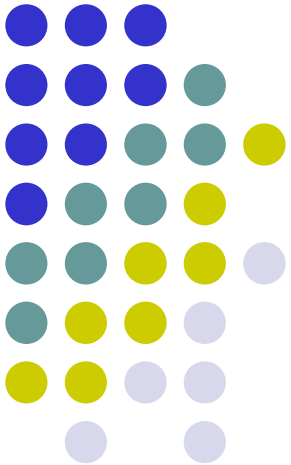
Laboratorio di Calcolatori 1

Corso di Laurea in Fisica
A.A. 2006/2007



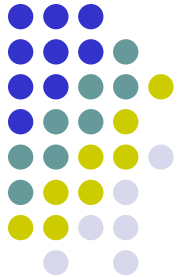
Dott.Davide Di Ruscio

Dipartimento di Informatica
Università degli Studi di L'Aquila

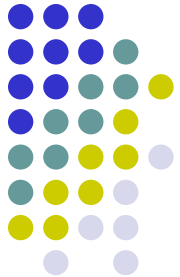
Lezione del 05/05/08

Nota

Questi lucidi sono tratti dal materiale distribuito dalla McGraw-Hill e basati su del materiale fornito dal Prof. Flammini Michele



Sommario (II parte)



Il Linguaggio C

- Caratteristiche generali
- Un linguaggio C semplificato ed esempi di semplici programmi
- Struttura di un programma C
- Direttive del pre-processore
- Parte dichiarativa:
  - tipi
  - definizioni di tipi
  - definizioni di variabili
- Parte esecutiiva
  - istruzione di assegnamento
  - istruzioni (funzioni) di input-output
  - istruzioni di selezione
  - istruzioni iterative
- Vettori mono e multidimensionali
- Funzioni e procedure
- File
- Allocazione dinamica di memoria
- Suddivisione dei programmi in piu' file e compilazione separata
- Algoritmi elementari
  - ricerca sequenziale e binaria
  - ordinamento di un vettore: per selezione, per inserimento, per fusione e a bolle
- Aspetti avanzati di programmazione
  - ricorsione
  - strutture dati dinamiche

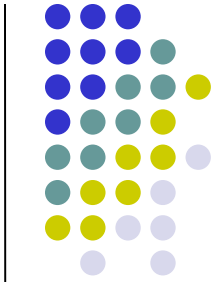
RIFERIMENTI

Ceri, Mandrioli, Sbattella

Informatica arte e mestiere

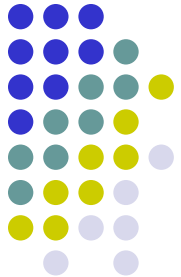
McGraw-Hill

Tipo di dati strutturati



- Un tipo strutturato non è caratterizzato da un valore semplice, ma da informazione aggregata in diverse componenti.
- Ad esempio un vettore o array consiste in una sequenza di elementi **consecutivi** e **omogenei**:
  - `int matricole_studenti[24];  /*variabile di tipo array*/`
  - `Matricole_studenti[4] = 123444; /*assegnamento all'elemento di indice 4*/`
- Il linguaggio C non possiede tipi predefiniti strutturati, tuttavia ha 4 costruttori di tipo:
  - **array**
  - **struct**
  - **pointer**
  - **union**
- Consentono di definire tipi strutturati anche complessi

Il costruttore struct



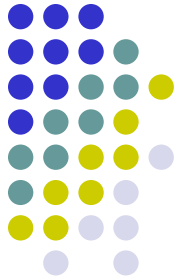
- Tipo impiegato: nome, cognome, codice fiscale, indirizzo, numero di telefono, eventuali stipendio, data di assunzione e via di seguito.

- Tipo famiglia: un certo insieme di persone, un patrimonio, costituito a sua volta da un insieme di beni, ognuno con un suo valore, un reddito annuo, spese varie, …

- Queste strutture informative sono eterogenee: l'array non si presta a questo tipo di aggregazione.

- Il costruttore di record (parola chiave **struct** in C) è la risposta a questo tipo di esigenze.

