

Il linguaggio C

Il record

Adrian	Ailincal	2900	21.0
Marco	Albergamo	31000	27.0
Leonardo	Amico	8000	21.0
Massimiliano	Arpino	1500	19.2
Valerio	Zerillo	45000	34.5

Il Record

Esempio:

Si vuole rappresentare adeguatamente l'astrazione "contribuente", caratterizzata dai seguenti attributi:

- **Nome,**
- **Cognome,**
- **Reddito,**
- **Aliquota**

Con gli strumenti visti finora, per ogni contribuente è necessario introdurre 4 variabili:

```
char Nome[20], Cognome[20];
int Reddito;
float Aliquota;
```

È una soluzione scomoda e non "astratta": le quattro variabili sono indipendenti tra di loro.

➔ È necessario un costrutto che consenta l'aggregazione dei 4 attributi nell'astrazione "contribuente": il record.

Tipi strutturati: il Record

Un record è un insieme finito di elementi, in generale non omogeneo:

- il numero degli elementi è rigidamente fissato a priori.
- gli elementi possono essere di tipo diverso.
- il tipo di ciascun elemento componente (campo) è prefissato.

Ad esempio:

NOME	COGNOME	REDDITO	ALIQUOTA
------	---------	---------	----------

Formalmente:

Il record è un tipo strutturato il cui dominio si ottiene mediante prodotto cartesiano:

dati n insiemi, $A_{c1}, A_{c2}, \dots, A_{cn}$, il prodotto cartesiano tra essi:

$$A_{c1} \times A_{c2} \times \dots \times A_{cn}$$

consente di definire un tipo di dato strutturato (il record) i cui elementi sono n -ple ordinate:

$$(a_{c1}, a_{c2}, \dots, a_{cn})$$

dove $a_{ci} \in A_{ci}$.

Ad esempio:

Il **numero complesso** può essere definito attraverso il prodotto cartesiano $R \times R$.

Fondamenti di Informatica L- A

Il Record in C: struct

Collezioni con un numero finito di campi (anche disomogenei) sono realizzabili in C mediante il costruttore di tipo strutturato `struct`.

Definizione di variabile di tipo struct:

```
struct {  
    <lista def. campi>  
} <id-variabile>;
```

```
struct {  
    char Nome[20],  
        Cognome[20];  
    int Reddito;  
    float Aliquota;  
} c1;
```

dove:

- `<lista def. campi>` è l'insieme delle definizioni dei campi componenti, costruita usando le stesse regole sintattiche della definizione di variabili:

```
<tipo1> <campo1>;  
<tipo2> <campo2>;  
...  
<tipoN> <campoN>;
```

```
char Nome[20],  
    Cognome[20];  
int Reddito;  
float Aliquota;
```

- `<id-variabile>` è l'identificatore della variabile di tipo record così definita.

Fondamenti di Informatica L- A

Il tipo struct

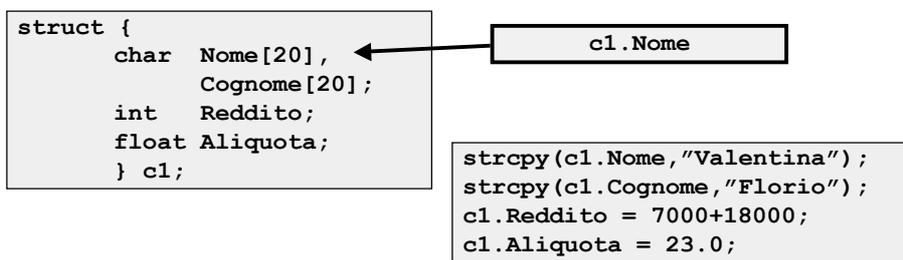
Accesso ai campi:

Per accedere (e manipolare) i singoli campi di un record si usa la notazione **postfissa** :

<id-variabile>.<componente>

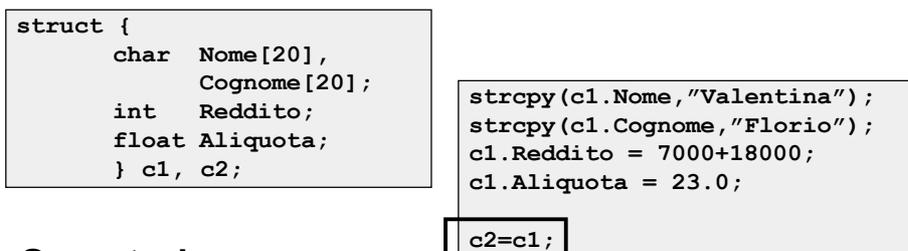
indica il valore del campo **<componente>** della variabile **<id-variabile>** .

→ I singoli campi possono essere manipolati con gli operatori previsti per il tipo ad essi associato.



Fondamenti di Informatica L- A

Il tipo struct: assegnamento



Operatori:

L'unico operatore previsto per dati di tipo struct è l'operatore di **assegnamento** (=):

→ è possibile l'assegnamento diretto tra record di tipo equivalente.

Fondamenti di Informatica L- A

Inizializzazione di record:

È possibile inizializzare i record in fase di definizione.

