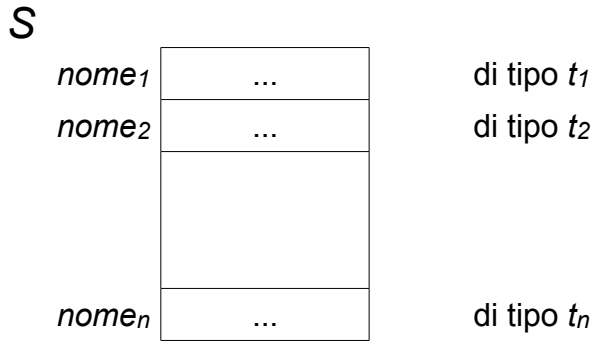
Fondamenti di Programmazione A
Appunti per le lezioni – *Gianfranco Rossi*

Tipi strutturati - *struct*

Struttura dati (concreta) *struct*: sequenza di n elementi ($n \geq 0$), rispettivamente di tipo $t_1,...,t_n$ (non necessariamente distinti), individuati singolarmente da n nomi simbolici $nome_1,...,$ $nome_n$.

Rappresentazione grafica:



dove: *S*: nome della struttura dati
nome$_1$,..., *nome*$_n$: campi della struttura dati

Creazione di una struttura dati *struct* in C++

Dichiarazione di variabile di tipo (definito tramite il costruttore di tipo) *struct*:

```
struct {t1 nome1;
        t2 nome2;
        ...
        tn nomen;} S;
```

dove: `S`: identificatore di var.= nome della struttura dati
`t1`,...,`tn`: tipi qualsiasi (primitivi, def. da utente)
= tipi degli elementi della struttura

`nome1`, ..., `nomen`: identificatori
= nomi degli elementi della struttura

Esempio:

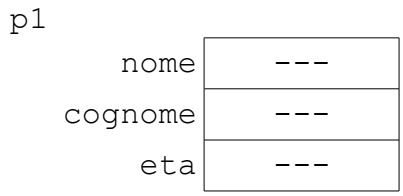
```
struct {char nome[32];
        char cognome[32];
        int eta;} p1;
```

dichiarazione di var. di nome `p1` e tipo `struct {...}`

rappresenta una struttura dati *struct* di nome `p1` costituita da due campi di tipo *array* di caratteri e un campo di tipo intero, rispettivamente di nomi `nome`, `cognome` ed `eta`.

n.b. `p1` = nome della var.; `struct {...}` = tipo della var. .

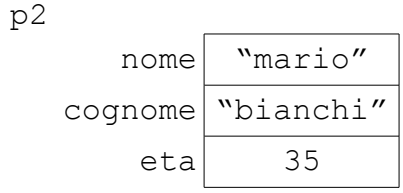
Graficamente:



Possibile anche dichiarazione con inizializzazione. Ad es.:

```
struct {char nome[32];
        char cognome[32];
        int eta;} p2 = {"mario","bianchi",35};
```

Graficamente:



Come fatto finora, la dichiarazione di una variabile di tipo *struct* introduce sia il nuovo tipo, sia la variabile di quel tipo (come già visto per *array*).

Possibile (preferibile) separare i due momenti: prima si definisce il nuovo tipo e gli si associa un nome; poi si dichiarano una o più variabili di quel tipo.

1. Dichiarazione del nuovo tipo con nome (ad es. `NomeTipo`)

```
struct NomeTipo {t1 nome1;
                 t2 nome2;
                    ...
                 tn nomen;};
```

dove: `NomeTipo`: identificatore
= nome del nuovo tipo `struct {...}`
(sinonimo di `struct {...}`)

n.b. Questa dichiarazione non alloca memoria; definisce soltanto un nuovo tipo e gli associa un nome.