Esempi (1)

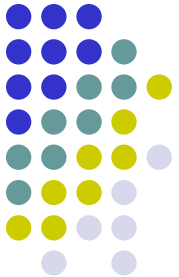


```
typedef char String[50];

typedef struct    {    int   Giorno;
                       int   Mese;
                       int   Anno;
                  }    Data;

typedef struct    {    String     Destinatario;
                       int        Importo;
                       Data       DataEmissione;
                  }    DescrizioneFatture;
```

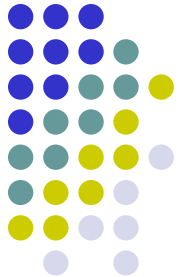
Esempi (2)



```
typedef enum    {On, Off}       AccType;

typedef struct  {   int          Canale;
                    AccType      Accensione;
                    double       CursoreLuminosita, CursoreColore,
                                 CursoreVolume;
                }   CanaliTV;
```

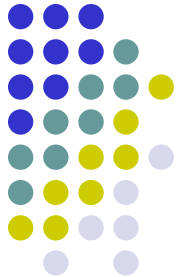
Esempi (3)



```
typedef enum {Dirigente, Impiegato, Operaio}
    CatType;

typedef struct  {       String      Nome;
                        String      Cognome;
                        int         Stipendio;
                        char        CodiceFiscale[16];
                        Data        DataAssunzione;
                        CatType     Categoria;
                }       Dipendenti;
```

Dichiarazione di variabili



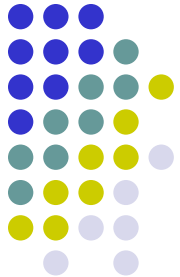
- La dichiarazione di variabili procede poi come al solito:

```
Dipendenti      Dip1, Dip2;
```

- Oppure è possibile definire il nuovo tipo in forma *anonima* e nello stesso tempo dichiarare le variabili:

```
struct  {       String          Nome;
                String          Cognome;
                int             Stipendio;
                char            CodiceFiscale[16];
                Data            DataAssunzione;
                CatType         Categoria;
        }       Dip1, Dip2;
```

Accesso alle componenti del record



- Per accedere alle singole componenti del record, si usa una notazione detta *dot notation*:

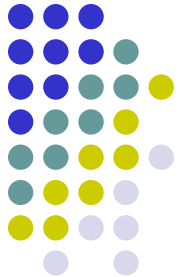
```
Dip1.Stipendio = Dip1.Stipendio + (Dip1.Stipendio*10) / 100;

Dip1.DataAssunzione.Giorno = 3;
Dip1.DataAssunzione.Mese = 1;
Dip1.DataAssunzione.Anno = 1993;
```

- Se si vuole sapere la prima lettere del cognome di Dip1 è A

```
if (Dip1.Cognome[0] == 'A') ...
```

Accesso alle componenti del record



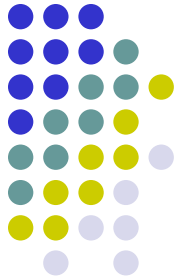
- Si supponga di avere dichiarato una variabile ArchivioFatture

```
DichiarazioneFatture        ArchivioFatture[1000];
```

- Per sapere se la fattura numero 500 è stata emessa entro il 2001 o no, e in caso affermativo, qual è il suo importo, si può scrivere il codice seguente:

```
if    (ArchivioFatture[500].DataEmissione.Anno <= 2000)
      printf("%d", ArchivioFatture[500].Importo);
else
      printf("La fattura in questione è stata emessa dopo il 2000\n");
```

Assegnamento tra record



- Come abbiamo visto, non è permesso scrivere un assegnamento tra array

```
Array2 = Array1;
```

- Invece:

```
Dip1 =  Dip2;
```

  è lecito e fa esattamente ciò che ci si aspetta: copia l'intera struttura Dip2 in Dip1, **comprese le sue componenti che sono costituite da array!**

- Non è invece possibile scrivere una condizione come la seguente:

```
if (Dip1 ==  Dip2)
```

- Il perché di questa stranezza risiede nel modo in cui in C sono realizzati gli array e potrà essere capito tra breve

