Corso di Laurea in INFORMATICA

a.a. 2012-2013

Algoritmi e Strutture Dati

MODULO N. 3

Strutture lineari di dati: Liste

Il concetto di sequenza. Le Liste: specifiche e realizzazioni attraverso rappresentazioni sequenziali e collegate. Esempi di problemi: epurazione, fusione di liste, ordinamento di liste

LE STRUTTURE DI DATI:

LINEARI : SEQUENZE

NON LINEARI : NELLE QUALI NON È INDIVIDUATA UNA SEQUENZA

A DIMENSIONE FISSA : IL CUI NUMERO DI ELEMENTI E' FISSO

A DIMENSIONE VARIABILE : IL CUI IL NUMERO DI ELEMENTI PUÒ VARIARE NEL TEMPO

OMOGENEE : I CUI DATI SONO DELLO STESSO TIPO

NON OMOGENEE : I CUI DATI NON SONO DEL MEDESIMO TIPO

Una struttura lineare: la Sequenza

GENERALMENTE RAPPRESENTA UNA ORGANIZZAZIONE DI DATI

- **Dinamica e lineare**
- **Contenente elementi generici (*item*), anche duplicati**
- **Ordine all'interno della sequenza è importante**

OPERATORI

- **E' possibile aggiungere / togliere elementi, specificando la posizione**
- **E' possibile accedere *direttamente* ad alcuni elementi (testa / coda)**
- **E' possibile accedere *sequenzialmente* a tutti gli altri elementi**

Altre strutture: set e dizionari

- **Struttura dati "generale": *insieme dinamico***
  - **Può crescere, contrarsi, cambiare contenuto**
  - **Operazioni base: inserimento, cancellazione, verifica appartenenza**
  - **Il tipo di insieme (= struttura) dipende dalle operazioni**
- **Elementi**
  - **Elemento: oggetto che può essere composto da:**
    - **campo chiave di identificazione**
    - **dati satellite**
    - **campi che fanno riferimento ad altri elementi dell'insieme**

LE STRUTTURE LINEARI DI DATI

SONO STRUTTURE DI DATI CHE SI SVILUPPANO IN UNA DIMENSIONE E POSSONO ESSERE CONSIDERATE COME SEQUENZE DI OGGETTI (vettori, liste, pile, code).

IN ESSE E' IMPORTANTE L'ESISTENZA DI UNA *RELAZIONE D'ORDINE* TRA GLI OGGETTI CHE SERVE AD INDIVIDUARLI E A SELEZIONARLI: CORRISPONDE AL PRINCIPIO CHE I COMPONENTI SIANO INDIVIDUATI DAL NUMERO D'ORDINE DELLA LORO POSIZIONE.

PER DISTINGUERE LE DIVERSE STRUTTURE E' RILEVANTE CONSIDERARE:

a) **I MODI DI INDIVIDUARE LA POSIZIONE IN CUI OPERARE NELLA SEQUENZA (*MODI DI ACCEDERE*)**

b) **I MODI DI AGIRE NELLA POSIZIONE INDIVIDUATA (*MODI DI OPERARE*)**

a) MODI DI ACCEDERE

- **ACCESSO DIRETTO**

IL VETTORE (ARRAY) E' IL TIPICO ESEMPIO DI ACCESSO DIRETTO: CONSENTE DI ACCEDERE AL SINGOLO COMPONENTE MEDIANTE IL NOME E IL MECCANISMO DELL'INDICE.

- **ACCESSO PER SCANSIONE**

LA LISTA E' UN ESEMPIO DI ACCESSO PER SCANSIONE: CONSENTE L'ACCESSO ALL'ELEMENTO GENERICO SOLO DOPO AVER SCANDITO GLI ELEMENTI CHE LO PRECEDONO.

- **ACCESSO AGLI ESTREMI**

  PILA E CODA SONO ESEMPI DI ACCESSO AGLI ESTREMI:

  PILA (LIFO LAST IN FIRST OUT)

  CODA (FIFO FIRST IN FIRST OUT)

b) MODI DI AGIRE

POSSIAMO AVERE I SEGUENTI TIPI DI OPERAZIONE:

- **LETTURA DEL VALORE DI UN COMPONENTE (ISPEZIONE)**
- **AGGIORNAMENTO DEL VALORE DI UN COMPONENTE (CAMBIO DI VALORE)**
- **INSERIMENTO DI UN NUOVO COMPONENTE (SCRITTURA)**
- **RIMOZIONE DI UN COMPONENTE (CANCELLAZIONE)**

SONO POSSIBILI DIVERSE COMBINAZIONI TRA I MODI DI ACCEDERE E I MODI D'OPERARE.

UN PROBLEMA RIGUARDA LA COMBINAZIONE TRA *ACCESSO DIRETTO* A TUTTA LA STRUTTURA E POSSIBILITÀ DI INSERIMENTO E RIMOZIONE DI COMPONENTI .

GENERALMENTE QUESTE OPERAZIONI ALTERANO LA NUMERAZIONE DEI COMPONENTI E CIÒ È IN CONTRASTO COL PRINCIPIO CHE I COMPONENTI SIANO INDIVIDUATI DAL NUMERO D'ORDINE DELLA LORO POSIZIONE.

TUTTE LE STRUTTURE LINEARI, TRANNE IL VETTORE, SONO DOTATE DI OPERATORI DI INSERIMENTO E RIMOZIONE DI COMPONENTI.

Nei testi di programmazione, quando ci si riferisce alla sequenza, si fa riferimento a possibili operazioni come:

SEQUENCE

% Restituisce **true** se la sequenza è vuota
boolean empty()

% Restituisce **true** se p è uguale a pos_0 oppure a pos_{n+1}
boolean finished(POS p)

% Restituisce la posizione del primo elemento
POS head()

% Restituisce la posizione dell'ultimo elemento
POS tail()

% Restituisce la posizione dell'elemento che segue p
POS next(POS p)

% Restituisce la posizione dell'elemento che precede p
POS prev(POS p)

SEQUENCE

% Inserisce l'elemento v di tipo ITEM nella posizione p.
% Ritorna la nuova posizione, che diviene il predecessore di p
POS insert(POS p, ITEM v)

% Rimuove l'elemento contenuto nella posizione p.
% Ritorna il successore di p, che diviene successore del predecessore di p
POS remove(POS p)

% Legge l'elemento di tipo ITEM contenuto nella posizione p
ITEM read(POS p)

% Scrive l'elemento v di tipo ITEM nella posizione p
write(POS p, ITEM v)

LISTE

E' UNA *STRUTTURA DATI DINAMICA*, FORMALMENTE UNA SEQUENZA FINITA, ANCHE VUOTA, DI ELEMENTI DELLO STESSO TIPO TRA I QUALI ESISTE UNA RELAZIONE DI ORDINE CHE CI CONSENTE DI PARLARE DI "*PRIMO ELEMENTO*" O "*j-ESIMO ELEMENTO*". GLI ELEMENTI, CUI SONO ASSOCIATE DELLE INFORMAZIONI, SONO DEFINITI ATOMI O NODI.

LA NOTAZIONE

$$L = < a_1, a_2, \ldots , a_n > \qquad n>=0$$

A CIASCUN ELEMENTO DI UNA LISTA VIENE ASSOCIATA UNA POSIZIONE **pos(i)**

E UN VALORE **a(i)**

LA DIFFERENZA COL CONCETTO DI INSIEME E' CHE MENTRE IN UN INSIEME UN ELEMENTO NON PUO' COMPARIRE PIU' DI UNA VOLTA, NELLA LISTA UNO STESSO ELEMENTO PUO' COMPARIRE PIU' VOLTE, IN POSIZIONI DIVERSE.

L'ACCESSO

- **SI PUO' ACCEDERE DIRETTAMENTE SOLO AL PRIMO ELEMENTO DELLA SEQUENZA.**

- **PER ACCEDERE AL GENERICO ELEMENTO OCCORRE SCANDIRE SEQUENZIALMENTE GLI ELEMENTI DELLA LISTA CHE LO PRECEDONO**

LE OPERAZIONI

OLTRE ALLE OVVIE OPERAZIONI DI ISPEZIONE E SCRITTURA E' POSSIBILE MODIFICARE LA STRUTTURA AGGIUNGENDO (INSERIMENTO) E TOGLIENDO (CANCELLAZIONE) ELEMENTI.

POICHE' LA LISTA E' A DIMENSIONE VARIABILE QUESTE OPERAZIONI CHE ALTERANO LA DIMENSIONE SONO CRITICHE MA FONDAMENTALI.

LA LISTA E' DUNQUE UNA STRUTTURA DATI DINAMICA.

LA LUNGHEZZA DI UNA LISTA E' IL NUMERO DEI SUOI ELEMENTI. SE IL NUMERO DI ELEMENTI E' ZERO LA LISTA SI DICE VUOTA.

LA LUNGHEZZA CONTA LE POSIZIONI, NON I SIMBOLI DISTINTI, COSI' UN SIMBOLO CHE COMPARE K VOLTE CONTRIBUISCE CON K UNITA' ALLA LUNGHEZZA DELLA LISTA.

SE $L = < a_1, a_2, \ldots, a_n >$ E' UNA LISTA

ALLORA PER OGNI i E j TALI CHE

$$1 \leq i \leq j \leq n$$

$< a_i, a_{i+1}, \ldots, a_j >$ E' UNA SOTTOLISTA DI L, OTTENUTA PARTENDO DALLA POSIZIONE i-esima E PRENDENDO TUTTI GLI ELEMENTI FINO ALLA POSIZIONE j-esima.

LA LISTA VUOTA < > E' SOTTOLISTA DI QUALSIASI LISTA.

USARE LE LISTE

NELLA LOGICA DELLA ASTRAZIONE DATI LO SFORZO DEL PROGETTISTA E' TESO A RENDERE POSSIBILE, AVENDO A DISPOSIZIONE GLI OPERATORI DEFINITI DA UNA OPPORTUNA ALGEBRA PER IL TRATTAMENTO DI UNA LISTA, LA SOLUZIONE DI UN GENERICO PROBLEMA, *EVITANDO DI ENTRARE NEI DETTAGLI RELATIVI A COME LA LISTA SI E' IMPLEMENTATA*.

