Liste semplici di strutture

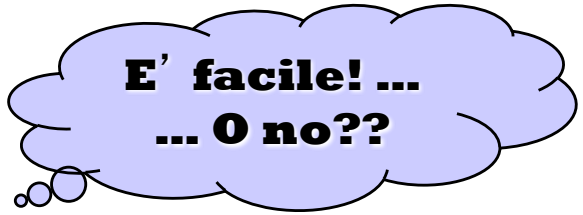
Obiettivi:

- Partendo dall'ADT lista semplice generalizzare ulteriormente la tipologia di elementi in lista, considerando strutture (struct)

GENERALIZZIAMO ULTERIORMENTE …

Realizziamo una lista di strutture per memorizzare una struttura che contiene parola e numero di linea del file da cui l'abbiamo letta (una specie di indice analitico)

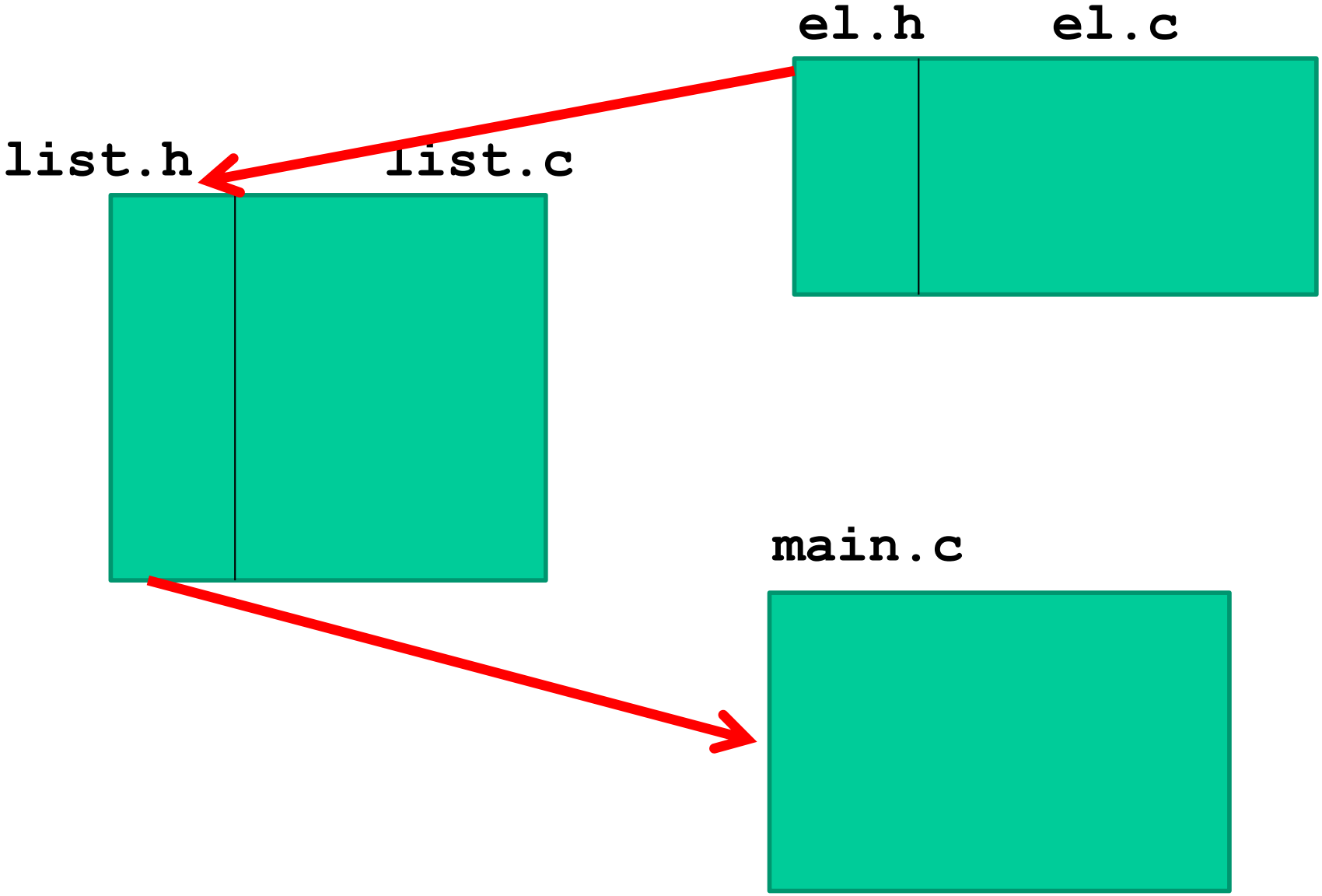
Un file sequenziale di tipo testo (ESPR.TXT) contiene stringhe (una per linea) costituite dai caratteri {a,b,c,*,+}.



