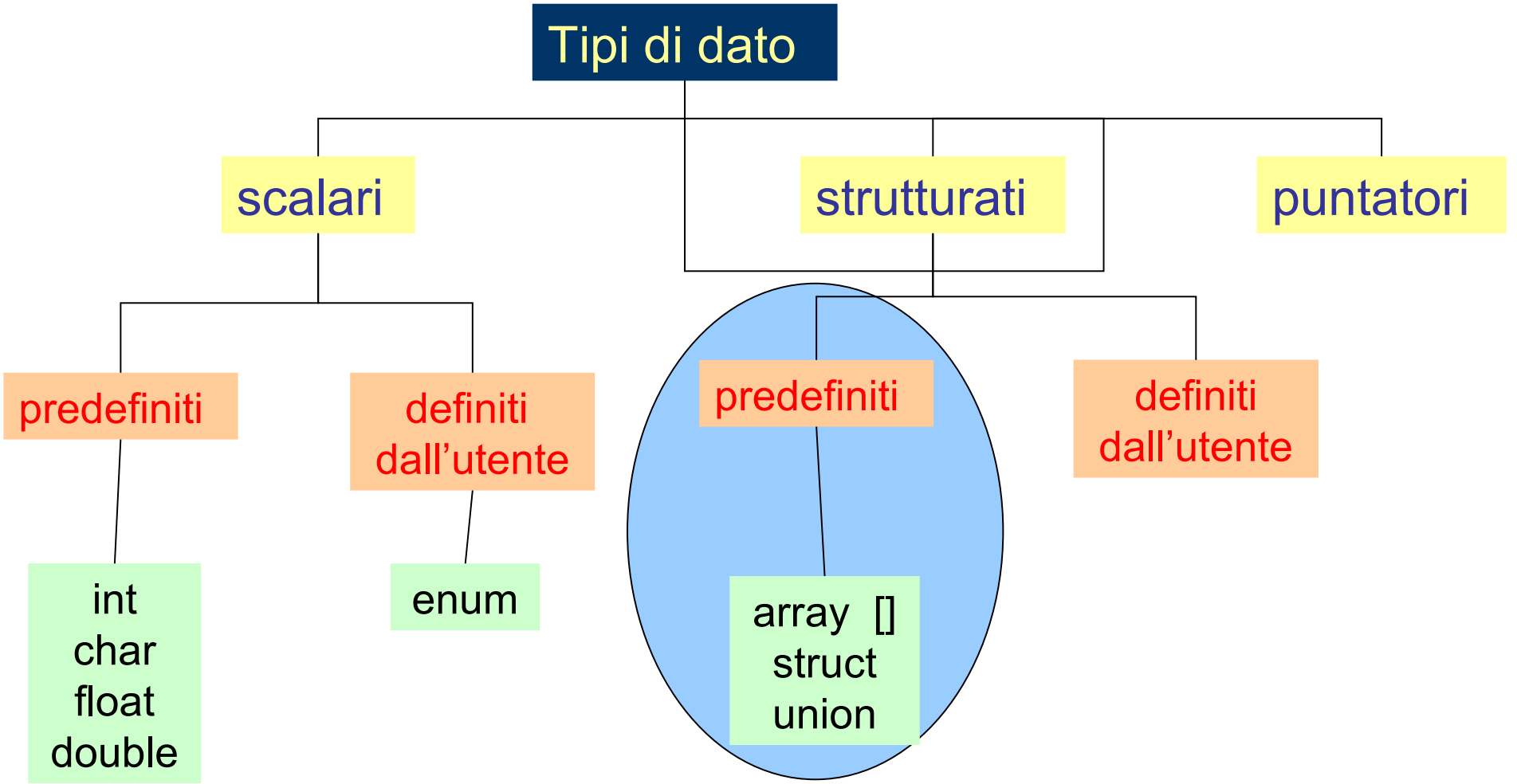
Tipi strutturati in C

Array

Tipi di dato in C



Tipi strutturati

- Il linguaggio C fornisce due costruttori fondamentali:
  - **[ ]** (array)
  - **struct** (strutture, record)

VETTORI

- Un vettore è un insieme finito di **N** variabili dello stesso tipo, ognuna identificata da un indice intero compreso fra **0** e **N-1**

STRUTTURE

- Una struttura è un insieme finito di variabili non necessariamente dello stesso tipo, ognuna identificata da un nome

Vettori

- Gli **array** (vettori) sono uno dei tipi di dato strutturati:
  - sono composti da **elementi omogenei** (tutti dello stesso tipo)
  - ogni elemento è identificato all'interno dell'array da un **numero d'ordine** detto **indice** dell'elemento
  - il numero di elementi del vettore è detto **lunghezza** (o **dimensione**) del vettore

Vettori: dichiarazione

Dichiarazione di variabili di tipo vettore

```
<tipo-elementi> <nome-array> [lunghezza];
```

Dove:

- tipo-elementi è il tipo degli elementi che l'array conterrà;
- nome-array è il nome identificatore dell'array (segue le stesse regole degli identificatori di variabile)
- lunghezza è una espressione costante (cioè nota a tempo di compilazione)

Vettori: esempio dichiarazione

Esempio:

```
int vet[6];
```

- Alloca un vettore di 6 elementi, ovvero 6 locazioni di memoria consecutive, ciascuna contenente un intero.
- **6** è la **lunghezza** del vettore.
- In C l'**indice degli elementi** va sempre da *0* a *lunghezza - 1*
- **vet[i]** denota l'elemento del vettore **vet** di indice **i**.
- Ogni elemento del vettore è una variabile.

