# ESERCIZI LISTE

# Esercizio: liste generiche

Un file binario (MEMO.DAT) contiene i dati relativi agli **appuntamenti** giornalieri di uno studio legale. Ciascun appuntamento è caratterizzato dal cognome della persona da incontrare (al più di 20 caratteri) e l'ora (numero intero da 7 a 20). Si realizzi un programma C che:

1. Costruisca in memoria centrale una **lista** che memorizzi, in modo **ordinato** in base ai cognomi, i nominativi delle persone che saranno ricevute e l'ora prevista.
2. Letto un cognome a terminale, utilizzi la lista per andare a visualizzare a quale ora tale persona sarà ricevuta.
3. Stampi a video l'elenco dei nominativi (uno per linea) contenuti nella lista, ciascuno preceduto dall'ora del proprio appuntamento.

# Esercizio: liste generiche

Ad esempio: Contenuto di MEMO.DAT:

| | |
|---|---|
| VERDI | 17 |
| ROSSI | 7 |
| BIANCHI | 9 |
| BLU | 7 |
| NERI | 17 |

Risultato della stampa del punto 3:

| | |
|---|---|
| 9 | BIANCHI |
| 7 | BLU |
| 17 | NERI |
| 7 | ROSSI |
| 17 | VERDI |

# Schema della soluzione: liste generiche

Suddivido il programma nei seguenti file:

- **list.c** funzioni di libreria per la gestione di liste
- **list.h** header file associato a list.c
- **el.c** funzioni di utilità dipendenti dalla rappresentazione di element_type
- **el.h** header file associato ad el.c (contiene la dichiarazione di element)
- **main.c** contiene il programma principale (funzione main)

# el.h

```
/* LIST ELEMENT TYPE - file el.h */
typedef struct {
   char cognome[20];
   int ora;
} element_type;

typedef enum {false, true} bool;
bool isequal(element_type, element_type);
bool isless(element_type, element_type);
void showel (element_type);
```

# el.c

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

// uguaglianza sul cognome
bool isequal(element_type e1, element_type e2) {
   return !strcmp(e1.cognome, e2.cognome);
}

// relazione d'ordine sul cognome
bool isless(element_type e1, element_type e2) {
   return (strcmp(e1.cognome, e2.cognome) < 0);
}

//mostra informazioni dell'elemento
void showel (element_type e) {
   printf("%s\t%d\n", e.cognome, e.ora);
}
```

# list.h

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

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

// PROTOTIPI DI FUNZIONE
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.c

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

list emptylist() {
   return NULL;
}

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

element_type head(list l) {
   if (empty(l))
      exit(1);
   else return(l->value);
}
```

# list.c

```
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)); //controlli
   t->value=e;
   t->next=l;
   return(t);
}
```

# list.c

```
// inserimento ordinato con possibili duplicazione

list ordins(element_type el, list l){
   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.c

```
element_type member (element_type el, list l)
{
  if  (!empty(l))
     if  (isequal(el, head(l)))
        return head(l);
    else
        return member(el, tail(l));
  else
     exit(1);
}
```

# main.c

```
/* PROGRAMMA PRINCIPALE - file main.c */
#include <stdio.h>
#include <stdlib.h>
#include "list.h"

main() {
   element_type e;
   list L, L1;
   FILE *f1;
   char C[20];
   int orario, i;

   L=emptylist();
```

# main.c

```
f1 = fopen("MEMO.DAT", "w");

do {
   printf("Inserisci cognome (fine per terminare):");
   scanf("%s",C);
   if (strcmp(C, "fine")==0)
      break;
   printf("Inserisci ora: ");
   scanf("%d", &orario);
   strcpy(e.cognome, C);
   e.ora=orario;
   fwrite(&e, sizeof(element_type), 1, f1);
} while (1);

fclose(f1);
```

# main.c

```
  f1 = fopen("MEMO.DAT", "rb");                    // DOMANDA 1
  while (fread(&e, sizeof(element_type), 1, f1) > 0)
     L = ordins(e, L);
  fclose(f1);