Si realizzi un programma C che:

a) riconosca le stringhe del file uguali a "a*b+c" oppure a "a+b*c" e le inserisca con i loro numeri di linea in una lista a puntatori, L1, ***ordinata sul campo stringa (a parità di stringa, si ordini in base alla posizione)***

COMPONENTI



GENERALIZZIAMO ULTERIORMENTE …

Si realizzi un programma C che:

a)riconosca le stringhe del file uguali a "a*b+c" oppure a "a+b*c" e le inserisca con i loro numeri di linea in una lista a puntatori, L1, ***ordinata sul campo stringa (a parità di stringa, si ordini in base alla posizione)***;

b)produca, a partire dalla lista L1 generata precedentemente, un file (UNICHE.TXT) di tipo testo in cui ogni stringa accettata compare seguita, sulla stessa linea, dalla posizione originale nel file ESPR.TXT

Esempio:

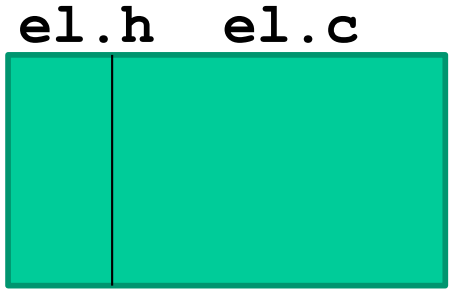
ESPR.TXT:	UNICHE:	
a*b+c	a*b+c	1
a**b+	a*b+c	4
a+b*c	a+b*c	3
a*b+c	a+b*c	5
a+b*c		

Interfaccia modulo elementi: el.h

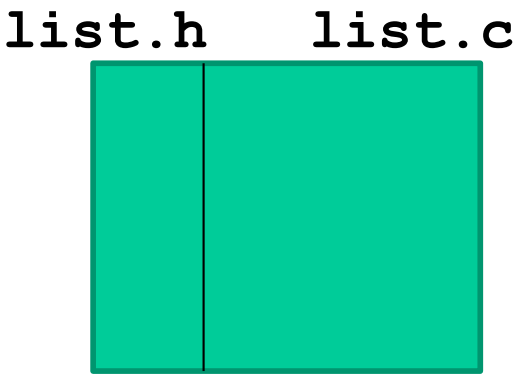
```
/* ELEMENT TYPE - file el.h*/
#ifndef ELEMENT_H
#define ELEMENT_H

#define N 30
typedef struct { char stringa[N];
                 int  pos; }  element;
typedef enum {false, true} boolean;


boolean isLess(element, element);
boolean isEqual(element, element);
element getElement(char *, int);
void printElement(FILE *,element);

#endif
```

el.c

Vediamo la parte implementazione del nuovo ADT degli elementi (strutture)



E' necessario cambiare le funzioni dell' ADT lista?



ADT LISTA: list.h

non cambia di molto

```
#include "el.h"

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

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

void fshowList(FILE *, list);            // NON PRIMITIVE
boolean member(element, list);
```

…

Implementazione modulo elementi: el.c

```
#include "el.h"
#include <stdio.h>
#include <string.h>

boolean isEqual(element e1, element e2) {
       return (!strcmp(e1.stringa,e2.stringa)); }

boolean isLess(element e1, element e2) {
    if (strcmp(e1.stringa,e2.stringa))
        return (strcmp(e1.stringa,e2.stringa)<0);
    else  return (e1.pos<e2.pos);  }

element getElement(char *s, int pos) {
       element el;
       strcpy(el.stringa,s);
       el.pos=pos;
       return el; }

void printElement(FILE *f, element el) {
       fprintf(f, "%s\t%d\n", el.stringa, el.pos); }
```

GENERALIZZIAMO ULTERIORMENTE …

Si realizzi un programma C che:

