

INFORMATICA GENERALE II

Ingegneria delle Telecomunicazioni

Università di Trento

A.A. 2003/2004

II Bimestre

Marco Roveri

roveri@irst.itc.it

Strutture e tipi elementari

Il problema

- Supponiamo di dover gestire una rubrica telefonica. Ogni elemento della rubrica è caratterizzato da un *nome*, l'anno di nascita della persona, un *numero di telefono*.
- Ci troviamo davanti al problema di memorizzare in una variabile I dati di alcune persone.
- Esempio:

```
void stampapersone(char * name, int year, char * phone) {  
    cout << "Nome = " << name << endl;  
    cout << "Anno = " << year << endl;  
    cout << "Phone = " << phone << endl;  
}  
int main () {  
    char * name = "Mario Rossi";  
    int year = 1965;  
    char * phone = "0461314326";  
    stampapersone(name, year, phone);  
}
```

Il problema

- Il programma precedente è corretto, ma...
 - Leggibilità:
 - Non risulta subito chiaro che le tre variabili sono attributi dello stesso “oggetto”.
 - Modificabilità:
 - Se vogliamo aggiungere la data di nascita, bisogna aggiungere una variabile, e poi modificare tutte le chiamate di funzione.
- Il primo problema può causare errori se per esempio si chiama la funzione stampapersona commettendo l'errore di passare il nome di una persona, il cognome di un'altra e l'età di una altra persona ancora.
- Il secondo problema può rendere difficile modificare il codice, e questo può a sua volta portare a errori se il codice non viene modificato nel modo giusto.

La soluzione

- Definire una variabile di tipo persona, che contiene al suo interno tutti i dati di una persona, ossia nome, anno di nascita, e numero di telefono.
- Non possiamo usare un array, perchè ci troviamo a manipolare dati eterogenei (stringhe di caratteri, interi, ...).
- Il C++ mette a disposizione dei costrutti per definire una variabile come un gruppo di variabili anche di tipo diverso. Ovvero **class** e **struct**. Ci soffermeremo sul costrutto **struct**. Il concetto di classe verrà introdotto nel corso successivo.

Strutture

- Una struttura è una n -upla ordinata di elementi, detti *membri* o *campi*, ciascuno dei quali ha uno specifico *tipo*, *nome* e *valore*
- Sintassi (definizione di un nuovo tipo):

```
struct new_struct_id {  
    tipo1 campo1;  
    ...  
    tipo $n$  campo $n$ ; } ;
```

Convenzioni

- In C++, ma anche in altri linguaggi come Java, gli oggetti (strutture e classi) hanno nomi capitalized (ovvero la prima lettera dell'identificativo maiuscola).

Definizione di una variabile di tipo **struct**

Può essere definita nei seguenti modi:

- *New_struct_id var_id;*
- **struct** *New_struct_id* {
 tipo₁ campo₁;
 ...
 tipo_n campo_n; } *var_id;*
- **struct** {
 tipo₁ campo₁;
 ...
 tipo_n campo_n; } *var_id;*
 - quest'ultima è detta struttura anonima

Esempio

```
// Dichiarazione di una struttura per rappresentare i dati
// di una persona
struct persona {
    char * name;
    int    year;
    char * phone;
};
```

```
// Dichiarazione di una struttura per rappresentare la data
struct data {
    int giorno;
    int mese;
    int anno;
};
```


Inizializzazione di una struttura

- Nella dichiarazione di una struttura possiamo introdurre una funzione che possiamo usare per inizializzare una variabile di tipo struttura.
- La funzione di inizializzazione non ha tipo di ritorno ed è detta “**costruttore**”.
- Esempio:

```
// Struttura per rappresentare la data
struct data {
    int giorno;
    int mese;
    int anno;
    data(int g, int m, int a) {
        giorno = g; mese = m; anno = a;
    }
};
```

```
data D = data(10,11,2003); //inizializza i campi di D
```

Strutture e puntatori

- Una struttura definisce un tipo, è quindi possibile definire dei puntatori a tali strutture.
- Di una variabile di tipo struttura si può determinare sia l'indirizzo della struttura che di ogni suo elemento.
- Similmente si può sapere lo spazio di memoria occupata dalla struttura e dai suoi singoli elementi.