  printf("\n Dammi il cognome da cercare: ");       // DOMANDA 2
  scanf("%s", e.cognome);
  L1=L;
  while (!empty(L1)){
     if (isequal(head(L1), e))
        showel(head(L1));
     L1 = tail(L1);
  }
  while (!empty(L)){                               // DOMANDA 3
     printf("%d\t%s\n", (L->value).ora, (L->value).cognome);
     L=tail(L);
  }
}
```

# Esercizio: liste di interi

Un file di testo (TEMP.TXT) contiene i dati relativi alle medie (int) di tutti gli **studenti** che devono accedere ad una sessione di laurea. Si realizzi un programma C che:

1. Costruisca in memoria centrale una lista che memorizzi tali medie in modo ordinato crescente e la stampi.
2. Letti due valori interi N e M, utilizzando la lista, visualizzi il valore delle medie comprese fra N e M ed un opportuno messaggio se non ne esistono.

Ad esempio, contenuto di TEMP.DAT:

90
100
98
110
88
87

intervallo

88 101

stampa

88 90 98 100

# Schema della soluzione: liste di interi

Suddivido il programma nei seguenti file:

- **list.c** funzioni di libreria per la gestione di liste
- **list.h** header file associato a list.c
- **el.c** funzioni di utilità dipendenti dalla rappresentazione di element_type
- **el.h** header file associato ad el.c (contiene la dichiarazione di element)
- **main.c** contiene il programma principale (funzione main)

# main.c

```
/* PROGRAMMA PRINCIPALE - file main.c */
#include <stdio.h>
#include <stdlib.h>
#include "list.h"

main(){
   int e, min, max;
   list L=emptylist();
   list L1;
   FILE *f1;
   int i;

/* DOMANDA 1 */
   f1 = fopen("TEMP.TXT", "r");
   while (fscanf(f1,"%d", &e) !=EOF )
      L=ordins(e, L);
   showlist(L);
   fclose(f1);
```

# main.c

```
/* DOMANDA 2 */
  printf("Dammi i due estremi (66-110): ");
  scanf("%d%d", &min, &max);
  L1=L;
  while (!empty(L1) && (head(L1)<min))
    L1=tail(L1);
  if  (empty(L1))
    printf("\nnessun valore");
  else{
    printf("\n");
    while ( !empty(L1) && (head(L1)<max)){
      printf("%d ", head(L1));
      L1=tail(L1);
    }
  }
  fclose(f1);
}
```

# Esercizio: liste generiche

Un file di testo ARCHIVIO.TXT contiene i dati (primo autore, titolo, numero di copie possedute, numero di copie in prestito) relativi ai differenti volumi conservati presso una **biblioteca**. Più precisamente, ogni riga del file contiene nell'ordine, separati da uno spazio bianco:

- autore (non più di 20 caratteri senza spazi intermedi);
- titolo (non più di 50 caratteri senza spazi intermedi);
- numero_possedute (int);
- numero_prestito (int).

Si realizzi un programma C che:

1. Legga il contenuto di ARCHIVIO.TXT e costruisca in memoria centrale un **vettore V di strutture** corrispondenti (si supponga che il file ARCHIVIO.TXT non possa contenere più di 30 righe). Si stampi a video il contenuto del vettore.
2. A partire da V, costruisca una **lista** L di interi contenente per ciascun volume il numero di copie disponibili nella biblioteca, ovvero la differenza fra il numero di copie possedute e il numero di copie in prestito. Si stampi a video il contenuto della lista L.

# Esercizio: liste generiche

3. Utilizzando L per ottenere la somma delle copie disponibili e V per la somma delle copie possedute, calcoli il rapporto fra volumi disponibili e volumi posseduti.

4. Utilizzando la lista di interi L, stampi il numero di riga di ARCHIVIO.TXT relativo al volume con più copie disponibili. In caso di più volumi con pari numero di copie disponibili, qualunque riga relativa a questi ultimi è considerata una risposta corretta.

# Esercizio: liste generiche

Ad esempio: Contenuto di ARCHIVIO.TXT:

Platone LaRepubblica 10 8
Aristotele Politica 12 3
Cartesio MeditazioniMetafisiche 12 12
Locke LetteraSullaTolleranza 6 0
Kant CriticaDellaRagionPratica 9 9
Hobbes Leviatano 3 1
Rousseau Emilio 7 7

Stampa di L:
[2, 9, 0, 6, 0, 2, 0]

# Schema della soluzione: liste generiche

Suddivido il programma nei seguenti file:

- **list.c** funzioni di libreria per la gestione di liste
- **list.h** header file associato a list.c
- **el.c** funzioni di utilità dipendenti dalla rappresentazione di element_type
- **el.h** header file associato ad el.c (contiene la dichiarazione di element)
- **main.c** contiene il programma principale (funzione main)

# main.c

