UNIVERSITY OF TRENTO - Italy

Laboratorio 16

DISI – aa 2024/2025

Pierluigi Roberti
Carmelo Ferrante

Università degli Studi di Trento
pierluigi.roberti@unitn.it

Inizializzazione di una struttura

Nella dichiarazione di una struttura possiamo introdurre una o più funzioni *con lo stesso nome della struttura*.

Tali funzioni (chiamate metodi) possono essere usate per *inizializzare* i campi della struttura e quindi una variabile di tipo struttura.

La funzione di inizializzazione non ha tipo di ritorno ed è detta "***costruttore***".

Esempio:

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

 Tdata d = Tdata(21, 3, 2006); //inizializza i campi di d
```

Inizializzazione di una struttura

Il C++ mette a disposizione due modi per dichiarare una variabile di un tipo non standard.

Ad esempio, se vogliamo dichiarare una variabile dal nome “oggi” di tipo Tdata come da dichiarazione precedente, possiamo procedere come sotto:

```
Tdata oggi(21, 3, 2006);

Tdata oggi = Tdata(21, 3, 2006);
```

Costruttori di una struttura

Si possono definire **vari costruttori** per una struttura a patto che essi differiscano per il numero e/o il tipo di parametri.

Il compilatore capisce quale costruttore si intende usare nei vari casi in base al numero e/o al tipo di parametri specificati nella chiamata.

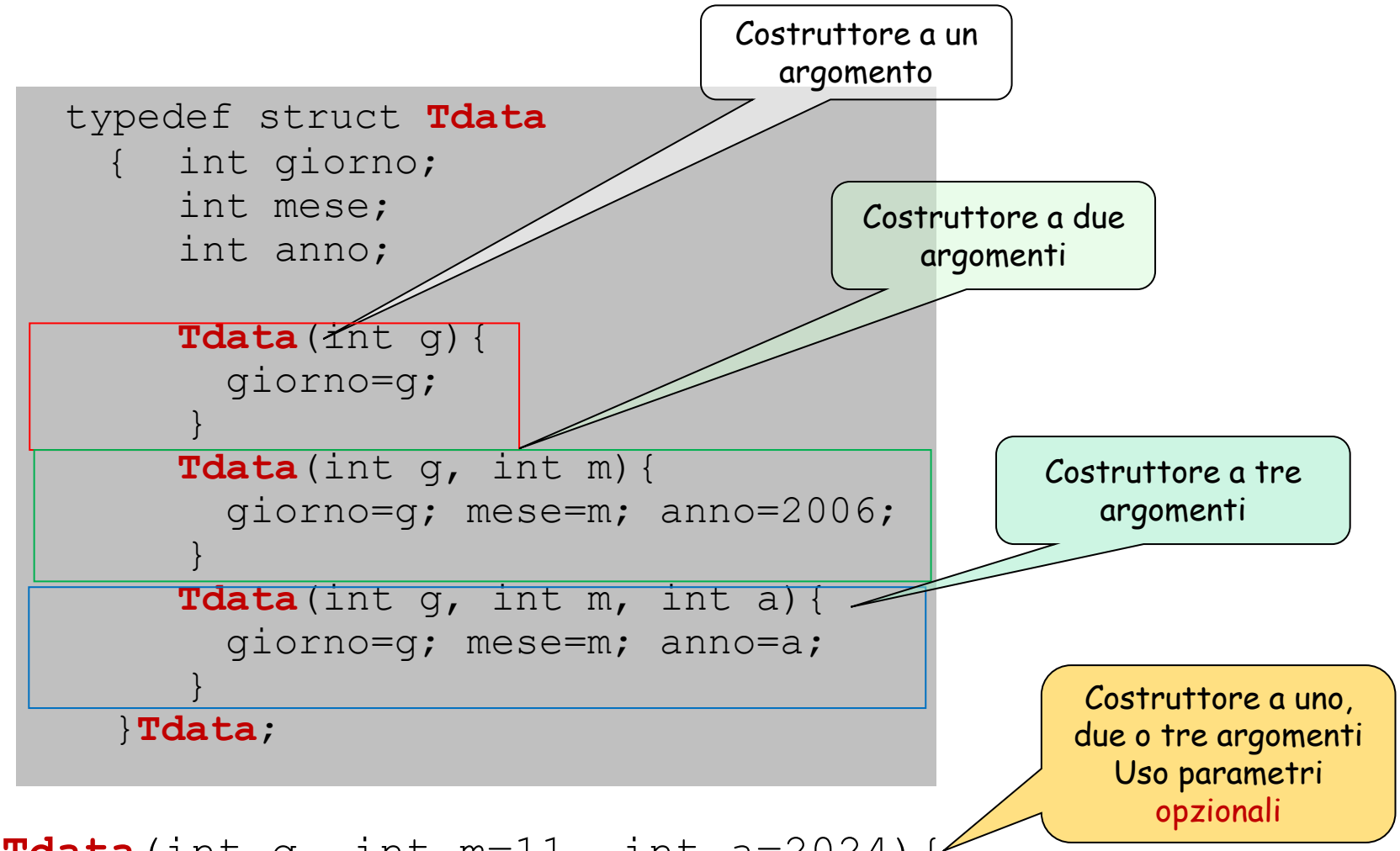
Una struttura ha sempre un **costruttore a zero parametri** (detto di **default**) che serve per dichiarare una variabile.

Esempio di come è implementato.

Non serve dichiararlo a meno che non ci siano altri costruttori!!!

