Fondamenti di Informatica T-1
modulo 2

Obiettivi

- Liste
- Liste
- Liste
- ...
- Liste!

Esercizio 1
(liste)

- I risultati di un appello d'esame di Fondamenti di Informatica vengono ***salvati su un file (di testo)***
- Su tale file vengono salvati ***solo i voti, separati da spazi o da newline***
- Scrivere un programma che, facendo uso delle ***liste di elementi interi***:
  - legga i valori salvati nel file, li memorizzi all'interno della lista, e li stampi a video
  - chieda all'utente un valore di soglia
  - memorizzi in ***due liste differenti i valori superiori*** (o uguali) e quelli ***inferiori a tale soglia***, stampando poi a video il contenuto di entrambe le liste

Esercizio 1- Soluzione
(liste)

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

int main(void)
{   FILE * fp;
    List startList, lowList, highList, temp;
    Element voto, soglia;

    if ( (fp=fopen("voti.txt", "r")) == NULL)
    {       perror("The file does not exist!");
            system("PAUSE"); exit(-1);
    } …
```

Esercizio 1 - Soluzione
(liste)

```
…
startList = emptyList();
lowList = emptyList();
highList = emptyList();

while (fscanf(fp, "%d", &voto) > 0)
    startList = cons(voto, startList);
fclose(fp);

showList(startList);
printf("Inserire soglia: ");
scanf("%d", &soglia);
…
```

Esercizio 1 – Soluzione – NO PRIMITIVE

(liste)

```
…
while(startList != NULL)
{
     if (startList->value < soglia)
     {
            temp = (List) malloc(sizeof(Item));
            temp->value = startList->value;
            temp->next = lowList; lowList = temp;
     }
     else
     {
            temp = (List) malloc(sizeof(Item));
            temp->value = startList->value;
            temp->next = highList; highList = temp;
     }
     startList = startList->next;
}
```

Esercizio 1 – Soluzione – CON PRIMITIVE
(liste)

```
…
while(!empty(startList))
{
     Element value = head(startList);
     if(value < soglia)
            lowList = cons(value, lowList);
     else
            highList = cons(value, highList);
     startList = tail(startList);
}
showList(lowList);
showList(highList);
// DEALLOCAZIONE: ATTENZIONE!!!!!!
while(!empty(lowList)) {
     temp = lowList;
     lowList = tail(lowList);
     free(temp);
} //analoga deallocazione serve per highList
system("PAUSE"); return (0);
}
```

Esercizio 2
(liste)

Intersezione e differenza fra liste

- Si leggano da standard input ***due liste di interi positivi*** (l'utente terminerà l'inserimento di ognuna con il valore 0)
- Scrivere le seguenti funzioni:
  - `List intersect(List l1, List l2)` riceve due liste e ***restituisce una terza lista contenente i valori presenti in entrambe***, utilizzando le primitive
  - `List diff(List l1, List l2)` restituisce una ***lista contenente i valori presenti in l1 che NON sono presenti in l2*** (l1- l2), senza usare le primitive

- ***Modificare*** le soluzioni precedenti facendo in modo che la ***lista risultato NON contenga elementi ripetuti***
- Deallocare correttamente le liste utilizzate

Esercizio 2 - suggerimenti
(liste)

- Scomporre in sottoproblemi
- Realizzare una funzione per verificare ***se un elemento è contenuto in una lista***
- Realizzare intersezione come insieme degli elementi della prima lista contenuti anche nella seconda
- Realizzare la differenza come l'insieme degli elementi della prima lista NON contenuti nella seconda
- Semplice modifica per evitare ripetizioni nel risultato
  - Aggiungere elementi solo se non già contenuti

Esercizio 2 - Soluzione
(liste)

CON PRIMITIVE:

```
Boolean contains(List l, Element e)
{
   Boolean found = false;
   while(!empty(l) && !found)
   {
       found = (head(l) == e);
       l = tail(l);
   }
   return found;
}
```

Esercizio 2 - Soluzione
(liste)

SENZA PRIMITIVE:

```
Boolean contains(List l, Element e)
{
   Boolean found = false;
   while(l != NULL && !found)
   {
       found = (l->value == e);
       l = l->next;
   }
   return found;
}
```

Esercizio 2 - Soluzione
(liste)