```
/* PROGRAMMA PRINCIPALE - file mainLibri.c */
#include <stdio.h>
#include <stdlib.h>
#include "list.h"
#define MAX 20

typedef  struct{
   char autore[21];
   char titolo[51];
   int possedute;
   int prestito;
} volume;
```

```
main() {
   volume e;
   list L, L1;
   FILE *f;
   volume V[MAX];
   int elementi=0, i, pos, max;
   int somma_possedute=0, somma_disponibili=0;
   L=emptylist();
```

# main.c

```
/* DOMANDA 1 */
  f = fopen("ARCHIVIO.TXT", "r");
  if (f==NULL) {
    printf("Impossibile aprire file di ingresso");
    exit(1);
  }
  while (fscanf(f, "%20s%50s%d%d\n", e.autore, e.titolo, &e.possedute, &e.prestito)>0)
    V[elementi++] = e;
  fclose(f);
  for (i=0; i<elementi; i++)
    printf("Volume %d: %s\t%s\t%d\t%d\n",i,V[i].autore, V[i].titolo, V[i].possedute,
V[i].prestito);

/* DOMANDA 2 */
  for (i=0; i<elementi; i++)
    L = cons(V[i].possedute-V[i].prestito,L);
  showlist(L);
```

# main.c

```
/* DOMANDA 3 */
   for (i=0; i<elementi; i++)
      somma_possedute += V[i].possedute;
   L1=L;
   while (!empty(L1)){
      somma_disponibili += head(L1);
      L1=tail(L1);
   }
   printf("Rapporto disponibili/possedute = %f\n", (float)somma_disponibili
/somma_possedute);
```

# main.c

```
/* DOMANDA 4 */
  i=0;
  max=-1;
  L1=L;
  while (!empty(L1)){
    if (head(L1)>max) {
      max = head(L1);
      pos=i;
    }
    L1=tail(L1);
    i++;
  }
  printf("Volume con più copie disponibili: %d", elementi-pos-1);
}
```

# Esercizio: liste generiche

È dato un file di testo CANZONI.TXT che contiene i dati di una serie di canzoni (non più di 20), una canzone per riga. Più precisamente, ogni riga contiene:

- autore (non più di 20 caratteri, senza spazi intermedi);
- uno e un solo spazio;
- titolo del brano (non più di 30 caratteri, senza spazi intermedi);
- uno e un solo spazio;
- durata in secondi (numero intero).

Scrivere un programma C che, dopo aver definito una struttura *brano*:

1. contenga una funzione *leggi_brani()* che, dato il nome del file, legga i dati delle canzoni dal file e li metta in un array di *brano* di nome *vettoreb*. Si mostri a video l'array così costruito.
2. Chieda all'utente il nome di un autore e costruisca una lista di interi i cui elementi sono dati dalla durata dei brani dell'autore indicato (si supponga che l'autore inserito dall'utente sia presente nell'elenco). Si visualizzi la lista costruita.
3. Contenga una funzione *media(list listab)* con la quale visualizzi la media della durata dei brani contenuti nella lista.

# main.c

