Lezione 7

Typedef
Struct
Array di struct
Allocazione dinamica array di struct
Ancora sui Puntatori: passaggio di parametri per riferimento

Informatica, Corso B - D. Tamascelli

Riassunto puntata precedente

- Allocazione dinamica di array
- Tipo puntatore
- Caricamento dati da file in vettori dinamicamente allocati

Oggi

- Definizione di nuovi tipi di dato
- t-uple
- Parametri formali puntatore

t-uple

Una t-upla è

- una n-upla ordinata
- di elementi **NON NECESSARIAMENTE** dello stesso tipo
- allocata in una zona di memoria senza soluzione di continuità
- ciascuna componente del quale è quindi accessibile se sono noti:

✓L'indirizzo di partenza

✓La dimensione (in byte) di ciascuna componente

- Esempio:

Scheda anagrafica

Nome	Stringa
Cognome	Stringa
Età	Intero
Altezza	Razionale (metri)
Stato civile	carattere

✓Informazione associata ad un individuo

✓Composta da diverse parti (campi)

✓Di diverso tipo

t-uple

Una t-upla, in C++ è

- una n-upla ordinata
- di **campi** (**NON NECESSARIAMENTE** dello stesso tipo)
- allocata in una zona di memoria senza soluzione di continuità

- Esempio (t-upla) C++:

Scheda anagrafica

Nome	Stringa
Cognome	Stringa
Età	Intero
Altezza	Razionale (metri)
Stato civile	carattere

```
struct anagrafica{

char nome[20];

char cognome[30];

int age;

int alt;

char status;

};
```

nome nuovo tipo di dato (derivato)

CAMPI

Come si usa una t-upla?

```
struct anagrafica{

char nome[20];

char cognome[30];

int age;

int alt;

char status;

};
```

```
anagrafica scheda1, scheda2;

scheda1.age = 17;

scheda1.status = 'c';

scheda1.alt = 179;

strcpy(scheda1.nome,"Dario");

strcpy(scheda1.cognome,  "Tamascelli");

scheda2  = scheda1; //!!!!!!!
```

sizeof(T)
byte
distanza

Notazione:

nomevar.campo: si usa come una normale variabile di tipo del campo

Assegnamento (struct-struct):

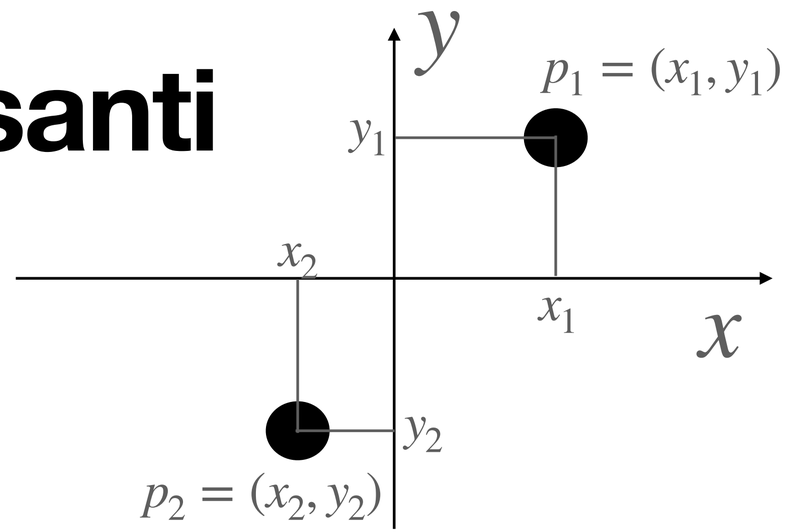
scheda1 = scheda2: copia campo-campo automatica. È come se, una volta dichiarata, la struct fosse un tipo, con tanto di operazione di copia.

