("COMUNI.DAT", "wb");

/* i codici nelle due liste sono gli
   stessi ma non nello stesso ordine */
while (!empty(L))
 {strcpy(prodotto.codice,
           head(L).codice);
  prodotto2=member(head(L),L2);
  prodotto.quantita=head(L).quantita+
                 prodotto2.quantita;
  fwrite(&prodotto,
       sizeof(element_type),1,f3);
  L = tail(L);
 }
fclose(f3);

printf("Quantita minima? ");
scanf("%d",&qty);
while (!empty(L1))
  {if ((L1->value).quantita>qty)
      showel(L1->value);
   L1 = tail(L1);
  }
}
```

ESERCIZIO n° 3

Sia dato un file binario teatro.dat che contiene le informazioni relative ai posti di un teatro per una certa rappresentazione. In particolare, ciascun record di teatro.dat contiene le seguenti informazioni:

numero: un intero che rappresenta univocamente il posto;

settore: un carattere appartenente all'insieme {*p*, *g*, *l*} (*p* significa platea, *g* sta per galleria, *l* sta per loggione) che rappresenta il settore del teatro in cui è collocato il posto;

prezzo: un intero che rappresenta il costo del biglietto relativo al posto considerato.

cliente: una stringa che rappresenta il cognome dell'eventuale persona che ha prenotato quel posto (in questo caso, il posto risulta occupato); se il campo cliente è vuoto (cioè, contiene la stringa nulla " "), significa che il posto è libero.

Scrivere un programma C che:

a partire dal file "teatro.dat" crei due liste L1 ed L2 che contengano rispettivamente i posti liberi (L1) ed i posti occupati (L2); le 2 liste devono essere ordinate in base al numero di posto.

data una richiesta di prenotazione dallo standard input, espressa attraverso un carattere appartenente all'insieme {*p*, *g*, *l*} (per specificare il tipo di settore richiesto) e dal cognome del richiedente, il programma deve eliminare da L1, se esiste, il primo elemento corrispondente al settore considerato. Se, invece, la richiesta non può essere soddisfatta (cioè, se non esiste un posto libero nel settore richiesto), il programma dovrà registrare la richiesta (cognome e settore) in un file di testo "rifiutati.txt".

Soluzione:

Suddivido il programma nei moduli list, el, main, rappresentati dai seguenti file:

list.c	implementazione del tipo di dato astratto lista
list.h	interfaccia del modulo
el.c	funzioni di utilità dipendenti dalla rappresentazione di element_type (definizione)
el.h	interfaccia
main.c	contiene il programma principale (funzione main())

File el.h:

```
/* LIST ELEMENT TYPE - file el.h*/
typedef struct {
              int numero;
              char settore;
              int prezzo;
              char cliente[30];
              } element_type;


int isequal(element_type, element_type);
int isless(element_type, element_type);
void showel (element_type);
```

File el.c:

```
/* LIST ELEMENT TYPE - file el.c*/
#include "el.h"
#include <string.h>

int isequal(element_type e1,
            element_type e2)
{return   (e1.settore==e2.settore); }

int isless(element_type e1, element_type e2)
       /* relazione d'ordine sul numero  */
{return (e1.numero<e2.numero);}


void showel (element_type e)
{printf("%d\t%c\t%d\t%s\n",e.numero,
         e.settore,e.prezzo, e.cliente);}
```


File list.h:

```
/* LIST INTERFACE  - file listp.h*/
#include "el.h"

typedef struct  list_element
     { element_type value;
       struct list_element  *next;
     } item;

typedef  item* list;

/*  PROTOTIPI DI FUNZIONE (extern) */
list emptylist();
int empty(list);
element_type head(list);
list tail(list);
list cons(element_type, list);
list ordins(element_type, list);
list delete( element_type, list);
int member(element_type e, list l);
```

File list.c:

```
/* LIST IMPLEMENTATION - file listp.c */
#include "list.h"
#include <stdlib.h>

list  emptylist()
{ return NULL; }

int  empty(list l)
{ return (l==NULL); }

element_type head(list l)
{
 if (empty(l)) { abort(); }
 else  return(l->value); }

list  tail(list l)
{
 if (empty(l)) { return emptylist(); }
  else         { return (l->next); }
}

list  cons(element_type e, list l)
{
 list t;
 t=(list)malloc(sizeof(item));
 t->value=e;
 t->next=l;
 return(t);
}
```



```
 list  ordins(element_type el, list l)
/*ins. ordinato con event. duplicazioni */
{element_type  e;
   if (empty(l))
      return(cons(el,l));
   else
     if (isless(el,head(l)))
                return(cons(el,l));
           else
              return(cons(head(l),
                    ordins(el,tail(l))) );
     }



list delete( element_type e,  list l)
{  if (empty(l)) return l;
   else
      if (isequal(head(l),e))
               return tail(l);
         else return cons(head(l),
                       delete( e, tail(l)));
}


int member(element_type e, list l
{  element_type h;
   h=head(l);
   if (empty(l)) return 0;
   else if(isequal(e,h)) return 1;
        else return member(e, tail(l));
}
```

File main.c