Ad esempio:

```

struct {  char   Nome[20],
          Cognome[20];
        int    Reddito;
        float  Aliquota;
} c1 = {"Valentina", "Florio", 7000+18000, 23.0 };

```

typedef struct

Il costruttore struct può essere utilizzato per dichiarare tipi non primitivi basati sul record:

Dichiarazione di tipo strutturato record:

```

typedef struct {
  <lista dichiarazioni campi>
} <id-tipo>;

```

dove:

- <lista definizioni campi> è l'insieme delle definizioni dei campi componenti;
- <id-tipo> è l'identificatore del nuovo tipo.

Ad esempio:

```

typedef struct {
  char   Nome[20],
         Cognome[20];
  int    Reddito;
  float  Aliquota;
} contribuente;

contribuente c1,c2,c3;
strcpy(c1.Nome, "Valentina");

```

typedef struct

Nota: non è consentito definire due tipi diversi con lo stesso nome. Però è possibile che un campo del record abbia lo stesso nome di una variabile. Come `Reddito`, nell'esempio:

```
typedef struct {  char Nome[20];
                  char Cognome[20];
                  int  Reddito;
                  float Aliquota;
                } contribuente;
contribuente c1, c2, c3;

unsigned int Reddito = 7000;

c1.Reddito = Reddito + 18000;
```

variabile.campo	variabile
(variabile strutturata)	(intero senza segno)

Fondamenti di Informatica L- A

Il tipo struct in sintesi

Riassumendo, la sintassi da adottare è:

```
[typedef] struct {
    <tipo-1> <nome_campo-1>;
    <tipo-2> <nome_campo-2>;
    ...
    <tipo-N> <nome_campo-N>;
} <nome>;
```

Vincoli:

- `<nome_campo-i>` è un identificatore stabilito che individua il campo *i*-esimo;
- `<tipo-i>` è un qualsiasi tipo, semplice o strutturato.
- `<nome>` è l'identificatore della struttura (o del tipo, se si usa `typedef`)

Uso:

- la struttura è una collezione di un numero fissato di elementi di vario tipo (`<tipo_campo-i>`);
- il singolo campo `<nome_campo-i>` di un record `R` è individuato mediante la notazione: `R.<nome_campo-i>`;
- se due strutture di dati di tipo `struct` hanno lo stesso tipo, allora è possibile l'assegnamento diretto.

Fondamenti di Informatica L- A

Esercizio sui record

Realizzare un programma che, lette da input le coordinate di un punto P del piano, sia in grado di applicare a P alcune trasformazioni geometriche (traslazione, e proiezioni sui due assi).

→ Rappresentiamo il punto del piano cartesiano mediante una **struct** di due campi (float), ciascuno associato a una particolare coordinata:

```
typedef struct{  
    _____  
} punto;
```

P

x
y

```
punto P; /* P è una variabile di tipo punto */
```

Fondamenti di Informatica L- A

Esercizio sui record

```
#include <stdio.h>
```

```
main()
```

```
{ typedef struct{float x,y;} punto;
```

```
  punto P;
```

```
  unsigned int op;
```

```
  float Dx, Dy;
```

```
/* lettura delle coordinate da input in P: */
```

```
printf("ascissa? ");
```

```
_____  
printf("ordinata? ");
```

```
_____
```

```
/* lettura dell'operazione richiesta assumendo le convenzioni:
```

```
0: termina
```

```
1: proietta sull'asse x
```

```
2: proietta sull'asse y
```

```
3: trasla di Dx, Dy */
```

```
_____
```

P

x
y

Fondamenti di Informatica L- A

Esercizio sui record

```

switch (op)
{
  case 1: proietta sull'asse x
  case 2: proietta sull'asse y
  case 3: printf("%s", "Traslazione?")
          scanf("%f%f", &Dx, &Dy);
          calcola
          nuove
          coordinate

  default: printf("errore!");
}
printf( "Nuove coordinate:
        %f\t%f\n" coordinate );
}

```

Fondamenti di Informatica L- A

Vettori e record

Non ci sono vincoli riguardo al tipo degli elementi di un vettore: si possono realizzare anche vettori di record (strutture tabellari).

Ad esempio:

```

typedef struct {
    char   Nome[20];
    char   Cognome[20];
    int    Reddito;
    float  Aliquota;
} Contribuente;

contribuente archivio[1000];

```

→ **archivio** è un vettore di 1000 elementi, ciascuno dei quali è di tipo **contribuente**

→ abbiamo realizzato un **vettore di record**, o **struttura tabellare**.

	Nome	Cognome	Reddito	Aliquota
0				
999				

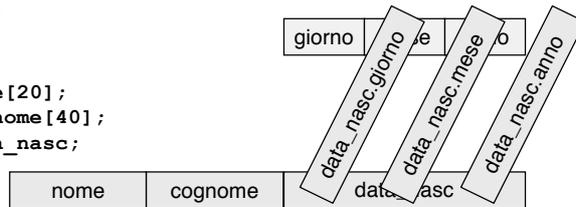
Fondamenti di Informatica L- A

Vettori e Record

Allo stesso modo, si possono fare **record di vettori** e **record di record**.

