Laboratorio di Programmazione

Corso di Laurea in Informatica

Gr. 3 (N-Z)

Università degli Studi di Napoli Federico II

A.A. 2022/23

A. Apicella

Structs (a.k.a. Records)

Aggregato di dati i cui tipi non sono per forza omogenei.
L'allocazione di una struct in memoria necessita di **tre** fasi:

- ***Dichiarazione***: ossia dichiarare l'esistenza del suo nome
  - non sempre necessaria, in quanto implicita nella *definizione*
- ***Definizione***: viene specificato *quali* elementi la struttura dovrà contenere.
  - in questa fase **non viene effettuata alcuna allocazione**, ma vengono soltanto *descritti* i *campi* che comporranno le eventuali strutture di quel tipo, quando e se allocate
- ***Allocazione***: effettiva allocazione in memoria dell'object (o degli object) di tipo struttura

```
struct Paziente;
```
⇦ Dichiarazione

```
struct Paziente
{
    float altezza;
    float peso;
    int   eta;
};
```
⇦ Definizione

```
int main()
{
    struct Paziente p;

    return 0;
}
```
⇦ Allocazione

Structs (a.k.a. Records)

Aggregato di dati i cui tipi non sono per forza omogenei.
L'allocazione di una struct in memoria necessita di **due** fasi:

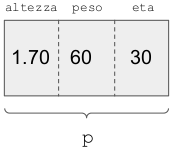
- ***Dichiarazione***: ossia dichiarare l'esistenza del suo nome
  - non sempre necessaria, in quanto implicita nella *definizione*
- ***Definizione***: viene specificato *quali* elementi la struttura dovrà contenere.
  - in questa fase **non viene effettuata alcuna allocazione**, ma vengono soltanto *descritti* i *campi* che comporranno le eventuali strutture di quel tipo, quando e se allocate
- ***Allocazione***: effettiva allocazione in memoria dell'object (o degli object) di tipo struttura

in memoria una volta **allocata**


```
int main()
{
    struct Paziente p;

    return 0;
}
```

⇐ Allocazione

```
struct Paziente
{
    float altezza;
    float peso;
    int   eta;
};
```



Initializing a Struct

2 tipi di inizializzazione:

1. sintassi classica
   - i valori dei campi devono seguire lo stesso ordine dato in fase di definizione

inizializzazione

```
int main()
{
    struct Paziente p = {1.70, 60, 30};

    return 0;
}
```

Initializing a Struct

2 tipi di inizializzazione:

1. sintassi classica
   - i valori dei campi devono seguire lo stesso ordine dato in fase di definizione
2. sintassi *.campo*= (a.k.a. *designators*)
   - i valori dei campi possono essere specificati in qualsiasi rodine

```
struct Paziente
{
    float altezza;
    float peso;
    int   eta;
};
```


```
int main()
{
    struct Paziente p = {.eta     = 30,
                         .peso    = 60,
                         .altezza =
                      1.30};
    return 0;
}
```

inizializzazione

Assigning values to a Struct

Attraverso l'operatore **"."** (**dot**), possono essere assegnati valori ai campi in qualsiasi momento durante la vita della struct

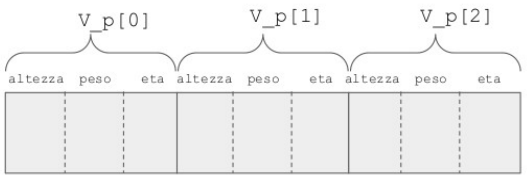
in memoria una volta **allocata**


```
int main()
{
    struct Paziente p;
    p.eta     = 30;
    p.peso    = 60;
    p.altezza = 1.70;
    return 0;
}
```

Assegnazione

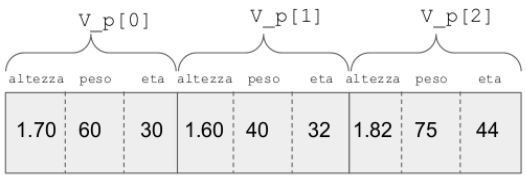
Array of struct

```
int main()
{
    struct Paziente V_p[3];
    return 0;
}
```



Array of struct

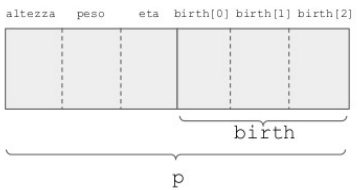
```
int main()
{
    struct Paziente V_p[3];
    for(int i=0; i<3; i=i+1)
    {
        printf("inserisci l'altezza del %d paziente: ", i+1);
        scanf("%f", &V_p[i].altezza);
        printf("inserisci il peso del %d paziente: ", i+1);
        scanf("%f", &V_p[i].peso)
        printf("inserisci l'età del %d paziente: ", i+1);
        scanf("%d", &V_p[i].eta)
    }
    return 0;
}
```



Array in struct

```
struct Paziente
{
    float altezza;
    float peso;
    int   eta;
    int birth[3];
};
```