```
#include <stdio.h>
#include "list.h"

main()
{
int i,k,trovato;
char c;
FILE *f1;
list L1, L2;
element_type e, e1;
char Svuota[]="";


/* domanda 1: lettura da magazzino */
L1=emptylist();
L2=emptylist();

f1=fopen("teatro.dat", "rb");
while
(fread(&e, sizeof(element_type), 1, f1)!=0)
   {  if (!strcmp(e.cliente, Svuota))
         L1=ordins(e,L1);
      else
         L2=ordins(e,L2);
   }
fclose(f1);
```



```
/*domanda 2: aggiornamento di L*/
fflush(stdin);
printf("Dammi il settore richiesto: ");
scanf("%c", &e.settore);
printf("Dammi il cognome del cliente: ");
scanf("%s", e.cliente);
if (member(e, L1))
      L1=delete(e1,L1);

else
   {  f1=fopen("rifiutati.txt", "w");
      fprintf(f1,"%s %c\n",e.cliente,
               e.settore);
      fclose(f1);
   }

}
```

ESERCIZIO n° 4

Esercizio n. 2 - 16 Febbraio 2004

E' dato un file di testo (SCHEDINA.TXT) contenente un elenco di risultati per alcune partite di calcio. Su ciascuna linea sono memorizzati il nome della squadra ospitante (max 20 caratteri), il nome della squadra ospite (max 20 caratteri) e a seguire un valore (1, 0, o 2) per indicare rispettivamente la vincita della squadra ospitante, il pareggio o la vittoria della squadra ospite. Si scriva un programma C strutturato in (almeno due) funzioni dedicate rispettivamente a:

leggere il contenuto del file e inserirlo in una lista L, ordinata in senso decrescente in base al valore del risultato (2, 1, 0);

a partire da L, stampare sul file FUORICASA.TXT, solo i nomi delle squadre vincitrici fuori casa (valore del risultato 2).

Esempio:

Contenuto di SCHEDINA.TXT: Roma Perugia 1

Milan Bologna 2

Nocerina Battipagliese 0

Inter Juventus 2

Contenuto di FUORICASA.TXT: Bologna

Juventus

Strutture dati:

Le strutture dati risultano del tipo:

```
typedef   struct
             {char squadra1[20];
              char squadra2[20];
              int  res;} el_type;
typedef struct nodo
          {el_type  value;
           struct nodo *next;  } NODO;
typedef NODO* list;
```

Struttura del programma:

La lista L viene inizializzata a lista vuota.

- Domanda a) :

Deve essere eseguito un ciclo di lettura (**while (!eof(f))**) delle stringhe dal file SCHEDINA.TXT (assegnato alla variabile **f** ed aperto in lettura).

Per ciascuna tripla (squadra1, squadra2 e risultato) letta (memorizzata in una variabile strutturaEL), si inserisce EL in L (con sinserimento ordinato). Si chiude il file.

- Domanda b):

Deve essere eseguito un ciclo di scansione (**while (L!=NULL)**) della lista L e per ciascun nodo di L, se il campo L->value.res è uguale a 2 si scrive nel file FUORICASA.TXT (assegnato alla variabile **f** ed aperto in scrittura) il campo L->value.squadra2.

File utilizzati:

main.c	programma principale
el.h	file header del tipo di dato astratto el_type
el.c	implementazione
list.h	file header del tipo di dato astratto lista ordinata
list.c	implementazione della libreria liste

```
/* MAIN FILE */
#include "lists.h"
#include <stdio.h>

list  crealista(FILE *f,list L);
void  creafile(FILE *f,list L);
void scrivifile (FILE * f, list L);

void  main ()
{  list  L = emptylist();
   el_type EL;
   FILE  *f;

/* DOMANDA a)  */
   f = fopen("SCHEDINA.TXT", "rt");
   L=crealista(f,L);
   close(f);

/* DOMANDA  b)  */
   f = fopen("FUORICASA.TXT", "wt");
   scrivifile(f,L);
   fclose(f);
}
```



```
list  crealista(FILE *f,list L)
{ el_type EL;
  while (!feof(f))
    { fscanf(f,"%s%s%d\n",EL.squadra1,
             EL.squadra2,& EL.res);
    /* INSERIMENTO NELLA LISTA */
          L = ordins(EL,L); }
}


void  scrivifile(FILE *f, list L)
{ el_type EL;
  while (!empty(L))
     { EL=head(L);
       if (EL.res==2)
             fprintf(f,"%s",EL.squadra2);
       L=tail(L);
   }
}
```

```
/* ELEMENT TYPE - file el.h*/

typedef   struct
             {char squadra1[20];
              char squadra2[20];
              int  res;} el_type;

typedef   enum {false,true}  bool;

bool isless(el_type, el_type);
```

```
/* ELEMENT TYPE - file el.c*/
#include "el.h"
#include <stdio.h>

bool isless(el_type e1, el_type e2)
{if (e1.res < e2.res) return true;
 else return false;}
```



```
/* LIST  INTERFACE  - file lists.h*/
#include "el.h"
typedef struct nodo
          {el_type  value;
           struct nodo *next; } NODO;
typedef NODO* list;