ESEMPI DELLA DIFFICOLTA' DI COMPRENDERE IL COMPITO REALIZZATO DA UN PEZZO DI CODICE SE NON SI FA USO DI APPROPRIATE TECNICHE DI ASTRAZIONE SONO DATI DI SEGUITO.

```
Const namesize = 20;
Type nameformat = packed array [ 1..namesize ] of char;
        pointer = ^node;
        node = record
                        name : nameformat;
                        next : pointer;
                    end;
Procedure BOOOO (newname: nameformat; var current, previuous :
       pointer; head: pointer; var found : boolean);
Var notfound:boolean;
Begin
      previous := nil; notfound := true; current := head;
      found := false;
            while notfound and (current <> nil) do
            with current^ do
                        if newname <= nameformat then
                                    notfound := false
                        else
                        begin
                                    previous := current;
                                    current := next;
                        end;

            if current <> nil then
                        if newname = current^.nameformat then found := true
                        else found := false
end
```

E' LA REALIZZAZIONE DELL'ALGORITMO DI RICERCA DI UN NOME IN UNA LISTA DI NOMI

1. **DEFINISCI IL NOME CERCATO E L'INIZIO LISTA**
2. **INIZIALIZZA *PRECEDENTE* E *CORRENTE* A INIZIO LISTA**
3. **PONI *TROVATO* A FALSO**
4. **MENTRE LA RICERCA CONTINUA E NON E' FINITA LA LISTA ESEGUI:**

   a) SE IL NOME CERCATO PRECEDE ALFABETICAMENTE IL NOME CORRENTE ALLORA

   a. 1) INTERROMPI LA RICERCA
   ALTRIMENTI

   a'. 1) PONI *PRECEDENTE* A *CORRENTE*

   a'. 2) AGGIORNA *CORRENTE* PUNTANDO ALL'ELEMENTO SUCCESSIVO

- **STABILISCI SE IL NOME CERCATO E' *TROVATO***
- **RESTITUISCI *PRECEDENTE*, *CORRENTE* E *TROVATO***

```
#include <stdio.h>
    #include <malloc.h>
    #include <string.h>

#define LUNGINFO 60          // allineamento a 64 byte
    //-------------------------------------------------
    typedef struct Nodo { char       Info[LUNGINFO];
                          struct Nodo *Next;
                        } NODO;
    //-------------------------------------------------

int  MenuUtente  ( void   );
    void Ins ( NODO * );
    void Es  ( NODO * );
    void Stampa      ( NODO * );
    void LiberaLista ( NODO * );

int main (void)
    {
      NODO  TestaLista;     // Puntatore di testa della lista
      int   Scelta;         // Operazione chiesta dall'utente;
                            // 0 per Exit

      strcpy (TestaLista.Info, "Archivio Clienti 2005");
```

```
TestaLista.Next = NULL;

        do
        { Scelta = MenuUtente();
          switch ( Scelta )
          {
            case 1: Ins ( &TestaLista );
                    break;
            case 2: Es ( &TestaLista );
                    break;
            case 3: Stampa      ( &TestaLista );
                    break;
          }
        }
        while (Scelta);
  LiberaLista ( &TestaLista );

        return 0;
      }
int MenuUtente
      ( void
      )
      { int Scelta;
```

```
  return Scelta;
     }
  printf ("\n-------------------------------------\n");
        printf ("  [1] ");
        printf ("  [2] ");
        printf ("  [3] ");
        printf ("  [0] ");

        do Scelta = getc(stdin); while (Scelta<'0' || Scelta>'3');

        Scelta -= '0'; // trasforma carattere ASCII in intero
void Ins
      ( NODO *Prec   // Puntatore al Nodo di Testa della lista
      )
      { char Buffer[200];
        NODO *Succ,
             *Temp;

        printf ("\n-------------------------------------\n");
        Temp = (NODO *)malloc( sizeof(NODO) );
  if (Temp)
        { printf ("  Nome nuovo elemento ? ");
```

```
 printf ("\n-----------------------------------\n");
       printf ("  Nome elemento ");
       scanf ("%s", Buffer);
       Buffer[LUNGINFO-1] = 0;  //NULL per sicurezza

  Succ = Prec->Next;
       while ( Succ && strcmp (Succ->Info, Buffer) )
       { Prec = Succ;
         Succ = Prec->Next;
       }

  if (Succ)

  { Prec->Next = Succ->Next;
         free (Succ);
       }
       else
       { printf ("  Elemento NON presente in lista.\n");
       }
     }

void Stampa
     ( NODO *Punt       // Puntatore al Nodo di Testa della lista
     )
     { printf ("\n-----------------------------------\n");
       printf ("  %s\n\n", Punt->Info );
```

```
while ( Punt->Next)
        { Punt = Punt->Next;
          printf ("  %s\n", Punt->Info );
        }
      }
void LiberaLista
     ( NODO *Prec   // Puntatore al Nodo di Testa della lista
     )
     { NODO *Succ;
  Succ = Prec->Next;

       // sgancia il puntatore di testa dalla lista
       Prec->Next = NULL;
  while ( Succ )
       { Prec = Succ;
         Succ = Prec->Next;
         free ( Prec );
       }
     }
```

E’ un esempio di gestione di una lista lineare creata già ordinata e di ricerca ed estrazione di un elemento dato.

Il puntatore di testa verrà posizionato in una variabile di tipo NODO, il cui campo Info potrà essere usato per informazioni generali sulla lista.

- I nodi necessari verranno allocati da memoria Heap.
- Il menu elenca tutte le scelte possibili all'utente e richiede la scelta di un'opzione. E’ presente l’inserimento ordinato di un nuovo elemento nella Lista. Il nodo necessario verrà allocato nella memoria Heap con malloc().
- Si utilizza il valore definito in LUNGINFO per marcare con NULL la fine della stringa di informazione, nel caso l'utente l'abbia inserita più lunga.
- Partendo dal primo elemento dopo il nodo di testa, si cerca il primo campo Info maggiore di Buffer e ci si ferma comunque in coda lista.
- Si ricerca ed estrae un elemento dalla Lista. Il nodo inutilizzato verrà rilasciato con free().
- Si ha una visualizzazione completa della Lista
- Si rilasciano tutti i nodi eventalmente presenti nella Lista con free(). Si azzera il campo Next nel puntatore di testa.

SPECIFICA

TIPI:

LISTA: INSIEME DELLE SEQUENZE L = < a_1, a_2, … , a_n >, n>=0, DI ELEMENTI DI TIPO TIPOELEM DOVE L'ELEMENTO i-ESIMO HA VALORE a_i E POSIZIONE pos(i)

BOOLEAN: INSIEME DEI VALORI DI VERITA'

OPERATORI:

CREALISTA : ( ) → LISTA
CREALISTA = L'
POST: L' = Λ, Λ = < > (sequenza vuota)

LISTAVUOTA : (LISTA) → BOOLEAN
LISTAVUOTA(L) = b
POST: b = TRUE SE L= < >
b = FALSE ALTRIMENTI

LEGGILISTA: (POSIZIONE, LISTA) → TIPOELEM
LEGGILISTA(p,L) = a
PRE: p = pos(i) $1 \leq i \leq n$
POST: a = a_i

SCRIVILISTA: (TIPOELEM,POSIZIONE,LISTA) → LISTA

SCRIVILISTA(a,p,L) = L'

PRE: p = pos(i) 1 $i \leq n$

POST: L' = < $a_1, a_2, \ldots, a_{i-1}, a, a_{i+1}, \ldots, a_n$ >

PRIMOLISTA: (LISTA) → POSIZIONE

PRIMOLISTA(L) = p

POST: p = pos(1) (se L= Λ pos(1)=pos(n+1))

FINELISTA : (POSIZIONE, LISTA) → BOOLEAN

FINELISTA(p,L) = b

PRE:p = pos(i) 1 $\leq i \leq n+1$

POST: b = TRUE SE p = pos(n+1)

b= FALSE ALTRIMENTI

SI PUO' SOSTITUIRE CON
ULTIMOLISTA(L)=p
POST: p=pos(n)

SUCCLISTA: (POSIZIONE, LISTA) → POSIZIONE

SUCCLISTA(p,L) = q

PRE: p = pos(i) 1 $\leq i \leq n$

POST: q = pos(i+1)

PREDLISTA: (POSIZIONE, LISTA) → POSIZIONE

PREDLISTA(p,L) = q

PRE: p = pos(i) 2 ≤i ≤ n

POST: q = pos(i-1)

INSLISTA: (TIPOELEM,POSIZIONE,LISTA) → LISTA

INSLISTA(a,p,L) = L'

PRE: p = pos(i) 1 ≤ i ≤ n+1

POST: L' = < a 1, a 2, … , a i-1, a, a i , a i+1, … , a n >, se 1 ≤ i ≤ n

L' = <a 1, a 2, … ,a n, a > , se i = n+1

(e quindi L' = < a > se i = 1 e L = < >)

CANCLISTA: (POSIZIONE, LISTA) → LISTA

CANCLISTA(p,L) = L'

PRE: p = pos(i) 1 ≤ i ≤ n

POST: L' = < a 1, a 2, … , a i-1, a i+1, … , a n >

DALLA SPECIFICA SEMANTICA EMERGE CHE PER ACCEDERE A UN ELEMENTO OCCORRE CONOSCERNE LA POSIZIONE COSA CHE NELLE LISTE E' POSSIBILE SOLO PER IL PRIMO ELEMENTO.

L'UNICO OPERATORE CHE DA' PER RISULTATO DIRETTAMENTE LA POSIZIONE E' PRIMOLISTA.

PER GLI ALTRI ELEMENTI LA POSIZIONE SI OTTIENE CONOSCENDO LA POSIZIONE DELL'ELEMENTO PRECEDENTE (O SEGUENTE) E APPLICANDO L'OPERAZIONE SUCCLISTA (O PREDLISTA). DUNQUE, PER ACCEDERE AD UN GENERICO ELEMENTO OCCORRE

SCANDIRE LA LISTA A PARTIRE DAL PRIMO ELEMENTO

L'OPERATORE LISTAVUOTA E' RIDONDANTE VISTO CHE PUO' ESSERE SOSTITUITO DA

FINELISTA (PRIMOLISTA (L) , L)

NOTA: ***IN ALCUNI TESTI SI TROVA UNA DIVERSA SPECIFICA DI PRIMOLISTA(L) CHE RESTITUISCE L'ELEMENTO PRIMO PIUTTOSTO CHE LA POSIZIONE. QUESTO CORRISPONDE ALLA SPECIFICA***

PRIMO(LISTA) ⟶ TIPOELEM

L'ALGEBRA CHE ABBIAMO PROPOSTO E' LA PIU' COMPLETA PER LISTE MODIFICABILI. TUTTAVIA POTREMMO PENSARE AD UN'ALGEBRA PIU' SEMPLICE CHE PREVEDA SOLO LE FUNZIONI DI AGGIORNAMENTO SUL PRIMO ELEMENTO DELLA LISTA.

INITLISTA: () → LISTA

LISTAVUOTA: (LISTA) → BOOLEAN

INSERISCITESTALISTA: (TIPOELEM, LISTA) → LISTA

TESTALISTA: (LISTA) → TIPOELEM

CANCELLAPRIMO: (LISTA) → LISTA

RESTOLISTA: (LISTA) → LISTA

LE OPERAZIONI ELEMENTARI PREVISTE SONO SUFFICIENTI A REALIZZARE OPERAZIONI PIU' COMPLESSE

L'ALGORITMO DI RICERCA DI UN NOME IN UNA LISTA DI NOMI (Pseudocodice che usa L'ALGEBRA DI LISTA)

```
dimnome = 20
tiponome: array di dimnome elementi di tipo char
p_cella: puntatore a cella
cella: tipo strutturato con componenti
    - nome: di tipo tiponome
    - successivo: di tipo p_cella
cosa: di tipo p_cella
posizione: di tipo p_cella

BOOOO2(cercato: di tipo tiponome; corrente, precedente: di tipo
posizione, L: di tipo cosa,  trovato: di tipo boolean)
    corrente ← PRIMOLISTA(L)
    precendente ← corrente
    continua ← TRUE
    trovato ← FALSE
    while (continua and not FINELISTA(corrente, L)) do
        nome_corr ← LEGGILISTA(corrente, L)
        if cercato <= nome_corr then
                continua ← FALSE
        else
            precedente ← corrente
            corrente ← SUCCLISTA(corrente, L)
        if not FINELISTA(corrente, L) then
                TROVATO  ← (cercato = nome_corr)
```

PROBLEMA

ELIMINAZIONE DI DUPLICATI:

DATA UNA LISTA L, I CUI ELEMENTI SIANO INTERI, ELIMINARE DA L GLI ELEMENTI CHE SONO DUPLICATI.