```
List intersect(List l1, List l2)
{
   Element cur;
   List intersection = emptyList();
   while(!empty(l1))
   //converrebbe iterare sulla lista più corta...
   {
        cur = head(l1);
        if(contains(l2, cur))
        intersection = cons(cur, intersection);
        l1 = tail(l1);
   }
   return intersection;
}
```

&& !contains(intersection, cur)
per evitare elementi ripetuti nel risultato

Esercizio 2 - Soluzione

(liste)

```
List diff(List l1, List l2) {
   Element cur;
   List difference = NULL, temp;
   while(!empty(l1))
        {
        cur = l1->value;
        if(!contains2(l2, cur))
        {
              temp = (List) malloc(sizeof(Item));
              temp->value = cur;
              temp->next = difference;
              difference = temp;
        }
        l1 = l1->next;
   }
   return difference;
}
```

&& !contains2(difference, cur)
per evitare elementi ripetuti nel risultato

Esercizio 3
(liste)

Si scriva una funzione ricorsiva `crossSelection()` che, ricevute in ingresso due liste di interi positivi `l1` e `l2`, restituisca una terza lista (eventualmente non ordinata) contenente gli interi di `l2` che sono nelle posizioni indicate dai valori di `l1` (si assuma per convenzione che il primo elemento di una lista sia in posizione 1)

Ad esempio, date due liste: `l1=[1,3,4]` e `l2=[2,4,6,8,10,12]`, la lista risultante deve contenere gli elementi di `l2` che sono in prima, terza e quarta posizione, cioè: `[2,6,8]`

Esercizio 3
(liste)

A tal scopo si realizzi una funzione ricorsiva di supporto **`select()`** che, ricevuti in ingresso una lista e un intero positivo rappresentante una posizione, restituisca l'intero della lista posto alla posizione specificata. La funzione deve restituire -1 qualora l'intero passato non corrisponda a nessuna posizione valida (si assuma comunque positivo l'intero passato)

Le funzioni **`crossSelection()`** e **`select()`** devono essere realizzate in modo ricorsivo, utilizzando il tipo di dato astratto **`list`**.

Si possono utilizzare le sole operazioni primitive definite durante il corso (che quindi possono NON essere riportate nella soluzione). Non si possono usare altre funzioni di alto livello

Esercizio 3 - Soluzione
(liste)

```
// Versione con primitive
element select(list l, int pos)
{
      if (empty(l)) return -1;
      else if (pos == 1) return head(l);
      else return select(tail(l), pos-1);
}


list crossSelection(list l1, list l2)
{
      if (empty(l1))
            return emptylist();
      else
            return cons(select(l2, head(l1)),
            crossSelection(tail(l1), l2));
}
```

Esercizio 3 - Soluzione
(liste)

```
// Versione con puntatori
element select(list l, int pos) {
      if (l==NULL) return -1;
      else if (pos == 1) return l->value;
      else return select(l->next, pos-1);
}

list crossSelection(list l1, list l2) {
      list l;
      if (l1==NULL) return NULL;
      else
      {
            l=(list) malloc(sizeof(item));
            l->value=select(l2, l1->value);
            l->next= crossSelection(l1->next, l2));
            return l;
      }
}
```

Esercizio 4
(liste)

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`** (al più di 20 caratteri senza spazi)
- **`titolo`** (al più di 50 caratteri senza spazi)
- **`numero_possedute`** (intero)
- **`numero_prestito`** (intero)

Esercizio 4
(liste)

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 4
(liste)

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 4
(liste)

Ad esempio: contenuto di ARCHIVIO.TXT

Salinger IlGiovaneHolden 10 8
Wallace InfiniteJest 12 3
Carver Cattedrale 12 12
Baricco Seta 6 0
Hornby ComeDiventare 9 9
Sartre LaNausea 3 1
Robbins NaturaMorta 7 7

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

Esercizio 4 - Soluzione
(liste)

Suddivido il programma nei seguenti file:

- **`list.c`**: funzioni di libreria per la gestione di liste
- **`list.h`**: header file associato a list.c
- **`element.h`**: dichiarazione di element
- **`mainLibri.c`**: programma principale

Esercizio 4 - Soluzione
(liste)