2. Dichiarazione di una o più variabili del tipo introdotto al punto 1

```
NomeTipo s1;     //struct di nome s1 e tipo
                 //NomeTipo

NomeTipo s2;     //struct di nome s2 e tipo
                 //NomeTipo
```

n.b. Questa dichiarazione alloca memoria

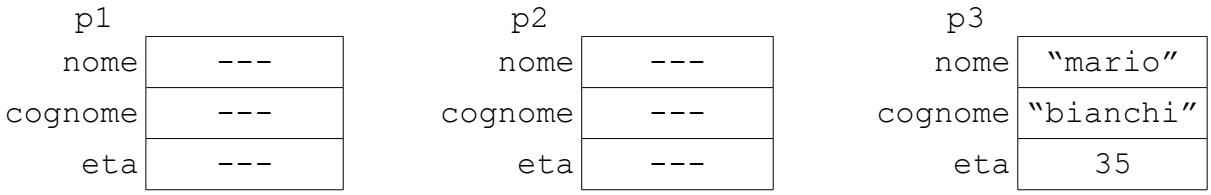
Esempio

```
struct persona { char nome[32];
                 char cognome[32];
                 int eta;};
```

→ nuovo tipo di nome `persona`

```
persona p1;
persona p2;
persona p3 = {"mario","bianchi",35};
```

→ tre variabili di tipo struct `persona`



n.b. `p1`, `p2` e `p3` hanno tutti e tre lo stesso tipo `persona`

Altro esempio – data del giorno.

```
struct data {int g;
             int m;
             int a;};
```

→ nuovo tipo di nome `data` (n.b. tre elementi tutti di tipo `int`)

```
data d1;
data d2 = {10,11,2009};
```

→ due variabili di tipo struct `data`

Dove inserire la dichiarazione di un tipo *struct*?

(Ovviamente) prima del suo uso. All'interno del corpo di una funzione (anche main) se se si vuole che sia visibile soltanto in essa; tra le dichiarazioni globali (più spesso) se si vuole che sia visibile in tutto il programma. Ad es.:

```
#include <iostream>
using namespace std;
struct data {int g;
             int m;
             int a;};
int main() {
    data d1,d2;
    ...
}
```

Operazione di selezione

Dato

```
struct NomeTipo { t1 c1;
                  t2 c2;
                    ...
                  tn cn;};
NomeTipo s;
```

l'espressione:

```
s.ci
```

seleziona il campo `ci` della struttura dati *struct* `s`.

N.B. L'espressione `NomeTipo.ci` non è corretta: la selezione si applica a variabili di tipo *struct*, non a tipi *struct*.

Esempio

```
p1.eta = 21;
p1.eta = p1.eta + 1;
```
(o, in modo equiv.: `p1.eta++`)

Altre operazioni primitive (su intere strutture dati *struct*, non su singoli elementi di *struct*)

- Possibile assegnamento tra *struct* dello stesso tipo

  Esempio

  ```
  p2 = p3;
  ```

  dove `p1` e `p3` sono *struct* di tipo `persona`.

  L'assegnamento copia campo a campo i valori della *struct* di destra nella *struct* di sinistra.

  N.B. Si noti che l'assegnamento avviene anche se alcuni campi delle *struct* coinvolte sono *array* (come nel caso di `persona`) (si ricordi che in generale l'assegnamento tra *array* non e' previsto come operazione primitiva).

- Non è prevista nessun'altra operazione, né primitiva né di libreria standard, su intere *struct*. In particolare non ci sono:
  - operazioni di input/output
  - operazioni di confronto.

Tutte queste operazioni devono essere realizzate a programma operando campo a campo.

Esempio – leggi e stampa dati di una persona

```
/* Leggi da std input i dati di una persona e
stampali su std output con opportuno formato
*/

#include <iostream>

using namespace std;

struct persona {
     char nome[32];
     char cognome[32];
     int eta;
  };

int main() {

    persona p;

    // lettura dati persona
    cout << "Dai il nome: ";
    cin >> p.nome;
    cout << "Dai il cognome: ";
    cin >> p.cognome;
    cout << "Dai l'eta': ";
    do cin >> p.eta; while (p.eta < 0 || p.eta > 150);

    // stampa dei dati letti
    cout << "\nNOME: " << p.nome << endl;
    cout << "COGNOME: " << p.cognome << endl;
    cout << "ETA': " << p.eta << endl;

    system("pause");

    return 0;
}
```

