Liste di Strutture

- Obiettivi:
  - Riprendere la nozione di strutture dato elementari per la gestione di collezioni di oggetti
  - Strutture dato tavole (tabelle), ma realizzate con liste collegate

Strutture dati

- Termine usato per indicare insiemi (collezioni) con elementi del dominio compositi, ad esempio:
  - tavole (dizionari);
  - liste;
  - insiemi;
  - pile e code;
  - alberi e grafi
- corredate delle operazioni per la loro gestione (efficiente)

Il tipo dato Dizionario (Tavola)

tipo `Dizionario`:
dati:
un insieme S di coppie $(elem, chiave)$.

operazioni:

`insert`$(elem\ e, chiave\ k)$
aggiunge a S una nuova coppia (e, k).

`delete`$(chiave\ k)$
cancella da S la coppia con chiave k.

`search`$(chiave\ k) \rightarrow elem$
se la chiave k è presente in S restituisce l'elemento e ad essa associato, e `null` altrimenti.

Tavole

- Tipo di dato astratto per rappresentare insiemi di coppie:

<chiave, attributi>

- Realizzabile – in memoria centrale – con liste semplici (strutture dati collegate)

Nome	Cognome	Reddito	AliquotaMax
Mario	Rossi	18324	27
Luca	Bianchi	1453253	43

Esercitazione 6.1 *rivisitata*

- Scrivere un programma che realizzi una rubrica telefonica come lista (ordinata) in memoria centrale. In particolare, ogni elemento della tavola è caratterizzato dalle seguenti informazioni:
  - Nome (chiave)
  - numero_telefono
- Il programma deve essere in grado di attuare varie richieste dell'utente:
- ***inserimento***: l'utente vuole inserire un nuovo record nell'archivio. Dati nome e numero di telefono della persona da inserire, il programma aggiunge un nuovo elemento alla lista (ovvero all'archivio).

Esercitazione 6.1 (*cont.*)

- ***cancellazione***: l'utente vuole eliminare un elemento dall'archivio. Dato un nome, il programma dovrà eliminare (se esiste) l'elemento con il nome specificato dalla lista.
- ***ricerca***: dato in ingresso il nome di una persona presente in archivio, si richiede la visualizzazione del numero di telefono relativo alla persona data;
- ***uscita***: l'utente richiede che il programma termini.

- L'interazione tra l'utente e il programma avviene in modo ciclico: l'utente può sottoporre una richiesta ad ogni ciclo ed il programma, sfruttando un meccanismo di selezione (per esempio **switch**) reagisce nel modo richiesto. L'esecuzione del programma si conclude quando l'utente richiede l'uscita.

Soluzione (come lista)

Rappresentazione collegata, come un lista di strutture (in memoria centrale). Il tipo di ciascun elemento della rubrica/lista è:

```
typedef struct {   char   nome[20];
                   char   tel[16];     } elemento;
```

La lista che rappresenta la rubrica è quindi del tipo:

```
typedef struct list_elemento {
                elemento value;
                struct list_element *next; } item;
typedef item *list;
```

Una variabile di tipo list mantiene il valore del puntatore radice; la ***dimensione logica*** è data dalla lunghezza della lista:

```
list R;
```

Dichiarazioni globali

```
#include <stdio.h>
#include <string.h>
typedef struct {  char  nome[20];
                  char  tel[16];    } elemento;

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

/* prototipi */
int menu(void);
list inserimento(list R);
list cancellazione(list R);
void ricerca(list R);
```

main

```
void main(void)
{int scelta, fine;
 list R = NULL;                    /* R è la radice */

 fine=0;
 do
 { scelta=menu();
   switch(scelta){
     case 1: R = inserimento(R); break;
     case 2: R = cancellazione(R); break;
     case 3: ricerca(R);  break;
     case 4: fine=1; break;
     default: printf("Scelta sbagliata\n");
              }
      }while (!fine);
}
```

menu

```
int menu(void)
{     int ris;
      printf("Scegli l'operazione:\n");
      printf("\t1\tInserimento\n");
      printf("\t2\tCancellazione\n");
      printf("\t3\tRicerca\n");
      printf("\t4\tUscita\n");
      printf("\n\nScelta:  ");
      scanf("%d", &ris);
      return ris;
}
```