/* operatori esportabili */
list  emptylist();
bool  empty(list l);
el_type  head(list l);
list  tail(list l);
list  cons(el_type e, list l);
list ordins(el_type, list);
```

```
/* LIST IMPLEMENTATION - file lists.c */
#include <stdio.h>
#include <stdlib.h>
#include "lists.h"

/* OPERAZIONI PRIMITIVE */
list  emptylist()
{ return NULL; };

bool  empty(list l)
{ return (l==NULL); };

el_type  head(list l)
{ if (empty(l)) { abort(); }
  else  return(l->value); }

list  tail(list l)
{ if (empty(l)) { abort(); return(0); }
  else          { return (l->next); }
}

list  cons(el_type e, list l)
{list t;
 t=(list)malloc(sizeof(NODO));
 t->value=e;
 t->next=l;
 return(t);
}
```



```
list   ordins(el_type el, list l)
{
          /* inserimento ordinato con
             possibili duplicazioni */
 if (empty(l)) return(cons(el,l));
  else
     {
        if (! isless(el,head(l)))
                return(cons(el,l));
        else return(cons(head(l),
                  ordins(el,tail(l))) );
     }
}
```

Si noti che l'uso del not (!) nella funzione ordins fa sì che la lista venga creata ordinata in senso decrescente.

ESERCIZIO n° 5

Esercizio n. 2 - 18 Febbraio 2005

E' dato un file di testo (TESI.DAT) contenente i dati relativi a tesi di laurea.

Il file contiene per ciascuna tesi, il titolo della tesi (stringa di 15 caratteri), un codice intero (parola chiave codificata da un intero) e un email del tesista (stringa di 20 caratteri).

Si scriva un programma C **strutturato in (almeno due) funzioni** dedicate rispettivamente a:

- leggere il contenuto del file e inserirlo in una lista L, ordinata in base al titolo;
- letto un intero da console, accendendo a L, produca una seconda lista LKEY con tutte e sole le tesi che si riferiscono a tale codice;

FACOLTATIVO:

accendendo a LKEY, stampi a video il numero di elementi della lista LKEY prodotta.

Strutture dati:

Le strutture dati risultano del tipo:

```
typedef   struct
             {char titolo[15];
              int  codice;
              char email[20];} el_type;

typedef struct nodo
          {el_type  value;
           struct nodo *next;  } NODO;

typedef NODO* list;
```

File utilizzati:

main.c	programma principale
el.h	file header del tipo di dato astratto el_type
el.c	implementazione
list.h	file header del tipo di dato astratto lista ordinata
list.c	implementazione della libreria liste

```
/* MAIN FILE */
#include "lists.h"
#include <stdio.h>

list  crealista(FILE *f,list L);
list  crealista2(int cod, list L);

void  main ()
{  list  L = emptylist();
   list  LKEY = emptylist();
   el_type EL;
   int cod;
   FILE  *f;

/* DOMANDA a)  */

   f = fopen("TESI.DAT", "rt");
   L=crealista(f,L);
   close(f);

/* DOMANDA  b)  */
   scanf("%d",&cod);
   LKEY=crealista2(cod,L);

/* DOMANDA  facoltativa)  */
/* stampa della lunghezza di LKEY */   }
```



```
list  crealista(FILE *f,list L)
{ el_type EL;
  while (!feof(f))
    { fscanf(f,"%s%d%s\n",EL.titolo,
             &EL.codice,EL.email);
    /* INSERIMENTO NELLA LISTA */
          L = ordins(EL,L); }
}


list  crealista2(int cod,list L)
{ list aux=emptylist();
  while (!empty(L))
    { if (head(L).codice==cod)
          aux=cons(head(L), aux);
      L=tail(L); }
  return aux;
}
```

```
/* ELEMENT TYPE - file el.h*/

typedef   struct
             {char titolo[15];
              int  codice;
              char email[20];} el_type;

typedef   enum {false,true}  bool;

bool isless(el_type, el_type);
```

```
/* ELEMENT TYPE - file el.c*/
#include "el.h"
#include <string.h>

bool isless(el_type e1, el_type e2)
{if (strcmp(e1.titolo,e2.titolo)<=0)
   return true;
 else return false;}
```



```
/* LIST  INTERFACE  - file list.h*/
#include "el.h"
typedef struct nodo
          {el_type  value;
           struct nodo *next; } NODO;
typedef NODO* list;

/* operatori esportabili */
list  emptylist();
bool  empty(list l);
el_type  head(list l);
list  tail(list l);
list  cons(el_type e, list l);
list ordins(el_type, list);
```

```
/* LIST IMPLEMENTATION - file lists.c */
#include <stdio.h>
#include <stdlib.h>
#include "lists.h"

/* OPERAZIONI PRIMITIVE */
list  emptylist()
{ return NULL; };

bool  empty(list l)
{ return (l==NULL); };

el_type  head(list l)
{ if (empty(l)) { abort(); }
  else  return(l->value); }

list  tail(list l)
{ if (empty(l)) { abort(); return(0); }
  else          { return (l->next); }
}

list  cons(el_type e, list l)
{list t;
 t=(list)malloc(sizeof(NODO));
 t->value=e;
 t->next=l;
 return(t);
}
```