```
// Struttura per rappresentare la data
 typedef struct Tdata {
   int giorno, mese, anno;
  Tdata() { }
 } Tdata;

 Tdata d;
```

Più costruttori di una struttura




```
typedef struct Tdata
  {  int giorno;
     int mese;
     int anno;

     Tdata(int g){
       giorno=g;
      }
     Tdata(int g, int m){
       giorno=g; mese=m; anno=2006;
      }
     Tdata(int g, int m, int a){
       giorno=g; mese=m; anno=a;
      }
   }Tdata;
```

```
Tdata(int g, int m=11, int a=2024){
   giorno=g; mese=m; anno=a;
}
```

Costruttore default di una struttura

- Quando il costruttore non viene definito, il C++ usa per costruire la struttura un **costruttore di default** che alloca memoria per i vari campi della struttura e lascia indefinito il loro valore.
- Esempio:

```
typedef struct Tdata {
   int giorno;
   int mese;
   int anno;
} Tdata;

 Tdata oggi;
```

Viene invocato il costruttore senza argomenti predefinito

Costruttori di una struttura

Appena definiamo un costruttore per la struct, *rinunciamo* ad usare il costruttore di default; il suo uso viene inibito dal compilatore.

In sua vece possiamo definire un costruttore senza argomenti.

Esempio:

```
typedef struct Tdata {
   int giorno;
   int mese;
   int anno;
   Tdata(){giorno=1; mese=1; anno=2000;}
   Tdata(int g, int m, int a){
     giorno=g; mese=m; anno=a;}
}Tdata;
```

In tal modo è ancora possibile scrivere dichiarazioni del tipo:

```
Tdata oggi;
```

Viene invocato il costruttore senza argomenti, *ma non quello predefinito*, bensì quello definito da noi.

Assegnamento e Strutture

Similmente ai costruttori per l'inizializzazione che abbiamo visto prima, esiste un altro costruttore detto ***costruttore per copia***.

- Il costruttore di inizializzazione è invocato dall'operatore con lo stesso nome della struttura (e.g., data(3,4,2004));
- Il costruttore per copia è invocato dall'operatore di assegnamento **=**
- Il costruttore per copia è invocato nel passaggio per copia di una variabile ad una funzione

Il **costruttore di copia** è una funzione **automaticamente definita** per ogni struct il cui comportamento è quello di far corrispondere ad uno ad uno i campi delle due struct cui si applica.

Il suo comportamento è molto **simile** a quello dell'**assegnamento**.

Costruttore di copia

Il costruttore di copia viene invocato in tre occasioni:

[1] Copia di variabile; ad esempio