Ad esempio:

```
typedef struct{ int giorno;  
               int mese;  
               int anno;  
} data;  
  
typedef struct{ char nome[20];  
               char cognome[40];  
               data data_nasc;  
} persona;  
  
persona P;  
...  
P.data_nasc.giorno=25;  
P.data_nasc.mese=3;  
P.data_nasc.anno=1992;  
...
```



Fondamenti di Informatica L- A

Esercizio

Scrivere un programma che acquisisca i dati relativi agli studenti di una classe:

- nome
- cognome
- voti: rappresenta i voti dello studente in 3 materie (italiano, matematica, inglese);

Il programma deve successivamente **calcolare** e **stampare**, per ogni studente, la **media dei voti ottenuti nelle 3 materie**.

Introduciamo un tipo di dato per rappresentare il generico studente:

```
typedef struct {  
    _____  
} studente;
```

La classe è rappresentata da un vettore di studenti:

```
studente classe[20];
```

Fondamenti di Informatica L- A

```

#include <stdio.h>
#define ita 0
#define mat 1
#define ing 2
#define N 20
typedef struct{
    } studente;

main()
{
    studente classe[N];
    float m;
    int i; int j;
    /* lettura dati */
    for(i=0;i<N; i++)
    {
        fflush(stdin);
        gets( nome i-mo studente );
        gets( cognome i-mo studente );
        for(j=ita; j<=ing; j++)
            scanf( voto j-ma materia i-mo studente );
    }

```

Fondamenti di Informatica L- A

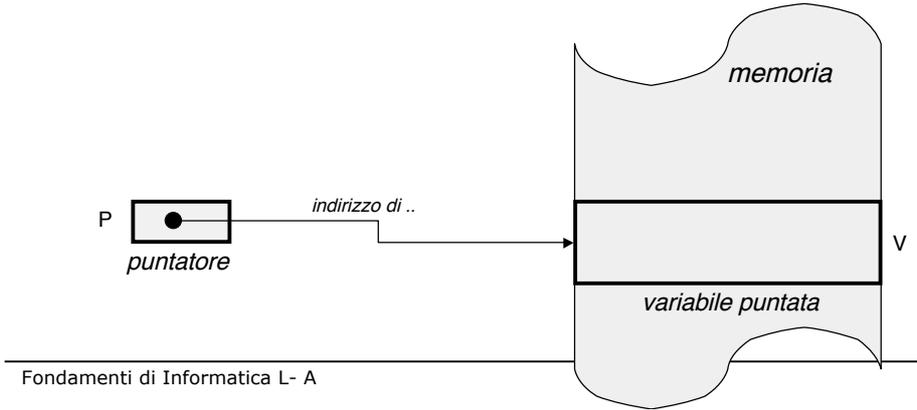
```

/* continua.. stampa delle medie */
for( i=0; i<N; i++ ) // N studenti
{
    for( m=0,j=ita; j<=ing; j++ ) // ita=0; ing=2
        aggiorna il totale dei voti... // m accumulatore
    printf( "media di %s %s: %f\n",
            media di <nome> <cognome>: <media> );
}
}

```

Fondamenti di Informatica L- A

Il linguaggio C I puntatori

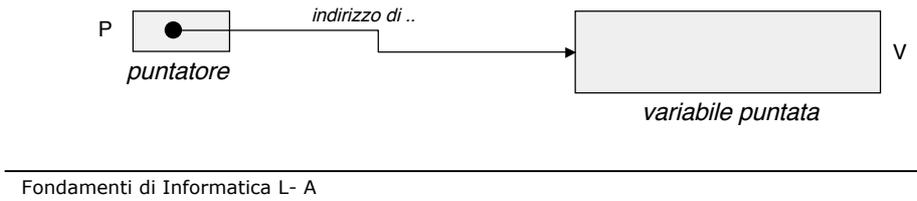


Il puntatore

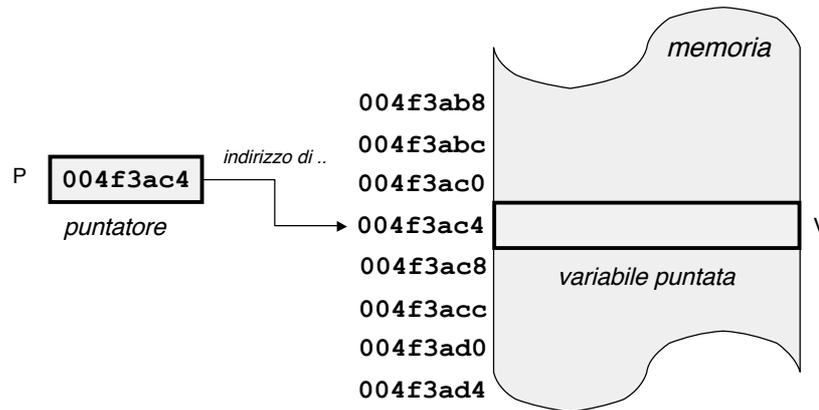
È un tipo di dato scalare, che consente di rappresentare gli **indirizzi** delle variabili allocate in memoria.

Dominio:

Il dominio di una variabile di tipo puntatore è un insieme di indirizzi: il valore di una variabile P di tipo puntatore può essere l'**indirizzo** di un'altra variabile (variabile **puntata**).



