Algoritmi e Strutture di Dati

Esercitazioni in linguaggio C

Strutture e liste

m.patrignani

Nota di copyright

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`

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;
```


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`

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;
```


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;
```


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`

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;
```


Definizione semplificata

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

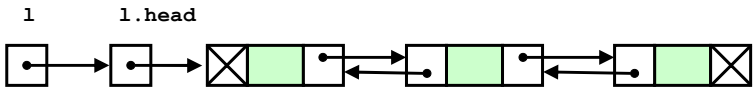
```
typedef struct elem {
     int info;
     struct elem* next;
     struct elem* prev;
} elemento;

typedef elemento* lista;
```

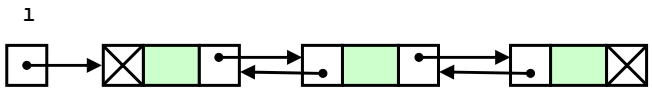
- Così la lista coincide con il valore che avrebbe il suo puntatore `head`

Confronto con lo speudocodice

- Lista in pseudocodice

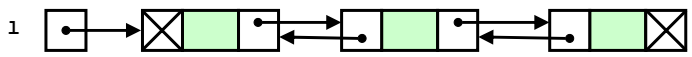


- Lista (semplificata) in linguaggio C
  - possibile esclusivamente se l'unico campo della lista sarebbe il campo `head`

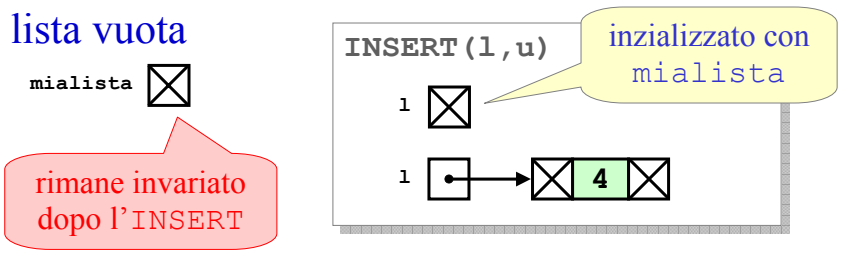


Riflessione sullo pseudocodice

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

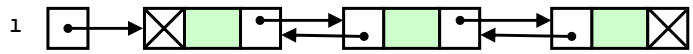


- Lanciamo `INSERT(mialista, 4)` con una lista vuota

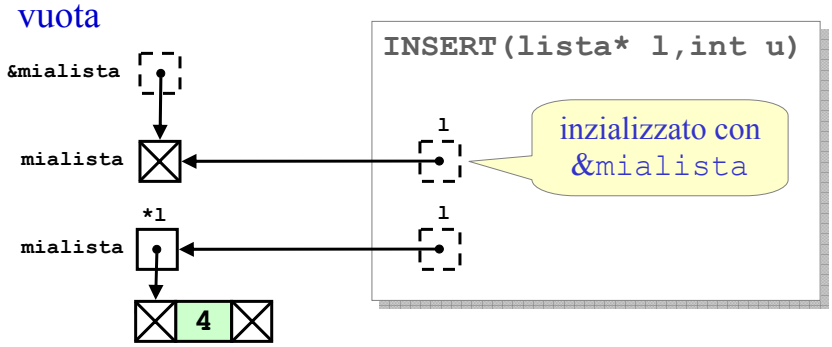


Uso della definizione semplificata in C

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



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



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)`