```
list   ordins(el_type el, list l)
{
          /* inserimento ordinato con
             possibili duplicazioni */
 if (empty(l)) return(cons(el,l));
  else
     {
        if (isless(el,head(l)))
                return(cons(el,l));
        else return(cons(head(l),
                  ordins(el,tail(l))) );
     }
}
```

ESERCIZIO n° 6

Esercizio n. 2 - 17 Febbraio 2006

Due file di binari (FILM1. DAT e FILM2.dat) contengono ciascuno i dati relativi ad alcuni film candidati al premio Oscar.

Ogni file contiene per ciascun film, il titolo del film (stringa di 15 caratteri), il cognome e nome del regista (stringhe di 15 caratteri), l'anno di produzione (intero), il numero di candidature (intero).

Si scriva un programma C strutturato in (almeno tre) funzioni dedicate rispettivamente a:

- inserire i dati dei film comuni ai due file (uguaglianza in base al titolo) in una lista L1, ordinata in senso non crescente in base al numero di candidature;
- letto un intero N da console, accendendo a L1, produca una seconda lista L2 (sempre ordinata in senso non crescente) con tutti e soli i film candidati ad almeno N oscar;
- stampi a video, in modo iterativo, i titoli dei film contenuti nella lista L2.

Strutture dati:

Ogni file contiene per ciascun film, il titolo del film (stringa di 15 caratteri), il cognome e nome del regista (stringhe di 15 caratteri), l'anno di produzione (intero), il numero di candidature (intero).

Le strutture dati risultano del tipo:

```
typedef   struct
             {char titolo[15];
              char cognome [15];
              char nome[15];
              int  anno;
              int cand;} el_type;

typedef struct nodo
          {el_type  value;
           struct nodo *next;  } NODO;

typedef NODO* list;
```

File utilizzati:

main.c	programma principale
el.h	file header del tipo di dato astratto el_type
el.c	implementazione
list.h	file header del tipo di dato astratto lista ordinata
list.c	implementazione della libreria liste

```
/* MAIN FILE */
#include "lists.h"
#include <stdio.h>

list  crealista(FILE *f, FILE *g,list L);
list  crealista2(int N, list L, list aux);
void showlist (list l);

void  main ()
{  list  L1 = emptylist();
   list  L2 = emptylist();
   int N;   FILE  *f;   FILE  *g;

/* DOMANDA a)  */
   f = fopen("FILM1.DAT", "rb");
   g = fopen("FILM2.DAT", "rb");
   L1=crealista(f,g,L1);
    close(f);
   close(g) ;

/* DOMANDA  b)  */
   scanf("%d",&N);
   L2=crealista2(N,L1,L2);

/* DOMANDA  c)  */
   showlist(L2);  }
```



```
list  crealista(FILE *f, FILE *g,list L)
{ bool trovato=false;
  el_type EL, EL2;
 while (fread(&EL, sizeof(el_type),1, f)!=0)
   { rewind(g);
      trovato=false;
    while (fread(&EL2, sizeof(el_type),1,
                g)!=0) && !trovato)
       { if (isequal(EL,EL2)
          { trovato=true;
            /* INSERIMENTO NELLA LISTA */
           L = ordins(EL,L); }
     }
}


list  crealista2(int N,list L, list aux)
{ while (!empty(L))
    { if (head(L).cand>=N)
          aux= ordins(head(L), aux);
      L=tail(L); }
  return aux;
}


void showlist (list l)
/*versione iterativa */
{  while (l!=NULL)
        {showel(l->value);
         l=l->next;
        }
}
```

```
/* ELEMENT TYPE - file el.h*/

typedef   struct
             {char titolo[15];
              char cognome [15];
              char nome[15];
              int  anno;
              int cand;} el_type;

typedef   enum {false,true}  bool;

bool isless(el_type, el_type);
bool isequal(el_type, el_type);
void showel(el_type);
```

```
/* ELEMENT TYPE - file el.c*/
#include "el.h"
#include <stdio.h>
#include <string.h>

bool isless(el_type e1, el_type e2)
{if (e1.cand<e2.cand)
   return true;
 else return false;}

bool isequal(el_type e1, el_type e2)
{if (strcmp(e1.titolo,e2.titolo)==0)
   return true;
 else return false;}

void showel(el_type e1)
{printf("%s\n",e1.titolo); }
```



```
/* LIST  INTERFACE  - file lists.h*/
#include "el.h"
typedef struct nodo
          {el_type  value;
           struct nodo *next; } NODO;
typedef NODO* list;

/* operatori esportabili */
list  emptylist();
bool  empty(list l);
el_type  head(list l);
list  tail(list l);
list  cons(el_type e, list l);
list ordins(el_type, list);
```

```
/* LIST IMPLEMENTATION - file lists.c */
#include <stdio.h>
#include <stdlib.h>
#include "lists.h"