Esempio



Fondamenti di Informatica L- A

Il puntatore in C

In C i puntatori si definiscono mediante il costruttore *.

Definizione di una variabile puntatore:

```
<TipoElementoPuntato> *<NomePuntatore>;
```

dove:

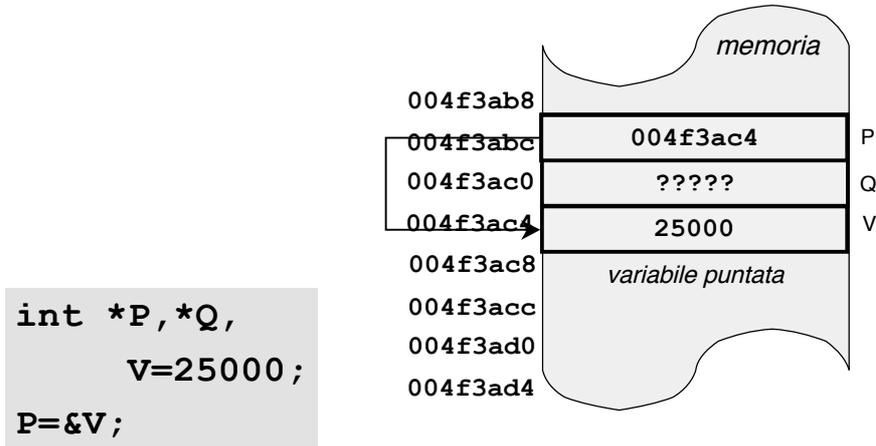
- <TipoElementoPuntato> è il tipo della variabile puntata
- <NomePuntatore> è il nome della variabile di tipo puntatore
- il simbolo * è il costruttore del tipo puntatore.

Ad esempio:

```
int *P; /*P è un puntatore a intero */
```

Fondamenti di Informatica L- A

Esempio



Fondamenti di Informatica L- A

Il puntatore in C

Operatori:

- **Assegnamento (=)**: è possibile l'assegnamento tra puntatori (dello stesso tipo). È disponibile la costante NULL, per indicare l'indirizzo nullo (0).
- **Dereferencing (*)**: è un operatore unario. Si applica a un puntatore e restituisce il valore contenuto nella cella puntata => serve per accedere alla variabile puntata.
- **Operatori aritmetici** (domani).
- **Operatori relazionali**: >, <, ==, !=
- **Operatore indirizzo &**: si applica ad una variabile e restituisce l'indirizzo della cella di memoria nella quale è allocata la variabile.

Ad esempio:

```
int *p1, *p2;
int A;
p1 = &A;
*p1 = 127;
p2 = p1;
p1 = NULL; /* NULL è la costante che vale 0 e
            denota il puntatore "nullo" */
```



Fondamenti di Informatica L- A

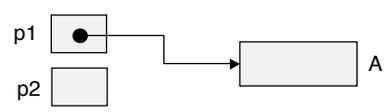
Il puntatore in C

Operatori:

- **Assegnamento (=)**: è possibile l'assegnamento tra puntatori (dello stesso tipo). È disponibile la costante NULL, per indicare l'indirizzo nullo (0).
- **Dereferencing (*)**: è un operatore unario. Si applica a un puntatore e restituisce il valore contenuto nella cella puntata => serve per accedere alla variabile puntata.
- **Operatori aritmetici** (domani).
- **Operatori relazionali**: >, <, ==, !=
- **Operatore indirizzo (&)**: si applica ad una variabile e restituisce l'indirizzo della cella di memoria nella quale è allocata la variabile.

Ad esempio:

```
int *p1, *p2;
int A;
p1 = &A;
*p1 = 127;
p2 = p1;
p1 = NULL;
```



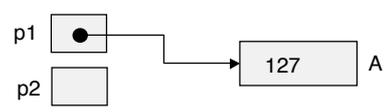
Il puntatore in C

Operatori:

- **Assegnamento (=)**: è possibile l'assegnamento tra puntatori (dello stesso tipo). È disponibile la costante NULL, per indicare l'indirizzo nullo (0).
- **Dereferencing (*)**: è un operatore unario. Si applica a un puntatore e restituisce il valore contenuto nella cella puntata => serve per accedere alla variabile puntata.
- **Operatori aritmetici** (domani).
- **Operatori relazionali**: >, <, ==, !=
- **Operatore indirizzo (&)**: si applica ad una variabile e restituisce l'indirizzo della cella di memoria nella quale è allocata la variabile.

Ad esempio:

```
int *p1, *p2;
int A;
p1 = &A;
*p1 = 127;
p2 = p1;
p1 = NULL;
```



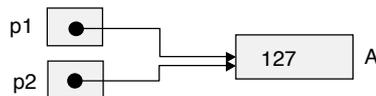
Il puntatore in C

Operatori:

- **Assegnamento (=)**: è possibile l'assegnamento tra puntatori (dello stesso tipo). È disponibile la costante NULL, per indicare l'indirizzo nullo (0).
- **Dereferencing (*)**: è un operatore unario. Si applica a un puntatore e restituisce il valore contenuto nella cella puntata => serve per accedere alla variabile puntata.
- **Operatori aritmetici** (domani).
- **Operatori relazionali**: >, <, ==, !=
- **Operatore indirizzo (&)**: si applica ad una variabile e restituisce l'indirizzo della cella di memoria nella quale è allocata la variabile.