Alcuni esempi di t-uple interessanti

puntoR2

x	$\mathbb{R}$
y	$\mathbb{R}$

```
struct puntoR2{
  double x;
  double y;
};
```




```
puntoR2 p1,p2;

puntoR2 v[5];

p1.x  = 0.2;

p1.y = 0.4;

p2 = p1;

p2.y = -p2.y;

for (int i=0; i<5; i++){

    v[i].x = 0.1  * i;

    v[i].y = 0.3*  v[i].t + 0.7;

}
```

```
int n;

puntoR2 *dv;

cin >> n;

dv = new puntoR2[n];

dv[2].x = -0.21;

....

delete [] dv;
```

Ricordate! puntoR2 è una t-upla ed è, dopo la definizione, un nuovo tipo di dato, derivato.

Ricordate! L'assegnamento tra due variabili t-upla dello stesso tipo è:

- possibile (a differenza dell'assegnamento tra array)
- comporta la copia (automatica) campo a campo.

Alcuni esempi di t-uple interessanti

myArrayFloat

size	ℕ
used	ℕ
dati	"n-upla"

```
struct myArrayfloat{
 int size;
 int used;
foat *raw;
};
```

```
myArrayFloat vf;
vf.size  = 10;
vf.used = 0;
vf.raw = new float[vf.size];
vf.raw[0] = 0.2;

....
```

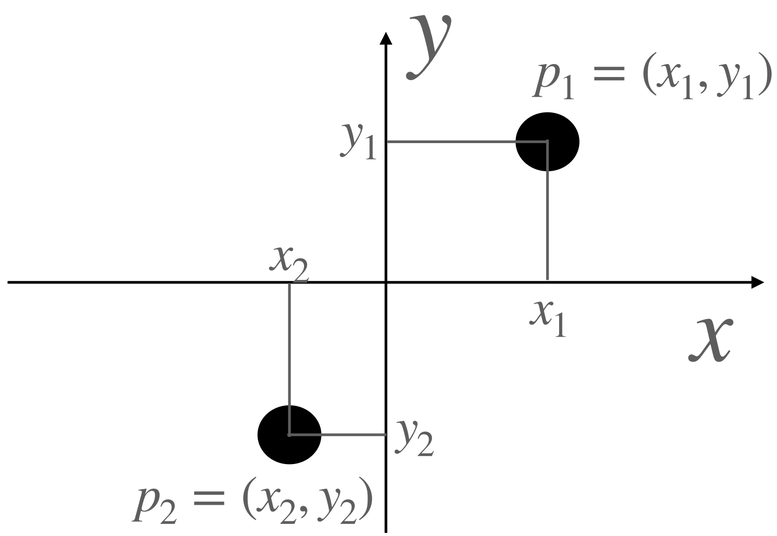
Le informazioni utili per la gestione di un array sono tre, e con le t-uple possiamo incapsulare queste 3 informazioni in una singola variabile.

Altro modo per definire nuovi tipi di dato

- Un punto sul piano è definito dalle sue due componenti, ovvero le coordinate rispetto ad un sistema di riferimento assegnato.
- Un vettore di due elementi (diciamo float) è quindi in grado di rappresentare un punto in $\mathbb{R}^2$

```
float p1[2];

float p2[2];
```

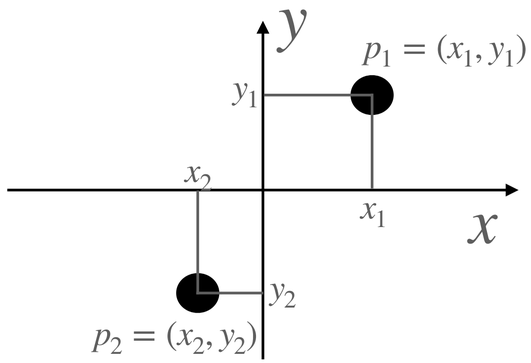
- Chiaramente un vettore di due float può rappresentare tante cose (ad esempio, altezza e larghezza della scrivania), ma in questo contesto rappresenta un punto in $\mathbb{R}^2$
- ...e alcune operazioni/grandezze sono "naturali" per punti in $\mathbb{R}^2$ e non su altre coppie di valori (ad esempio la distanza).



Definizione nuovi tipi di dato

Abbreviazione o nomi semanticamente significativi?

