



ALMA MATER STUDIORUM  
UNIVERSITÀ DI BOLOGNA

DIPARTIMENTO DI  
INFORMATICA - SCIENZA E INGEGNERIA

# STRUTTURE

COSIMO LANEVE

`cosimo.laneve@unibo.it`

CORSO 00819 – PROGRAMMAZIONE

# ARGOMENTI (SEZIONE 10.1 SAVITCH)

1. dichiarazioni di strutture
2. l'operazione di selezione e i campi
3. strutture passate come parametri e ritornate come valori di funzioni
4. esempi/esercizi

# PROLOGO


una **struttura** è un dato composto da elementi **in generale differenti** tra loro (eterogenei), raggruppati sotto un unico nome

\* le strutture sono anche chiamati **record**

\* anche gli array sono dei raggruppamenti di dati, **ma dello stesso tipo**

**esempi:**    giorno    19            12            2001  
                         day            month        year

studente	Marco	Bianchi	Unibo	13	10	1975
	nome	cognome	istituto	<u>day</u>	<u>month</u>	<u>year</u>
				data_di_nascita		

volo aereo	AZ506	B747	255	
	n_volo	tipo_areo	n_posti	nome_passeggeri
				(array)

# STRUTTURE/ESEMPI

in C++ è possibile definire tipi di dato che raggruppano in una unica struttura informazioni di tipo diverso

**occorre specificare:**

- \* il nome di ciascuna componente (i componenti sono detti **campi**)
- \* il tipo di informazione memorizzato nelle componenti

**esempi:**

```
struct data {  
    int day ;  
    int month ;  
    int year ;  
};
```

ricordatevi di  
questo ;

è il nome del  
tipo struttura

```
struct volo_aereo {  
    char n_volo [5] ;  
    char tipo_aereo [5] ;  
    int n_posti ;  
    struct passeggero nome_passeggero[255];  
};
```

```
struct studente {  
    char nome [10] ;  
    char cognome [10] ;  
    char istituto [10] ;  
    struct data data_nasc ;  
};
```

una struttura che  
contiene un'altra struttura

# STRUTTURE/DICHIARAZIONE

**sintassi:** `struct id {type1 id_list1 ; type2 id_list2 ; ... ; typen id_listn ; }`  
`;`

- \* l'identificatore `id` è il nome del tipo struttura
- \* ogni `id_listi` è un elenco di uno o più nomi di componenti separati da virgole;
- \* il tipo di ogni componente in `id_listi` è `typei` (`typei` è un tipo semplice o un tipo precedentemente definito)

**osservazione:** la dichiarazione di struttura serve a quantificare lo spazio di memoria necessario per le variabili di quel tipo

una volta definita una struttura, è possibile dichiarare variabili di quel tipo

```
data oggi, giorno_fortunato ;  
studente Pinco_Pallino, Marco_Rossi ;  
volo_aereo volo_Parigi ;
```

# STRUTTURE/OPERAZIONE DI SELEZIONE

**selezione:** operazione per accedere ai campi (*dot-notation*)

**sintassi:**  $lhs\_expression ::= \begin{array}{l} . \ . \ . \\ | \ lhs\_expression \ . \ identifier \end{array}$

**esempi:** `volo_Parigi.n_posti = 250 ;`  
`oggi.day = oggi.day + 2 ;`

# STRUTTURE/OPERAZIONE DI SELEZIONE

esempio:

```
Pinco_Pallino.nome[0] = 'P' ;
```

*è un array*

*selezione su array*

**NOTATE** la differenza tra l'operazione di selezione nelle strutture e quella negli array

*questa differenza consente di evitare errori di tipo*

\* in un array **A** con due elementi è sempre possibile fare **A[0] = A[1] ;**  
perchè **A[0]** e **A[1]** hanno lo stesso tipo

\* in una struttura **A** con due campi "zero" e "uno", l'assegnamento

**A.zero = A.uno ;**

richiede che i due campi abbiano lo stesso tipo (**ed in generale non è così**)

# STRUTTURE/OPERAZIONE DI COPIA

è possibile copiare e assegnare strutture

```
Marco_Rossi = Pinco_Pallino ;
```

**risultato:** tutti i campi di `Pinco_Pallino` sono copiati nei corrispondenti campi di `Marco_Rossi`

\* anche quando i campi sono array (vedi `nome`, `cognome`, `istituto`)

\* in ciò le strutture si differenziano dagli array

**vincolo:** non è possibile confrontare strutture



# STRUTTURE/INIZIALIZZAZIONE

è possibile inizializzare una struttura quando viene dichiarata

```
struct date {  
    int month ;  
    int day ;  
    int year ;  
} ;
```