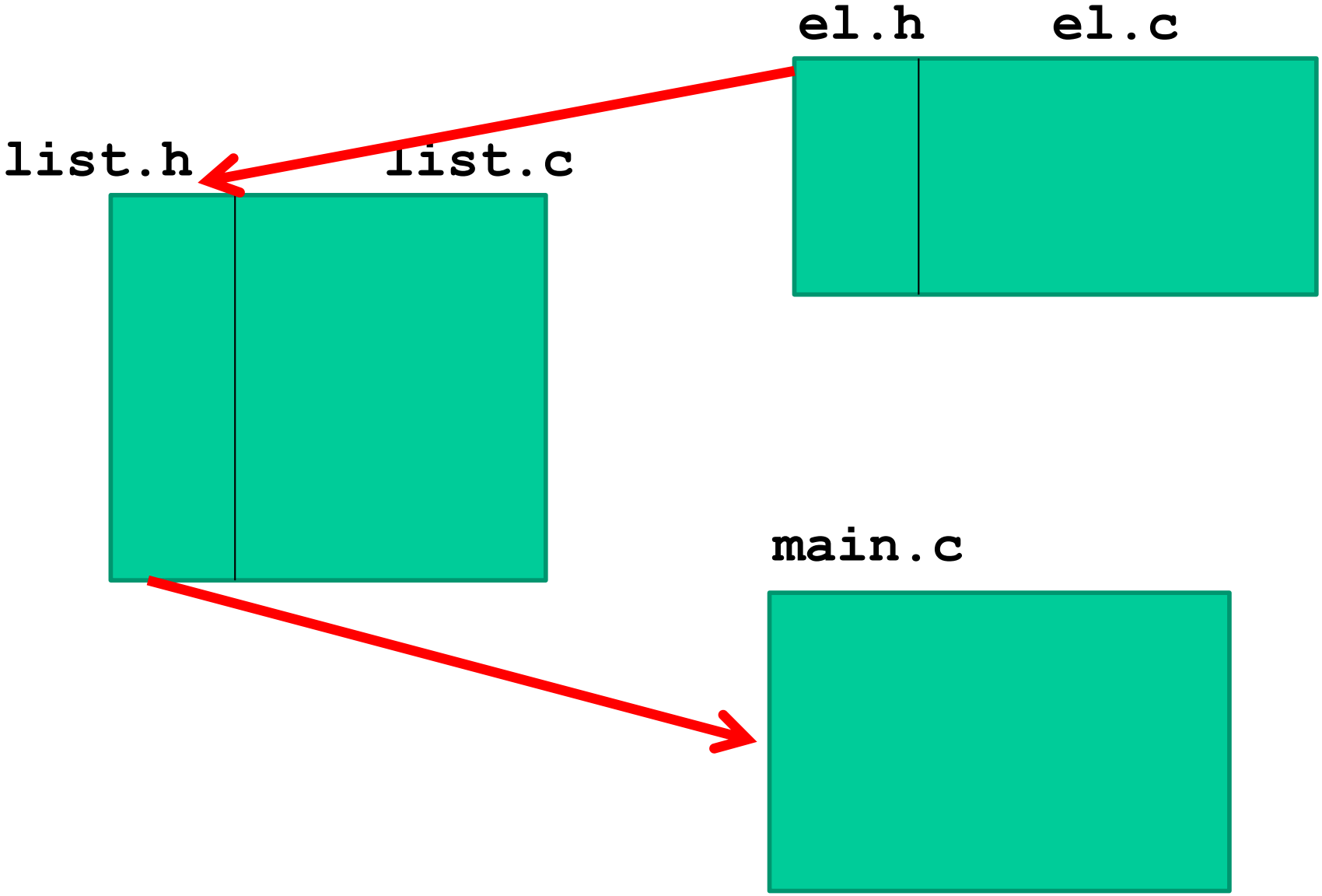
a)riconosca le stringhe del file uguali a "a*b+c" oppure a "a+b*c" e le inserisca con i loro numeri di linea in una lista a puntatori, L1, ***ordinata sul campo stringa (a parità di stringa, si ordini in base alla posizione)***;

b)produca, a partire dalla lista L1 generata precedentemente, un file (UNICHE.TXT) di tipo testo in cui ogni stringa accettata compare seguita, sulla stessa linea, dalla posizione originale nel file ESPR.TXT