/* OPERAZIONI PRIMITIVE */
list  emptylist()
{ return NULL; };

bool  empty(list l)
{ return (l==NULL); };

el_type  head(list l)
{ if (empty(l)) { abort(); }
  else  return(l->value); }

list  tail(list l)
{ if (empty(l)) { abort(); return(0); }
  else          { return (l->next); }
}

list  cons(el_type e, list l)
{list t;
 t=(list)malloc(sizeof(NODO));
 t->value=e;
 t->next=l;
 return(t);
}
```



```
list   ordins(el_type el, list l)
{
          /* inserimento ordinato con
             possibili duplicazioni */
 if (empty(l)) return(cons(el,l));
  else
     {
        if (! isless(el,head(l)))
                return(cons(el,l));
        else return(cons(head(l),
                  ordins(el,tail(l))) );
     }
}
```

Oppure:

```
list   ordins(element_type el, list root)
{element_type  e;
 int trovato=0;
 list aux, prec, current; /*inserimento ordinato
                    con possibili duplicazioni */
 aux=(list)malloc(
          sizeof(struct list_element));
  aux->value = el;
  aux->next = NULL;
  if (root==NULL)    /* primo elemento */
       root=aux;
  else
   {current=root;
    while((current!=NULL)&& !trovato)
       {if (isless(current->value,el))
          { prec=current;
            current=current->next; }
        else trovato=1; }

    if (current==NULL)
         {prec->next=aux;}   /*in fondo */
    else
         if (current==root)
             {aux->next=root;
              root=aux;}    /* in testa */
         else
             {prec->next=aux;
                    /* inserisce in mezzo */
              aux->next=current;}
       }
   }
 return root;
}
```


ARGOMENTO: Alberi binari di ricerca

ESERCIZIO n° 1

Un file testo (PAROLE.TXT) contiene un elenco di parole, una per linea. Ciascuna parola è lunga al piu` 20 caratteri e può comparire piu` volte nel file.

Un secondo file testo (ESCLUSE.TXT) contiene parole in copia unica (una per linea) che occorre cancellare dal file precedente.

Si definisca un programma C che:

Per operare tale cancellazione si crea in memoria centrale un albero binario di ricerca T contenente le parole del file PAROLE.TXT che ***non compaiono*** nel file ESCLUSE.TXT (la stessa parola può essere inserita piu` volte nell'albero);

Riscriva il contenuto dell'albero T sul file PAROLE.TXT, in modo che il file sia ordinato;

Soluzione:

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

typedef struct{char parola[30];} el_type;
typedef int boolean;

typedef struct nodo {
          el_type value;
          struct nodo *left, *right;
       } NODO;

typedef NODO *tree;
```

```
boolean uguale(el_type e1,el_type e2);
boolean minore(el_type e1, el_type e2);
tree cons_tree(el_type e, tree l, tree r);
tree ord_ins(el_type e, tree t);
void inorder_write(tree t,FILE *f1);

main()
{  tree T;
   FILE *f1,*f2;
   el_type e1,e2;
   char s[30];
   int trovato;

   /*domanda 1: */
   T=NULL;
   f1=fopen ("parole.txt","r");
   f2=fopen ("escluse.txt","r");

   fscanf(f1,"%s", &e1.parola);
   while (!feof(f1))
   {  trovato=0;
      rewind(f2);
      fscanf(f2, "%s", &e2.parola);
      while(!feof(f2) && !trovato)
         {
         if (uguale(e1,e2))
            trovato=1;
         else fscanf(f2,"%s",&e2.parola);
         }
      if (!trovato)
         T=ord_ins(e1,T);
      fscanf(f1,"%s", &e1.parola);
      }
   fclose(f1);
   fclose(f2);
```



```
/* domanda 2 */
   f1=fopen("parole.txt", "w");
   inorder_write(T,f1);
   fclose(f1);
}

boolean uguale(el_type e1,el_type e2)
{  return (!strcmp(e1.parola,e2.parola)); }

boolean minore(el_type e1, el_type e2)
{return (strcmp(e1.parola,e2.parola)<0||
         !strcmp(e1.parola,e2.parola));}

tree ord_ins(el_type e, tree t)
{  if (t==NULL)
      return
          cons_tree(e, NULL, NULL);
   else
      if (minore(e,t->value))
         return
           cons_tree(t->value,
              ord_ins(e,left(t)),right(t));
      else
         return
           cons_tree(t->value,left(t),
                     ord_ins(e,right(t)));
}

tree cons_tree(el_type e, tree l, tree r)
{ tree t;
  t=(NODO *)malloc(sizeof(NODO));
  t->value=e;
  t->left=l;
  t->right=r;
  return t; }

void inorder_write(tree t,FILE *f1)
{
   if (!empty(t))
      {inorder_write(left(t),f1);
       fprintf(f1,"%s\n",root(t).parola);
```

```
        inorder_write(right(t),f1);
      }
   return;
}
```


Seconda soluzione:

Suddivido il programma in moduli, rappresentati dai seguenti file:

el_tree.c

el_tree.h

tree.c

tree.h

main.c

File el_tree.h

```
#include <stdio.h>

typedef struct{char parola[30];} el_type;
typedef int boolean;

void shownode(el_type e);
boolean minore(el_type e1, el_type e2);
boolean uguale(el_type e1,el_type e2);
```