Il costruttore puntatore

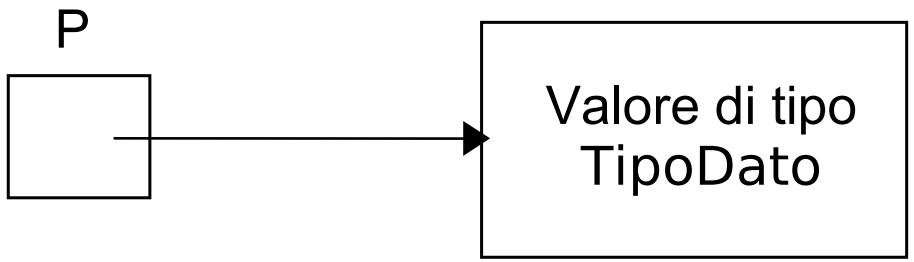
- Il puntatore indica l'indirizzo della variabile cui fa riferimento
- Dichiarazione di una variabile puntatore:

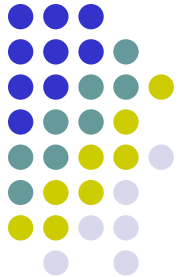
```
typedef     TipoDato     *TipoPuntatore;
```

definisce il tipo denominato TipoPuntatore come un puntatore a una cella contenente un valore di tipo TipoDato

- Quindi il valore di una variabile P di tipo TipoPuntatore è l'indirizzo di una variabile, il cui tipo è TipoDato
- **Dereferenziazione**: *P indica la cella di memoria il cui indirizzo è contenuto in P

```
typedef                 TipoDato     *TipoPuntatore;
TipoPuntatore  P;
TipoDato       x;
```





- L’operatore unario & significa “indirizzo di” ed è il duale dell’operatore ‘*’.

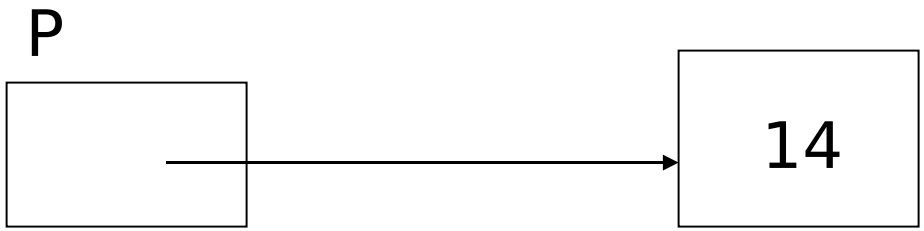
```
typedef     TipoDato    *TipoPuntatore;
TipoPuntatore   P, Q;
TipoDato    y, z;

P = &y;
Q = &z;
P = Q;
```

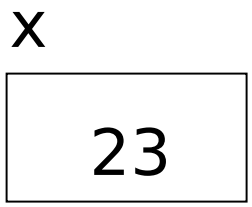
- y e z sono di tipo TipoDato mentre P e Q sono *puntatori* a variabili di tipo TipoDato.

Esempi (1)

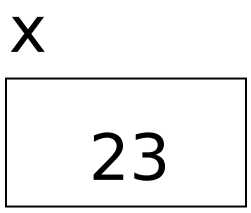
*P = 14;



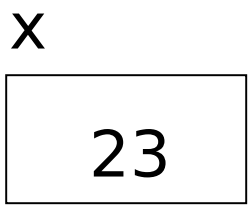
x = 23;



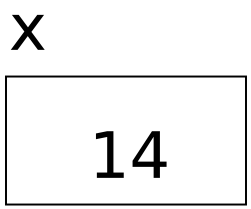
*P = x;