Esempio:

ESPR.TXT:	UNICHE:	
a*b+c	a*b+c	1
a**b+	a*b+c	4
a+b*c	a+b*c	3
a*b+c	a+b*c	5
a+b*c		

COMPONENTI



Il main (1)

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

void  main (void)
{  list  L1 = emptylist();
   element EL;
   char  s[N];
   int  pos=0;
   FILE  *f, *g;

   f = fopen("ESPR.TXT",”rt");
   if(f==NULL)
      { printf("errore apertura espr.txt \n");
        exit(-1);     }
```

Il main (2)

```
//DOMANDA a) ciclo di scansione del file

   while (fscanf(f, "%s", s) !=EOF)
      {  pos++;
         if ( (strcmp(s,"a*b+c")==0) ||
              (strcmp(s,"a+b*c")==0) )
           {  EL = getElement(s,pos);

           // e inserimento ordinato in L1
              L1 = insord(EL,L1);
            }
       }
   fclose(f);
```

Il main (3)

```
// DOMANDA b)

   g = fopen("UNICHE.TXT", "wt");
   if(g==NULL)
      { printf("errore apertura uniche.txt \n");
        exit(-2);      }

   fshowList(g,L1);     // MA ANCHE: showList(L1);
   fclose(g);
}
```

```
a*b+c 1
a*b+c 4
a+b*c 3
a+b*c 5
```

ADT LISTA: *ora*

```
void fshowList(FILE *f, list l) {

      while (!empty(l)) {

            printElement(f, head(l));

            l = tail(l);    }
}
```

NOTA: **printElement** stampa su file testo le componenti di head(l):

```
void printElement(FILE *f, element el) {
      fprintf(f, "%s\t%d\n", el.stringa, el.pos); }
```

ADT LISTA: *ora*

```
void fshowList(FILE *f, list l) {

     if(!empty(l)) {

          printElement(f, head(l));

               fshowList(f, tail(l));   }
}
```

NOTA: **printElement** stampa su file testo le componenti di head(l):

```
void printElement(FILE *f, element el) {
       fprintf(f, "%s\t%d\n", el.stringa, el.pos); }
```

TO DO … MORE

c) trovi, a partire dalla lista L1 generata precedentemente, la parola più frequente

Esempio:

ESPR.TXT:

a*b+c

a**b+

a+b*c

a*b+c

a+b*c

Conto quante volte compare la prima parola nella lista, e quante volte la seconda (meglio se con una sola scansione della lista), poi determino il maggiore.

ADT LISTA: *ora*

```
void conta(list l, int *c1, int *c2) {

  while (!empty(l)) {

     if(strcmp(head(l).value,"a*b+c")==0) (*c1)++;
     else (*c2)++;
      l = tail(l);       }
}
```