```
#include <stdio.h>
#include <stdlib.h>
#include "list.h"
#define NUMEROBRANI 20
#define DIMAUTORE 21
#define DIMTITOLO 31

typedef  struct{
   char autore[DIMAUTORE];
   char titolo[DIMTITOLO];
   int durata;
} brano;
```

# main.c

```
/* ---------- domanda 1 -------- */
void leggi_brani(char nomefile[], brano v[], int* pindice){
   brano x;
   FILE *f = fopen(nomefile, "r");
   if (f==NULL) {
      printf("Impossibile aprire file di ingresso");
      exit(1);
   }
   while (fscanf(f, "%s%s%d\n", x.autore, x.titolo, &x.durata)>0) {
      v[*pindice] = x;
      (*pindice)++;
   }
   fclose(f);
}
```

# main.c

```
void mostraVettoreBrani(brano elenco[], int dim){
   int i;
   for (i=0; i<dim; i++)
      printf("\nautore:%s\ntitolo:%s\ndurata:%ds\n", elenco[i].autore,
elenco[i].titolo, elenco[i].durata);
}

double media(list listab) {                    // DOMANDA 3
   double ris = 0.0;
   int n = 0;
   list listaux;
   listaux = listab;
   while (!empty(listaux)) {
      ris = ris + head(listaux);
      listaux = tail(listaux);
      n++;
   }
   return ris/n;
}
```

# main.c

```
main(){
   brano vettoreb[NUMEROBRANI];
   list listab = emptylist();
   int dim = 0, i;
   FILE *f;
   char autore[DIMAUTORE];

// DOMANDA 1
   leggi_brani("CANZONI.TXT", vettoreb, &dim);
   mostraVettoreBrani(vettoreb, dim);
```

# main.c

```
// DOMANDA 2
   printf("\nInserire l'autore: ");
   scanf("%s",autore);
   for (i=0; i<dim; i++)
      if  (strcmp(vettoreb[i].autore,autore)==0)
         listab = cons(vettoreb[i].durata, listab);
   showlist(listab);

// DOMANDA 3
   printf("\nLa durata media dei brani e': %lf\n", media(listab));
}
```

# Gestione testo tramite liste

**Esercizio**: **gestione di un testo**

Dato un file di testo, trasformarlo spezzando tutte le righe di lunghezza superiore a *dim* in sequenze di righe di lunghezza *dim*.

Trasformare poi il testo in una sequenza di righe tutte di dimensione *dim*, tranne, eventualmente, l'ultima

# Gestione testo tramite liste

```
#include <stdio.h>
#include <stdlib.h>
typedef int bool;
#define TRUE  1
#define FALSE 0

struct StructCar {                    /* rappresentazione della lista di caratteri */
  char car;
  struct StructCar *next_car;         /* puntatore al carattere successivo */
};

typedef struct StructCar TipoCar;
typedef TipoCar *TipoPuntCar;

struct StructRiga {                   /* rappresentazione della lista di righe */
  TipoPuntCar inizio;                 /* puntatore iniziale alla lista di caratteri */
  struct StructRiga *next_riga;       /* puntatore alla riga successiva */
};

typedef struct StructRiga TipoRiga;
typedef TipoRiga *TipoPuntRiga;

typedef TipoPuntRiga TipoTesto;
```

# Gestione testo tramite liste

```
void SpezzaRiga(TipoPuntRiga riga, int dim)
  /* Spezza una riga in una parte di lunghezza dim ed in un resto. */
{
  TipoPuntCar car_corr;                 /* puntatore al car. corrente nella riga corrente */
  TipoPuntRiga resto;                   /* puntatore usato per creare la riga con il resto della riga spezzata */
  int num_car;                          /* contatore del numero di caratteri nella riga corrente */

  car_corr = riga->inizio;
  if (car_corr != NULL) {                /* la riga non è vuota */
    for (num_car = 1;
        car_corr->next_car != NULL && num_car < dim;
                    /* la riga corrente non è terminata e non si è ancora raggiunto il carattere in posizione dim */
        num_car++)
                car_corr = car_corr->next_car;

    if (car_corr->next_car != NULL) {           /* la riga corrente viene spezzata */
      resto = malloc(sizeof(TipoRiga));         /* alloca il nodo iniziale per una nuova riga + controlli*/
      resto->inizio = car_corr->next_car;       /* la nuova riga contiene il resto della riga spezzata */
      car_corr->next_car = NULL;                /* chiude la riga spezzata */