```
Tdata oggi(23,11,2022);
Tdata copia_oggi = oggi;  //Costruttore di copia
```

[2] Passaggio di parametri per valore;

[3] Quando si usa l'istruzione "return".

Ogni campo è copiato singolarmente!!!

Esempio di come è implementato il costruttore di copia.

Non serve dichiararlo!!!

```
typedef struct Tdata {
   int giorno, mese, anno;
Tdata (const Tdata& d) {
   giorno = d.giorno;
   mese = d.mese;
   anno = d.anno;
 }
 } Tdata;
```

NOTA costruttore copia

```
typedef struct Tdato {
     char c;
} Tdato;
```

- **Costruttore per copia di default** (già esistente)

```
Tdato(const Tdato& d){
     c = d.c
}
```

- Tutto ok
- Non necessita re-implementarlo!!

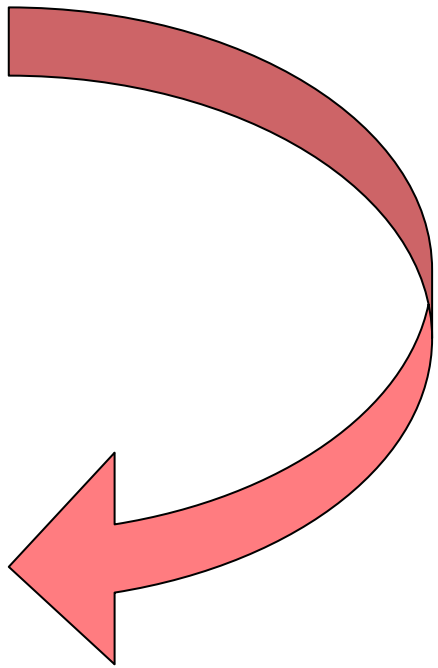
NOTA costruttore copia

```
typedef struct Tdato {
     char* s;
} Tdato;
```

Reimplementare il costruttore per copia di default

```
Tdato(const Tdato& d){
     s = d.s;
}
```

- Altrimenti ho problemi!!!!
- Problema noto come Shallow Copy
- Soluzione: Deep Copy



```
Tdato(const Tdato& d){
     s = new char[strlen(d.s)+1];
     strcpy(s, d.s);
}
```

Distruttore di una struttura

~ ALT+126

Esempio:

```
typedef struct Tdata {
   int giorno;
   int mese;
   int anno;
   ~Tdata(){cout << "data distrutta"<<endl;}
   Tdata(){giorno=mese=anno=0;}
   Tdata(int g, int m, int a){
     giorno=g; mese=m; anno=a;}
}Tdata;
```

racchiudere la dichiarazione della variabile tra parentesi graffe, in modo da poter "vedere" cosa stampa il distruttore, altrimenti invocato al termine del programma

```
{
   Tdata oggi;
}
```

Viene invocato il distruttore

Funzione (metodo) in una struttura

Esempio:

```
typedef struct Tdata {
    int giorno;
    int mese;
    int anno;
    ~Tdata(){cout << "data distrutta"<<endl;}
    Tdata(){giorno=mese=anno=0;}
    Tdata(int g, int m, int a){
      giorno=g; mese=m; anno=a;}
    void stampa()const{
      printf("%d/%d/%d",giorno,mese,anno);
     }
};
```

const => nel metodo stampa non è possibile modificare attributi struct!!

nel main dichiarare la variabile ed invocare la funzione di stampa:

```
Tdata datanascita(12,6,1989);
datanascita.stampa();
```

Viene invocato la funzione (metodo) di stampa

Costruttore di copia