è possibile dichiarare ed inizializzare una variabile in questo modo:

```
date today = { 10, 24, 2014 } ;
```

**risultato:** i campi month, day e year di today sono  
inizializzati a 10, 24, 2014 rispettivamente

# STRUTTURE COME PARAMETRI DI FUNZIONI

le strutture **possono essere passati come argomenti di funzioni**  
(come gli array) **ed anche essere restituite come valori**

**esempio:**

potete anche scrivere  
struct point

```
struct point { int x ; int y ; } ;  
point make_point(int a, int b){  
    point tmp ;  
    tmp.x = a ; tmp.y = b ;  
    return(tmp) ;  
}
```

**problema:** cosa accade se si definisce

```
void incr_point (point a){  
    a.x = a.x + 1 ;  
    a.y = a.y + 1 ;  
}
```

e nel main si invoca

```
. . . point b ;  
        b.x = 1 ; b.y = 1 ;  
incr_point(b) ; . . .
```

# STRUTTURE COME PARAMETRI DI FUNZIONI/CONT.

**risposta:** a differenza degli array, il passaggio delle strutture come parametro è **per valore**, **compreso il caso in cui un campo della struttura è un array (viene copiato)**

**è possibile passare una struttura per riferimento:**

```
void incr_point (point& a) {  
    a.x = a.x + 1 ;  
    a.y = a.y + 1 ;  
}
```

se nel main si invoca

```
. . .  
point b ;  
b.x = 1 ; b.y = 1 ;  
incr_point(b) ;  
. . .
```

allora l'incremento dei due campi è visibile nel chiamante...

# ESERCIZIO: IL TIPO DI DATO PILA

implementare il tipo di dato pila di interi (LIFO) (per mezzo di strutture e array)

\* l'implementazione deve fornire le operazioni: creazione di una pila vuota, test per pila vuota, inserimento e estrazione

```
const int length = 1000;

struct stack { int data[length];
               int top;
} ;

stack new_stack() {
    stack tmp;
    tmp.top = length;
    return(tmp) ;
}

bool is_empty(stack s) {
    return(s.top == length) ;
}
```

non esiste un elemento in  
data[length]: se top==length  
significa che la pila è vuota

# STRUTTURE/TIPO DI DATO PILA/CONT

```
stack push(stack s, int e){
    if (s.top == 0) { cout << "error, stack full\n" ; }
    else { s.top = s.top - 1; s.data[s.top] = e ; }
    return(s) ;
}

stack pop(stack s){
    if (s.top == length) { cout << "error, empty stack\n"; }
    else { cout << "pop" << s.data[s.top] << "\n" ;
           s.top = s.top + 1; }
    return(s) ;
}
```

**osservazione:** la complessità computazionale di ogni operazione è costante

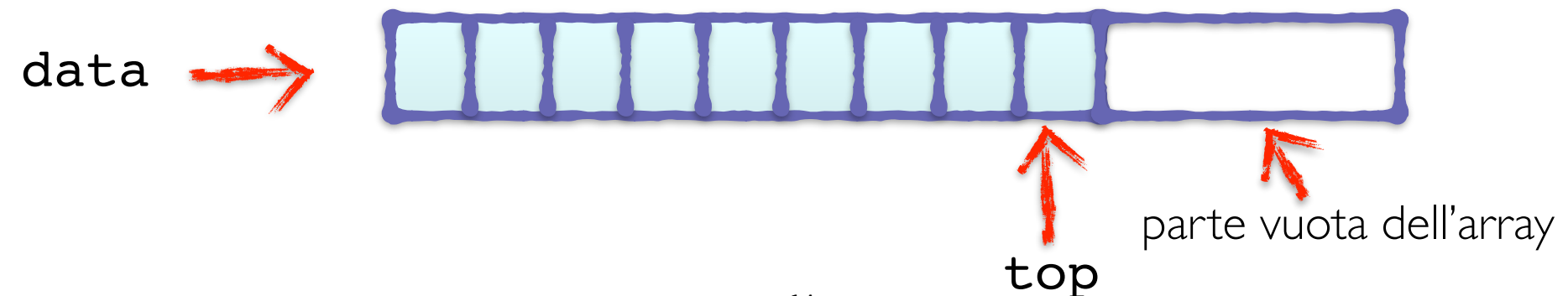
**problemi:** (1) **le risposte erronee "a video" non sono ideali** -- *non ci dovrebbero essere errori!*

(2) non c'è incapsulamento: la struttura degli elementi di tipo stack è visibile all'utente, che può modificarli senza usare le funzioni implementate