```
/* PROGRAMMA PRINCIPALE – file mainLibri.c */

#include <stdio.h>
#include <stdlib.h>
#include "list.h"
#define MAX 20

typedef struct{
        char autore[20];
        char titolo[50];
        int possedute;
        int prestito;
} volume;
```

Esercizio 4 - Soluzione
(liste)

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


      // DOMANDA 1
      f = fopen("ARCHIVIO.TXT", "r");
      if (f==NULL)
      {
            printf("Impossibile aprire file di ingresso");
            exit(1);
      }
```

Esercizio 4 - Soluzione
(liste)

```
while (fscanf(f,"%s%s%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);
```

Esercizio 4 - Soluzione
(liste)

```
// DOMANDA 2
for (i=0; i<elementi; i++)
      L = cons(V[i].possedute-V[i].prestito,L);
showlist(L);
// in che ordine viene stampata la lista?


// 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);
```

Esercizio 4 - Soluzione
(liste)

```
        // DOMANDA 4
        i=0;
        max=-1;

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

Esercizio 5
(liste)

Una ditta di impianti elettrici tiene un registro dei lavori effettuati per i diversi clienti su diversi file binari e in un file di testo registra per ogni cliente il nome del file in cui sono registrati i lavori che lo riguardano

In particolare, per ogni lavoro effettuato la ditta salva su un file binario strutture dati di tipo **`Work`**: ogni struttura contiene il **nome del cliente** (stringa di al più 64 caratteri utili), il **giorno** in cui il lavoro è stato fatto (int che rappresenta il giorno nell'anno corrente) e l'**importo** ai fini della fatturazione (float)
I lavori relativi ad uno stesso cliente risiedono tutti nello stesso file, ma uno stesso file contiene i dati dei lavori relativi a più clienti

La ditta salva poi, su ogni riga di un file di testo che funge da "chiave", il nome del cliente (sempre una stringa di al più 64 caratteri utili) e, separato da uno spazio, il nome del file in cui i lavori del cliente sono salvati (stringa di al più 64 caratteri utili). Si deve realizzare un programma che calcoli l'ammontare delle fatture per tutti i clienti della ditta

Esercizio 5
(liste)

Esempio di contenuto del file di testo:

```
PaoloBellavista marzo.dat
PaolaMello aprile.dat
FedericoChesani marzo.dat
CarloGiannelli marzo.dat
Marco Montali giugno.dat
```

Realizzare:

```
list findBills(char * fileName, char * clientName);
```

che, ricevuti in ingresso il nome del file binario e il nome del cliente, restituisca una lista contenente i soli importi relativi ai lavori eseguiti per il cliente specificato (si presti attenzione al fatto che in uno stesso file ci possono essere i dati relativi a più clienti)

Si supponga a tal fine di poter disporre del tipo di dato astratto **`list`** visto a lezione, definito per il tipo primitivo **`float`**, e si supponga di disporre anche di tutte le funzioni primitive presentate a lezione

Esercizio 5
(liste)

Un programma **`main()`**, che chieda all'utente il nome del file di testo in cui sono registrate le coppie nome cliente-nome file; per ogni cliente registrato in tale file, il programma deve stampare a video il nome cliente, la lista importi relativi, e la loro somma

Al fine di stampare le liste, il candidato ipotizzi di avere a disposizione la funzione **`showlist(…)`** opportunamente modificata, e che quindi non deve essere riportata nella soluzione

```
#define DIM 65
typedef struct work
{
        char name[DIM];
        int day;
        float bill;
} Work;
```

Esercizio 5 - Soluzione
(liste)

```
list findBills(char * fileName, char * clientName)
{
    FILE * fWorks;
    list result;
    Work temp;
    if ((fWorks = fopen(fileName, "rb")) == NULL)
    {
        printf("Error opening the file %s\n", fileName);
        exit(-1);
    }
    result = emptylist();
    while (fread(&temp, sizeof(Work), 1, fWorks) > 0 )
        if (strcmp(temp.name, clientName) == 0)
                result = cons(temp.bill, result);
    fclose(fWorks);
    return result;
}
```

Esercizio 5 - Soluzione
(liste)