```
void stampaFunz (Tdata d) {
   cout << "data in Funz " << &d << endl;
   d.stampa();
}
Tdata foo (){
   Tdata d(5, 12, 2016);
   cout << "data in foo " << &d << endl;
   return d;
}
int main (){
   Tdata d1(1, 11, 2016);
   cout << "data in main " << &d1 << endl;
   stampaFunz(d1);
   Tdata d2;
   d2 = foo();
   cout << "data2 in main" << &d2 << endl;
   Tdata d3 = foo();
   cout << "data3 in main" << &d3 << endl;
}
```

- Output

  Data in main 0x9ffe10
  Data in Funz 0x9ffe20
  1/11/20
  Data distrutta
  Data in foo 0x9ffe30
  Data distrutta
  Data2 in main 0x9ffe00
  Data in foo 0x9ffdf0
  Data3 in main 0x9ffdf0
  Data distrutta
  Data distrutta
  Data distrutta

Passaggio parametri a funzoni

```
typedef struct Tdato { int val; } Tdato;
```

Invocazione

- Per valore
  - f1(d);
- Per indirizzo
  - f2(&d);
- Per riferimento
  - f3(d);

Prototipo funzioni

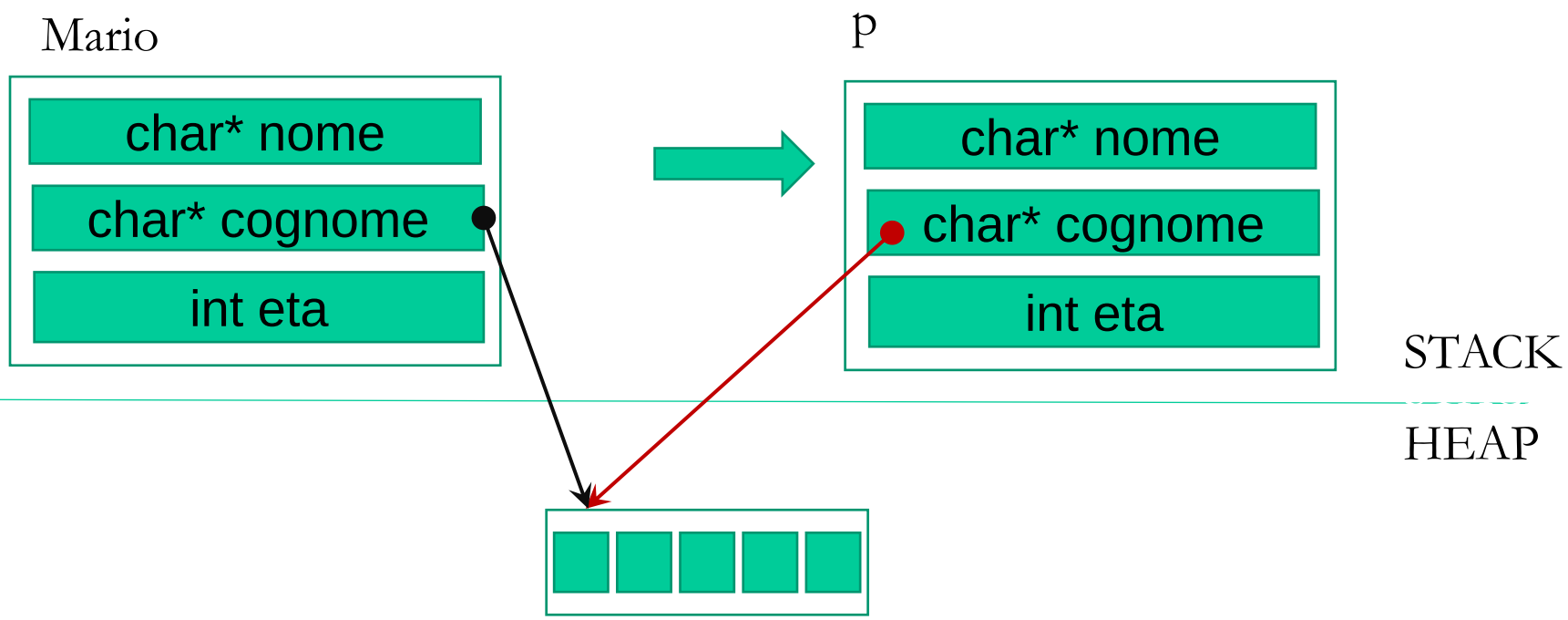
- Per valore
  - void f1(Tdato x);
  - // x.val
  - Modifica di x è locale alla funzione
- Per indirizzo
  - void f2(const Tdato***** x);
  - // x->val o (*x).val
  - La modifica di x (possibile se non specificato const) modifica d
- Per riferimento
  - void f3(**const** Tdato**&** d);
  - // x.val
  - La modifica di x (possibile se non specificato const) modifica d

Passaggio parametri a funzoni

- **Se** Tdato contiene valori definiti solo in memoria Stack
  - `typedef struct Tdato { int val; } Tdato;`
  - Possibile usare passaggio di parametri per
    - Valore
    - Indirizzo
    - Riferimento

- **Se** Tdato contiene campi allocati dinamicamente (definiti in memoria Heap)
  - `typedef struct Tdato { int* val; } Tdato;`
  - Possibile usare passaggio di parametri per
    - Indirizzo
    - Riferimento
  - NON usare passaggio per valore
    - Problemi dovuti a come il costruttore per copia è definito
    - Dettagli esulano dal corso, ma presenti in slide «approfondimenti»

p = Mario




