

Algoritmi e Strutture di Dati

Esercitazioni in linguaggio C

Strutture e liste

m.patrignani

c030-strutture-e-liste-01 copyright ©2017 patrignani@dia.uniroma3.it

Nota di copyright

- queste slides sono protette dalle leggi sul copyright
- il titolo ed il copyright relativi alle slides (inclusi, ma non limitatamente, immagini, foto, animazioni, video, audio, musica e testo) sono di proprietà degli autori indicati sulla prima pagina
- le slides possono essere riprodotte ed utilizzate liberamente, non a fini di lucro, da università e scuole pubbliche e da istituti pubblici di ricerca
- ogni altro uso o riproduzione è vietata, se non esplicitamente autorizzata per iscritto, a priori, da parte degli autori
- gli autori non si assumono nessuna responsabilità per il contenuto delle slides, che sono comunque soggette a cambiamento
- questa nota di copyright non deve essere mai rimossa e deve essere riportata anche in casi di uso parziale

c030-strutture-e-liste-01 copyright ©2017 patrignani@dia.uniroma3.it

Definizione di una struttura

- Definizione e uso standard di una struttura

```
struct miastruct {  
    int minimo;  
    int massimo;  
};  
  
struct miastruct a, b, c;
```

- Il nome della struttura è `miastruct`
- “`struct miastruct`” può essere usato come un tipo per definire le variabili `a`, `b`, `c`

c030-strutture-e-liste-01 copyright ©2017 patrignani@dia.uniroma3.it

Accesso ai campi di una struttura

- Definizione di `struct miastruct`

```
struct miastruct {  
    int minimo;  
    int massimo;  
};
```

- Si accede ai campi di una struttura con il costrutto `.` (punto)

```
struct miastruct a;  
a.minimo = 3;
```

c030-strutture-e-liste-01 copyright ©2017 patrignani@dia.uniroma3.it

Accesso ai campi di una struttura

- Definizione di struct miastruct

```
struct miastruct {
    int minimo;
    int massimo;
};
```

- Disponendo di un puntatore alla struttura si accede ai campi con il costrutto -> (freccia)

```
struct miastruct a;
struct miastruct* b = &a;
b->minimo = 3;
```

- b->minimo equivale a (*b).minimo

c030-strutture-e-liste-01 copyright ©2017 patrignani@dia.uniroma3.it

Definizioni alternative di struttura

- Possiamo dichiarare contestualmente le variabili

```
struct miastruct {
    int minimo;
    int massimo;
} a, b, c;
```

- Possiamo rinunciare al nome della struttura
 - ma così non potremo più usarla nel resto del codice

```
struct {
    int minimo;
    int massimo;
} a, b, c;
```

c030-strutture-e-liste-01 copyright ©2017 patrignani@dia.uniroma3.it

Definizioni alternative di struttura

- Possiamo definire un tipo mediante il costrutto `typedef`

```
typedef <definizione> <nometipo>;
```

```
typedef struct miastruct {
    int minimo;
    int massimo;
} miastruttura;
```

- Questo ci consente di sostituire il tipo “`struct miastruct`” con il tipo “`miastruttura`”

```
miastruttura a,b,c;
```

c030-strutture-e-liste-01 copyright ©2017 patrignani@dia.uniroma3.it

Definizioni ricorsive di struttura

- Alcune definizioni di struttura contengono riferimenti a se stesse

```
typedef struct elem {
    int info;
    struct elem* next;
} elemento;
```

- Nelle definizioni ricorsive si utilizza esclusivamente il tipo “**`struct <nome>`**”
 - non si può utilizzare il tipo definito dal `typedef`

c030-strutture-e-liste-01 copyright ©2017 patrignani@dia.uniroma3.it

Definizione di una lista

- Definizione degli elementi della lista

```
typedef struct elem {  
    int info;  
    struct elem* next;  
    struct elem* prev;  
} elemento;
```