Esempio – data minore

```
/* Leggi due date (G M A) e determina qual'e' la maggiore
delle due; ad es.: 1 8 1975 > 12 10 1972.
*/

#include <iostream>

using namespace std;

struct data {
     int g;
     int m;
     int a;
  };

int main() {

  data d1, d2;

  // lettura date
  cout << "Dai la prima data (g m a): ";
  cin >> d1.g >> d1.m >> d1.a;
  cout << "Dai la seconda data (g m a): ";
  cin >> d2.g >> d2.m >> d2.a;

  // confronto date lette
  bool prima = false;
  if (d1.a > d2.a) prima = true;
  else if (d1.a == d2.a && d1.m > d2.m) prima = true;
  else if (d1.a == d2.a && d1.m == d2.m && d1.g > d2.g) prima = true;

  // stampa risultato
  if (d1.a == d2.a && d1.m == d2.m && d1.g == d2.g)
         cout << "Le due date sono uguali" << endl;
  else
    {cout << "La data " << d1.g << '/' << d1.m << '/' << d1.a;
     if (prima) cout << " e' maggiore " << endl;
          else  cout << " e' minore " << endl;
     cout << "della data " << d2.g << '/' << d2.m << '/' << d2.a
          << endl;
    }

  system("pause");

  return 0;
}
```

Forme più complesse di tipi strutturati costruiti tramite *struct*

1. struct annidati

Possibile utilizzare *struct* all'interno di altre *struct*

Esempio

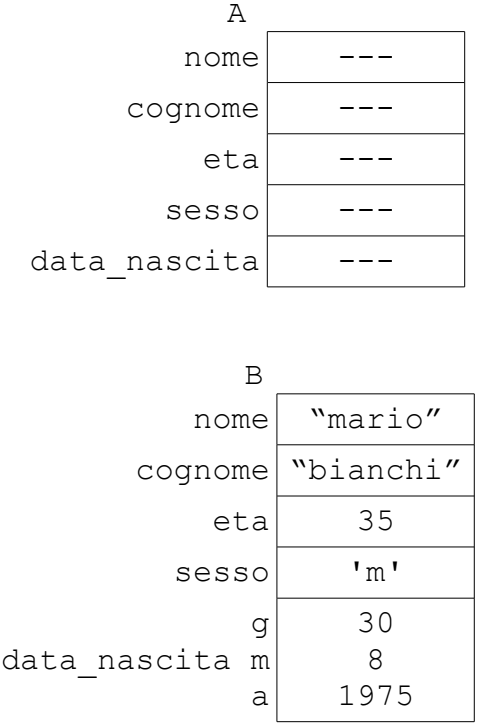
```
struct dati_anagrafici {
                char nome[32];
                char cognome[32];
                int eta;
                char sesso;
                data data_nascita;
               };
```

utilizza il tipo *struct* `data` come tipo di uno dei suoi campi

Due variabili di tipo `dati_anagrafici`

```
dati_anagrafici A;
dati_anagrafici B = {”mario”,”bianchi”,
                     32,'m',{30,8,1975}};
```

Graficamente:



Esempio d'uso:

```
cout << "Anno nascita: " << B.data_nascita.a;
```

n.b. L'operatore '.' è associativo a sinistra

2. struct contenente *array*

Possibile utilizzare *array* all'interno di altre *struct*

Esempio:
campi `nome` e `cognome` in *struct* `persona` e `dati_anagrafici`

Esempio d'uso:

```
cout << "Iniziale cognome: " << p1.cognome[0];
```

3. array di *struct*

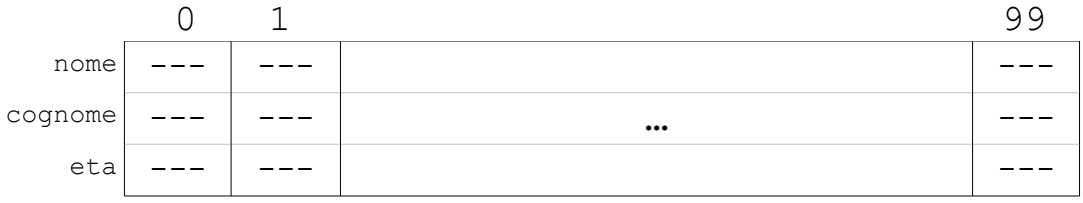
Possibile definire *array* i cui elementi sono *struct*

Esempio:

```
Persona elenco[100];
```

dichiara un *array* di nome `elenco` costituito da *100* elementi di tipo `persona`

Graficamente



Esempi d'uso:

```
elenco[0].eta = 35;
cout << elenco[1].cognome[0];
for(int i = 0; i < 100; i++)
        elenco[i].eta++;
```