```
void stampa (const Tpersona p){…}
int main(){
    Tpersona Mario;
    stampa(Mario);   //passaggio di valore
    //…
}
```

Quando p nella funzione stampa viene deallocato al termine della funzione stessa, provoca la deallocazione della memeoria HEAP associata a nome e cognome!!!

Esercizio 1 – cmp + valore

- Aggiungere alla struttura Tdata il metodo

```
int cmp(Tdata d)const;
```

  - Restituisce
    - -1 se la data è precedente a d
    - 1 se la data è successiva a d
    - 0 se la data è uguale a d
- Invocare il metodo valutando il risultato ritornato:

```
Tdata oggi = Tdata(4, 11, 2018);
Tdata ieri(3, 11, 2018);
cout << ogg.cmp(ieri) << endl;
```

```
int cmp(Tdata d)const {
    if (aa<d.aa)  { return -1; }
    if (aa>d.aa)  { return +1; }
    if (mm<d.mm)  { return -1; }
    if (mm>d.mm)  { return +1; }
    if (gg<d.gg)  { return -1; }
    if (gg>d.gg)  { return +1; }
    return 0;
}
```

Esercizio 1 – cmp + reference

- Aggiungere alla struttura Tdata il metodo

```
int cmp(const Tdata& d)const;
```

  - Restituisce
    - -1 se la data è precedente a d
    - 1 se la data è successiva a d
    - 0 se la data è uguale a d
- Invocare il metodo valutando il risultato ritornato:

```
Tdata oggi = Tdata(4, 11, 2018);
Tdata ieri(3, 11, 2018);
cout << ogg.cmp(ieri) << endl;
```

```
int cmp(const Tdata& d)const {
    if (aa<d.aa)  { return -1; }
    if (aa>d.aa)  { return +1; }
    if (mm<d.mm)  { return -1; }
    if (mm>d.mm)  { return +1; }
    if (gg<d.gg)  { return -1; }
    if (gg>d.gg)  { return +1; }
    return 0;
}
```

Esercizio 1 – cmp + indirizzo

- Aggiungere alla struttura Tdata il metodo