```
int main() {
       FILE * fClients; list tempBill; float totalBill;
       char fileClients[DIM], clientName[DIM], fileName[DIM];

       printf("Inserire il nome del file dei clienti: ");
       scanf("%s", fileClients);

       if ((fClients = fopen(fileClients, "r")) == NULL)
       {
           printf("Error opening the file %s\n", fileClients);
           exit(-1);
       }
```

Esercizio 5 - Soluzione
(liste)

```
        while (fscanf(fClients, "%s %s",
                                    clientName, fileName) != EOF )
        {
                totalBill = 0;
                tempBill = findBills(fileName, clientName);
                printf("%s: ", clientName);
                showlist(tempBill);
                while(! empty(tempBill))
                {
                        totalBill = totalBill + head(tempBill);
                        tempBill = tail(tempBill);
                }
                printf(" Total bill: %6.2f\n", totalBill);
        }
        fclose(fClients);
        return 0;
}
```

Esercizio 6
(liste)

Gestione di un negozio di videogame

- Un negozio di videogame vuole automatizzare parte della propria gestione
- Il negozio salva mensilmente ***lo stato dei propri articoli su un file di testo*** e ***traccia su un secondo file di testo tutte le vendite*** effettuate
- Lo scopo del programma è
  - Costruire ***una lista per i videogiochi*** aggiornandone il ***numero di copie disponibili*** in seguito alle vendite
  - Generare una ***lista di videogiochi acquistabili da bambini***
  - Salvare in un ***file di testo*** gli ***ordini da effettuare*** per riportare il magazzino ad avere un certo numero di copie per ogni gioco

Esercizio 6
(liste)

- Ogni riga del file memorizza lo stato mensile e contiene i dati di un videogioco
  - codice intero
  - titolo, possibilmente contenente spazi, di esattamente 30 caratteri
  - carattere di identificazione del tipo di gioco (‘P’ = picchiaduro, ‘A’ = avventura, ‘R’ = rompicapo, ‘O’ = altro)
  - numero di copie disponibili
  - voto medio dato dagli utenti
- In un secondo file di testo il negozio tiene traccia dei ***videogiochi venduti*** nell’arco del mese
  - Ogni riga del file contiene il codice identificativo di un gioco venduto (una singola copia)
  - Si prevede la possibilità che ci siano delle righe del file con codici errati (vanno saltati)

Esercizio 6
(liste)

Implementare le seguenti funzioni

- **`void printGames(list games)`**
  - Deve stampare il contenuto di una lista di videogiochi (si realizzi la funzione ricorsivamente usando le primitive)
- **`boolean loadFromFile(char *fileName, list *games)`**
  - Realizza la lista di videogiochi di partenza prendendo i dati dal file di nome fileName (si utilizzino le primitive)
- **`boolean updateAvailability(char *fileName, list games)`**
  - Aggiorna il numero di copie dei videogiochi, leggendo le vendite dal file di nome fileName
    - Se trova un codice non “riconosciuto”, stampa un messaggio di warning e va avanti

  (si utilizzi notazione a puntatori, no primitive)

Esercizio 6
(liste)

- `list gamesForKids(list games, float threshold)`
  - Restituisce la lista di videogiochi acquistabili da un bambino nel mese corrente (si realizzi la funzione ricorsivamente e usando le primitive)
  - Si adotti la seguente politica per scegliere se un videogioco è acquistabile da un bambino
    - Sono acquistabili unicamente i giochi disponibili (ovvero quelli per cui il numero di copie è positivo)
    - I picchiaduro non sono acquistabili da un bambino
    - Tutti i giochi di avventura sono acquistabili da un bambino
    - Per gli altri giochi, si indichino solo quelli per cui il voto medio dagli utenti è superiore alla soglia data

Esercizio 6
(liste)

- **`Boolean saveOrdersToFile`**
  `(char *fileName, list games, int qty)`
  - Salva i codici, i titoli e il numero di copie da ordinare per ogni videogioco (si realizzi la funzione iterativamente usando le primitive)
  - Si adotti la seguente politica
    - I quantitativi da ordinare devono riportare tutti i giochi alla quantità decisa **`qty`**
    - Ovviamente, quindi, un videogioco è presente nella lista solo se per esso è effettivamente necessario ordinare copie aggiuntive

Esercizio 6
(liste)