Ad esempio:

```
int *p1, *p2;  
int A;  
p1 = &A;  
*p1 = 127;  
p2 = p1;  
p1 = NULL;
```



Fondamenti di Informatica L- A

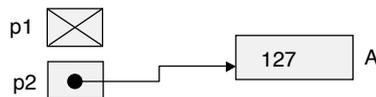
Il puntatore in C

Operatori:

- **Assegnamento (=)**: è possibile l'assegnamento tra puntatori (dello stesso tipo). È disponibile la costante NULL, per indicare l'indirizzo nullo (0).
- **Dereferencing (*)**: è un operatore unario. Si applica a un puntatore e restituisce il valore contenuto nella cella puntata => serve per accedere alla variabile puntata.
- **Operatori aritmetici** (domani).
- **Operatori relazionali**: >, <, ==, !=
- **Operatore indirizzo (&)**: si applica ad una variabile e restituisce l'indirizzo della cella di memoria nella quale è allocata la variabile.

Ad esempio:

```
int *p1, *p2;  
int A;  
p1 = &A;  
*p1 = 127;  
p2 = p1;  
p1 = NULL;
```



Fondamenti di Informatica L- A

Il puntatore in C: * e &

Operatore Indirizzo &:

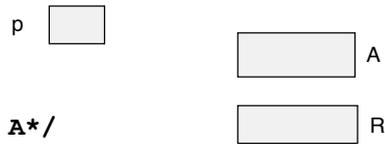
- & si applica solo ad oggetti che esistono in memoria (quindi, già definiti).
- & non è applicabile ad espressioni.

Operatore Dereferencing *:

- consente di accedere ad una variabile specificandone l'indirizzo
- l'indirizzo rappresenta un modo alternativo al nome (*alias*) per accedere e manipolare la variabile:

Ad esempio:

```
float *p;  
float R, A;
```



```
p=&A; /* *p è un alias di A*/  
R=2;  
*p=3.14*R; /* A è modificato */
```

Fondamenti di Informatica L- A

Il puntatore in C: * e &

Operatore Indirizzo &:

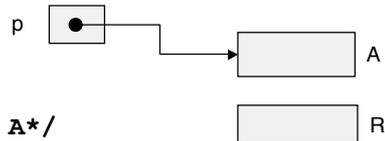
- & si applica solo ad oggetti che esistono in memoria (quindi, già definiti).
- & non è applicabile ad espressioni.

Operatore Dereferencing *:

- consente di accedere ad una variabile specificandone l'indirizzo
- l'indirizzo rappresenta un modo alternativo al nome (*alias*) per accedere e manipolare la variabile:

Ad esempio:

```
float *p;  
float R, A;
```



```
➔ p=&A; /* *p è un alias di A*/  
R=2;  
*p=3.14*R; /* A è modificato */
```

Fondamenti di Informatica L- A

Il puntatore in C: * e &

Operatore Indirizzo &:

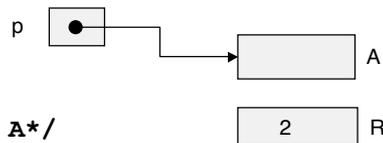
- & si applica solo ad oggetti che esistono in memoria (quindi, già definiti).
- & non è applicabile ad espressioni.

Operatore Dereferencing *:

- consente di accedere ad una variabile specificandone l'indirizzo
- l'indirizzo rappresenta un modo alternativo al nome (*alias*) per accedere e manipolare la variabile:

Ad esempio:

```
float *p;  
float R, A;
```



```
p=&A; /* *p è un alias di A*/  
R=2;  
*p=3.14*R; /* A è modificato */
```

Fondamenti di Informatica L- A

Il puntatore in C: * e &

Operatore Indirizzo &:

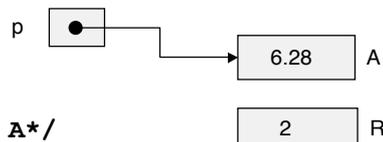
- & si applica solo ad oggetti che esistono in memoria (quindi, già definiti).
- & non è applicabile ad espressioni.

Operatore Dereferencing *:

- consente di accedere ad una variabile specificandone l'indirizzo
- l'indirizzo rappresenta un modo alternativo al nome (*alias*) per accedere e manipolare la variabile:

Ad esempio:

```
float *p;  
float R, A;
```



```
p=&A; /* *p è un alias di A*/  
R=2;  
*p=3.14*R; /* A è modificato */
```

Fondamenti di Informatica L- A

Puntatore come costruttore di tipo

Il costruttore di tipo "*" può essere anche usato per dichiarare tipi non primitivi basati sul puntatore.

```
Dichiarazione di un tipo puntatore:

typedef <TipoElementoPuntato> *<NomeTipo>;
```

- <TipoElementoPuntato> è il tipo della variabile puntata
- <NomePuntatore> è il nome del tipo dichiarato.