```
int cmp(const Tdata* pd)const;
```

  - Restituisce
    - -1 se la data è precedente a d
    - 1 se la data è successiva a d
    - 0 se la data è uguale a d
- Invocare il metodo valutando il risultato ritornato:

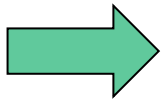
```
Tdata oggi = Tdata(4, 11, 2018);
Tdata ieri(3, 11, 2018);
cout << ogg.cmp(&ieri) << endl;
```

```
int cmp(const Tdata* pd)const{
    if (aa<pd->aa)  { return -1; }
    if (aa>pd->aa)  { return +1; }
    if (mm<pd->mm)  { return -1; }
    if (mm>pd->mm)  { return +1; }
    if (gg<pd->gg)  { return -1; }
    if (gg>pd->gg)  { return +1; }
    return 0;
}
```

Esercizio 2

- Dividere il progetto in più file: creare libreria “persona.h”
- Definire una struttura Tpersona contente:

```
char* nome;
char* cognome;
int eta;
```



```
string nome;
string cognome;
int eta;
```

- Definire
  - Costruttore di default
    nome = “Jane”, cognome = “Doe”
  - Costruttore specifico per definire nel main:
    `Tpersona Anna("Anna", "Rossi", 18);`
  - Distruttore
  - Metodo Stampa
  - Funzione di Stampa
  - Metodo Confronta
- NB utilizzare funzioni della libreria string
  (a seconda del compilatore)
  - #include <cstring>
  - #include <string>

Esercizio 2 – nota

- Nei costruttuori
  - Allocare il campo stringa della dimensione opportuna (in riferimento alla stringa passata come parametro)
  - Copiare la stringa passata come parametro nel campo

```
Tpersona(){
   //assegnamenti alternativi
   nome = new char[strlen("Jane")+1];
   strcpy(nome, "Jane");
   nome = "Jane";
}
```

Se attributo nome dichiarato di tipo string è sufficiente scrivere:

nome = " Jane";

- Nel distruttuore
  - De-allocare il campo stringa (se allocate dinamicamente con new)

```
~Tpersona(){
   delete [] nome;
}
```

- Pamateri delle funzioni
  - Passaggio per INDIRIZZO o per RIFERIMENTO
  - Struct contiene puntatore → passaggio per valore può creare problemi

Esercizio 3 parte 1

- Date le seguenti strutture dati

```
Tpunto: float x, y;
Tfigura: Tpunto punti[MAXP]; int n; // MAXP vale 10
```

- Definire
  - `Tpunto`
    - `Costruttori:` default (x e y pari a 1.0) e specifico (x e y passati come parametri)
    - Metodi:
      - `float distanzaDa0()const;`
      - `float distanza(const Tpunto p) const;`
        `// oppure float distanza(const Tpunto& p) const;`
      - `void stampa() const;`
  - `Tfigura`
    - `Costruttori:` default (nessun punto=> `n=0`) e specifico (numero punti passato come parametri e inizializzare i punti in modo **casuale** tra 0.0 e 10.0)
    - Metodi
      - `float lunghezza() const;`
      - `Tpunto puntoPiuLontanoDa0() const;`
      - `void stampa() const;`

      `// deve invocare il metodo stampa di Tpunto!!!;`

Esercizio 3 parte 2

- Nel main dichiarare una variabie f contenente 3 punti nel modo seguente:

```
Tfigura f = Tfigura(3);
//oppure
Tfigura f(3);
```

- Invocare I seguenti metodi:

```
cout << “Punti della figura:";
f.stampa();
cout << endl;

cout << "lunghezze:";
cout << f.lunghezza() << endl;

cout << "punto piu' distante di f:";
p = f.puntoPiuLontanoDa0();
p.stampa();
cout << endl;
```

Esercizio 4 – parte1

- Estendere l'esercizio precedente con struttura dati

  `Tdisegno``: Tfigura figure*; int nf;`

  `figure:` array da allocare dinamicamente – chiedere all'utente la dimensione **`nf`** (`figure = new Tfigura[`**`nf`**`];`)
- Definire
  - `Tdisegno`
    - `Costruttori:` default (3 figure) e specifico (numero figure passato come parametro e inizializzare ogni figura con un numero casuale di punti compreso tra 0 e MAXP)
    - `Distruttore`
    - Metodi:
      - `Tfigura figuraConLunghezzaMaggiore()` **`const;`**
      - `Tfigura figuraConPuntoPiuDistanteDa0()` **`const;`**
      - `void stampa()` **`const;`**
- Note
  - **`nf`**`:` dimensione massima dell'array
  - L'array è completamente utilizzato (numero elementi da considerare nell'array **`nf`**)

Esercizio 4 – parte2

- Estendere l'esercizio precedente con funzioni per salvare su file
  - `void salvaDisegno (const Tdisegno *d);`
    `// oppure void salvaDisegno (const Tdisegno& d);`
  - `void leggiDisegno (const Tdisegno *d);`

- Estendere l'esercizio precedente con funzioni per salvare su file in modalità binaria
  - `void salvaDisegnoBin (Tdisegno *d);`
    `// oppure void salvaDisegnoBin (const Tdisegno& d);`
  - `void leggiDisegnoBin (Tdisegno *d);`

- Note
  - File testo: disegno.txt
  - Il salvataggio del disegno in modalità BINARIA **DEVE** essere fatto salvando su due file DISTINTI
    - dimensione → disegno_dim_bin.txt
    - dati → disegno_bin.txt

Esercizio 4 – parte2

- `void salvaDisegno (Tdisegno d);`
  - Numero di figure nel disegno
  - Per ogni disegno
    - Numero dei punti del disegno
    - Per ogni punto del disegno
      - Coordinata del punto in formato [x, y]

- Esempio disegno.txt

```
3
8
[7.200000 8.500000]
[8.000000 3.800000]
[6.500000 6.900000]
[9.600000 6.800000]
[4.900000 2.200000]
[5.100000 6.700000]
[6.300000 6.100000]
...
```

Esercizio 4 – parte2

- `void salvaDisegnoBin (Tdisegno d);`
  - File: disegno_dim_bin.txt
    - Salvare il numero di figure in disegno
  - File
    - dati → disegno_bin.txt
    - Salvare array figure in formato binario
      - Con unica fwrite
- Non possibile/complicato salvare il disegno come
  - `fwrite(&d, sizeof(Tdisegno), 1, file)`
  - Problemi dovuti ad allocazione dinamica dell'array figure
- Possibile/facile salvare l'array figure con unica fwrite
  - `fwrite(figure, sizeof(Tfigure), d.n_max_f, file)`
  - Ogni variabile di tipo Tfigura ha una dimensione nota
    - L'array dei punti è definito in modo statico

Esercizio 4 – parte2 (nota)

- Dettaglio nota

```
// i puntatori hanno dimensione 8 byte
sizeof(int) : 4
sizeof(int*) : 8
sizeof(char) : 1
sizeof(char*) : 8

sizeof(Tdisegno) : 16

sizeof(int) : 4
sizeof(Tfigura*) : 8

sizeof(Tfigura) : 84
```

8+4 = 12 e non 16…

Da considerare allineamento delle strutture dati

(generalmente allineamento a 8 byte per architetture a 64 bit)

sizeof() **NON** considera la memoria allocata dinamicamente nell'HEAP

Esempio di problema Shallow copy