Inserimento (ordinato in lista)

L'inserimento ordinato determina la memorizzazione dell'elemento dato da standard input nella posizione deputata nella lista :

```
list inserimento (list R)
{     elemento el;
      printf("\nInserire nome:  ");
      scanf("%s",el.nome);
      printf("\nInserire numero:  ");
      scanf("%s",el.tel);
      return insord(el,R);
}
```

Non c'è il problema di riempire tutte le locazioni con la lista ... Finché c'è memoria heap disponibile

Insord in lista (*iterativo*)

```
list insord(elemento el, list l) {
  list pprec, patt = l, paux;
  int trovato = 0;
  while (patt!=NULL && !trovato) {
     if(strcmp(el.nome,patt->value.nome)<=0)
         trovato = 1;
     else { pprec = patt;
            patt = patt->next; }
  }
  paux = (list) malloc(sizeof(item));
  paux->value = el; paux->next = patt;
  if (patt==l) return paux;
  else { pprec->next = paux; return l; }
}
```

Insord in lista (*ricorsivo*)

```
list insord(elemento el, list l) {
  if (l == NULL)
          return cons(el, l);
  else {
    if (strcmp(el.nome,l->value.nome)<=0)
            return cons(el, l);
    else {
            l->next =insord(el,l->next);
            return l;  }
   }
}
```

cancellazione

Eliminazione dalla rubrica (data come parametro) dell'elemento identificato attraverso il suo nome (letto da input):

```
list cancellazione(list R)
{ elemento e;
  printf("\nInserire nome:  ");
  scanf("%s",e.nome);
  return delete(el,R);
}
```

```
list delete(elemento el, list L) {

  int trovato = 0;
  list aux=L, prev=NULL;
  if (aux != NULL)
    if (strcmp(aux->value.nome,el.nome)==0)
       { L = aux->next;
         free(aux);  }
    else
    { while ((aux != NULL) && (!trovato))
        if (strcmp(aux->value.nome,el.nome)==0) trovato=1;
         else
          { prev = aux;
             aux = aux->next;        }
      if (aux != NULL)
        { prev->next = aux->next;
          free(aux);} }
  return L;        //il return qui per tutti i casi
}
```

ricerca

Ricerca nella rubrica (data come parametro) del telefono di un elemento attraverso il suo nome (letto da input):

```
void ricerca(list R) {
  list k=NULL;          /* qui k puntatore di list */
  elemento e;
  printf("\nInserire nome:  ");
  scanf("%s",e.nome);
  k=individua(R,e);
  if (k!=NULL)
   {  printf("\nTelefono di %s ...\n", k->value.nome);
      printf("\è  %s ...\n", k->value.tel);
            }
  else printf("\n%s\t non trovato\n", e.nome);
}
```

Individua (... *member*)

Realizza la ricerca sequenziale, caso peggiore O(**DIM**)

```
list individua(list R, elemento e) {
     list L = R;
     int trovato=0;
     while ((L!=NULL) && (!trovato)) {
            if (!strcmp(e.nome, L->value.nome))
                          trovato=1;
            else L=L->next; }

       if (trovato) return L;
       else return NULL;
}
```

In caso di chiave composta …

Se la chiave fosse costituita da più attributi (ad esempio **nome** e **cognome**):

```
list individua(list R, elemento e) {
     list L = R;
     int trovato=0;
     while ((L!=NULL) && (!trovato)) {
       if (!strcmp(e.nome, L->value.nome) &&
         (!strcmp(e.cognome, L->value.cognome))
                        trovato=1;
             else L=L->next; }

      if (trovato) return L;
      else return NULL;
}
```

Tavola come lista collegata

- Vantaggi:
  - La lista non deve essere definita staticamente, è una struttura dati dinamica, cresce finché c'è spazio nell'heap
  - Cambia il tipo degli elementi (campo value), ma le operazioni in pratica restano quasi le stesse (tranne i confronti, *sul campo chiave*)
- Difetti
  - Ricerca solo sequenziale (no ricerca binaria ... *la ritroveremo negli alberi binari di ricerca*)