       /* inserisce la riga con il resto prima della prossima riga di testo */
      resto->next_riga = riga->next_riga;
      riga->next_riga = resto;
    }
  }
}  /* SpezzaRiga */
```

# Gestione testo tramite liste

```
void RidimensionaTesto(TipoTesto *testo, int dim)
  /* Trasforma testo spezzando tutte le righe di lunghezza superiore a dim in
     sequenze di righe di lunghezza dim. */
{
  TipoPuntRiga riga_corr;

  riga_corr = *testo;
  while (riga_corr != NULL) {
    SpezzaRiga(riga_corr, dim);                 /* spezza la riga corrente */
    riga_corr = riga_corr->next_riga;           /* si posiziona sulla riga successiva */
  }
}  /* RidimensionaTesto */
```

# Gestione testo tramite liste

```
void CancellaRigheIniziali Vuote(TipoTesto *testo)
  /* Cancella da testo tutte le righe iniziali vuote. */
{
  TipoPuntRiga raux;
  bool continua = TRUE;        /* indica se si deve continuare a cancellare */

  while (*testo != NULL && continua) {
    if ((*testo)->inizio == NULL) {                    /* la riga è vuota */
      raux = *testo;
      *testo = (*testo)->next_riga;
      free(raux);
    }
    else   /* si è arrivati ad una riga non vuota e si smette di cancellare */
      continua = FALSE;
  }
}  /* CancellaRigheIniziali Vuote */
```

# Gestione testo tramite liste

```
void UnicaRiga(TipoTesto *testo)
  /* Trasforma testo collegando tutte le righe in un'unica riga. */
{
  TipoPuntCar car_corr;                 /* puntatore al carattere corrente  */
  TipoPuntRiga riga_succ;               /* ptr alla riga che segue quella con il carattere corrente */
  TipoPuntRiga raux;

  CancellaRigheIniziali Vuote(testo);
  if (*testo != NULL) {                                          /* testo punta alla prima riga non vuota */
            car_corr = (*testo)->inizio;
            riga_succ = (*testo)->next_riga;
            while (riga_succ != NULL) {                          /* esistono ancora righe successive */
                      while (car_corr->next_car != NULL) {
                                          /* non si e` ancora alla fine della riga */
                                car_corr = car_corr->next_car;
                      }
                      car_corr->next_car = riga_succ->inizio;  /* collega la riga successive in coda a quella corrente  */

                    /* cancella il nodo di riga relativo alla riga successiva */
                      raux = riga_succ;
                      riga_succ = riga_succ->next_riga;
                      free(raux);
           }
           (*testo)->next_riga = NULL;          /* testo termina dopo la prima riga */
}
}  /* UnicaRiga */
```

# Gestione testo tramite liste

```
void CompattaTesto(TipoTesto *testo, int dim)
  /* Trasforma testo in una sequenza di righe tutte di dimensione dim, tranne
     eventualmente l'ultima. */
{
  UnicaRiga(testo);
  RidimensionaTesto(testo, dim);
}  /* CompattaTesto */
```

# Gestione testo tramite liste

```
bool LeggiRiga(FILE *fi, TipoPuntCar *inizio_riga)
  /* Legge una riga di testo dal file fi e restituisce in *inizio_riga il puntatore iniziale alla lista di caratteri della
riga letta. Restituisce FALSE se si è giunti alla fine del file e la riga letta è vuota.  */
{
  TipoPuntCar caux;
  char ch;