- Il linguaggio consente al programmatore di assegnare nomi a tipi di dato (semplici o derivati), tramite il costrutto typedef



typedef float sp;

definisce il tipo di dato sp come sinonimo di float

typedef float puntoR2[2];

definisce il tipo di dato puntoR2 come sinonimo di float[2]

- Una volta definito, il tipo si può usare come si sono sempre usati i tipi: stanziando variabili di quel tipo

Definizione nuovi tipi di dato

```
typedef float sp;

typedef float puntoR2[2];
```

L'uso del typedef consente quindi da un lato di rinominare i soliti tipi. Il linguaggio ne fa largo uso nelle sue librerie:

```
typedef unsigned int size_t
```

```
sp media, varianza;

puntoR2 p1, p2;

puntoR2 vpunti[5];

puntoR2 *vdyn;
```

Come si usano queste variabili?

Sapendo cosa sono i tipi definiti, è abbastanza semplice.

```
varianza  = 3.2;

p1[0] = 0.2;

cin >> p1[1];puntoR2 vpunti[5];

p2[0]=2.f;

p2[1] = p2[0];
```

Definizione nuovi tipi di dato

- Per poter usare in modo opportuno un tipo di dato è quindi necessario sapere come è fatto. Ma in fondo è sempre stato così.....
- Come "si chiama" l'ascissa del terzo elemento del vettore vpunti?

```
puntoR2 vpunti[5];
```

```
vpunti[2][0]
```

vpunti :array di puntoR2;

vpunti[2]: terza componente del vettore: è un puntoR2

vpunti[2][0]: è la prima componente di un puntoR2

Operazioni e Funzioni?

- Il typedef permette di assegnare nomi a tipi semplici o derivati, quindi a cose che il linguaggio conosce.
- È uno strumento comodo per il programmatore, che riesce a dare nomi semantici alle cose, e quindi ad autodocumentare il codice.
- A livello del compilatore, i nomi dei tipi assegnati tramite typedef vengono "risolti" nei corrispondenti tipi noti al linguaggio.

Noi scriviamo

```
dp varianza;
```

Il compilatore legge

```
double varianza
```

Noi scriviamo

```
puntoR2 pl;
```

Il compilatore legge

```
float pl[2];
```

Noi scriviamo

```
puntoR2 vpunti[5]
```

Il compilatore legge

```
float  vpunti[5][2];
```

Riassumendo

- Il linguaggio fornisce dei tipi di dato "primitivi": `float`, `int`, `char`, `double`, `bool`
- Attraverso gli array possiamo definire tipi derivati "n-upla di T": `float[10]`, `float[5]`, `int[4]`,....
- Attenzione che il tipo è dato dal tipo degli elementi E dal numero degli elementi
- Attraverso le `struct`, possiamo definire tipi derivati t-upla, con campi di tipo diverso
- Il costrutto typedef consente di rinominare tipi di dato semplici o derivati, assegnando loro nomi "semantici"

struct: le useremo, e parecchio

typedef: sappiate che si può fare.

Puntatori e passaggio parametri

Tipi indirizzo

- variabile tipo `char`: contiene informazione di tipo carattere (8 bit, 1 byte)
- variabile tipo `int`: contiene informazione di tipo intero (32 bit, 4 byte)
- variabile tipo `float`: contiene informazione di tipo razionale (sp) (32 bit, 4 byte)
- variabile tipo `double`: contiene informazione di tipo razionale (dp) (64 bit, 8 byte)

- variabile tipo `char *`: contiene informazione di tipo **indirizzo** di una zona di memoria contenente un `char`.
- variabile tipo `int *`: contiene informazione di tipo **indirizzo** di una zona di memoria contenente un `int`.
- variabile tipo `float *`: contiene informazione di tipo **indirizzo** di una zona di memoria contenente un `float`.
- variabile tipo `double *`: contiene informazione di tipo **indirizzo** di una zona di memoria contenente un `double`.