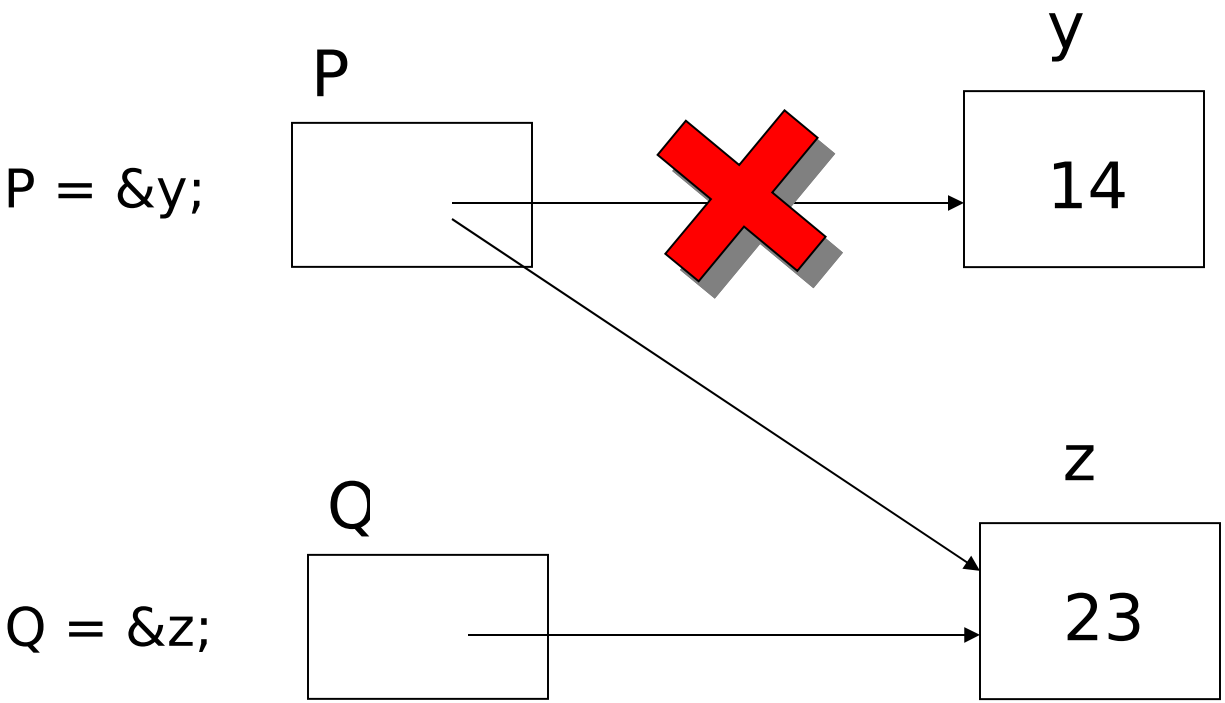
Esempi (2)



x = *P;



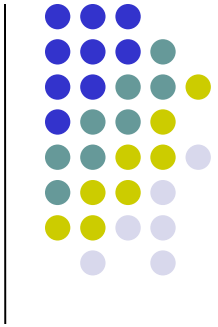
Esempi (3)



P = Q;

Attenzione: si noti la differenze tra le istruzioni P = Q e *P = *Q

Puntatori e tipi



```
typedef TipoDato *TipoPuntatore;
typedef AltroTipoDato *AltroTipoPuntatore;
TipoDato *Puntatore;
TipoDato **DoppioPuntatore;
TipoPuntatore P, Q;
AltroTipoPuntatore P1, Q1;
TipoDato x, y;
AltroTipoDato z, w;
```

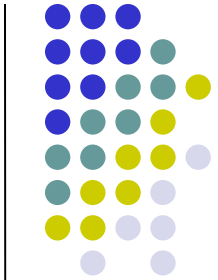
Istruzioni corrette:

```
Puntatore = &y;
DoppioPuntatore = &P;
Q1 = &z;
P = &x;
P = Q;
*P = *Q;
*Puntatore = x;
P = *DoppioPuntatore;
z = *P1;
Puntatore = P;
```

Istruzioni scorrette:

```
 P1 = P;                          (warning)
 w = *P;                          (error)
*DoppioPuntatore = y;             (warning)
 Puntatore = DoppioPuntatore;     (warning)
*P1 = *Q;                         (error)
```

Una tipica abbreviazione del C...



- Definiamo un record e dichiariamo una variabile puntatore:

```
typedef struct {        int     PrimoCampo;
                        char    SecondoCampo;
               } TipoDato;

TipoDato        x, *P;
P = &x;
```

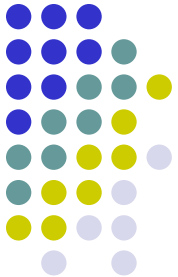
- Accesso al campo PrimoCampo di x, attraverso il puntatore P, usando la *dot notation*:

```
(*P).PrimoCampo = 12;        /* Inserisce 12 nel campo PrimoCampo di x */
```

- Esiste una sistassi abbreviata:

```
P–>PrimoCampo = 12;        /* Inserisce 12 nel campo PrimoCampo di x */
```

Riassumendo e completando

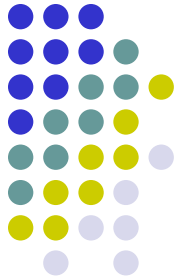


Operazioni applicabili a variabili puntatori:

- assegnamento dell'indirizzo di una variabile tramite l'operatore unario &;
- assegnamento del valore di un altro puntatore;
- assegnamento del valore speciale NULL. Se una variabile puntatore ha valore NULL, *P è indefinito: P non punta ad alcuna informazione significativa.