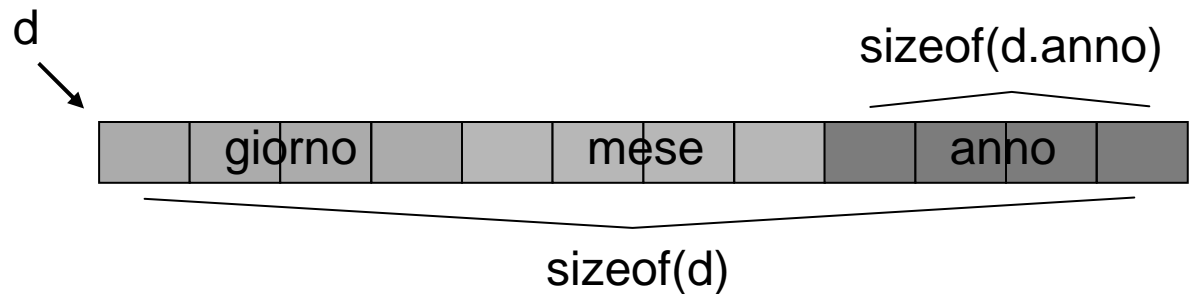
Accesso agli elementi di una struttura

- Se `s` è una struttura e `field` è un identificatore di un campo, allora `s.field` denota il campo della struttura
- Se `ps` è un puntatore ad una struttura avente `field` come identificatore di campo, allora è possibile scrivere `ps->field` al posto di `(*ps).field`
- Esempio:

```
struct complex { double re, im; };  
complex c; complex *pc = &c;  
c1.re = 2.5; pc->im = 3;
```

Strutture e memoria

```
struct data {  
    int giorno;  
    int mese;  
    int anno;  
}
```



```
data d;  
cout << "Indirizzo di d = " &d << endl;  
cout << "Indirizzo di d.giorno = " << &d.giorno << endl;  
cout << "Sizeof(d) = " << sizeof(d) << endl;  
cout << "Sizeof(d.anno) = " << sizeof(d.anno) << endl;
```

Assegnamento e Strutture

- A differenza degli array, l'assegnazione è definita per le variabili di tipo struct
- L'assegnazione di strutture avviene per valore e quindi vengono copiati tutti i valori dei membri
- Esempio:
struct complex { double re, im; };
complex c1, c2;
c1.re = 2.5; c1.im = 3;
c2 = c1; //assegnazione di struct

Assegnamento e Strutture

- Per gestire gli array in modo che possano essere copiati, si può incapsulare un tipo array come membro di una **struct**

- Esempio:

```
struct int_array { int ia[3]; };  
int_array sa, sb;  
sa.ia[0]=1;sa.ia[1]=2;sa.ia[2]=3;  
sb = sa; //l'array viene copiato!  
cout << sb.ia[0] << sb.ia[1]  
      << sb.ia[2] << endl;
```

Assegnamento e Strutture

- Similmente ai costruttori per l'inizializzazione che abbiamo visto prima esiste un altro costruttore detto *costruttore per copia*. Il primo è invocato dall'operatore **new**, il secondo dall'operatore di assegnamento **=**.
- Il costruttore per copia, di default, copia strutture membro a membro.

Array di strutture

- Dato che una struttura definisce un tipo, possiamo anche definire array di strutture.
- Definita una relazione d'ordine tra due strutture, possiamo adattare gli algoritmi di ordinamento visti in precedenza per lavorare su interi a lavorare su strutture generiche.