Ad esempio:

```
typedef float *tpf;
tpf p;
float f;
p=&f;
*p=0.56;
```

Puntatori: controlli di tipo

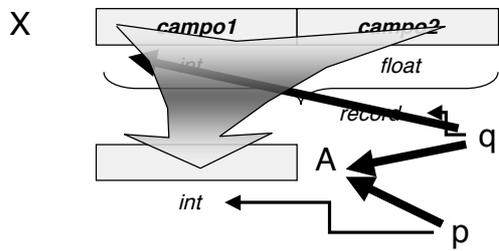
Nella definizione di un puntatore è **necessario** indicare il tipo della variabile puntata.

→ il compilatore **può** effettuare controlli statici sull'uso dei puntatori.

Esempio:

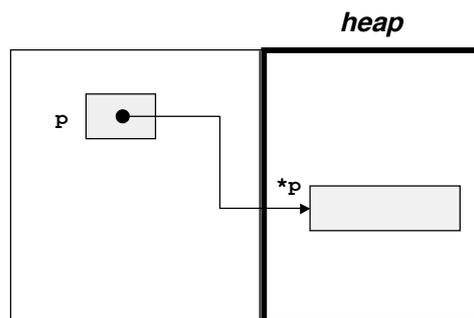
```
typedef struct{int campo1; float campo2;}record;
int A, *p;
record X, *q;

p = &A;
q = p;
/* warning! */
q = &X;
*p = *q;
/* errore! */
```



→ Viene segnalato dal compilatore (**warning**) il tentativo di utilizzo di un puntatore a un tipo diverso rispetto a quello per cui è stato definito.

Variabili dinamiche



Fondamenti di Informatica L- A

Variabili automatiche e dinamiche

In C è possibile classificare le variabili in base al loro tempo di vita.

Due categorie:

- variabili **automatiche**
- variabili **dinamiche**

Variabili automatiche:

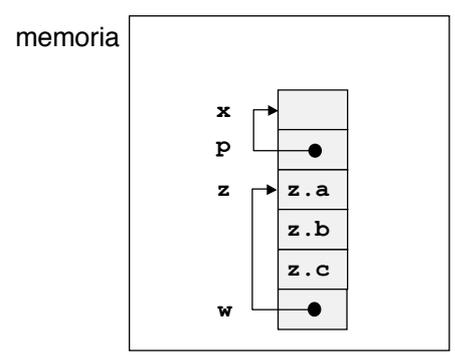
- L'allocazione e la deallocazione di variabili automatiche è effettuata **automaticamente** dal sistema (senza l'intervento del programmatore).
- Ogni variabile automatica ha un **nome**, attraverso il quale la si può riferire.
- Il programmatore non ha la possibilità di influire sul tempo di vita di variabili automatiche.

→ tutte le variabili viste finora rientrano nella categoria delle **variabili automatiche**.

Fondamenti di Informatica L- A

Variabili automatiche

```
int *p, x;  
struct {int a, b, c;} z, *w;
```



Variabili dinamiche

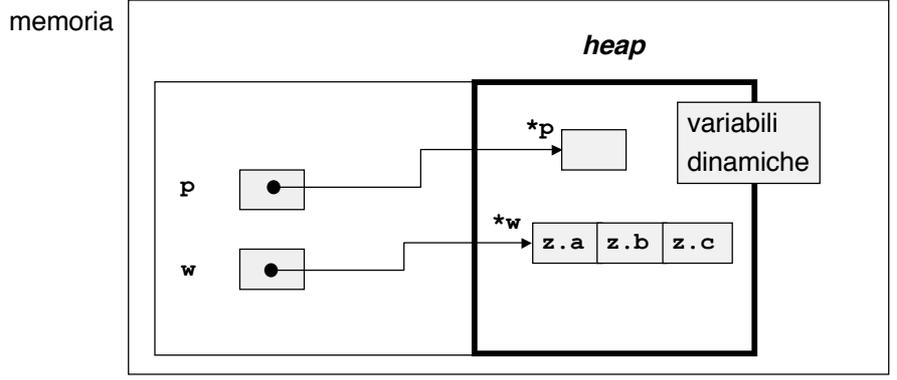
Variabili dinamiche:

- Le variabili dinamiche devono essere allocate e deallocate **esplicitamente** dal programmatore.
- L'area di memoria in cui vengono allocate le variabili dinamiche si chiama **heap**.
- Le variabili dinamiche non hanno un **identificatore**, ma possono essere riferite soltanto attraverso il loro indirizzo (mediante i **puntatori**).
- Il tempo di vita delle variabili dinamiche è l'intervallo di tempo che intercorre tra l'allocazione e la deallocazione (che sono impartite esplicitamente dal programmatore).

Variabili dinamiche

```
int *p;
struct {int a, b, c;} *w;
```

variabili automatiche



Variabili Dinamiche in C

Il C prevede funzioni standard di allocazione e deallocazione per variabili dinamiche:

- **Allocazione:** `malloc`
- **Deallocazione:** `free`

`malloc` e `free` sono definite a livello di **sistema operativo**, mediante la libreria standard `<stdlib.h>` (da includere nei programmi che le usano).