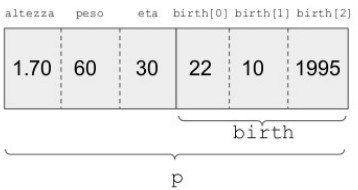
```
int main()
{
    struct Paziente p;

    return 0;
}
```



Array in struct

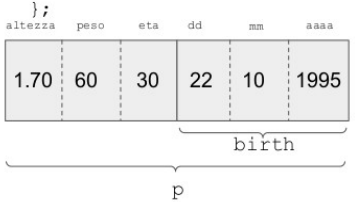
```
struct Paziente
{
    float altezza;
    float peso;
    int   eta;
    int birth[3];
};
```

```
int main()
{
    struct Paziente p;
    p.eta       = 30;
    p.peso      = 60;
    p.altezza   = 1.70;
    p.birth[0] = 22;
    p.birth[1] = 10;
    p.birth[2] = 1995;
    return 0;
}
```



struct in struct

```
struct Data
{
    int dd;
    int mm;
    int aaaa;
};
struct Paziente
{
    float altezza;
    float peso;
    int   eta;
    struct Data birth;
};
```




```
int main()
{
    struct Paziente p;
    p.eta        = 30;
    p.peso       = 60;
    p.altezza    = 1.70;
    p.birth.dd   = 22;
    p.birth.mm   = 10;
    p.birth.aaaa = 1995;
    return 0;
}
```

Array of struct in struct

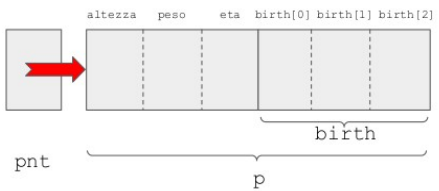
```
struct Aula
{
    int classe;
    char sezione;
};

struct Scuola
{
    char indirizzo[30];
    struct Aula aule[10];
};

struct Quartiere
{
    int cap;
    struct Scuola scuole[5];

};
```

```
int main()
{
    Quartiere q;
    q.scuole[0].aule[2].classe  = 3;
    q.scuole[0].aule[2].sezione = 'B';

    return 0;
}
```

Pointers to struct

```
struct Paziente
{
    float altezza;
    float peso;
    int   eta;
    int birth[3];
};
```

```
int main()
{
    struct Paziente p;
    struct Paziente* pnt;
    pnt = &p;

    return 0;
}
```

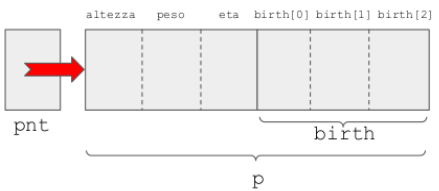


Pointers to struct

```
struct Paziente
{
    float altezza;
    float peso;
    int   eta;
    int birth[3];
};
```

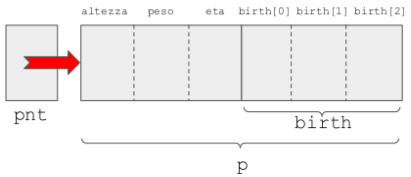
accesso ai campi tramite puntatore…

```
int main()
{
    struct Paziente p;
    struct Paziente* pnt;
    pnt = &p;
    *pnt.altezza = 50;
    return 0;
}
```



Pointers to struct

```
struct Paziente
{
     float altezza;
     float peso;
     int   eta;
     int birth[3];
};
```



accesso ai campi tramite puntatore…

```
int main()
{
     struct Paziente p;
     struct Paziente* pnt;
     pnt = &p;
     *pnt.altezza = 50;
     return 0;
}
```

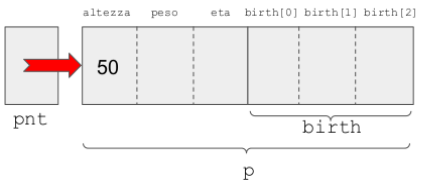
errore! Precedenza operatore . maggiore di *

`*pnt.altezza`

prova ad accedere al campo altezza di `pnt` (che non è una struct, ma un puntatore)

Pointers to struct

```
struct Paziente
{
     float altezza;
     float peso;
     int   eta;
     int birth[3];
};
```



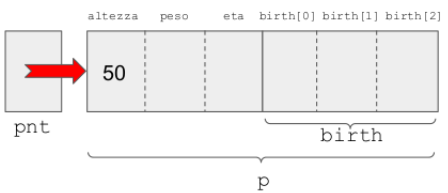
accesso ai campi tramite puntatore…

```
int main()
{
     struct Paziente p;
     struct Paziente* pnt;
     pnt = &p;
     (*pnt).altezza = 50;
     return 0;
}
```

ok!

Pointers to struct

```
struct Paziente
{
    float altezza;
    float peso;
    int   eta;
    int birth[3];
};
```



accesso ai campi tramite puntatore…

```
int main()
{
    struct Paziente p;
    struct Paziente* pnt;
    pnt = &p;
    pnt->altezza = 50;
    return 0;
}
```

equivalentemente si può usare l'operatore ->

`A->B` ⇔ `(*A).B`