NE DAREMO L'ALGORITMO USANDO GLI OPERATORI E SENZA FARE RIFERIMENTO AD UNA PARTICOLARE REALIZZAZIONE.

DISPONENDO DI UN LINGUAGGIO DI PROGRAMMAZIONE CHE CONSENTA LA ASTRAZIONE FUNZIONALE E LA ASTRAZIONE DATI SI PUO' FARE RIFERIMENTO ALLA STRUTTURA LISTA SENZA ENTRARE NEI DETTAGLI REALIZZATIVI.

Pseudocodice

```
EPURAZIONE(lista L)
    p ← PRIMOLISTA(L)
    while not FINELISTA(p, L) do
        q ← SUCCLISTA(p, L)
        while not FINELISTA(q, L) do
            if LEGGILISTA(p,L) = LEGGILISTA(q,L) then
                CANCLISTA(q,L) else
                q ← SUCCLISTA(q, L)
            p ← SUCCLISTA(p,L)
```

REALIZZARE LE LISTE

PER REALIZZARE UNA LISTA DISTINGUIAMO DUE RAPPRESENTAZIONI FONDAMENTALI:

- **RAPPRESENTAZIONE SEQUENZIALE:**

  LA LISTA E' RAPPRESENTATA USANDO UNA TABELLA INDICIZZATA MONODIMENSIONALE.

- **RAPPRESENTAZIONE COLLEGATA:**

  L'IDEA E' QUELLA DI MEMORIZZARE GLI ELEMENTI DELLA LISTA ASSOCIANDO AD OGNUNO DI ESSI UNA PARTICOLARE INFORMAZIONE (RIFERIMENTO) CHE PERMETTA DI INDIVIDUARE LA LOCAZIONE IN CUI E' MEMORIZZATO L'ELEMENTO SUCCESSIVO.

RAPPRESENTAZIONE CON VETTORE

UNA LISTA PUO' ESSERE RAPPRESENTATA USANDO UN VETTORE.

POICHE' IL NUMERO DI ELEMENTI CHE COMPONGONO LA LISTA PUO' VARIARE SI UTILIZZA UNA VARIABILE PRIMO PER IL VALORE DELL'INDICE DELLA COMPONENTE DEL VETTORE IN CUI E' MEMORIZZATO IL PRIMO ELEMENTO DELLA LISTA E SI UTILIZZA UN'ALTRA VARIABILE LUNGHEZZA PER INDICARE IL NUMERO DI ELEMENTI DAI QUALI E' COMPOSTA LA LISTA RAPPRESENTATA OPPURE CHE MI INDICA L'ULTIMO ELEMENTO UTILE PER CONTROLLARE IL *FINELISTA*.

QUESTA RAPPRESENTAZIONE CONSENTE DI REALIZZARE MOLTO SEMPLICEMENTE ALCUNE DELLE OPERAZIONI DEFINITE PER LA LISTA.

IL VERO PROBLEMA RIGUARDA L'INSERIMENTO E LA RIMOZIONE DI COMPONENTI

LISTA

(4 5 1 21 45)

1	4
2	5
3	1
4	21
5	45
6	78
7	12
8	1
9	-5
10	0
11	-2
12	61

PRIMO

1

LUNGHEZZA

5

LISTA

(4 5 1 11 21 45)

1	4
2	5
3	1
4	11
5	21
6	45
7	12
8	1
9	-5
10	0
11	-2
12	61

PRIMO

1

LUNGHEZZA

6

IL BANALE INSERIMENTO IN TERZA POSIZIONE DI UN NUOVO ELEMENTO 11 CAUSA LO SPOSTAMENTO VERSO IL BASSO DEL 4° E DEL 5° ELEMENTO DELL'ARRAY

LA RAPPRESENTAZIONE COLLEGATA

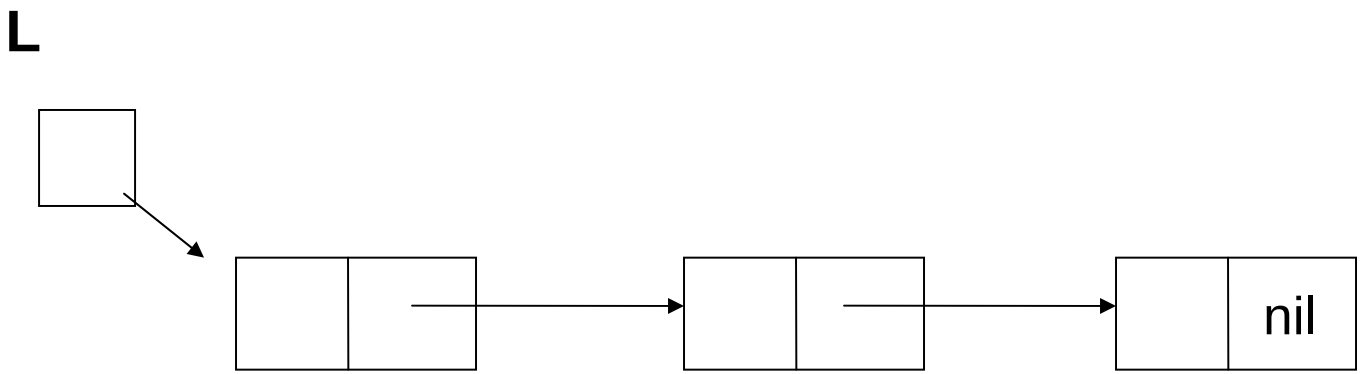
USATA PER OVVIARE ALL'INCONVENIENTE DI RIMANERE VINCOLATI DALLA STATICITA' DELLA RAPPRESENTAZIONE, QUANDO CI SIANO OPERAZIONI DI MODIFICA DELLA STRUTTURA LISTA.

L'IDEA FONDAMENTALE E' QUELLA DI MEMORIZZARE GLI ELEMENTI DELLA LISTA ASSOCIANDO AD OGNUNO DI ESSI UN RIFERIMENTO CHE PERMETTE DI INDIVIDUARE LA LOCAZIONE IN CUI E' MEMORIZZATO L'ELEMENTO SUCCESSIVO.

PER VISUALIZZARE TALE RAPPRESENTAZIONE SI USA UNA NOTAZIONE GRAFICA IN CUI

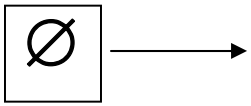
- **GLI ELEMENTI SONO RAPPRESENTATI MEDIANTE NODI**
- **I RIFERIMENTI MEDIANTE ARCHI CHE COLLEGANO NODI**

LA NOTAZIONE GRAFICA

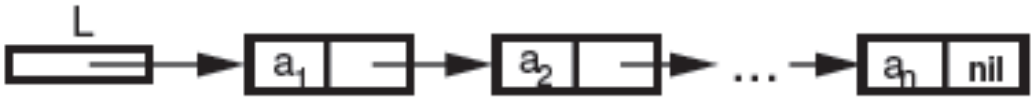


SI PUO' NOTARE CHE SI USA UN RIFERIMENTO AL PRIMO ELEMENTO DELLA LISTA (RIFERIMENTO INIZIALE) E UN SIMBOLO SPECIALE nil o ∅ COME RIFERIMENTO ASSOCIATO ALL'ULTIMO NODO.

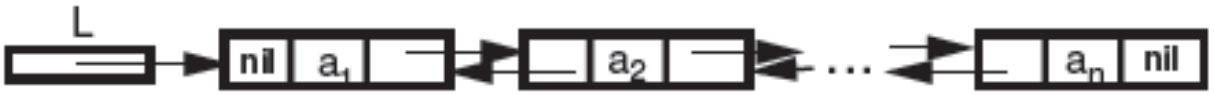
NEL CASO LA LISTA SIA CREATA MA VUOTA, TALE SIMBOLO COMPARE DIRETTAMENTE NEL RIFERIMENTO INIZIALE.



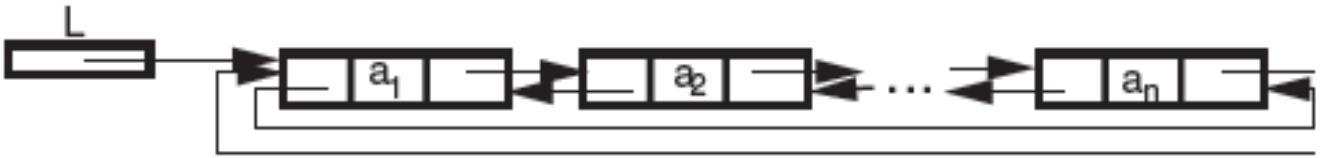
Rappresentazione collegata- esempi



monodirezionale



bidirezionale

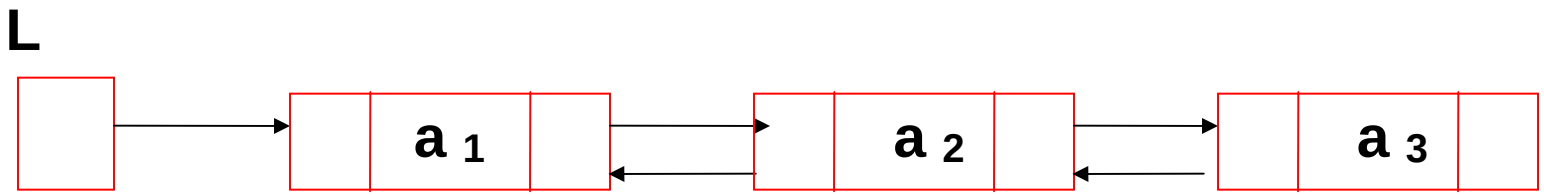


bidirezionale circolare

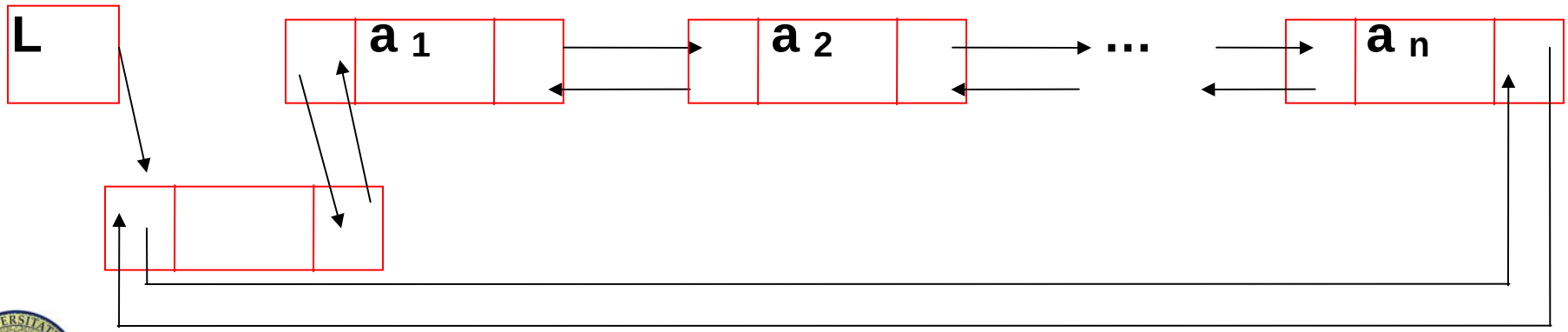
monodirezionale con sentinella

LA RAPPRESENTAZIONE A DOPPI RIFERIMENTI O SIMMETRICA

OGNI ELEMENTO CONTIENE, OLTRE AL RIFERIMENTO AL NODO SUCCESSIVO, ANCHE IL RIFERIMENTO AL PRECEDENTE (BIDIREZIONALE).