```
int main()
{
   List games;
   List kids;
   loadFromFile("lista.txt", &games);
   printGames(games);
   printf("-------UPDATE-------\n");
   updateAvailability("acquisti.txt", games);
   printGames(games);
   printf("-------G4KIDS-------\n");
   kids = gamesForKids(games, 4);
   printGames(kids);
   saveOrdersToFile("ordine.txt", games, 5);
   //DEALLOCAZIONE DELLE LISTE!!!
   return 0;
}
```

Esercizio 6
(liste)

```
787) dis-avventura              (A) 2       2.200000
981) noia e ancora noia         (O) 3       3.100000
421) the incredible machine     (R) 7       8.500000
753) super pang                 (O) 4       8.600000
642) zak mckracken              (A) 6       9.500000
574) ti spiezzo in due          (P) 4       4.500000
125) monkey island 1            (A) 9       9.200000
123) tekken 3                   (P) 7       8.100000
-------UPDATE-------
CODICE NON RICONOSCIUTO: 111
787) dis-avventura              (A) 2       2.200000
981) noia e ancora noia         (O) 1       3.100000
421) the incredible machine     (R) 3       8.500000
753) super pang                 (O) 0       8.600000
642) zak mckracken              (A) 5       9.500000
574) ti spiezzo in due          (P) 1       4.500000
125) monkey island 1            (A) 4       9.200000
123) tekken 3                   (P) 5       8.100000
-------G4KIDS-------
787) dis-avventura              (A) 2       2.200000
421) the incredible machine     (R) 3       8.500000
642) zak mckracken              (A) 5       9.500000
125) monkey island 1            (A) 4       9.200000
```

Esercizio 6
(liste)

- Contenuto del file **`ordine.txt`**

787 dis-avventura 3
981 noia e ancora noia 4
421 the incredible machine 2
753 super pang 5
574 ti spiezzo in due 4
125 monkey island 1 1

Esercizio 6 - Soluzione
(liste)

"Element.h":

```
typedef struct
{
      int code;
      char title[31];
      char type;
      int nItems;
      float rate;
} Element;
```

Esercizio 6 - Soluzione

(liste)

```
void printGames(List games)
{
  if(!empty(games))
  {
     Element game = head(games);
     printf("%d) %s \t(%c) %d \t %f\n",
                          game.code,
                               game.title,
                               game.type,
                               game.nItems,
                               game.rate);
     printGames(tail(games));
  }
}
```

Esercizio 6 - Soluzione
(liste)

```
Boolean loadFromFile(char *fileName, List *games)
{
   FILE *fp;
   Element curGame;
   *games = emptyList();
   if((fp = fopen(fileName, "r")) == NULL)
   {
       perror("Error opening file: ");
       return false;
   }
   …
```

Esercizio 6 - Soluzione
(liste)

```
        …
        while(fscanf(fp, "%d%30c%c%d%f",
                         &curGame.code,
                         curGame.title,
                         &curGame.type,
                         &curGame.nItems,
                         &curGame.rate) != EOF)
        {
              curGame.title[30] = '\0';
              *games = cons(curGame, *games);
        }
        fclose(fp);
        return true;
}
```

Esercizio 6 - Soluzione

(liste)

```
Boolean updateAvailability(char *fileName, List games)
{
   FILE *fp;
   int code;
   Element* game;
   if((fp = fopen(fileName, "r")) == NULL)
   { perror("Error opening file: "); return false; }
   while(fscanf(fp, "%d", &code) != 1)
   {
        game = findElementByCode(games, code);
        if(game != NULL)
               game->nItems--;
        else
               printf("CODICE NON RICONOSCIUTO: %d\n", code);
   }
   return true;
}
```

Esercizio 6 - Soluzione

(liste)

```
Element* findElementByCode(List games, int code)
{
  List cur;
  Element* found = NULL;
  cur = games;
  while(cur != NULL && found == NULL)
  {
      if(cur->value.code == code)
            found = &cur->value;
      else
            cur = cur->next;
  }
  return found;
}
```

Esercizio 6 - Soluzione

(liste)

```
List gamesForKids(List games, float threshold)
{
  List kids;
  Element game;
  if(empty(games))
      return emptyList();
  kids = gamesForKids(tail(games), threshold);
  game = head(games);
  if(game.nItems > 0 && game.type != 'P' &&
      (game.type == 'A' || game.rate >= threshold))
  {
      kids = cons(game, kids);
  }
  return kids;
}
```

Esercizio 6 - Soluzione

(liste)