- Definizione di lista

```
typedef struct {  
    elemento* head; // primo elemento  
    elemento* last; // ultimo elemento  
} lista;
```

c030-strutture-e-liste-01 copyright ©2017 patrignani@dia.uniroma3.it

Definizione semplificata

- Quando la lista ha il solo campo head si può semplificare ulteriormente la sua definizione

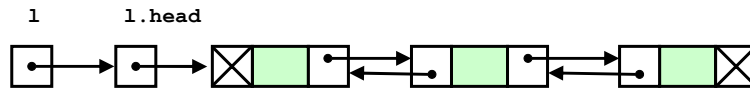
```
typedef struct elem {  
    int info;  
    struct elem* next;  
    struct elem* prev;  
} elemento;  
  
typedef elemento* lista;
```

- Così la lista coincide con il valore che avrebbe il suo puntatore head

c030-strutture-e-liste-01 copyright ©2017 patrignani@dia.uniroma3.it

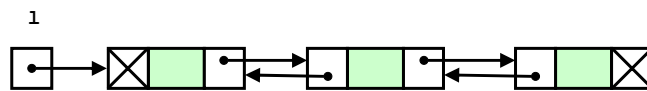
Confronto con lo pseudocodice

- Lista in pseudocodice



- Lista (semplificata) in linguaggio C

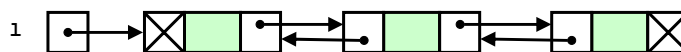
– possibile esclusivamente se l'unico campo della lista sarebbe il campo `head`



c030-strutture-e-liste-01 copyright ©2017 patrignani@dia.uniroma3.it

Riflessione sullo pseudocodice

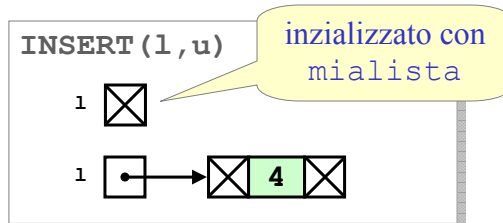
- La definizione di lista in pseudocodice non può essere semplificata
- Infatti, supponiamo di adottare in pseudocodice



- Lanciamo `INSERT(mialista, 4)` con una lista vuota

`mialista`

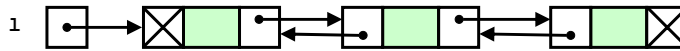
rimane invariato
dopo l'INSERT



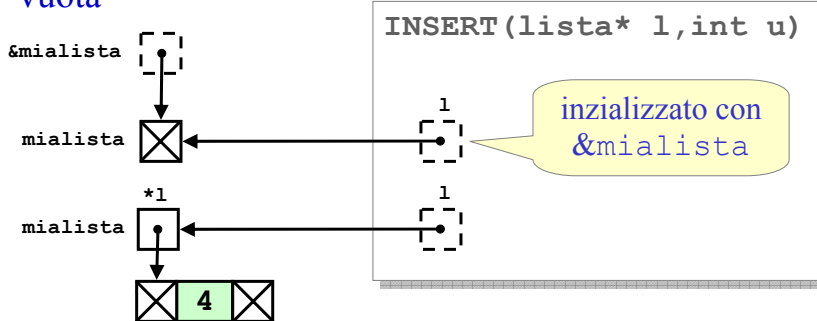
c030-strutture-e-liste-01 copyright ©2017 patrignani@dia.uniroma3.it

Uso della definizione semplificata in C

- Nel linguaggio C il problema può essere scavalcato tramite il costrutto &



- Lanciamo `INSERT(&mialista, 4)` con una lista vuota



c030-strutture-e-liste-01

copyright ©2017 patrignani@dia.uniroma3.it

Esercizio

- Scrivi il codice in linguaggio C che definisce il tipo `lista` doppiamente concatenata di interi realizzata con oggetti e puntatori
 - non è necessario afferire in tempo costante all'ultimo elemento della lista
- Definisci anche le funzioni:
 - `void INSERT(lista* l, int i)`
 - `void DELETE(lista* l, int i)`

c030-strutture-e-liste-01

copyright ©2017 patrignani@dia.uniroma3.it