LA LISTA PUO' ESSERE "ESPANSA" CON UNA CELLA IN PIU' PER LA REALIZZAZIONE CIRCOLARE CHE PUNTI ALLA TESTA E ALLA FINE DELLA LISTA



CON QUESTA RAPPRESENTAZIONE SI HA IL VANTAGGIO DI:

- **POTER SCANDIRE LA LISTA IN ENTRAMBE LE DIREZIONI**
- **POTER INDIVIDUARE FACILMENTE L'ELEMENTO CHE PRECEDE**
- **POTER REALIZZARE LE OPERAZIONI DI INSERIMENTO SENZA DOVER USARE VARIABILI AGGIUNTIVE**

RAPPRESENTAZIONE COLLEGATA REALIZZATA CON CURSORI

E' SEMPRE UTILIZZATO UN VETTORE (ARRAY MONODIMENSIONALE) PER L'IMPLEMENTAZIONE DELLA LISTA, MA SI RIESCE A SUPERARE, ATTRAVERSO I RIFERIMENTI, IL PROBLEMA DELL'AGGIORNAMENTO (INSERIMENTO O CANCELLAZIONE DI UN ELEMENTO)

I RIFERIMENTI SONO REALIZZATI MEDIANTE CURSORI, CIOE' VARIABILI INTERE O ENUMERATIVE, IL CUI VALORE E' INTERPRETATO COME INDICE DI UN VETTORE.

SI DEFINISCE UN VETTORE SPAZIO CHE :

1) **CONTIENE TUTTE LE LISTE, OGNUNA INDIVIDUATA DA UN PROPRIO CURSORE INIZIALE**
2) **CONTIENE TUTTE LE CELLE LIBERE, ORGANIZZATE IN UNA LISTA, DETTA "LISTALIBERA"**

Esempio: Disponiamo di tre diverse liste L, M, S

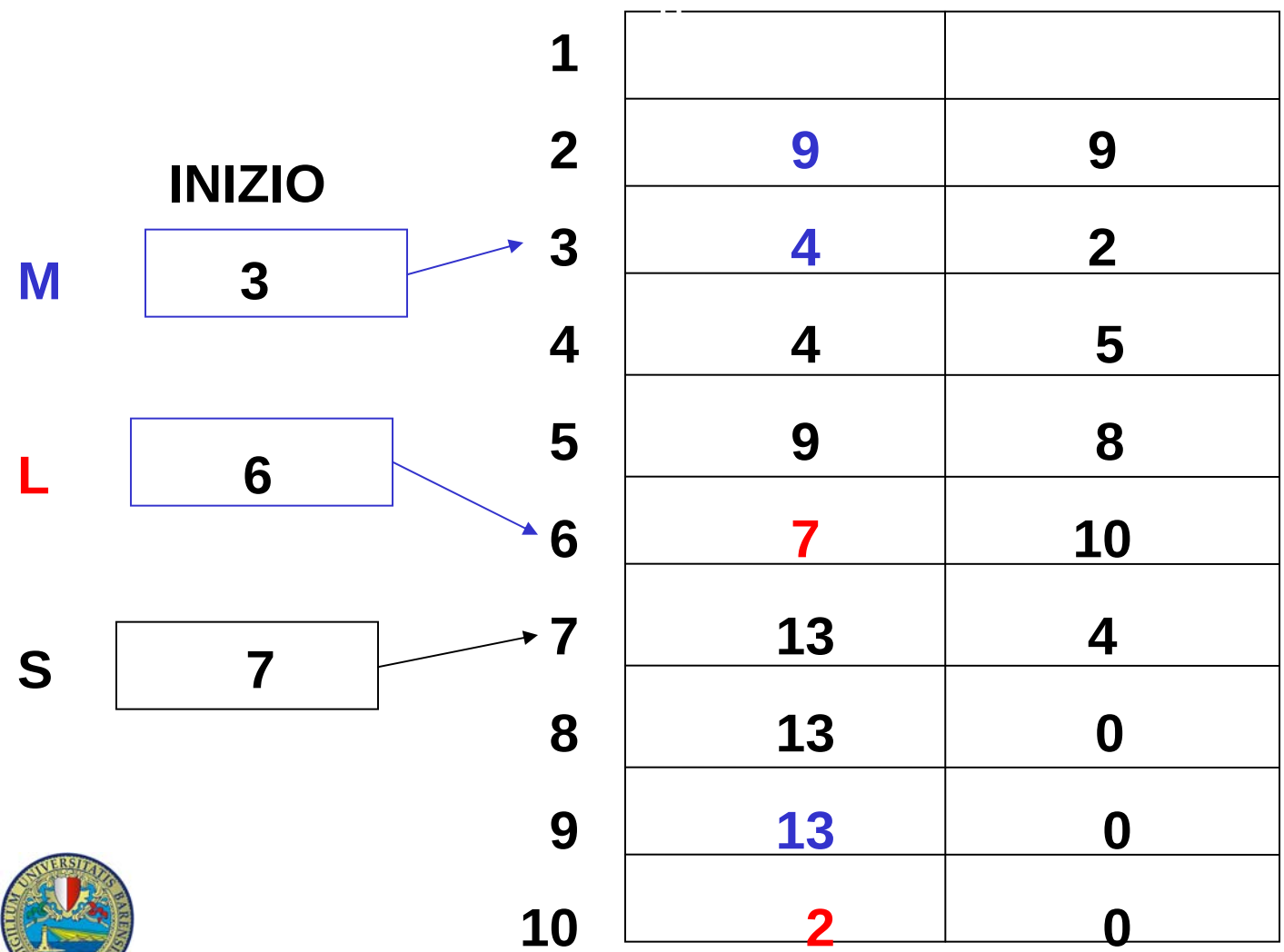
L = <7, 2> M = <4, 9, 13> S = <13, 4, 9, 13>

POSSIAMO USARE UN UNICO VETTORE SPAZIO PER RAPPRESENTARE LE TRE LISTE.

LA COMPONENTE DI SPAZIO HA DUE CAMPI: NEL CAMPO "*ELEMENTO*" E' MEMORIZZATO IL CONTENUTO DEL NODO, NEL CAMPO "*SUCCESSIVO*" E' IL RIFERIMENTO AL PROSSIMO NODO. NEL CASO SI SCELGA LA REALIZZAZIONE A DOPPI RIFERIMENTI CI SARA' ANCHE UN CAMPO "*PRECEDENTE*"

OGNI LISTA E' RICOSTRUIBILE INIZIANDO DALLA COMPONENTE DELL'ARRAY CORRISPONDENTE AL VALORE DI INIZIO.

SPAZIO


elemento
Succ.
INIZIO
M
3
L
6
S
7
1
2
3
4
5
6
7
8
9
10
9
9
4
2
4
5
9
8
7
10
13
4
13
0
13
0
2
0

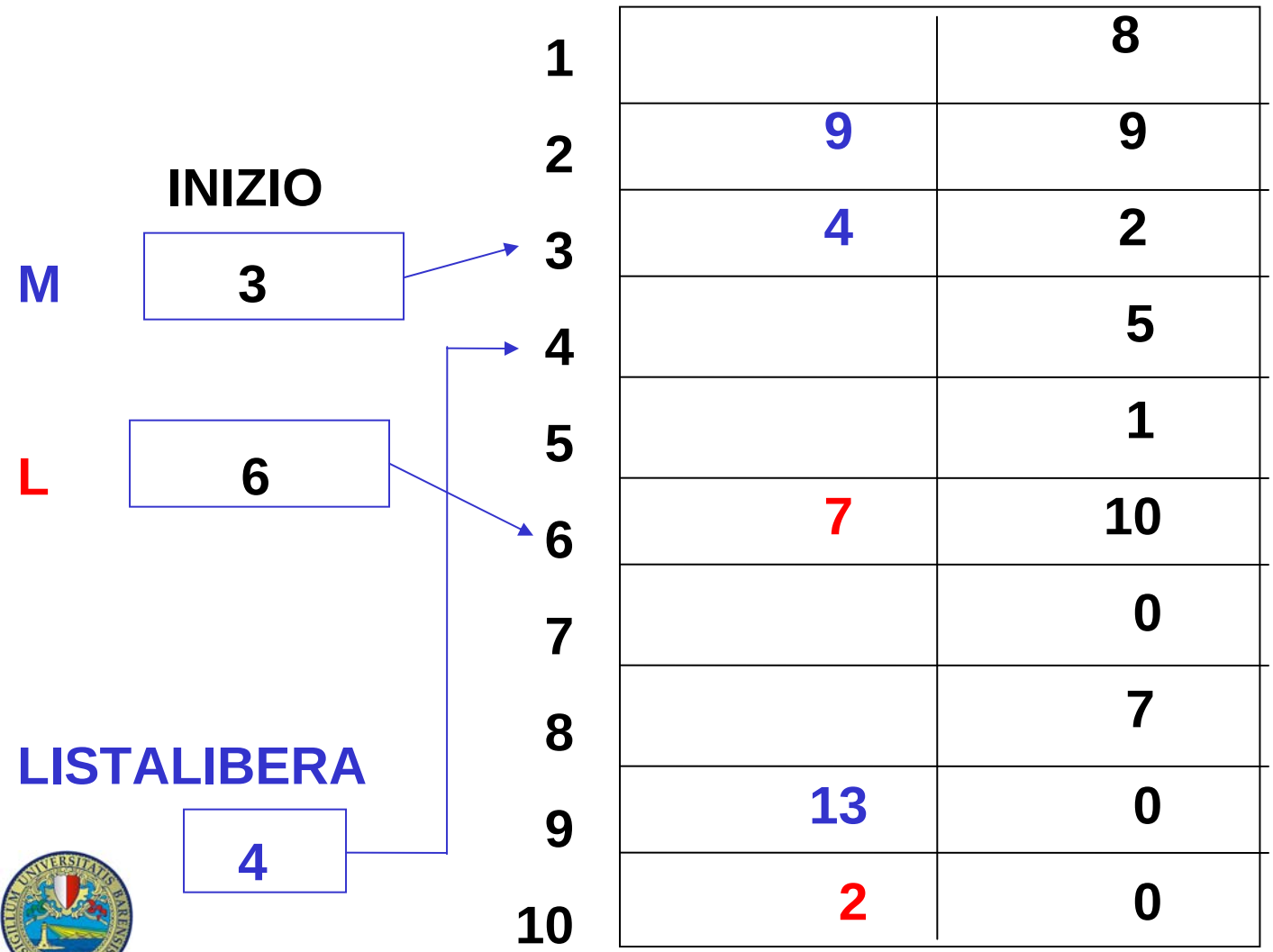
NEL CAMPO "ELEMENTO" DELLA COMPONENTE INDIRIZZATA DA INIZIO E' MEMORIZZATO IL VALORE DEL PRIMO ELEMENTO. SEGUENDO I RIFERIMENTI (CURSORI) SI TROVANO GLI ELEMENTI SUCCESSIVI DELLA LISTA.