```
...
       // nel main:
int conta1=0, conta2=0;
conta(L1,&conta1,&conta2);
```

TO DO … MORE

d) produca, a partire dalla lista L1 generata precedentemente, un file (UNICHE.TXT) di tipo testo in cui ogni stringa accettata compare ***una sola volta*** seguita, sulla stessa linea, dalle posizioni originali nel file ESPR.TXT

Esempio:

ESPR.TXT:

a*b+c

a**b+

a+b*c

a*b+c

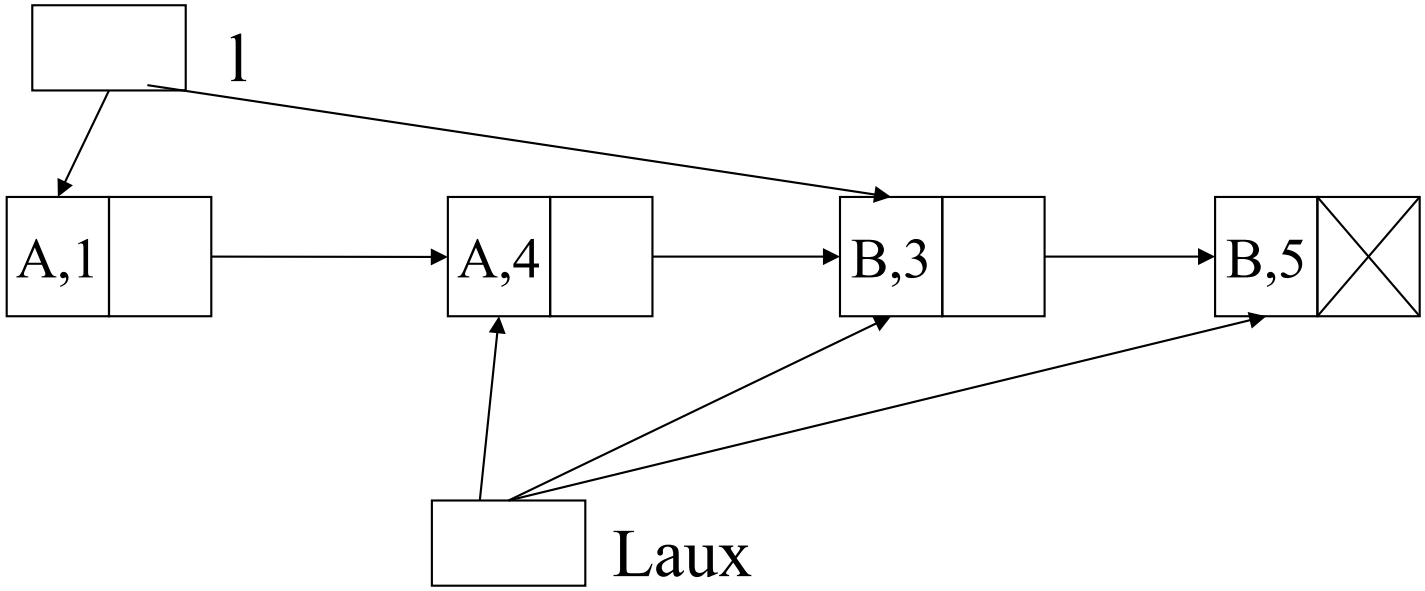
a+b*c

UNICHE:

a*b+c 1 4

a+b*c 3 5

Come scandire la lista?



A	1	4
B	3	5

Scansione innestata, avanzo con Laux (che parte da tail(l)) finché trovo nodi con la stessa stringa (o arrivo alla fine della lista); non appena trovo una stringa diversa, avanzo con l fino a questo nodo e riprendo la scansione innestata (partendo con Laux=tail(l)).

REALIZZATE QUESTA FUNZIONE

```
void fshowList2(FILE *f, list l);
```

Nuova funzione *ad hoc* per questa stampa

