

# Informatica 1

Corso di Laurea Triennale in Matematica

Gianluca Rossi

`gianluca.rossi@uniroma2.it`

Dipartimento di Matematica  
Università di Roma "Tor Vergata"

## 10: Strutture e liste - Parte prima

- Permettono di raggruppare in un unico “contenitore” più variabili (*campi*) disomogenee;
- Utili per raccogliere in un unico “contenitore” le grandezze che descrivono un'unica entità;
- Il nuovo “contenitore” può essere trattato come un nuovo tipo composto;
- Permettono di organizzare il codice in maniera più razionale.

## struct

Si definisce elencando i campi che contiene ed i relativi tipi.

Definizione

```
struct nome1{  
    tipo1 campo1;  
    tipo2 campo2, campo3;  
};
```

Dichiarazione

```
struct nome nomevariabile2;
```

Accesso ai campi della **struct**: operatore .

```
nomevariabile1.campo2
```

Definizione + dichiarazione

```
struct nome2{  
    tipo1 campo1;  
    tipo2 campo2;  
} nomevariabile1;
```

# Esempio: Area del rettangolo

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

```
#include <math.h>
```

```
struct point {  
    double x, y;  
};
```

```
struct rectangle{  
    struct point p1;  
    struct point p2;  
};
```

```
main(){  
    struct rectangle R;  
    double area;  
    scanf( "%lf, %lf", &(R.p1.x), &(R.p1.y));  
    scanf( "%lf, %lf", &(R.p2.x), &(R.p2.y));  
    area = fabs(R.p1.x - R.p2.x)*fabs(R.p1.y - R.p2.y);  
    printf("area = %lf\n", area);  
}
```

# Esempio: Quadrato in Rettangolo

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

```
#include <math.h>
```

```
struct point {  
    double x, y;  
};
```

```
struct rectangle {  
    struct point p1;  
    struct point p2;  
};
```

```
struct square {  
    struct point p;  
    double lenght;  
};
```

```
main(){  
    struct rectangle R;  
    struct square Q;  
    scanf("%lf, %lf", &(Q.p.x), &(Q.p.y));  
    scanf("%lf", &(Q.lenght));  
    R.p1.x = Q.p.x;  
    R.p1.y = Q.p.y;  
    R.p2.x = Q.p.x + Q.lenght;  
    R.p2.y = Q.p.y + Q.lenght;  
    printf("( %lf, %lf), ( %lf, %lf) \n", \  
        R.p1.x, R.p1.y, R.p2.x, R.p2.y);  
}
```

Limiti dei vettori nella rappresentazione di collezioni di oggetti:

- La dimensione deve essere fissata al momento della creazione.
- La riallocazione di memoria potrebbe comportare un dispendio di risorse.
- La memoria allocata deve essere contigua.
- Inserire un nuovo elemento potrebbe comportare la traslazione degli elementi successivi.

## Liste: Definizione

Una lista  $L$  di elementi in  $X$  è l'insieme vuoto o un elemento di  $X$  seguito da una lista  $L'$ .

$$L = \begin{cases} \emptyset \\ e \circ L' \end{cases} \quad \text{dove } e \in X \text{ ed } L' \text{ è una lista}$$

## Liste: Operatori

$$\mathbf{size}(L) = \begin{cases} 0 & \text{se } L = \emptyset \\ 1 + \mathbf{size}(L') & \text{Se } L = e \circ L' \end{cases}$$

$$\mathbf{next}(L) = \begin{cases} \emptyset & \text{se } L = \emptyset \\ L' & \text{se } L = e \circ L' \end{cases}$$

$$\mathbf{getelem}(L) = \begin{cases} e & \text{se } L = e \circ L' \\ \emptyset & \text{altrimenti} \end{cases}$$



## Liste: Operatori (2)

$$i \in \mathbb{N}, \quad \text{get}(L, i) = \begin{cases} L & \text{se } i = 0 \\ \text{get}(L', i - 1) & \text{se } L = e \circ L' \\ \emptyset & \text{altrimenti} \end{cases}$$

$$i \in \mathbb{N} \quad e' \in X$$

$$\text{insert}(L, e', i) = \begin{cases} e \circ \text{insert}(L', e', i - 1) & \text{se } i > 0 \text{ e } L = e \circ L' \\ e' \circ L & \text{se } i = 0 \\ \text{errore} & \text{altrimenti} \end{cases}$$

$$\begin{aligned}\text{insert}(\langle a, b, c, d, e \rangle, x, 2) &= \langle a \rangle \circ \text{insert}(\langle b, c, d, e \rangle, x, 1) \\ &= \langle a, b \rangle \circ \text{insert}(\langle c, d, e \rangle, x, 0) \\ &= \langle a, b \rangle \circ \langle x \rangle \circ \langle c, d, e \rangle \\ &= \langle a, b, x, c, d, e \rangle\end{aligned}$$

## Esercizio

Definire l'operatore **delete** che, con input la lista  $L$  ed un intero  $i$  restituisce la lista ottenuta eliminando da  $L$  l'elemento in posizione  $i$ .

# Liste: Implementazione

- Si deve conoscere la posizione del primo elemento della lista.
- Per ogni elemento della lista, deve essere nota la posizione del prossimo elemento.
- Ogni elemento della lista può essere definito come un contenitore contenente due entità: l'informazione vera e propria ed il riferimento all'elemento successivo.
- I due dati sono disomogenei tuttavia possono essere raccolti in una **struct**.

# Liste: Implementazione

```
struct listelem{  
    tipo info;  
    struct listelem *next;  
};
```

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

```
struct listelem{  
    char info;  
    struct listelem *next;  
};
```

```
main(){  
    struct listelem primo, secondo, terzo;  
    primo.info = 'a'; primo.next = &secondo;  
    secondo.info = 'b'; secondo.next = &terzo;  
    terzo.info = 'c'; terzo.next = NULL;  
    printf("%c\n", \ /* ' ha precedenza su '*' */  
        (*primo.next).info);  
}
```

...	...
b	9A0
D63	9A1
	9A2

...	...
a	B1C
9A0	B1D
	B1E

...	...
c	D63
NULL	D64
	D65

...

...