- l'operazione di dereferenziazione, indicata dall'operatore *;
- il confronto basato sulle relazioni ==, !=, >, <, <=, >=;
- operazioni aritmetiche
- l'assegnamento di indirizzi di memoria a seguito di operazioni di allocazione esplicita di memoria
  (gli ultimi due casi verranno trattati in seguito);

Attenzione ai “rischi” dei puntatori



- Effetti collaterali (*side effects*):

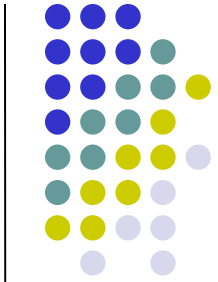
```
*P = 3;
*Q = 5;
P = Q;
                                    /* a questo punto *P = 5 */
*Q = 7;
```

- A questo punto *Q = 7, ma anche *P = 7

- Un assegnamento esplicito alla variabile puntata da Q determina un assegnamento nascosto alla variabile puntata da P.

- Caso particolare di *aliasing*, ovvero del fatto che uno stesso oggetto viene identificato in due modi diversi.

Array e puntatori



- L'operatore **sizeof** produce il numero di byte occupati da ciascun elemento di un array o da un array nel suo complesso.
- Se si usano quattro byte per la memorizzazione di un valore **int**:

```
int     a[5];
```

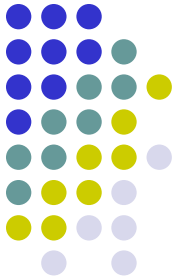
- Allora:

```
sizeof(a[2])
```

- Restituisce il valore 4 e:

```
sizeof(a)
```

- Restituisce il valore 20.
- Il nome di una variabile di tipo array viene considerato in C come l'indirizzo della prima parola di memoria che contiene il primo elemento della variabile di tipo array (lo 0-esimo …).
- Se ne deduce:
  - `a` "punta" a una parola di memoria esattamente come un puntatore;
  - `a` punta sempre al primo elemento della variabile di tipo array (è un puntatore "fisso" al quale non è possibile assegnare l'indirizzo di un'altra parola di memoria).

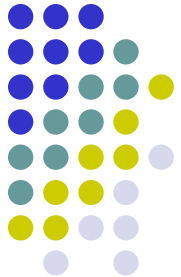


- *Il C consente di eseguire operazioni di somma e sottrazione su puntatori*.
- Se `p` e `a` forniscono l'indirizzo di memoria di elementi di tipo opportuno, `p+i` e `a+i` forniscono l'indirizzo di memoria dell'*i*-esimo elemento successivo di quel tipo
- Se `i` è una variabile intera:
  la notazione `a[i]` è equivalente a `*(a+i)`
- Analogamente, se `p` è dichiarato come puntatore a una variabile di tipo **`int`**:
  la notazione `p[i]` è equivalente a `*(p+i)`.
- Ne segue che:

  `p = a` è equivalente a `p = &a[0];`

  `p = a+1` è equivalente a `p = &a[1];`
- Mentre non sono ammessi assegnamenti ad `a` del tipo:

```
a = p;
a = a +1;
```



- Se `p` e `q` puntano a due diversi elementi di un array,
- `p–q` restituisce un valore intero pari al *numero di elementi* esistenti tra l'elemento cui punta `p` e l'elemento cui punta `q`.
- *Non la differenza tra il valore dei puntatori*.
- Supponendo che il risultato di `p–q` sia pari a `3` e supponendo che ogni elemento dell'array sia memorizzato in 4 byte, la differenza tra l'indirizzo contenuto in `p` e l'indirizzo contenuto in `q` darebbe 12.