Variabili Dinamiche

Allocazione di variabili dinamiche:

La memoria dinamica viene allocata con la funzione standard `malloc`. La **sintassi** da usare è:

```
punt = (tipodato *)malloc(sizeof(tipodato));
```

dove:

- `tipodato` è il tipo della variabile puntata
- `punt` è una variabile di tipo `tipodato *`
- `sizeof()` è una funzione standard che calcola il numero di byte che occupa il dato specificato come argomento
- è necessario convertire esplicitamente il tipo del valore ritornato (*casting*):
`(tipodato *) malloc(..)`

Significato:

La `malloc`

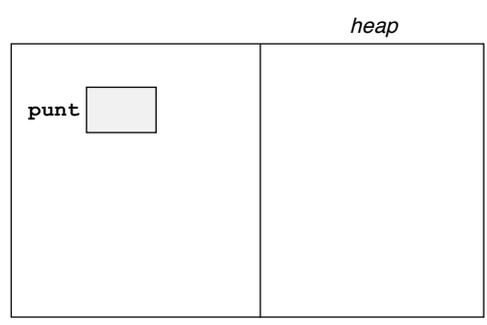
- provoca la creazione di una variabile dinamica nell'**heap** e
- restituisce l'**indirizzo** della variabile creata.

Variabili Dinamiche

Esempio:

```
#include <stdlib.h>
typedef int *tp;
➔ tp punt;
...
punt=(tp )malloc(sizeof(int));
```

`*punt=12;`



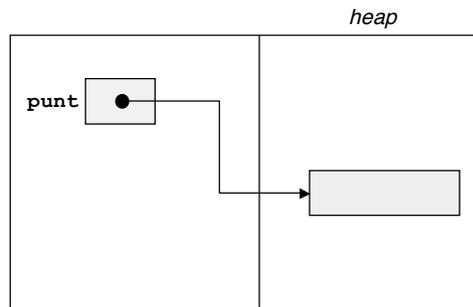
Variabili Dinamiche

Esempio:

```
#include <stdlib.h>
typedef int *tp;
tp punt;
...
```

➔ `punt=(tp)malloc(sizeof(int));`

`*punt=12;`



Fondamenti di Informatica L- A

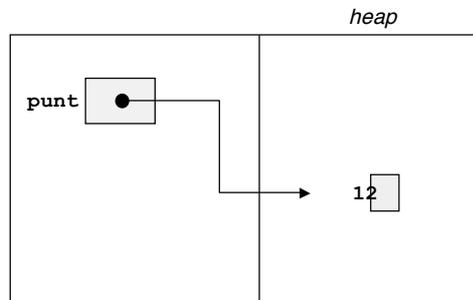
Variabili Dinamiche

Esempio:

```
#include <stdlib.h>
typedef int *tp;
tp punt;
...
```

`punt=(tp)malloc(sizeof(int));`

➔ `*punt=12;`



Fondamenti di Informatica L- A

Variabili dinamiche

3

Deallocazione:

Si rilascia la memoria allocata dinamicamente con:

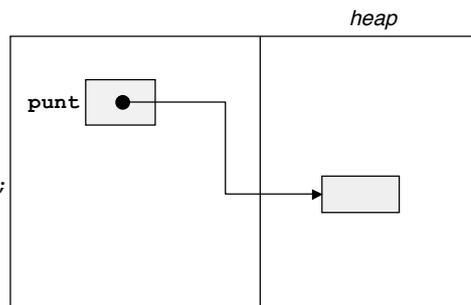
free (punt) ;

dove `punt` è l'indirizzo della variabile da deallocare.

→ Dopo questa operazione, la cella di memoria occupata da `*punt` viene liberata:
`*punt non esiste più.`

Esempio:

```
#include <stdlib.h>
typedef int *tp;
tp punt;
...
punt=(tp )malloc(sizeof(int));
*punt=12;
...<uso di punt>...
free (punt);
```

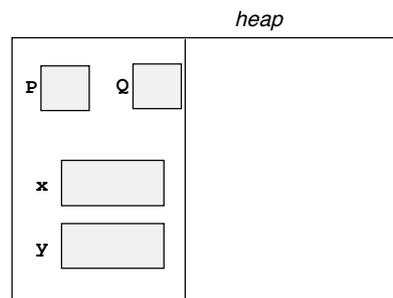


Fondamenti di Informatica L- A

Puntatori e variabili dinamiche: esempio

```
#include <stdio.h>

main()
{ int *P, *Q, x, y;
  x=5;
  y=14;
  P=(int *)malloc(sizeof(int));
  Q=(int *)malloc(sizeof(int));
  *P = 25;
  *Q = 30;
  *P = x;
  y = *Q;
  P = &x;
}
```

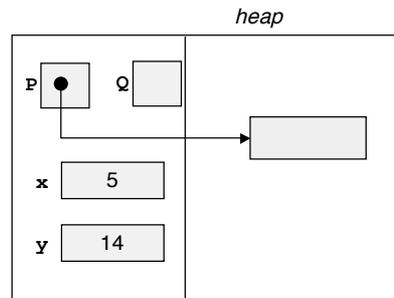


Fondamenti di Informatica L- A

Puntatori e variabili dinamiche: esempio

```
#include <stdio.h>

main()
{ int *P, *Q, x, y;
  x=5;
  y=14;
  P=(int *)malloc(sizeof(int));
  Q=(int *)malloc(sizeof(int));
  *P = 25;
  *Q = 30;
  *P = x;
  y = *Q;
  P = &x;
}
```