```
Boolean saveOrdersToFile
      (char* fileName, List games, int qty)
{
  FILE* fp;
  int orderNumber;
  Element game;
  List gamesToRead = games;
  if ((fp = fopen(fileName,"w")) == NULL)
  {
     perror("Error opening file: ");
     return false;
  }
  …
```

Esercizio 6 - Soluzione
(liste)

```
   …
   while(!empty(gamesToRead))
   {
       game = head(gamesToRead);
       orderNumber = qty - game.nItems;
       if(orderNumber > 0)
             fprintf(fp, "%d %s %d\n",
                                   game.code,
                                   game.title,
                                   orderNumber);
       gamesToRead = tail(gamesToRead);
   }
   fclose(fp);
   return true;
}
```

Esercizio 7
(liste)

- La segreteria studenti di una facoltà tiene traccia dei voti degli studenti tramite un file di testo “elenco.txt”
- In ogni riga di tale file vengono memorizzati i seguenti dati (tutti relativi allo stesso studente):
  - Matricola (al più 14 caratteri), seguito dal carattere separatore ‘;’
  - Cognome (al più 63 cartteri) , seguito dal carattere separatore ‘;’
  - Nome (al più 63 caratteri), seguito dal carattere separatore ‘;’
  - Il numero di esami sostenuti (un intero) e, di seguito, tutti i voti degli esami. Tali voti sono tutti espressi tramite un intero, e sono separati tra di loro da uno spazio.

```
2148000974787;Chesani;Federico Domenico;5 30 28 26 30 19
2141000867548;Giannnelli;Carlo Maria Francesco;8 30 30 30 30 30 30 30 30
```

Esercizio 7
(liste)

- Non è noto a priori quanti studenti siano memorizzati nel file
- Non è noto a priori quanti esami abbia sostenuto ogni studente (tale valore è però indicato nel file in ogni riga, prima dell'elenco dei voti)
- Ogni volta che uno studente passa un'esame, la segreteria aggiunge nel file di testo una nuova riga con aggiornato l'esame relativo allo studente (e non cancella la riga precedente). Dopo un po' quindi:

```
2148000974787;Chesani;Federico Domenico;5 30 28 26 30 19
2141000867548;Giannnelli;Carlo Maria Francesco;8 30 30 30 30 30 30 30 30
2148000974787;Chesani;Federico Domenico;6 30 28 26 30 19 29
2148000974787;Chesani;Federico Domenico;7 30 28 26 30 19 29 30
```

Esercizio 7
(liste)

1. Definire in un appostio file “`element.h`” una struttura dati di tipo `element` adatta a rappresentare i dati di ogni studente. A tal scopo, si noti che non essendo noto a priori quanti esami ha sostenuto ogni studente, sarà necessario prevedere un puntatore ad un’area di memoria allocata dinamicamente. In fase di lettura, sarà poi necessario allocare memoria suficiente...

Esercizio 7
(liste)

2. Definire in un apposito modulo "`elementUtil.h / elementUtil.c`" le seguenti funzioni:
   - `void printElement(element el);`

     Stampa ordinata delle informazioni relative ad uno studente
   - `int compareElement(element el1, element el2);`

     Confronto tra due studenti; restituisce 0 se el1 ed el2 hanno lo stesso cognome, nome, matricola e numero di esami; retituisce un valore negativo/positivo se el1 precede/segue el2. el1 precede/segue el2 se ciò vale, in ordine, per il cognome, il nome, la matricola, ed infine il numero di esami. Per i valori di tipo stringa si consideri l'ordinamento lessico-grafico.
   - `READ_CODE readElementFromFile(FILE * fp, element * dest);`

     Legge i dati relativi ad uno studente dal file FP e li salva in dest.
   - `int equals(element el1, element el2);`

     Restituisce un valore interpretabile come vero se i due elementi riguardano la stessa matricola, cognome e nome (ignorando quindi il numero di esami)

Esercizio 7
(liste)

3. Realizzare un main che esegua le seguenti funzioni:

   - legga da file e memorizzi in una lista di element tutti gli studenti memorizzati, comprese le ripetizioni

   - stampi a video tale lista per controllare che la lettura sia andata a buon fine

   - Definisca una funzione `list eliminaRipetuti(list l);` che generi una nuova lista, ordinata secondo i criteri già esposti, e che non contenga elementi ripetuti. Nel caso ci siano elementi ripetuti, si deve salvare nella nuova lista il record più aggiornato, cioè quello con più esami sostenuti...

   - Al termine, il programma deve avere cura di de-allocare tutta la memoria allocata dinamicamente...

Esercizio 7 - Soluzione
(liste)

element.h