  *inizio_riga = malloc(sizeof(TipoCar));
  caux = *inizio_riga;
  ch = getc(fi);
  while (ch != '\n' && ch != EOF) {
    caux->next_car = malloc(sizeof(TipoCar));
    caux = caux->next_car;
    caux->car = ch;
    ch = getc(fi);
  }
  caux->next_car = NULL;
  caux = *inizio_riga;
  *inizio_riga = (*inizio_riga)->next_car;
  free(caux);
  return (ch != EOF || *inizio_riga != NULL);
}  /* LeggiRiga */
```

# Gestione testo tramite liste

```
void LeggiTesto(char *nomefile, TipoTesto *testo)
  /* Restituisce un testo le cui righe vengono lette dal file di nome nomefile. Ogni riga è costituita dalla lista dei
caratteri della corrispondente riga nel file. */
{
  TipoPuntRiga raux;
  TipoPuntCar caux;
  FILE *fi;

  fi = fopen(nomefile, "r");
  if (fi == NULL) {
    fprintf(stderr, "Errore nell'apertura in lettura di %s", nomefile);
    exit (1);
  }

  *testo = malloc(sizeof(TipoRiga));       /* controlli */
  raux = *testo;
  while (LeggiRiga(fi, &caux)) {
    raux->next_riga = malloc(sizeof(TipoRiga));
    raux = raux->next_riga;
    raux->inizio = caux;
  }
  raux->next_riga = NULL;
  raux = *testo;
  *testo = (*testo)->next_riga;
  free(raux);
  fclose(fi);
}  /* LeggiTesto */
```

# Gestione testo tramite liste

```
void ScriviRiga(FILE *fi, TipoPuntCar riga)
  /* Scrive riga sul file fi. */
{
  while (riga != NULL) {
    putc(riga->car, fi);
    riga = riga->next_car;
  }
  putc('\n', fi);
}  /* ScriviRiga */
```

# Gestione testo tramite liste

```
void ScriviTesto(char *nomefile, TipoTesto testo)
  /* Scrive testo sul file di nome nomefile. */
{
  FILE *fi;

  fi = fopen(nomefile, "w");
  if (fi == NULL) {
    fprintf(stderr, "Errore nell'apertura in scrittura di %s", nomefile);
    exit (1);
  }

  while (testo != NULL) {
    ScriviRiga(fi, testo->inizio);
    testo = testo->next_riga;
  }
  fclose(fi);
}  /* ScriviTesto */
```

# Gestione testo tramite liste

```
void CancellaRiga(TipoPuntCar *riga)
  /* Cancella riga liberando la memoria occupata. */
{
  TipoPuntCar caux;

  while (*riga != NULL) {
    caux = *riga;
    *riga = (*riga)->next_car;
    free(caux);
  }
}  /* CancellaRiga */


void CancellaTesto(TipoTesto *testo)
  /* Cancella testo liberando la memoria occupata. */
{
  TipoPuntRiga raux;

  while (*testo != NULL) {
    CancellaRiga(&(*testo)->inizio);
    raux = *testo;
    *testo = (*testo)->next_riga;
    free(raux);
  }
}  /* CancellaTesto */
```

# Gestione testo tramite liste

```
int main(void)
{
  TipoTesto testo;
  int dim;

  do {
            printf("Immetti la lunghezza massima delle righe di testo: ");
            scanf("%d", &dim);
            if (dim <= 0)
              printf("La lunghezza massima deve essere un intero positivo.\n");
  } while (dim <= 0);
  LeggiTesto("dati.txt", &testo);
  RidimensionaTesto(&testo, dim);
  ScriviTesto("datiou1.txt", testo);
  CancellaTesto(&testo);
  LeggiTesto("dati.txt", &testo);
  CompattaTesto(&testo, dim);
  ScriviTesto("datiou2.txt", testo);
  CancellaTesto(&testo);
  return 0;
}  /* GestioneTesti */
```