Esempio

```
int vet[6], a;
vet[0] = 15;
a = vet[0];
vet[1] = vet[0] + a;
printf("%d %d", vet[0], vet[1]);
```

indice	elemento	variabile
0	15	vet[0]
1	30	vet[1]
2	?	vet[2]
3	?	vet[3]
4	?	vet[4]
5	?	vet[5]

Vettori: esempio dichiarazione

ATTENZIONE:

- La lunghezza del vettore deve essere una espressione costante, cioè nota a tempo di compilazione

```
int N;
float v[N];
```

ERRORE
Il compilatore non sa quanta memoria allocare per l'array

Inizializzazione di vettori

- Gli elementi del vettore possono essere inizializzati con valori costanti contestualmente alla dichiarazione del vettore .

  Esempio: `int n[4] = {11, 22, 33, 44};`

- l'inizializzazione deve essere contestuale alla dichiarazione

  Esempio: `int n[4];`
  `n = {11, 22, 33, 44};` **errore!**

- se ci sono meno inizializzatori di elementi, quelli rimanenti vengono posti a 0

  Esempio: `int n[10] = {3};` azzera i rimanenti 9 elementi del vettore
  `float af[5] = {0.0}` pone i 5 elementi pari a 0,0
  `int x[5] = {};` **errore!**

- se ci sono più inizializzatori di elementi, si ottiene un errore di sintassi

  Esempio: `int v[2] = {1, 2, 3};` **errore!**

- se si mette una lista di inizializzatori, si può evitare di specificare la lunghezza (viene presa la lunghezza della lista)

  Esempio: `int n[] = {1,2,3};` equivale a `int n[3] = {1,2,3};`

Manipolazione di vettori

Manipolazione di vettori

- avviene solitamente attraverso cicli **for**
- l'indice del ciclo varia in genere da 0 a lunghezza - 1
- conviene definire la lunghezza come una costante (uso della direttiva di compilazione **#define**)
  - Es:
```
#define LUNG 6
float v[LUNG];
```

Operazione su vettori

- In C l'unica operazione possibile sugli array è l'accesso agli elementi.
- Non si possono effettuare direttamente delle assegnazioni tra vettori.

Esempio:

```
int a[3] = {11, 22, 33};
int b[3];
b = a;
```

errore! (inoltre non farebbe quello che ci aspettiamo)

Esempio: lettura e stampa di un vettore

```
#include <stdio.h>
#define DIM 5

int main () {
  int v[DIM]; /* vettore di DIM elementi, indicizzati da 0 a DIM-1*/
  int i;

  for (i = 0; i < DIM; i++) {
    printf("Inserisci l'elemento di indice %d: ", i);
    scanf("%d", &v[i]);
  }

  printf("Indice Elemento\n");
  for (i = 0; i < DIM; i++) {
    printf("%d   %d\n", i, v[i]);
  }
}
```

Esempio: somma degli elementi di un vettore

```
#define DIM 10
int a[DIM], i, somma = 0;

... /* lettura degli elementi del vettore */

for (i = 0; i < DIM; i++)
      somma = somma + a[i];
printf("Somma: %d", somma);
```

Esempio: calcolo del massimo di un vettore

```
#define DIM 10
int a[DIM], i, max;

... /* lettura degli elementi del vettore */
max = a[0];
for (i = 1; i < DIM; i++)
  if (a[i] > max) max = a[i];
printf("Massimo del vettore: %d", max);
```

Array multidimensionali

- Non è un nuovo tipo, ma si definiscono come vettori di vettori

Dichiarazione di vettori multidimensionali

```
<tipo-elementi> <nome-array> [lung_1][lung_2]…[lungN];
```

- Per ogni dimensione ***i*** l'indice va da ***0*** a ***lung_i-1***

Esempio:

```
int mat[3][4];
```
→ array bidimensionale di 3 righe per 4 colonne (ovvero **matrice** 3 X 4)

Array multidimensionali

Accesso agli elementi di una matrice

Esempio:

```
int i, mat[3][4];
...
i = mat[0][0];          elemento di riga 0 e colonna 0 (primo elemento)
mat[2][3] = 28;         elemento di riga 2 e colonna 3 (ultimo elemento)
mat[2][1] = mat[0][0] * mat[1][3];
```

- Come per i vettori, l'unica operazione possibile sulle matrici è l'accesso agli elementi tramite l'operatore []

Esempio: lettura e stampa di matrice

```
#include <stdio.h>
#define RIG 2
#define COL 3
int main() {
  int mat[RIG][COL];
  int i, j;
 /* lettura matrice */
  printf("Lettura matrice %d x %d;\n", RIG, COL);
  for (i = 0; i < RIG; i++)
    for (j = 0; j < COL; j++)
      scanf("%d", &mat[i][j]);
 /* stampa matrice */
  printf("La matrice e':\n");
  for (i = 0; i < RIG; i++) {
    for (j = 0; j < COL; j++)
      printf("%6d ", mat[i][j]);
    printf("\n");
  }
}
```