UTILIZZANDO I CURSORI, SI PUO' DEFINIRE LA POSIZIONE pos(i) DELL'ELEMENTO i–ESIMO DI L UGUALE AL VALORE DEL CURSORE ALLA CELLA DEL VETTORE SPAZIO CHE CONTIENE L'ELEMENTO (i-1)-esimo, SE $2 \leq i \leq n+1$, UGUALE A 0 SE i = 1.

ANALOGAMENTE, SI DEFINISCE pos(n+1) UGUALE AL CURSORE ALLA CELLA DI SPAZIO CHE CONTIENE L'ELEMENTO n-esimo SE $n \geq 1$, O UGUALE A 0 ALTRIMENTI. LA LISTA VUOTA SI INDICA CON L=∅.

PER POTER AGGIORNARE UNA LISTA COSI' REALIZZATA SORGE IL PROBLEMA DI INDIVIDUARE LA POSIZIONE DI UNA COMPONENTE NON ANCORA UTILIZZATA NELL'ARRAY.

SI USA UNA LISTALIBERA MEMORIZZATA UTILIZZANDO LO STESSO ARRAY SPAZIO PER RACCOGLIERE IN MODO COLLEGATO LE COMPONENTI LIBERE.



LISTALIBERA RAPPRESENTA UN SERBATOIO DA CUI PRELEVARE COMPONENTI LIBERE DELL'ARRAY E IN CUI RIVERSARE LE COMPONENTI DELL'ARRAY CHE NON SONO PIU' UTILIZZATE PER LA LISTA.

GRAZIE ALLA LISTALIBERA, CON QUESTA RAPPRESENTAZIONE L'INSERIMENTO E LA ELIMINAZIONE DI UN ELEMENTO NON RICHIEDONO LO SPOSTAMENTO DI ALTRI ELEMENTI DELLA LISTA.

PERO' E' NECESSARIO GESTIRE L'ARRAY SPAZIO E AL CONTEMPO LA LISTALIBERA.

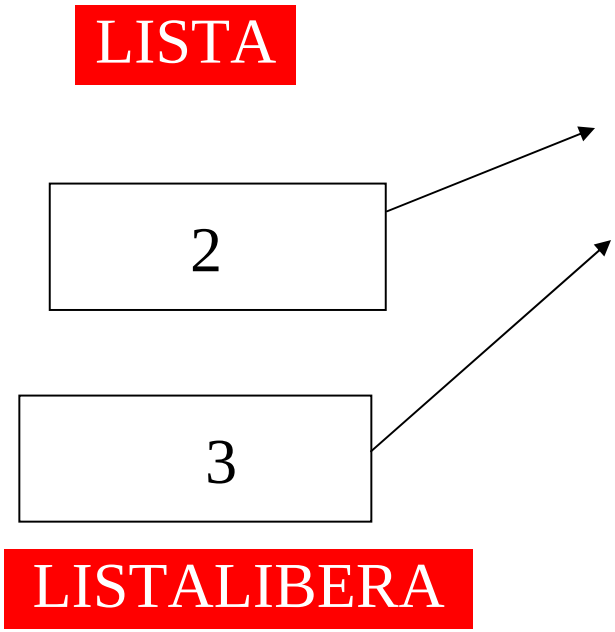
QUANDO SI DEVE INSERIRE UN NUOVO ELEMENTO NELLA LISTA, SI PRELEVA UNA COMPONENTE DELLA LISTALIBERA E LA SI UTILIZZA PER MEMORIZZARE IL NUOVO ELEMENTO, COLLEGANDOLO IN MODO OPPORTUNO AGLI ALTRI ELEMENTI DELLA LISTA.

UNA LISTA PUO' ESSERE DEFINITA AD ESEMPIO:

```
Type    lista=0..maxlung;
        posizione:lista
var listalibera:lista
        spazio: array[0..maxlung-1] of record
        precedente:posizione
        elemento: tipoelem
        successivo: posizione
```

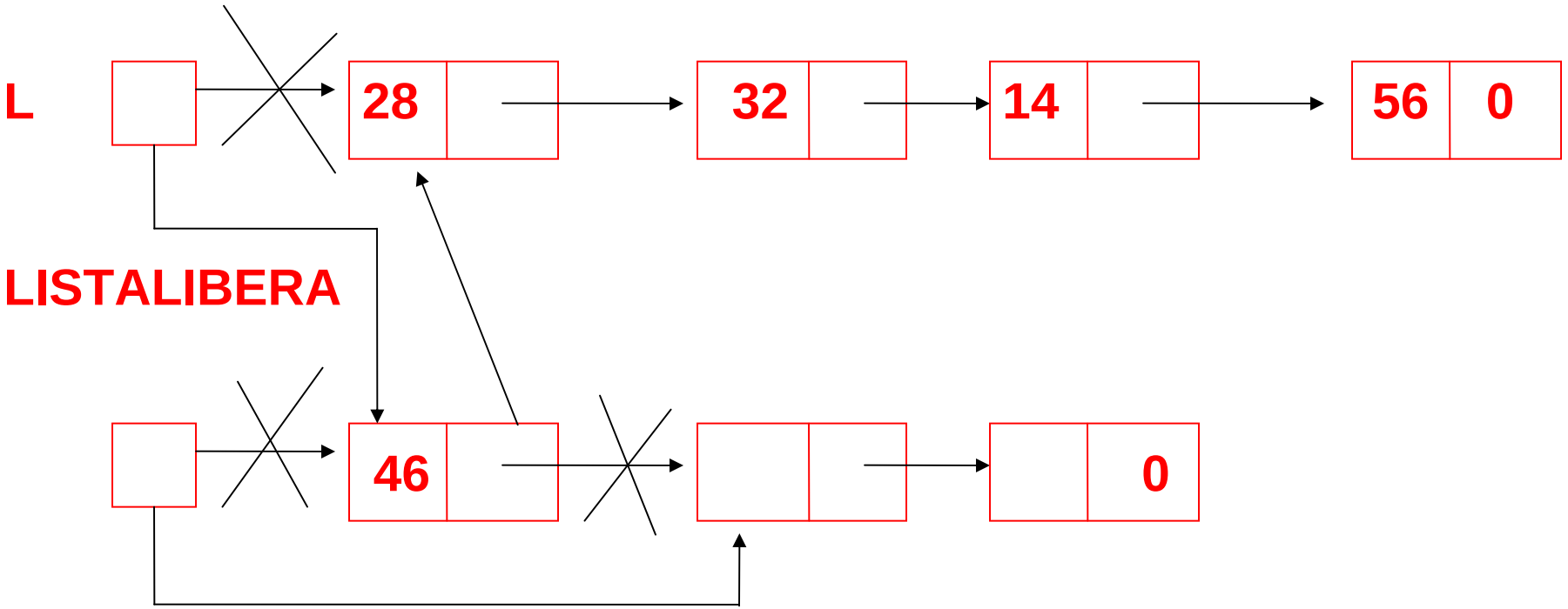
ES.:

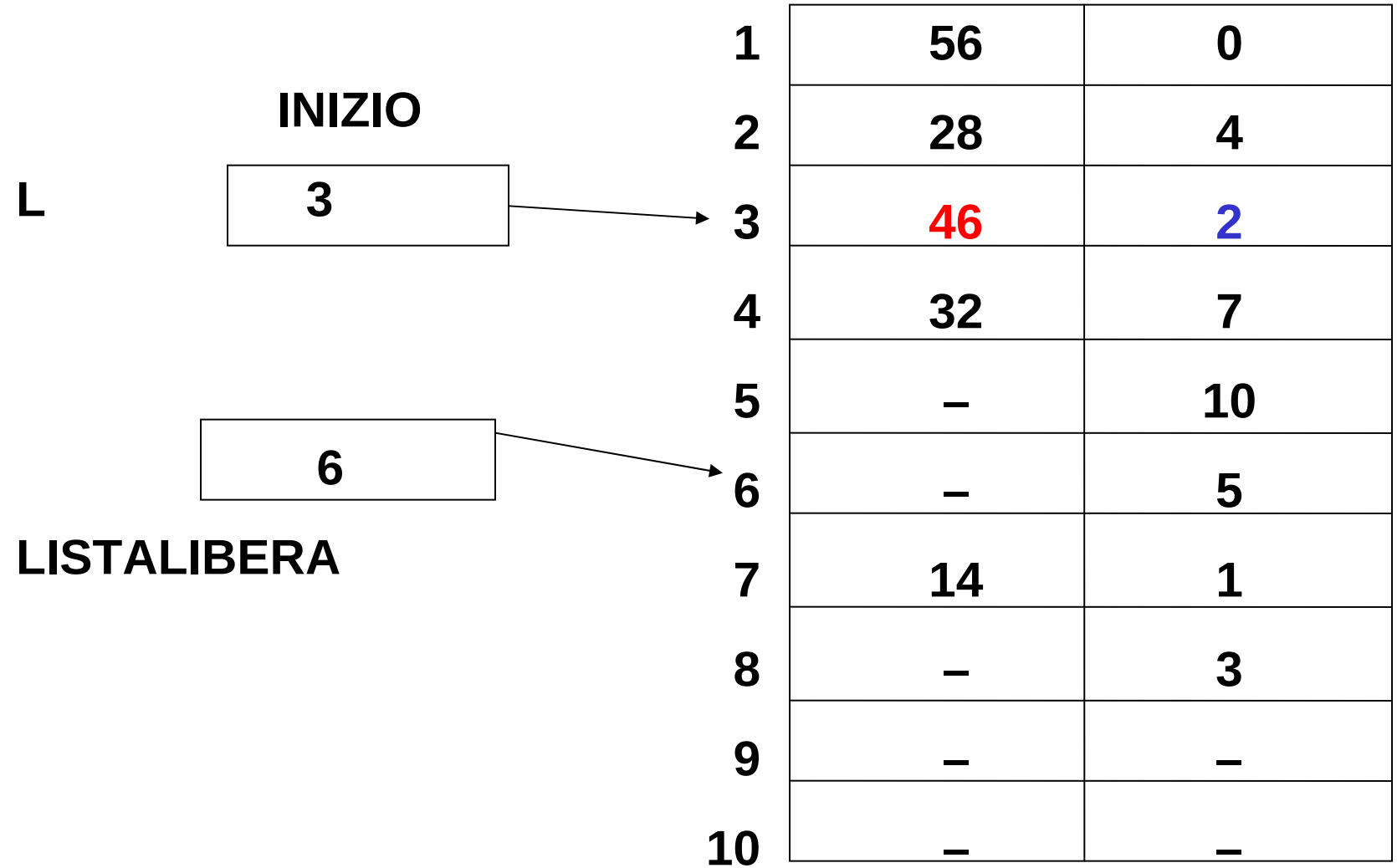
L = 28, 32, 14, 56



1	56	0
2	28	4
3	–	6
4	32	7
5	–	10
6	–	5
7	14	1
8	–	0
9	–	8
10	–	9

SE SI VUOLE INSERIRE UN NUOVO ELEMENTO (= 46) IN TESTA ALLA LISTA SI FA USO DELLA PRIMA COMPONENTE DELLA LISTA LIBERA

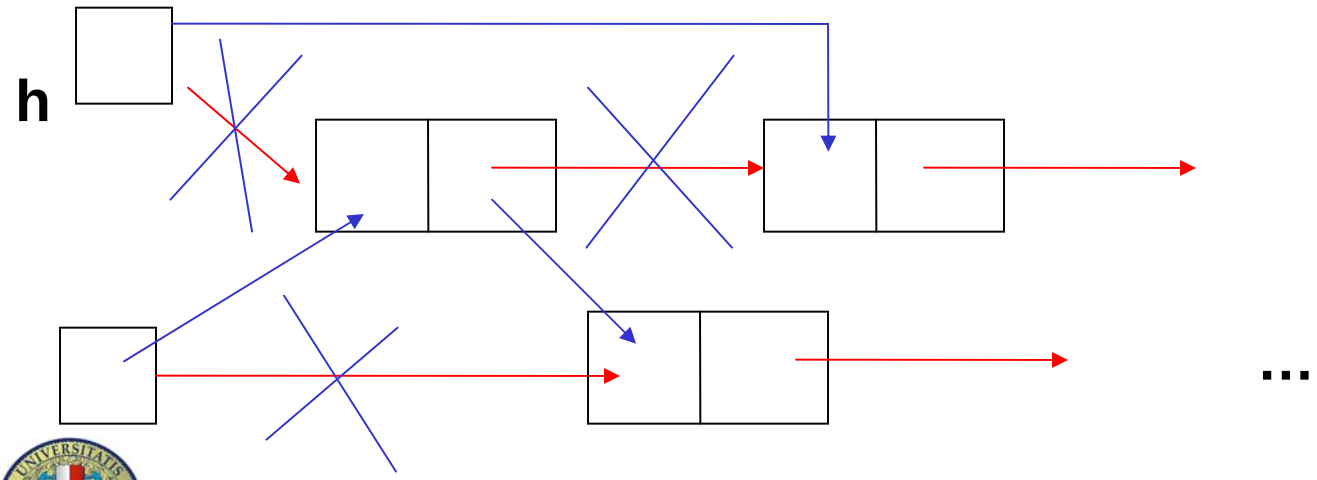



INIZIO
L
3
6
LISTALIBERA
1 56 0
2 28 4
3 46 2
4 32 7
5 – 10
6 – 5
7 14 1
8 – 3
9 – –
10 – –

E' FONDAMENTALE DISPORRE DELLA PROCEDURA SPOSTA(h,k) CHE TRASFERISCE LA CELLA PUNTATA DA h SPOSTANDOLA PRIMA DELLA CELLA PUNTATA DA k

SPOSTA(posizione h, posizione k)

temp ← spazio[k].successivo
spazio[spazio[h].precedente].successivo ← spazio[h].successivo
spazio[spazio[h].successivo].precedente ← spazio[h].precedente
spazio[h].precedente ← spazio[k].precedente
spazio[spazio[k].precedente].successivo ← h
spazio[h].successivo ← k
spazio[k].precedente ← k
k ← h
h ← temp



LE OPERAZIONI ESEGUIBILI SULLA LISTALIBERA SONO:

- **LA INIZIALIZZAZIONE, CHE SERVE A COLLEGARE TRA LORO LE VARIE COMPONENTI DELL'ARRAY CHE FANNO PARTE INIZIALMENTE DELLA LISTALIBERA;**
- **IL PRELIEVO DI UNA COMPONENTE DELLA LISTALIBERA SE ESISTE;**
- **LA RESTITUZIONE DI UNA COMPONENTE ALLA LISTALIBERA.**

IN PSEUDOCODICE

La lista deve essere stata creata (vale a dire *listalibera* ← *0*)

```
INIZIALIZZA_LISTA_LIBERA()
    listalibera ← 1
    for i=1 to maxlung – 1 do begin
        SPAZIO[i].successivo ← (i+1)
        SPAZIO[i].precedente ← (i-1)
    end
```

PER QUANTO RIGUARDA GLI OPERATORI DELL'ALGEBRA CONSIDERIAMO QUELLI DI MODIFICA DELLA STRUTTURA.

IN PSEUDOCODICE

```
INSLISTA(tipoelem a, posizione p, lista L)
   if  listavuota(listalibera) then error
   else begin
      SPOSTA(listalibera,p)
      SPAZIO[p].elemento ← a
   end

CANCLISTA(posizione p, lista L)
         SPOSTA(p,listalibera)
```

CONSIDERAZIONI SULLA REALIZZAZIONE DI UNA RAPPRESENTAZIONE COLLEGATA CON CURSORI

A FAVORE

+UNO DEGLI SVANTAGGI DELLA RAPPRESENTAZIONE SEQUENZIALE E' SUPERATO: L'INSERIMENTO E LA ELIMINAZIONE DI UN ELEMENTO NON RICHIEDONO LO SPOSTAMENTO DI ALTRI ELEMENTI NELLA LISTA

+LA COMPLICAZIONE DATA DALLA NECESSITA' DI GESTIRE LA LISTA LIBERA E' COMPENSATA DAL FATTO CHE GLI AGGIORNAMENTI RICHIEDONO UN NUMERO LIMITATO DI OPERAZIONI

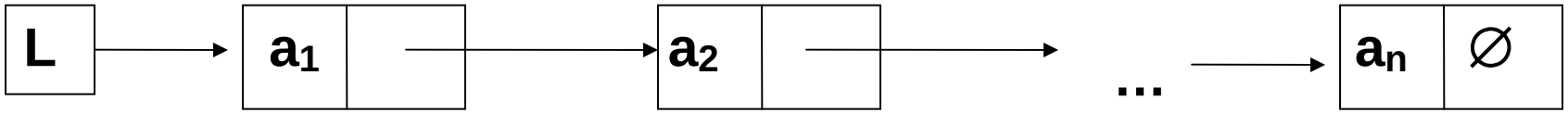
A SFAVORE

–RIMANGONO GLI SVANTAGGI CONNESSI ALL'USO DELL'ARRAY: LA DIMENSIONE DELL'ARRAY RAPPRESENTA UN LIMITE ALLA CRESCITA DELLA LISTA E LA QUANTITA' IN MEMORIA UTILIZZATA NON DIPENDE DALLA LUNGHEZZA EFFETTIVA DELLA LISTA

–RISPETTO ALLA RAPPRESENTAZIONE SEQUENZIALE VI E' UN'ULTERIORE OCCUPAZIONE DI MEMORIA, VISTA LA NECESSITA' DI MEMORIZZARE I RIFERIMENTI.

RAPPRESENTAZIONE COLLEGATA DI LISTA REALIZZATA MEDIANTE PUNTATORI

SI CONSIDERI LA REALIZZAZIONE DI UNA LISTA *MONODIREZIONALE SEMPLIFICATA*. VI E' UNA STRUTTURA DI n "CELLE", TALE CHE L'i-esima CELLA CONTIENE L'i-esimo ELEMENTO DELLA LISTA E L'INDIRIZZO DELLA CELLA CHE CONTIENE L'ELEMENTO SUCCESSIVO

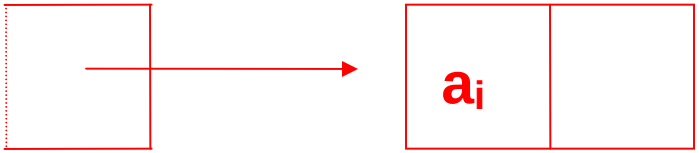


- LA PRIMA CELLA E' INDIRIZZATA DA UNA VARIABILE L DI TIPO PUNTATORE
- L'ULTIMA CELLA PUNTA A UN VALORE CONVENZIONALE ∅ NIL
- GLI INDIRIZZI SONO NOTI ALLA MACCHINA MA NON AL PROGRAMMATORE
- LA POSIZIONE **pos(i)** E' UGUALE AL VALORE DEL PUNTATORE ALLA CELLA CHE CONTIENE L'i-esimo ELEMENTO (ATOMO)

$1 \leq i \leq n$

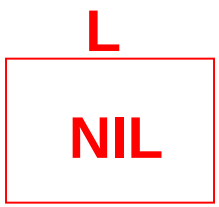
DUNQUE IN GENERALE

pos (i)



PER NUMERO DI ELEMENTI DIVERSO DA ZERO (n > 1)

MENTRE PER n = 0 OVVERO QUANDO LA LISTA E' VUOTA



IN MOLTI LINGUAGGI DI PROGRAMMAZIONE NIL E' UN PARTICOLARE VALORE CHE SI PUO' ASSEGNARE AD UN PUNTATORE PER INDICARE CHE NON PUNTA A NIENTE

L'implementazione di una struttura dati dinamica come la lista generalmente richiede che il linguaggio abbia delle funzionalità di allocazione dinamica della memoria (come la funzione malloc del linguaggio C o il pointer del Pascal).
In assenza di queste funzionalità, una struttura dati dinamica può essere implementata “simulando” l'allocazione dinamica con una variabile dimensionata per il numero massimo di elementi, ma in questo caso si perdono alcuni dei benefici di una struttura dati dinamica.

D'altra parte è importante osservare che non è sufficiente che una struttura dati sia allocata dinamicamente perché sia una struttura dati dinamica: è necessario che la struttura dati consenta di modificare il numero di elementi contenuti durante l'esecuzione del programma.

L’uso scorretto di puntatori può portare a malfunzionamenti le cui cause sono molto difficili da individuare.

Una variabile di tipo puntatore non inizializzata, ovvero alla quale cioè non è mai stato assegnato l'indirizzo di alcun oggetto può comportare che la variabile contenga un valore "casuale" oppure il valore null.

La dereferenziazione di un puntatore "non valido" spesso genera un errore di sistema o un'eccezione. Nel caso peggiore in cui il puntatore contiene un valore "casuale" essa potrebbe portare a una violazione grave della coerenza interna della memoria del programma, con risultati non prevedibili.

Alcuni linguaggi tentano di limitare l'uso dei puntatori o addirittura di eliminarli completamente (un esempio è Java); all'eliminazione dei puntatori deve corrispondere in genere l'introduzione di altri meccanismi che consentano di ottenere risultati analoghi a quelli per i quali si usano solitamente i puntatori (le limitazioni imposte da Java all'uso dei puntatori, per esempio, sono controbilanciate dal suo meccanismo di garbage collection).