```
void fshowList2(FILE *f, list l) {
  list Laux; int fine;
  while (!empty(l))
       {  EL=head(l);
          fprintf(f,"%s",EL.stringa);
          fprintf(f,"\t%d", EL.pos);
          Laux=tail(l); fine=0;
          while ((!empty(Laux)&&!fine)
              if (!strcmp(Laux->value.stringa,
                          l->value.stringa) )
                 { fprintf(f,"\t%d",Laux->value.pos);
                   Laux=tail(Laux);        }
              else fine=1;
          fprintf(f,"\n");
          l=Laux;
       }
}
```

CONCLUSIONE

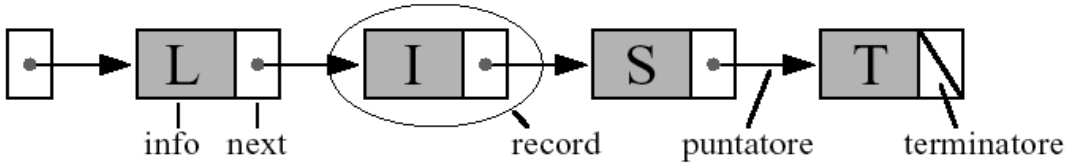
Che cosa possiamo trarre come ulteriore conclusione?

L'obiettivo del corso è presentare strumenti e approcci, e far sì che li assimiliate non che li impariate a memoria …

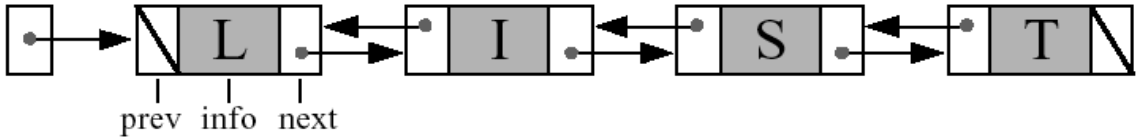
In sede d'esame si valuta se sapete lavorare / gestire le strutture dati viste, anche al di là delle funzioni base o derivate tipiche …

… una funzione non tipica su liste e/o alberi è ***sempre*** richiesta nel testo … fate esercizi, scrivete e fate girare qualche programma!!

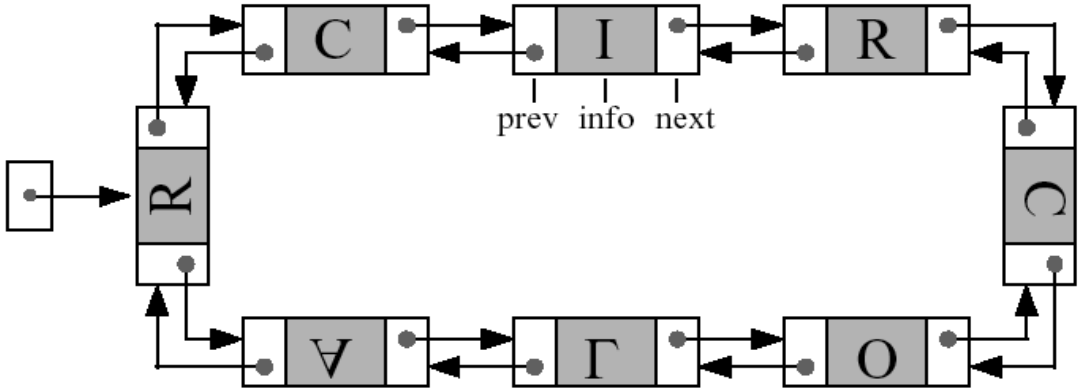
Altri esempi di strutture dati collegate



lista semplice



lista doppiamente collegata



lista circolare doppiamente

e anche ... alberi binari, alberi n-ari (liste di liste)

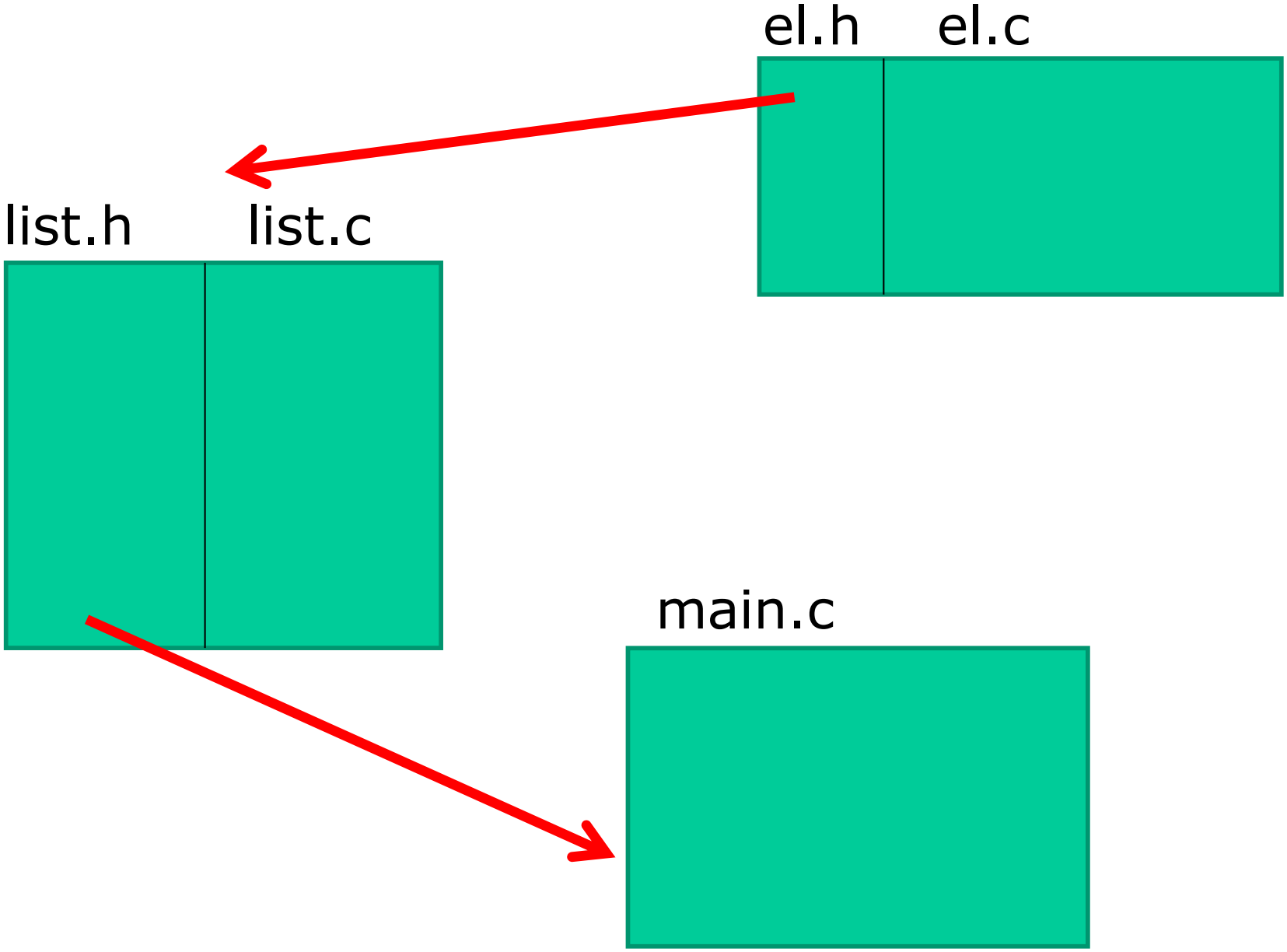
Esercizio 1. Liste di interi

Si legga una sequenza di numeri interi da tastiera, dopo ogni inserimento chiedere all'utente se vuole continuare, quindi:

- Creare due liste L1 e L2 con **inserimento ordinato**.
- Creare una funzione **append** che, date due liste, restituisca una terza lista contenente gli elementi della prima seguiti dagli elementi della seconda: **list append(list, list)**
- Creare funzione **merge** che, date due liste ordinate, restituisca una lista ordinata contenente gli elementi delle due liste date (gli elementi possono essere ripetuti): **list merge(list, list)**

Si astragga il concetto di lista in modo da rendere l'implementazione il più strutturata possibile.

Esercizio 1. Struttura



Esercizio 2. Liste di strutture

Si vuole tenere in memoria le informazioni riguardanti gli studenti iscritti ad un corso di informatica. Il corso è organizzato in due gruppi, si utilizzi una lista per ogni gruppo.

Si astragga il concetto di lista in modo da rendere l'implementazione il più strutturata possibile.

Esercizio 2. Liste di strutture

Per fare ciò si cominci a implementare una libreria (**studente.h**, **studente.c**) contenente la definizione della struttura **studente** e le funzioni principali per la sua manipolazione.