```
#ifndef ELEMENT
#define ELEMENT

#define NAME_LENGTH 64
#define SURNAME_LENGTH 64
#define ID_LENGTH 15

typedef struct {
   char name[NAME_LENGTH];
   char surname[SURNAME_LENGTH];
   char id[ID_LENGTH];
   int * votes;
   int num_votes;
} element;


#endif
```

Esercizio 7 - Soluzione
(liste)

elementUtil.h

```
#ifndef ELEMENTUTIL
#define ELEMENTUTIL

#include <stdio.h>
#include "element.h"

#define READ_FAILED 0
#define READ_SUCCESS 1

typedef int READ_CODE;

void printElement(element el);
int compareElement(element el1, element el2);
READ_CODE readElementFromFile(FILE * fp, element * dest);
int equals(element el1, element el2);

#endif
```

Esercizio 7 - Soluzione

(liste)

elementUtil.c

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

#include "generalUtil.h"
#include "elementUtil.h"

void printElement(element el) {
   int i;
   printf("   %s %s, %s: ", el.id, el.surname, el.name);
   printf("%d esami superati.\n", el.num_votes);
   printf("     Voti: ");
   for (i=0; i<el.num_votes; i++) {
        printf("%d", el.votes[i]);
        if (i<el.num_votes-1)
                printf(" ");
   }
}
…
```

Esercizio 7 - Soluzione

(liste)

elementUtil.c

```
int compareElement(element el1, element el2) {
   int result;
   result = strcmp(el1.surname, el2.surname);
   if (result == 0) {
        result = strcmp(el1.name, el2.name);
        if (result == 0) {
               result = strcmp(el1.id, el2.id);
               if (result == 0) {
                      if (el1.num_votes <el2.num_votes)
                             result = -1;
                      else
                             if (el1.num_votes > el2.num_votes)
                                    result = 1;
               }
        }
   }
   return result;
}
```

Esercizio 7 - Soluzione

(liste)

elementUtil.c

```
int equals(element el1, element el2) {
   if (
        strcmp(el1.id, el2.id) == 0 &&
        strcmp(el1.surname, el2.surname) == 0 &&
        strcmp(el1.name , el2.name) == 0
        )
        return 1;
   else
        return 0;
}
```

Esercizio 7 - Soluzione

(liste)

elementUtil.c

```
READ_CODE readElementFromFile(FILE * fp, element * dest) {
    int result = READ_SUCCESS; int i;
    char sepRead;

    sepRead = readField(dest->id, ';', fp);
    if (sepRead == ';' && result==READ_SUCCESS)
          sepRead = readField(dest->surname, ';', fp);
    else result = READ_FAILED;
    if (sepRead == ';' && result==READ_SUCCESS)
          sepRead = readField(dest->name, ';', fp);
    else result = READ_FAILED;
    if (sepRead == ';' && result==READ_SUCCESS) {
          fscanf(fp, "%d", &(dest->num_votes));
          dest->votes = (int*) malloc(sizeof(int) * dest->num_votes);
          for (i=0; i< dest->num_votes && result == READ_SUCCESS; i++)
                    if (fscanf(fp, "%d", &((dest->votes)[i])) != 1)
                              result = READ_FAILED;
          if (result == READ_SUCCESS)
                    fgetc(fp);
    }
    else result = READ_FAILED;
    return result;
}
```

Esercizio 7 - Soluzione

(liste)

generalUtil.h

```
#ifndef GENERALUTIL
#define GENERALUTIL
#include <stdio.h>

char readField(char buffer[], char sep, FILE *f);
#endif
```