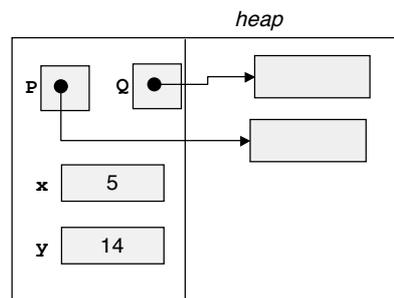


Fondamenti di Informatica L- A

Puntatori e variabili dinamiche: esempio

```
#include <stdio.h>

main()
{ int *P, *Q, x, y;
  x=5;
  y=14;
  P=(int *)malloc(sizeof(int));
  Q=(int *)malloc(sizeof(int));
  *P = 25;
  *Q = 30;
  *P = x;
  y = *Q;
  P = &x;
}
```

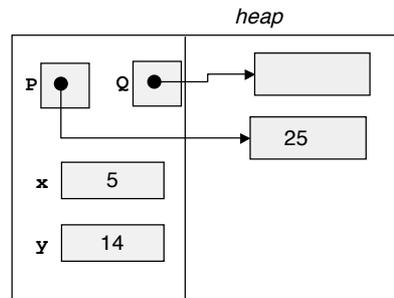


Fondamenti di Informatica L- A

Puntatori e variabili dinamiche: esempio

```
#include <stdio.h>

main()
{ int *P, *Q, x, y;
  x=5;
  y=14;
  P=(int *)malloc(sizeof(int));
  Q=(int *)malloc(sizeof(int));
  *P = 25;
  *Q = 30;
  *P = x;
  y = *Q;
  P = &x;
}
```

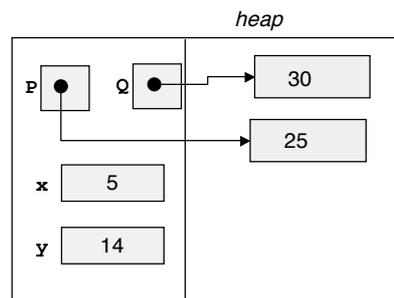


Fondamenti di Informatica L- A

Puntatori e variabili dinamiche: esempio

```
#include <stdio.h>

main()
{ int *P, *Q, x, y;
  x=5;
  y=14;
  P=(int *)malloc(sizeof(int));
  Q=(int *)malloc(sizeof(int));
  *P = 25;
  *Q = 30;
  *P = x;
  y = *Q;
  P = &x;
}
```

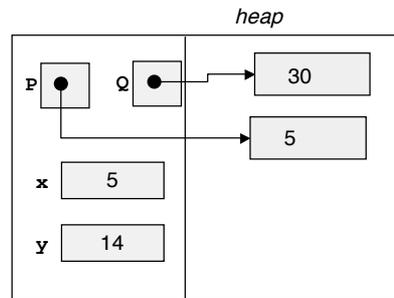


Fondamenti di Informatica L- A

Puntatori e variabili dinamiche: esempio

```
#include <stdio.h>

main()
{ int *P, *Q, x, y;
  x=5;
  y=14;
  P=(int *)malloc(sizeof(int));
  Q=(int *)malloc(sizeof(int));
  *P = 25;
  *Q = 30;
  *P = x;
  y = *Q;
  P = &x;
}
```

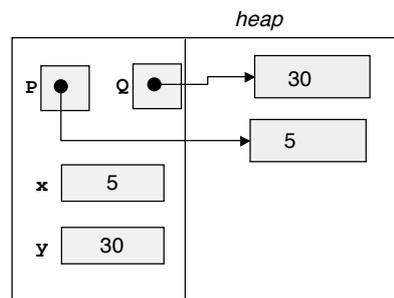


Fondamenti di Informatica L- A

Puntatori e variabili dinamiche: esempio

```
#include <stdio.h>

main()
{ int *P, *Q, x, y;
  x=5;
  y=14;
  P=(int *)malloc(sizeof(int));
  Q=(int *)malloc(sizeof(int));
  *P = 25;
  *Q = 30;
  *P = x;
  y = *Q;
  P = &x;
}
```

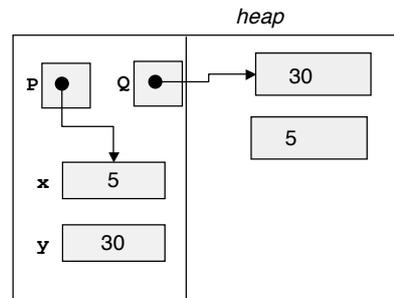


Fondamenti di Informatica L- A

Puntatori e variabili dinamiche: esempio

```
#include <stdio.h>

main()
{ int *P, *Q, x, y;
  x=5;
  y=14;
  P=(int *)malloc(sizeof(int));
  Q=(int *)malloc(sizeof(int));
  *P = 25;
  *Q = 30;
  *P = x;
  y = *Q;
  P = &x;
}
```



→ l'ultimo assegnamento ha come effetto collaterale la **perdita dell'indirizzo** di una variabile dinamica (quella precedentemente referenziata da `P`) che rimane allocata ma **non è più utilizzabile!**