```
#define DIM_STR 20
typedef struct {
        char nome[DIM_STR];
        char cognome[DIM_STR];
        float media;
} element;
```

Esercizio 2. Liste di strutture

Creare le funzioni:

- **boolean isLess(element, element);**
  Controlla se un elemento è minore dell'altro
- **boolean isEqual(element, element);**
  Controlla uguaglianza fra due elementi
- **element getElement(void);**
  Legge in input da tastiera un elemento
- **void printElement(element);**
  Stampa a video un elemento

Usare queste librerie per astrarre il tipo di elemento delle liste dall'implementazione.

Esercizio 2. Liste di strutture

Il programma deve leggere da un file **studenti1.txt** un numero N di studenti (N definito dall'utente in input) inserendo gli studenti presenti nel file nella lista **l1** e leggere da **studenti2.txt** un numero M di studenti inserendoli nella lista **l2**.
NB: in **l1** e **l2** effettuare l'inserimento **ordinato** in base al cognome (e al nome in caso di cognomi uguali).

Visualizzare un menu nel con le seguenti opzioni:

- Visualizzazione di tutti gli studenti.
- Ricerca di uno studente (per nome e cognome).
- Cancellazione di uno studente da una lista.
- Append delle due liste.
- Copia di una delle due liste in una terza.
- Merge di due liste.
- Stampa del numero degli studenti nelle due liste.

Si astragga il concetto di lista in modo da rendere l'implementazione il più strutturata possibile.