IN PASCAL

IL TIPO PUNTATORE E' UN TIPO DI DATO I CUI VALORI RAPPRESENTANO INDIRIZZI DI LOCAZIONI DI MEMORIA. LE OPERAZIONI USUALMENTE DISPONIBILI SU UNA VARIABILE P SONO:

- *L'ACCESSO ALLA LOCAZIONE IL CUI INDIRIZZO E' MEMORIZZATO IN P*
- *LA RICHIESTA DI UNA NUOVA LOCAZIONE DI MEMORIA E LA MEMORIZZAZIONE DELL'INDIRIZZO IN P (new)*
- *IL RILASCIO DELLA LOCAZIONE DI MEMORIA IL CUI INDIRIZZO E' MEMORIZZATO IN P (dispose)*

Esempi:

punta–intero: alias di tipo puntatore a intero

x: variabile di tipo intero

y: variabile di tipo *punta–intero*

x INDICA UN INTERO

y INDICA IL RIFERIMENTO AD UNA LOCAZIONE CHE CONTIENE UN INTERO (SE IN ESECUZIONE y = 1031, NELLA CELLA CON QUESTO INDIRIZZO VI SARA' UN INTERO; SE L'INTERO CONTENUTO E' 5 PER ACCEDERVI INDICHERO' puntatore(y) CHE DARA' 5)

IN C L'OPERATORE:

& INDICA L'INDIRIZZO DI (data una cosa, puntala)

*** INDICA IL DIFFERIMENTO (dato un puntatore, preleva la cosa puntata)**

int cosa; //definisce "cosa"

Int *ptr_cosa; //definisce "puntatore a cosa"

Cosa=4

Ptr_cosa= &cosa; //punta alla cosa

***ptr_cosa=5; pone "cosa" uguale 5**

free corrisponde a DISPOSE

malloc corrisponde a NEW

UNA POSSIBILE DICHIARAZIONE DI TIPO GENERICA

posizione: tipo puntatore a cella
cella: tipo strutturato con componenti
- elemento di tipo tipoelem
- successivo di tipo posizione

lista: alias per il tipo posizione

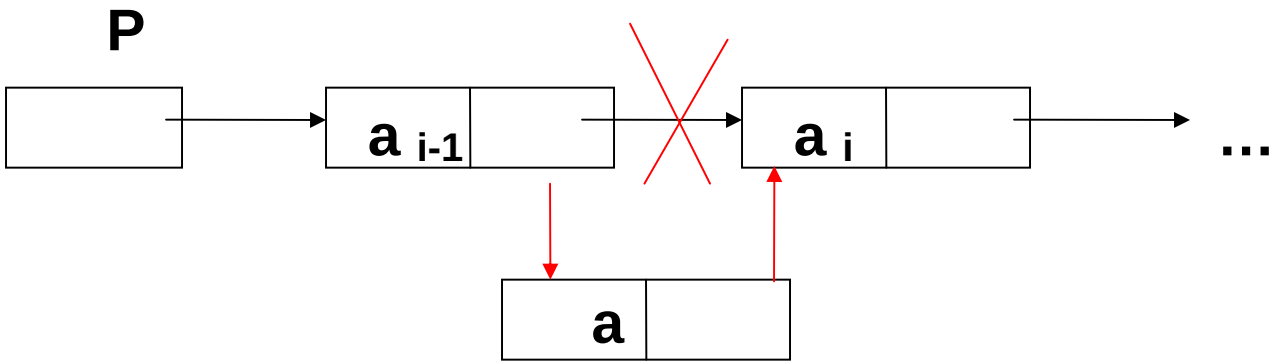
UNA POSSIBILE DICHIARAZIONE DI TIPO LISTA IN PASCAL

```
TYPE
   cella = RECORD
      elemento : tipoelem;
      successivo : ^cella;
      END;
   lista = ^cella;
   posizione = ^cella;
```

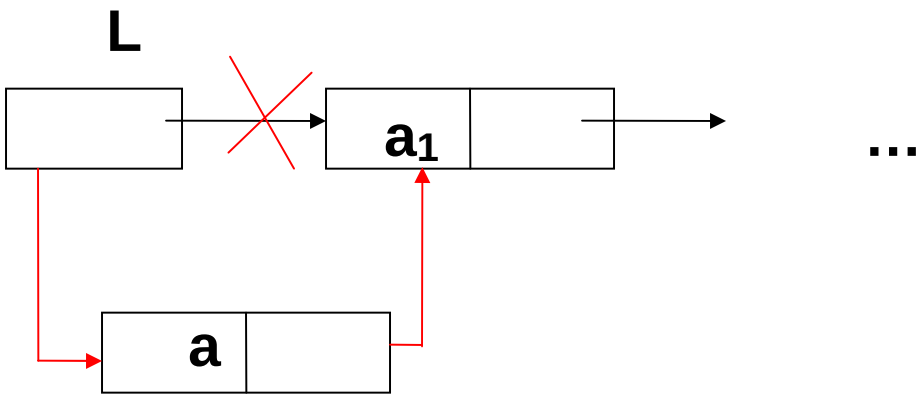
UNA POSSIBILE REALIZZAZIONE DI LISTA COLLEGATA CON PUNTATORE IN C:

```
/* richiede la definizione preliminare del tipo per gli elementi della lista

typedef…TipoElemLista;

*/

struct cella{

   TipoElemLista info;  /*elemento della lista*/

   struct cella *succ; /*puntatore al successivo*/

};

typedef struct cella TipoNodoLista;

typedef TipoNodoLista *TipoLista;
```

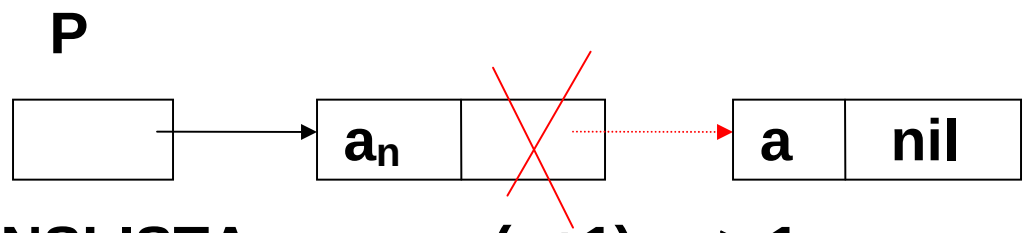
L'AGGIORNAMENTO DEI PUNTATORI



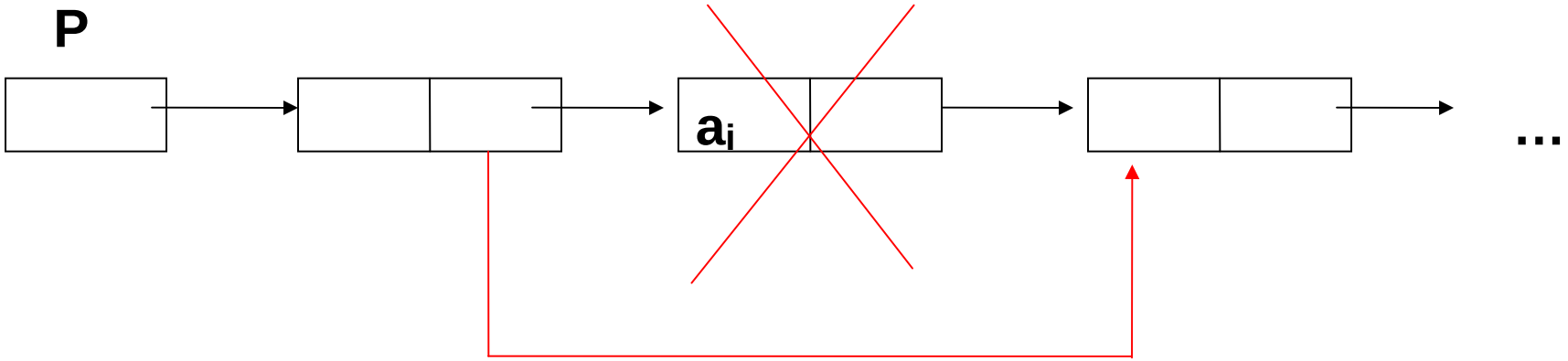
INSLISTA p = pos (i) i ≥ 2 (a)



INSLISTA p = pos (1), n ≥ 1 (b)



INSLISTA p = pos (n+1), n ≥ 1 **(c)**



CANCLISTA p = pos (i), i ≥ 2 **(d)**

UNA POSSIBILE REALIZZAZIONE DELL'OPERATORE INSLISTA (LISTA CON DOPPIO PUNTATORE) IN C :

```
typedef…int Tipoelem;
struct cella{
    Tipoelem info;
    struct cella *precedente, *succ, ;
    };
typedef struct cella *lista, *posizione;
Void Inslista (Tipoelem,posizione*,lista);
. . . . . .
Void Inslista(Tipoelem a, posizione*p, lista L)
    {
Struct cella *temp;
    temp = (struct cella*)malloc(sizeof(struct cella));
    temp→precedente=(*p)→precedente;
    temp→successivo=*p;
    temp→precedente→successivo=temp;
    (*p)→precedente=temp;
    temp→info=a;
    (*p)=temp;
}
```

ESEMPI DI PROBLEMI:

FUSIONE DI LISTE ORDINATE

DATE DUE LISTE ORDINATE, PRODURRE UNA TERZA LISTA OTTENUTA DALLA FUSIONE DELLE PRIME DUE E CHE RISPETTA LO STESSO ORDINAMENTO

L'ACCESSO ALL'ULTIMO ELEMENTO PUO' SOLO AVVENIRE MEDIANTE SCANSIONE SEQUENZIALE. QUESTO VINCOLO RENDE IL NOTO ALGORITMO REALIZZATO PER DUE VETTORI ORDINATI INAPPLICABILE ALLE LISTE.

NON SERVE IL CONFRONTO DEGLI ULTIMI DUE ELEMENTI PER STABILIRE SUBITO QUALE SEQUENZA VENGA ESAURITA PER PRIMA, E DI CONSEGUENZA PERDE SENSO ANCHE IL CONFRONTO FRA L'ULTIMO ELEMENTO DELLA PRIMA SEQUENZA AD ESAURIRSI E IL PRIMO ELEMENTO DELL'ALTRA.

FUSIONE DI LISTE ORDINATE

L'ALGORITMO DI FUSIONE FRA LISTE E' SOSTANZIALMENTE FONDATO SUL SEGUENTE CICLO:

MENTRE NON E' FINITA LISTA1 E NON E' FINITA LISTA2

CONFRONTA I DUE ELEMENTI CORRENTI DI LISTA1 E LISTA2

MEMORIZZA L'ELEMENTO MINORE IN LISTA3 ED AGGIORNA L'ELEMENTO CORRENTE DELLA LISTA INTERESSATA

DOPO DI CHE

SE LISTA1 E' FINITA COPIA IL RESTO DI LISTA2 IN LISTA3

SE LISTA2 E' FINITA COPIA IL RESTO DI LISTA1 IN LISTA3

Pseudocodice

```
fusione(Lista1, Lista2, Lista3  di tipo Lista)
    crealista(Lista3)
    p1 ← primolista(Lista1)
    p2 ← primolista(Lista2)
    p3 ← primolista(Lista3)
    while (not finelista(p1, Lista1) and not finelista (p2, Lista2)) do
        elem1 ← leggilista(p1, Lista1)
        elem2 ← leggilista(p2, Lista2)
        if elem1 < elem2 then
            inslista(elem1, p3, Lista3)
            p1 ← succlista (p1, Lista1)
        else
            inslista(elem2, p3, Lista3)
            p2 ← succlista (p2, Lista2)
        p3 ← succlista (p3, Lista3)
    while not finelista (p1, Lista1) do
        inslista (leggilista(p1, Lista1), p3, Lista3)
        p1 ← succlista (p1, Lista1)
        p3 ← succlista (p3, Lista3)
    while not finelista (p2, Lista2) do
        inslista (leggilista(p2, Lista2), p3, Lista3)
        p2 ← succlista (p2, Lista2)
        p3 ← succlista (p3, Lista3)
```

PROBLEMA: ORDINAMENTO DI UNA LISTA

DATA UNA LISTA, ORDINARE GLI ELEMENTI IN ORDINE CRESCENTE RISPETTO AD UNA RELAZIONE D'ORDINE $\leq$.