Esercizio 7 - Soluzione

(liste)

generalUtil.c

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

char readField(char buffer[], char sep, FILE *f)
{
    int i = 0;
    char ch = fgetc(f);
    while (ch != sep && ch != 10 && ch != EOF)
    {
        buffer[i++] = ch;
        ch = fgetc(f);
    }
    buffer[i] = '\0';
    return ch;
}
```

Esercizio 7 - Soluzione

(liste)

list.h

```
#ifndef LIST
#define LIST

#include "element.h"
#include "elementUtil.h"

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

typedef  item* list;
typedef  int boolean;

/*  PROTOTIPI DI FUNZIONE (extern) */

/* PRIMITIVE  */
list emptylist(void);
boolean empty(list);
list cons(element, list);
element head(list);
list tail(list);

/* NON PRIMITIVE */
void showlist(list);
boolean member(element, list);
list insord(element el, list l);

#endif
```

Esercizio 7 - Soluzione

(liste)

list.c

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

/* OPERAZIONI PRIMITIVE */
list  emptylist(void)              /* costruttore lista vuota */
{ return NULL; }

boolean  empty(list l) /* verifica se lista vuota */
{ return (l==NULL); }

list  cons(element e, list l)
{ list t;       /* costruttore che aggiunge in testa alla lista */
  t=(list)malloc(sizeof(item));
  t->value=e;
  t->next=l;
  return(t);
}

element head(list l) /* selettore testa lista */
{ if (empty(l))
     exit(-1);
  else return (l->value);
}

list  tail(list l)         /* selettore coda lista */
{  return (l->next);
}
```

Esercizio 7 - Soluzione

(liste)

list.c

```
/*  ATTENZIONE:  la showlist() e' stat ridefinita per trattare un generico element  */
void   showlist(list l) {
    /* VERSIONE ITERATIVA: */
    printf("Content of the list:\n");
    while (!empty(l)) {
           printElement(head(l));
           printf("\n");
           l=tail(l);
    }
    printf("Fine lista.\n");
}

boolean member(element el, list l)
/* appartenenenza di un elemento - ricorsiva  */
/* versione generica: presuppone l'esistenza di equals(...) */
{ if (empty(l)) return 0;
  else { if (equals(el,head(l))) return 1;
       else return member(el,tail(l)); }
}

/* versione generica: presuppone l'esistenza di compareElement(...) */
list insord(element el, list l) {
  if (empty(l)) return cons(el,l);
  else if (compareElement(el, head(l)) <= 0) return cons(el,l);
  else return cons(head(l), insord(el,tail(l)));
}
```

Esercizio 7 - Soluzione
(liste)

main.c

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

#include "list.h"
#include "elementUtil.h"

list eliminaRipetuti(list l) {
     list lOrd = emptylist();
     list result = emptylist();
     list temp;

     while (!empty(l)) { // ordino la lista
            lOrd = insord(head(l), lOrd);
            l = tail(l);
     }

     temp = lOrd; // elimino i ripetuti
     while (!empty(lOrd)) {
            if (!member(head(lOrd), tail(lOrd)))
                        result = insord(head(lOrd), result);
            lOrd = tail(lOrd);
     }

     lOrd = temp; //de-alloco la lista temporanea lOrd
     while (!empty(lOrd)) {
            temp = tail(lOrd);
            free (lOrd);
            lOrd = temp;
     }
     return result;
}
```

Esercizio 7 - Soluzione
(liste)

main.c

```
int main() {
    list l; list l2; list lTemp;
    FILE * fp;
    element temp;
    int readOk;

    l = emptylist();
    if ((fp=fopen("elenco.txt", "r"))== NULL) {
          printf("Errore nell'apertura del file...\n");
          exit(1);
    }
    do {
          readOk = readElementFromFile(fp, &temp);
          if (readOk == READ_SUCCESS)
                    l = cons(temp, l);
    } while (readOk == READ_SUCCESS);
    fclose(fp);
    showlist(l);

    l2 = eliminaRipetuti(l);
    showlist(l2);

    …
```

Esercizio 7 - Soluzione
(liste)

main.c

```
    …

    while (!empty(l2)) {
          lTemp = tail(l2);
          free(l2);
          l2= lTemp;
    }
    while (!empty(l)) {
          lTemp = tail(l);
          free(head(l).votes);
          free(l);
          l = lTemp;
    }

    system("PAUSE");
    return 0;
}
```