File el_tree.c

```
#include "el_tree.h"

/* albero di interi*/

void shownode(el_type e)
{ printf("%s\t", e.parola);
}

boolean minore(el_type e1, el_type e2)
{return (strcmp(e1.parola,e2.parola)<0||
         !strcmp(e1.parola,e2.parola));}


boolean uguale(el_type e1,el_type e2)
{  return (!strcmp(e1.parola,e2.parola)); }
```


File tree.h

```
#include "el_tree.h"

typedef struct nodo {
          el_type value;
          struct nodo *left, *right;
       } NODO;

typedef NODO *tree;

tree emptytree();
 /* operazioni primitive esportate: */

tree emptytree();
boolean empty(tree t);
el_type root(tree t);
tree left(tree t);
tree right(tree t);
tree cons_tree(el_type e, tree l, tree r);

/* operazioni non primitive esportate */

void preorder(tree t);
void postorder(tree t);
void inorder(tree t);
tree ord_ins(el_type e, tree t);
boolean member(el_type e, tree t);
boolean member_ord(el_type e, tree t);
```

file tree.c

```
#include <stdlib.h>
#include "tree.h"

tree emptytree()
{  return NULL; }

/* operazioni primitive esportate: */
boolean empty(tree t)
{ return (t==NULL);}

el_type root(tree t)
{ if (empty(t))
       exit(0);
  else return (t->value); }

tree left(tree t)
{  if (empty(t))
       exit(0);
   else return (t->left); }

tree right(tree t)
{  if (empty(t))
       exit(0);
   else return (t->right); }

tree cons_tree(el_type e, tree l, tree r)
{ tree t;
  t=(NODO *)malloc(sizeof(NODO));
  t->value=e;
  t->left=l;
  t->right=r;
  return t;
}
```



```
/* operazioni non primitive esportate */

void preorder(tree t)
{
   if (!empty(t))
       {  preorder(left(t));
          preorder(right(t));
          shownode(root(t));
       }
   return;
}

void postorder(tree t)
{
   if (!empty(t))
       {  shownode(root(t));
          postorder(left(t));
          postorder(right(t));

       }
   return;
}

void inorder(tree t)
{
   if (!empty(t))
       {  inorder(left(t));
          shownode(root(t));
          inorder(right(t));

       }
   return;
}
```

```
tree ord_ins(el_type e, tree t)
{
   if (empty(t))
      return
          cons_tree(e, emptytree(),
                       emptytree());
   else
      if (minore(e,root(t)))
         return
           cons_tree(root(t),
              ord_ins(e,left(t)),right(t));
      else
         return
           cons_tree(root(t),left(t),
                     ord_ins(e,right(t)));
}

boolean member(el_type e, tree t)
{
   if (empty(t))
      return 0;
   else
      if (uguale(e, root(t)))
         return 1;
      else
         return( member(e,left(t))||
                 member(e, right(t)));
}
```



```
boolean member_ord(el_type e, tree t)
{
   if (empty(t))
      return 0;
   else
      if (uguale(e, root(t)))
         return 1;
      else
         if (minore(e,root(t)))
            return member_ord(e, left(t));
         else
            return member_ord(e, right(t));
}
```

File main.c

```
#include"tree.h"
void inorder_write(tree t,FILE *f1);

main()
{   tree T;
    FILE *f1,*f2;
    el_type e1,e2;
    char s[30];
    int trovato;

    /*domanda 1: */
    T=emptytree();
    f1=fopen ("parole.txt","r");
    f2=fopen ("escluse.txt","r");

    fscanf(f1,"%s", &e1.parola);
    while (!feof(f1))
    {   trovato=0;
        rewind(f2);
        fscanf(f2, "%s", &e2.parola);
        while(!feof(f2) && !trovato)
            {
            if (uguale(e1,e2))
               trovato=1;
            else fscanf(f2,"%s",&e2.parola);
            }
        if (!trovato)
            T=ord_ins(e1,T);
        fscanf(f1,"%s", &e1.parola);
        }
    fclose(f1);
    fclose(f2);
    /* domanda 2 */
    f1=fopen("parole.txt", "w");
    inorder_write(T,f1);
    fclose(f1);
}
```



```
void inorder_write(tree t,FILE *f1)
{
    if (!empty(t))
        {inorder_write(left(t),f1);
         fprintf(f1,"%s\n",root(t).parola);
         inorder_write(right(t),f1);
        }
    return;
}
```

ESERCIZIO n° 2

Esercizio n. 2 - 16 Febbraio 2007

Due file di tipo testo contengono dati relativi a film (FILM.TXT) e registi (DIRECTOR.TXT). Il primo file contiene per ciascun film, su una riga titolo del film (stringa di 20 caratteri), nome del regista (stringa di 20 caratteri) e costo (intero). Il secondo file contiene per ciascun regista, su una riga il nome (stringa di 20 caratteri), l'anno di nascita e l'anno di morte (interi, se vivente l'anno di morte vale 0).