Tipi indirizzo (2)

`T *` : è un tipo di dato

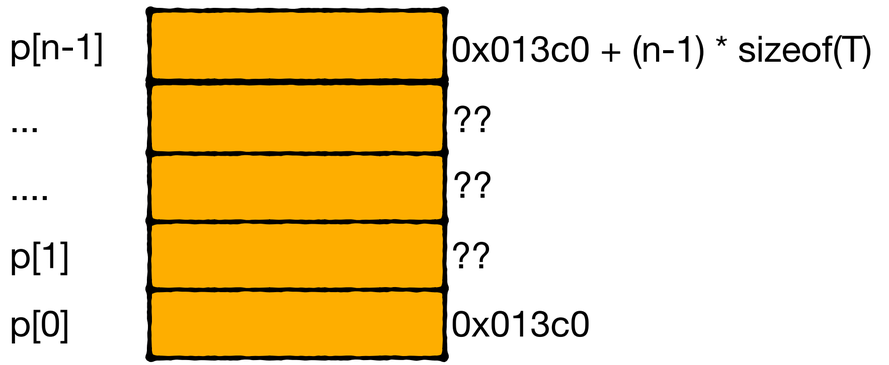
`T * p`: è una variabile di tipo `T *` capace di contenere l'indirizzo di una cella di memoria contenente un dato di tipo `T`.

```
float * p;

int n;

cin >> n;

p = new float[n];
```

Et voilá: abbiamo creato un'area di memoria capace di contenere n float, il cui **indirizzo è stato registrato in p**!

Ancora più attenzione

- Abbiamo visto che la ragione che consente a funzioni di modificare array passati come parametro è il fatto che alla funzione arriva l'indirizzo di inizio dell'array (e il tipo).



`float media(float v[],int dim)  = float media (float * v, int dim)`

- Un parametro formale array (float []) è quindi in realtà sempre stata una variabile di tipo puntatore a float (float *).
- Avendo l'indirizzo di una zona di memoria la funzione poteva andare a scriverci dentro.

Quindi....

Quindi...

Quindi....: possiamo usare i puntatori per far uscire (esportare) diversi valori, anche di tipo diverso, dalla funzione, usando i side-effects in modo sistematico!

- Supponiamo che una funzione abbia un parametri formale int *p

```
float * f(int *p);
```

- La funzione riceve quindi in ingresso un indirizzo, che viene assegnato alla variabile (locale) p
- A questo punto l'indirizzo può essere usato "come" veniva usato per scrivere nelle componenti di un vettore passato alla funzione....più o meno

```
float *f(int p){

float *v = NULL;

*p = 5;

v = new float[*p];

return v;
}
```

```
float *vett;

int dim;

vett = f( &p);
```

Quindi...

```
float * f(int *p);

float *f(int p){

float *v = NULL;

*p = 5;

v = new float[*p];

return v;
}
```

p è di tipo int *, quindi può contenere l'indirizzo di una cella di memoria contenente un intero.

*p si legge: cella di memoria all'indirizzo scritto in p

...quindi *p = 5 scrive nella cella di memoria il cui indirizzo è

scritto in p

....quindi, se il parametro attuale &dim viene passato....

...la funzione scrive il valore 5 in dim!!!!!