```
using namespace std;
typedef struct Tpersona {
      char* nome;
      Tpersona() {
            nome = new char[strlen("Jane")+1];
            strcpy(nome, "Jane");
            cout << "Persona Inizializzata per default\n";
      }
      Tpersona(char* _nome) {
            nome = new char[strlen(_nome)+1];
            strcpy(nome, _nome);
            cout << "Persona Inizializzata in modo specifico\n";
      }
      ~Tpersona() {
            cout << "Distruggo: " << nome << " " << ".\n";
            delete[] nome;
            cout << "Persona distrutta\n";
      }
} Tpersona; //end struct
```

Esempio di problema Shallow copy

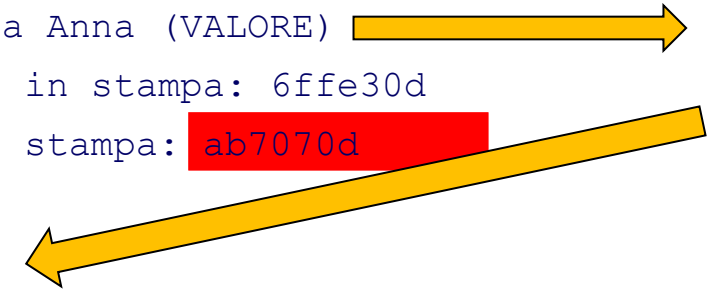
```
// passaggio per valore
void stampa(const Tpersona p){
      printf("locazione persona in stampa: %0xd", &p);
      printf("\nlocazione nome in stampa: %0xd\n", p.nome);
      cout << p.nome << " " << endl;
}
// p viene deallocato alla fine della funzione!!!

// passaggio per riferimento
void stampa2(const Tpersona* p){
      printf("locazione persona in stampa2: %0xd", p);
      printf("\nlocazione nome in stampa2: %0xd\n", p->nome);
      cout << p->nome << " " << endl;
}
```

Esempio di problema Shallow copy

```
int main(){
      {
        Tpersona Anna("Anna");
        Tpersona Mario=Anna;
        printf("mario: %0xd\n", &Mario);
        printf("mario nome: %0xd\n", Mario.nome);
        printf("anna: %0xd\n", &Anna);
        printf("anna nome: %0xd\n", Anna.nome);
        cout << "invocazione stampa Anna (VALORE)" << endl;
        stampa(Anna);
        cout << "invocazione stampa Anna (RIFERIMENTO)" << endl;
        stampa2(&Anna);
        cout << "Prossima istruzione: chiusura blocco" << endl;
      }
      cout << "Blocco istruzioni chiuso" << endl;
      return 0;
}
```

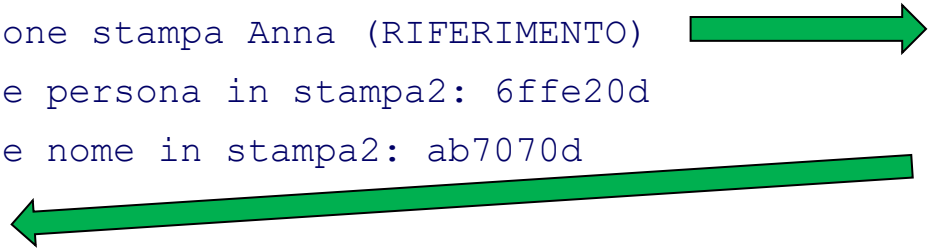
Esempio di problema Shallow copy

```
Persona Inizializzata in modo specifico
mario: 6ffe10d
mario nome: ab7070d
anna: 6ffe20d
anna nome: ab7070d
invocazione stampa Anna (VALORE)
locazione persona in stampa: 6ffe30d
locazione nome in stampa: ab7070d
Anna
Distruggo: Anna .
Persona distrutta
invocazione stampa Anna (RIFERIMENTO)
locazione persona in stampa2: 6ffe20d
locazione nome in stampa2: ab7070d
Anna
Prossima istruzione: chiusura blocco
Distruggo: Anna .
Persona distrutta
Distruggo: Anna .
Persona distrutta
Blocco istruzioni chiuso
```



Con passaggio per VALORE creata copia della variabile.

Copia distrutta prima di uscire dalla funzione.



Con passaggio per INDIRIZZO **NON** è creata copia della variabile.

Manca invocazione del distruttore.