Si scriva un programma C **strutturato in (almeno) tre funzioni** che svolgono i seguenti punti:

a) crea un albero binario di ricerca T, ordinato in base al nome del regista, che contiene in ciascun nodo: nome del regista, titolo del film e anno di morte del regista;
b) legge a terminale il nome di una persona e, accedendo all'albero T, determini quanti film ha diretto come regista;
c) accedendo a T stampi a video l'elenco ordinato dei registi viventi (anno di morte uguale a 0).

È possibile utilizzare ***librerie C*** (ad esempio per le stringhe).

Nel caso si strutturi a moduli l'applicazione qualunque ***libreria utente*** va riportata nello svolgimento.

Soluzione:

```
/* file el.h */
typedef struct {   char titolo[20];
                       char nome[20];
                       int costo;} tipo_film;
typedef struct {   char nome[20];
                       int nascita;
                       int morte;}tipo_regista;
typedef struct {   char nome[20];
                       char titolo[20];
                       int morte;} el_type

typedef enum{FALSO, VERO} boolean;
```



```
boolean isequal(tipo_regista, tipo_film);
boolean isless (el_type , el_type)
el_type crea_el_type(tipo_regista, tipo_film);
void showel_vivo(el_type);
```

```
/* file el.c */
#include <string.h>
#include "el.h"

el_type crea_el_type(tipo_regista reg, tipo_film film)
{   el_type el;
    strcpy(el.nome, reg.nome);
    strcpy(el.titolo, film.titolo);
    el.morte = reg.morte;
    return el; }

boolean isequal(tipo_regista el1, tipo_film el2)
{   return !strcmp(el1.nome, el2.nome); }

boolean isless(el_type el1, el_type el2)
{   if (strcmp(el1.nome, el2.nome) < 0)
    return TRUE;
    else return FALSE; }

void showel_vivo(el_type el)
{   if (el.morte == 0) printf("%s\n", el.nome);}
```

```
/* file tree.h */
#include "el.h"

typedef struct nodo
{   el_type value;
    nodo *left;
    nodo *right;
} nodo;

typedef nodo* tree;
tree emptytree();
boolean empty(tree T);
el_type root(tree t);
tree left (tree t);
tree right (tree t);
tree cons_tree(el_type e, tree l, tree r);
tree ord_ins(el_type e, tree t);
void inorder (tree t);
int ricerca_numero_film(tree T, char nome[],
                                        int numero);
```



```
/* file tree.c */
#include "tree.h"
#include "el.h"

tree emptytree()
{   return NULL;
}
boolean empty(tree T)
{   return (T == NULL);
}
el_type root(tree t)
{   if (!empty(t)) return (t->value);
    else
    {   printf("Albero vuoto\n");
        exit(-1);
    }
}
tree left (tree t)
{   if (empty(t)) return(NULL);
    else return(t->left);
}

tree right (tree t)
{   if (empty(t)) return(NULL);
    else return(t->right);
}

tree cons_tree(el_type e, tree l, tree r)
{   tree t;
    t = (nodo *) malloc(sizeof(nodo));
    t-> value = e;
    t-> left = l;
    t-> right = r;
    return t;
}
```

```
tree ord_ins(el_type e, tree t)
{   if (empty(t))
      return(cons_tree(e, emptytree(), emptytree()));
    else
    {   if (isless(e, root(t)))
           return(cons_tree(root(t),
                        ord_ins(e,left(t)), right(t)));
       else return (cons_tree(root(t),
                       left(t), ord_ins(e, right(t))));
    }
}

void inorder(tree t)
{   if (!empty(t))
    { inorder(left(t));
      showel_vivo(root(t));
      inorder(right(t));
    }
}


int  ricerca_numero_film  (tree  T,  char  nome[],  int
numero)
/* versione tail ricorsiva */
{   if (empty(T)) return numero;
    else
    { if ((strcmp(root(T).nome, nome) == 0)
          numero++;
      if ((strcmp(root(T).nome, nome) < 0)
{   return   ricerca_numero_film   (T->right,   nome,
numero);   }
      else
  return ricerca_numero_film (T->left, nome, numero);
    }
}
```



```
/* file main.c */
#include "tree.h"
#include <stlib.h>
#include <stdio.h>
#include <string.h>
void ricerca(tree T);

void main (void)
{   tree T = emptytree();
    FILE * f1, f2;

    /* Domanda A */
    f1 = fopen("FILM.TXT", "rt");
    f2 = fopen("DIRECTOR.TXT", "rt");
    if ((f1 == NULL) || (f2 == NULL)) exit(-1);
    else
        T=creafile(f1,f2,T);

    /* Domanda B */
    ricerca(T);

    /* Domanda C */
    printf ("Registi ancora vivi\n");
    inorder(T);
}
```

```
tree creafile (FILE *f1,FILE * f2, tree T)
{ tipo_film film;
  tipo_regista regista;
  el_type el;
  boolean trovato;

  while (f1 != EOF)
  {  fscanf(f1,  "%s  %s  %d",  film.titolo,  film.nome,
&film.costo);
    rewind(f2);
    trovato=FALSO;
    while ((f2 != EOF)&& !trovato)
     {   fscanf(f2,   "%s   %d   %d",   regista.nome,
&regista.nascita, &regista.morte);
      if (isequal(regista, film))
       {   el = crea_el_type(regista, film);
           T=ord_ins(el, T);
           trovato = TRUE;   }
     }
   }
}

void ricerca(tree T)
{   char nome[20];
    printf("Inserisci il regista da ricercare: ");
    scanf("%s", nome);
    printf ("Il regista %s ha diretto %d film.\n",
            nome, ricerca_numero_film(T, nome, 0));
}
```