PER ORDINARE UNA LISTA NON SI POSSONO UTILIZZARE GLI ALGORITMI PROPOSTI PER GLI ARRAY IN QUANTO BASATI TUTTI SULL'ACCESSO DIRETTO AGLI ELEMENTI.

IL METODO NOTO COME MERGE SORT, DI SEGUITO DESCRITTO PER ARRAY, PREVEDE LA DIVISIONE IN PARTIZIONI DELLA STRUTTURA E LA SUCCESSIVA RICOMPOSIZIONE ATTRAVERSO FUSIONE DELLE PARTIZIONI PER CHIAMATE RICORSIVE.

QUESTO CONSENTE DI RICAVARE L'ORDINAMENTO ATTRAVERSO LA FUSIONE, MA RICHIEDE LA POSSIBILITA' DI DEFINIRE E INDIRIZZARSI A PARTIZIONI.

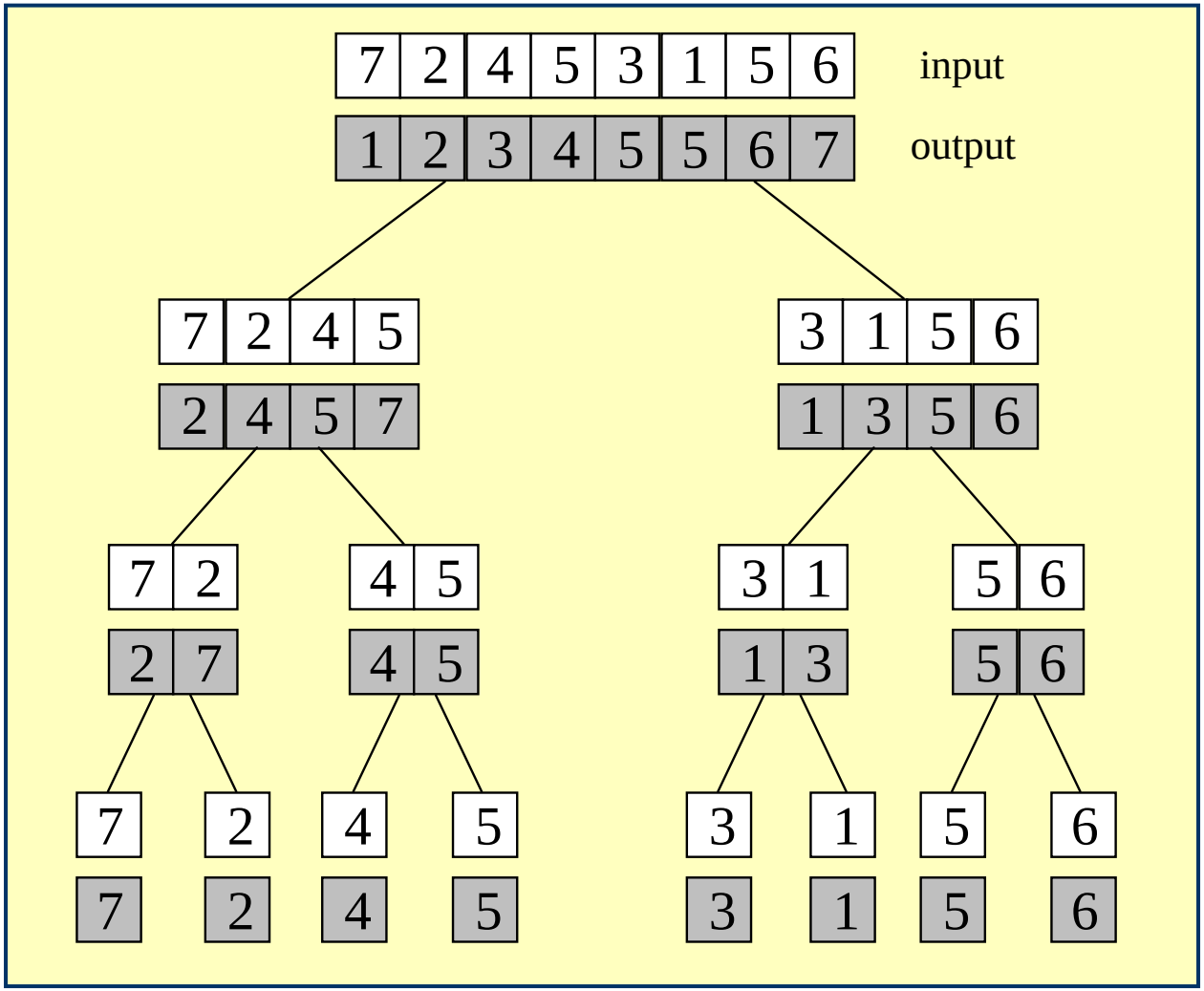
GENERALMENTE PER ORDINARE SEQUENZE SI USA. L'ORDINAMENTO NATURALE (O NATURAL MERGE SORT),

MergeSort

Usa la tecnica del divide et impera:

1. Divide: dividi l'array a metà o preservando un criterio di ordinamento parziale
2. Risolvi il sottoproblema ricorsivamente
3. Impera: fondi le due sottosequenze (parzialmente) ordinate

Esempio di esecuzione



Input ed output delle chiamate Ricorsive con divisione in partizioni successive

NATURAL MERGE SORT

INTRODUCIAMO UN CRITERIO DI *ORDINAMENTO PARZIALE* PER DIVIDERE E FONDERE.

DATA UNA LISTA $L = a_1\ a_2\ \ldots\ a_n$, DIREMO CHE UNA SOTTOSEQUENZA

$$a_i\ a_{i+1}\ \ldots\ a_k$$

COSTITUISCE UNA CATENA (O RUN) SE ACCADE CHE:

$a_{i-1} > a_i$

$a_j \leq a_{j+1}$ **per ogni j = i, i+1, … , k -1**

$a_k > a_{k+1}$

UNA LISTA E' QUINDI UNA SEQUENZA DI CATENE.

ESEMPIO

L = 82 16 14 15 84 25 77 13 75 4

L E' OTTENUTA DALLA CONCATENAZIONE DI SEI CATENE DI VARIA LUNGHEZZA

LA FUSIONE DI DUE SEQUENZE DI N CATENE PRODUCE UNA SINGOLA SEQUENZA DI N CATENE.

FONDENDO LE CATENE CHE COMPONGONO UNA LISTA SI OTTIENE UNA NUOVA LISTA CON UN NUMERO DIMEZZATO DI CATENE. RIPETENDO QUESTA OPERAZIONE AD OGNI PASSO, DOPO AL PIU' $LOG_2\ N$ PASSI LA LISTA SARA' COMPLETAMENTE ORDINATA.

ESEMPIO

L = 82 16 14 15 84 25 77 13 72 4

1° PASSO: FONDIAMO CATENE CONSECUTIVE:

L = 16 82 14 15 25 77 84 4 13 72

2° PASSO:

L = 14 15 16 25 77 82 84 4 13 72

3° PASSO:

L = 4 13 14 15 16 25 72 77 82 84

MA COME DETERMINARE LE SEQUENZE DI CATENE DA FONDERE?

VI E' UNA FASE DI DISTRIBUZIONE IN CUI LE CATENE DI L SONO DISTRIBUITE ALTERNATIVAMENTE A DUE LISTE AUSILIARIE A E B. AL TERMINE DI QUESTA FASE A E B CONTERRANNO OGNUNA N/2 CATENE ORIGINARIE DI L.

ESEMPIO

A : 82 14 15 84 13 72

B : 16 25 77 4

SI NOTI CHE IL NUMERO DI CATENE EFFETTIVE IN A E B PUO' ANCHE ESSERE MINORE DI N/2, POICHE' DOPO LA DISTRIBUZIONE DUE O PIU' CATENE ORIGINARIE IN L FORMANO UN'UNICA CATENA IN A (O B).

AD ESEMPIO, B HA EFFETTIVAMENTE DUE CATENE:

B : 16 25 77 4

FONDENDO LE CATENE DISTRIBUITE SI PUO' CREARE UNA NUOVA LISTA L IL CUI *GRADO DI ORDINAMENTO* E' MAGGIORE. L'ALGORITMO DI ORDINAMENTO ALTERNA FASI DI DISTRIBUZIONE A FASI DI FUSIONE, FINO A QUANDO SI OTTIENE UNA LISTA CON UN'UNICA CATENA.

```
Ordinamento-Naturale ( L di tipo lista)
    repeat
        crealista(A)
        crealista(B)
        distribuisci(L, A, B)
        numero_catene ← 0
        crealista(L)
        merge(A, B, L, numero_catene)
    until numero_catene = 1
```

LA FASE DI DISTRIBUZIONE SI BASA SUL RICHIAMO DELLA PROCEDURA copiaCatena.

```
distribuisci (L, A e B di tipo lista);
    pl ← primolista(L)
    pa ← primolista (A)
    pb ← primolista (B)
    repeat
        copiaCatena(pl, L, pa, A)
        if not finelista(pl, L) then
            copiaCatena(pl, L, pb, B)
    until finelista(pl, L)
```

MENTRE LA FASE DI FUSIONE SI BASA SUL RICHIAMO DELLA PROCEDURA fondiCatena. QUESTA PRODUCE UNA SINGOLA CATENA FUSA IN L A PARTIRE DA UNA CATENA DELLA LISTA A ED UNA DELLA LISTA B. E' IN QUESTA PROCEDURA CHE SI SFRUTTA LA RELAZIONE D'ORDINE SUGLI ELEMENTI (VEDI CONFRONTO <)

```
merge(A, B, L di tipo lista; numero_catene di tipo integer)

    pa ← primolista(A)
    pb ← primolista(B)
    pl ← primolista(L)
    while not finelista(pa,A) and not finelista(pb,B) do
        fondiCatena(pa, A, pb, B, pl, L)
        numero_catene ← numero_catene + 1
    while not finelista(pa,A) do
        copiaCatena(pa, A, pl, L);
        numero_catene ← numero_catene + 1
    while not finelista(pb,B) do
        copiaCatena(pb, B, pl, L)
        numero_catene ← numero_catene + 1
```

```
fondiCatena(pa di tipo posizione A, pb di tipo posizione, B
      ditipo lista, pl di tipo posizione, L di tipo lista)

  repeat
    if leggilista(pa, A) < leggilista(pb, B) then
      copia(pa, A, pl, L, finecatena)
      if finecatena then
        copiaCatena (pb, B, pl, L)
      else
        copia(pb, B, pl, L, finecatena)
        if finecatena then
          copiacatena(pa, A, pl, L)
  until finecatena
```

```
copiaCatena (pa di tipo posizione;
             A di tipo lista;
             pb di tipo posizione;
             B di tipo lista );
  repeat
    copia (pa, A, pb, B, finecatena)
  until finecatena
```

INFINE LA PROCEDURA COPIA SARA' DATA DA:

```
copia (pa di tipo posizione;
       A di tipo lista;
       pl di tipo posizione;
       L di tipolista; finecatena di tipo boolean);


  elemento ← leggilista(pa, A)
  inslista(elemento, pl, L)
  pa ← succlista(pa, A)
  pl ← succlista(pl, L)
  if finelista(pa, A) then
    finecatena ← true
  else
    finecatena ← (elemento > leggilista (pa, A))
```