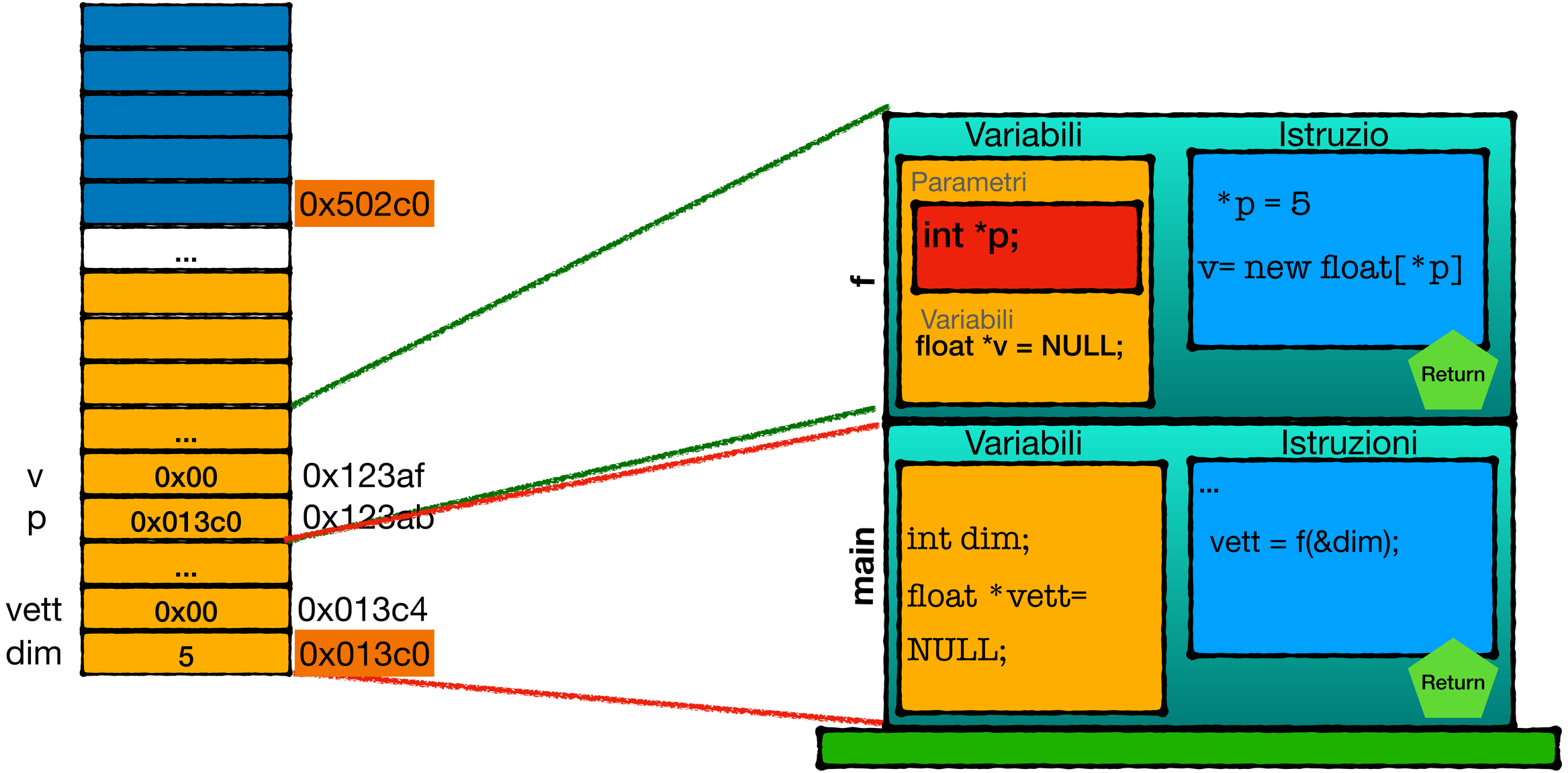
Esempio chiamata della funzione

```
float *vett = NULL;

int dim;

vett = f( &dim);
```

&p : l'operatore (unario) & estrae l'indirizzo del suo

argomento, in questo caso l'indirizzo della variabile

int p


0x502c0
...
...
v
0x00
0x123af
p
0x013c0
0x123ab
...
vett
0x00
0x013c4
dim
5
0x013c0
f
Variabili
Parametri
int *p;
Variabili
float *v = NULL;
Istruzio
*p = 5
v= new float[*p]
Return
main
Variabili
int dim;
float *vett=
NULL;
Istruzioni
...
vett = f(&dim);
Return

Parametri puntatore

DON'T PANIC!!!!

- Puntatori: Introdotti per poter gestire l'allocazione dinamica della memoria
- Hanno messo in luce un meccanismo particolare di passaggio di informazione tra funzioni
- Il meccanismo prevede di passare ad una funzione l'indirizzo di una variabile (zona di memoria) e di sfruttarlo per poter accedere a variabili NON locali della funzione
- Originariamente usato solo per variabili di tipo array (di qualsiasi tipo), ora può essere usato per "esportare" più valori di una funzione tramite side effects.