Array di strutture

```
struct data {  
    int giorno; int mese; int anno;  
}
```

```
int compare(data A, data B) {  
    if (A.year < B.year) return 1;  
    if ((A.year == B.year) && (A.month < B.month)) return 1;  
    if ((A.year == B.year) && (A.month == B.month) && (A.day < B.day)) return 1;  
    return 0;  
}
```

```
void swap(data &A, data &B) {  
    data C = A;  
    A = B;  
    B = C;  
}
```

```
// ordinamento di un insieme di date  
void bubblesort(data A[], int N) {  
    for( int i = 0; i < N - 1; i++)  
        for( int j = N - 1; j > i; j--)  
            if (compare(A[j], A[j-1]))  
                swap(A[j-1], A[j]);  
}
```

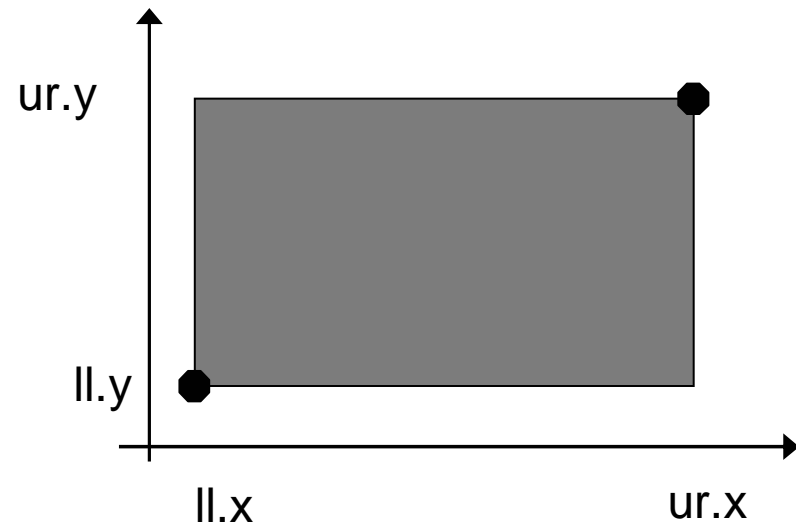
Strutture di Strutture

- È possibile definire strutture di strutture.

```
struct punto {  
    double x;  
    double y;  
}
```

```
struct rettangolo {  
    punto ll;  
    punto ur;  
}
```

```
double area(rettangolo g) {  
    double base = (g.ur.x - g.ll.x);  
    double altezza = (g.ur.y - g.ll.y);  
    return (base*altezza);  
}
```



Strutture ricorsive

- La seguente definizione non è lecita:

```
struct S
{ int value;
  S next; //definizione circolare!
};
```

- La seguente definizione è lecita:

```
struct S
{ int value;
  S *next;
};
```

Infatti ogni puntatore occupa lo stesso spazio di memoria indipendentemente dal suo tipo.

Strutture mutuamente ricorsive

- La seguente definizione non è lecita:

```
struct S1
{ int value;
  S2 *next; //S2 ancora indefinito
};
```

```
struct S2
{ int value;
  S1 *next;
};
```

- S2 non è stato ancora definito al momento della definizione di S1

Strutture mutuamente ricorsive

- Dichiarando prima s2 risulta invece lecita:
`struct s2; // dichiarazione di s2`

```
struct s1
{ int value;
  s2 *next; // Ok!
};
```

```
struct s2 // definizione di s2
{ int value;
  s1 *next;
};
```

Definizione di tipi

- In C++ possiamo definire degli identificativi per riferirsi a tipi definiti dall'utente o primitivi.

- Sintassi:

```
typedef <tipo> <new_type_id>;
```

Definizione di tipi

- Esempi:

```
typedef struct Point_ Point;  
typedef struct Point_ * PointPtr;  
typedef char * stringa;
```

```
Point P = Point(7,0);
```

```
PointPtr Q = new Point(1,9);
```

```
stringa s = "Prova";
```