Esempio di prova scritta (A.A. 2002-03 e successivi):

Esercizio 1:

Dato il seguente codice in linguaggio C, dove il tipo **`list`** rappresenta una lista di interi e **`cons`** la funzione di inserimento in testa a una lista:

```
#include <stdio.h>
. . .
void main(void)
{ int i, trovato=0;;
  list L=NULL;

  do
     { scanf("%d",&i);
       L=cons(i,L);  }
  while (i>0);

  scanf("%d",&i);

  while ( (L!=NULL) && !trovato) /* (1) */
      if (i==L->value) trovato=1;
      else L=L->next;

  if (trovato)
      printf(‘’%d’’,L->value);

}
```

Si indichi cosa fa questo frammento di programma;

Se ne indichi la complessità in termini di numero di test condizionali eseguiti rispettivamente dall'istruzione **`while`** (test sottolineato), ipotizzando che da input siano stati inseriti i valori da N sino a 0 e successivamente il valore 0.

SOLUZIONE Es. n° 1:

Il programma, tramite un ciclo do, acquisisce da input una sequenza di interi positivi (terminata da un valore nullo o minore di 0) e la inserisce in una lista (inserimento in testa). Legge poi un valore intero e lo cerca nella lista.

Il numero di test condizionali eseguiti dall'istruzione **`while`** (test sottolineato), ipotizzando che da input siano stati inseriti i valori da N sino a 0 e successivamente il valore 0, è pari a (N+1)+1. Infatti, la lista ha N+1 elementi e il test del ciclo while viene eseguito un numero di volte pari alla lunghezza della lista (N+1) più 1.

Esercizio 2:

Un file di testo (CORSI.TXT) contiene l'elenco dei corsi attivati in una facoltà universitaria, uno per linea, ciascuno seguito, sulla linea successiva, da un numero intero che rappresenta il numero di frequentanti il corso.

Esempio:

CORSI.TXT:
Matematica
30
Geometria
50
Fisica
10

Si scriva un programma in C che:

- Legga il contenuto del file e inserisca i nomi dei corsi e il numero di frequentanti in una lista L1 ordinata in base al numero;
- A partire da L1, crei un file testo (ORDINATI.TXT), dove i corsi sono scritti ordinati in base al numero di iscritti

Esempio:

ORDINATI.TXT:
Fisica
10
Matematica
30
Geometria
50

Qualunque ***libreria utente*** utilizzata va riportata o inserita nello svolgimento.

SOLUZIONE Es. n° 2:

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

typedef struct {int numero;
                char nome[50];   }
                              element_type;

typedef struct   list_element
     { element_type value;
       struct list_element  *next;
      } item;

typedef  item* list;


/* PROTOTIPI */
int isless(element_type, element_type);
list ordins(element_type, list);

void main(void)
{element_type    e;
 list   L;
 FILE *f1;
 FILE *f2;
```



```
/* DOMANDA a */
 f1 = fopen("CORSI.TXT", "rt");
 L=NULL;
 while (!feof(f1))
   {
      fscanf(f1,"%s", &e.nome);
      fscanf(f1,"%d", &e.numero);
      L=ordins(e, L);
    }
 fclose(f1);


/* DOMANDA b */
f2 = fopen("ORDINATI.TXT", "wt");
while (L!=NULL)
   {
       fprintf(f1,"%s\n", (L->value).nome);
       fprintf(f1,"%d\n",(L->value).numero);
       L= L->next;
   }
}
       /*fine main*/



/* FUNZIONI AUSILIARIE */


int isless(element_type e1, element_type e2)
      /* relazione d'ordine sulla parola */
{return (strcmp(e1.parola, e2.parola)<0);}
```

```
list   ordins(element_type el, list root)
{element_type  e;
 int trovato=0;
 list aux, prec=NULL, current;
          /* inserimento ordinato con
             possibili duplicazioni */
 aux=(list)malloc(
          sizeof(struct list_element));
  aux->value = el;
  aux->next = NULL;
  if (root==NULL)    /* primo elemento */
       root=aux;
  else
   {current=root;
    while((current!=NULL)&& !trovato)
       while (isless(current->value,el))
          { prec=current;
            current=current->next; }
    if ((current==NULL) && (prec !=NULL))
         {prec->next=aux;}   /*in fondo */
    else
         if (current==root) /* prec==NULL */
             {aux->next=root;
              root=aux;}    /* in testa */
         else
             {prec->next=aux;
                    /* inserisce in mezzo */
              aux->next=current;}
       }
   }
 return root;
}
```