# ESERCIZIO: TIPO DI DATO INSIEME

**esercizio:** implementare il tipo di dato insieme di interi (per mezzo di strutture e array)

- \* l'implementazione deve fornire le seguenti operazioni: **creazione** di un insieme vuoto, appartenenza di un elemento all'insieme, intersezione e unione
- \* implementiamo un insieme attraverso una struttura con due campi: un campo **data** che è un array e un campo **top** che contiene l'indice dell'ultimo elemento valido nell'array



# ESERCIZIO: TIPO DI DATO INSIEME

```
const int length = 1000;

struct set {
    int data[length];
    int top;
} ;

set empty_set() {
    set tmp ;
    tmp.top = -1 ;
    return(tmp) ;
}
```

non esiste un elemento  
in data[-1]: se top == -1  
significa che l'insieme è vuoto

# STRUTTURE/TIPO DI DATO INSIEME/CONT.

gli elementi nell'array sono ordinati

**le operazioni saranno più efficienti**

```
bool is_in(set s, int e){  
    if (s.top == -1) return(false) ;  
    else {  
        int l = 0 ;  
        int r = s.top ;  
        int m ;  
        bool found = false ;  
  
        while ((l <= r) && !found) {  
            m = (l+r)/2 ;  
            if (s.data[m] == e) found = true ;  
            else if (s.data[m] > e) r = m-1 ;  
            else l = m+1 ;  
        }  
  
        return(found) ;  
    }  
}
```

algoritmo di ricerca binaria  
perché l'array è ordinato



# STRUTTURE/TIPO DI DATO INSIEME/CONT.

```
set intersection(set s1, set s2){
    if ((s1.top == -1) || (s2.top == -1)) return(empty_set()) ;
    else { set s ;
        int i = 0 ;
        int j = 0 ;
        s = empty_set() ;

        while ((i <= s1.top) && (j <= s2.top)) {
            if (s1.data[i] == s2.data[j]) {
                s.top = s.top + 1 ;
                s.data[s.top] = s1.data[i] ;
                i = i+1 ;
                j = j+1 ;
            } else if (s1.data[i] < s2.data[j]) i = i+1 ;
            else j = j+1 ;
        }

        return(s) ;
    }
}
```

trascrizione di due array ordinati  
in un terzo array (ordinato)

# STRUTTURE/TIPO DI DATO INSIEME/CONT.

```
set set_union(set s1, set s2){
    if (s1.top == -1) return(s2) ;
    else if (s2.top == -1) return(s1) ;
    else { set s = empty_set() ;
        int i1 = 0 ;
        int i2 = 0 ;
```

unione di due array ordinati

```
while ((i1 <= s1.top) && (i2 <= s2.top) && (s.top < length)){
    if (s1.data[i1] < s2.data[i2]) {
        s.top = s.top + 1 ; s.data[s.top] = s1.data[i1] ; i1 = i1+1 ;
    } else if (s1.data[i1] > s2.data[i2]){
        s.top = s.top + 1 ; s.data[s.top] = s2.data[i2] ; i2 = i2+1 ;
    } else {
        s.top = s.top + 1 ; s.data[s.top] = s1.data[i1] ;
        i1 = i1+1 ; i2 = i2+1 ;
    }
}
```

copia  
delle parti  
restanti

```
while ((i1 <= s1.top) && (s.top < length)){
    s.top = s.top + 1 ; s.data[s.top] = s1.data[i1] ; i1 = i1+1 ;
}
while ((i2 <= s2.top) && (s.top < length)){
    s.top = s.top + 1 ; s.data[s.top] = s2.data[i2] ; i2 = i2+1 ;
}
if ((i1 <= s1.top) || (i2 <= s2.top)) cout << "write error" << endl ;
return(s) ;
}
}
```

# STRUTTURE/ESERCIZI

1. scrivere un programma che definisce una struct studente, chiede all'utente di inserire i dati di uno studente e stampa poi il nome dello studente e la media dei suoi voti
2. scrivere un programma che definisce una struttura giorno dell'anno, chiede all'utente di inserire un giorno e calcola quanti giorni sono passati dall'inizio dell'anno
3. scrivere un programma che gestisce gli studenti di informatica, con le seguenti funzioni:
  - a. inserimento di uno studente
  - b. inserimento di un esame
  - c. calcolo della media dei voti di uno studente
  - d. calcolo dell'eta media degli studenti
4. scrivere un programma che funziona da agenda telefonica, con le seguenti funzioni:
  - a. inserimento di una persona nell'agenda
  - b. ricerca di una persona per nome
  - c. cancellazione di una persona