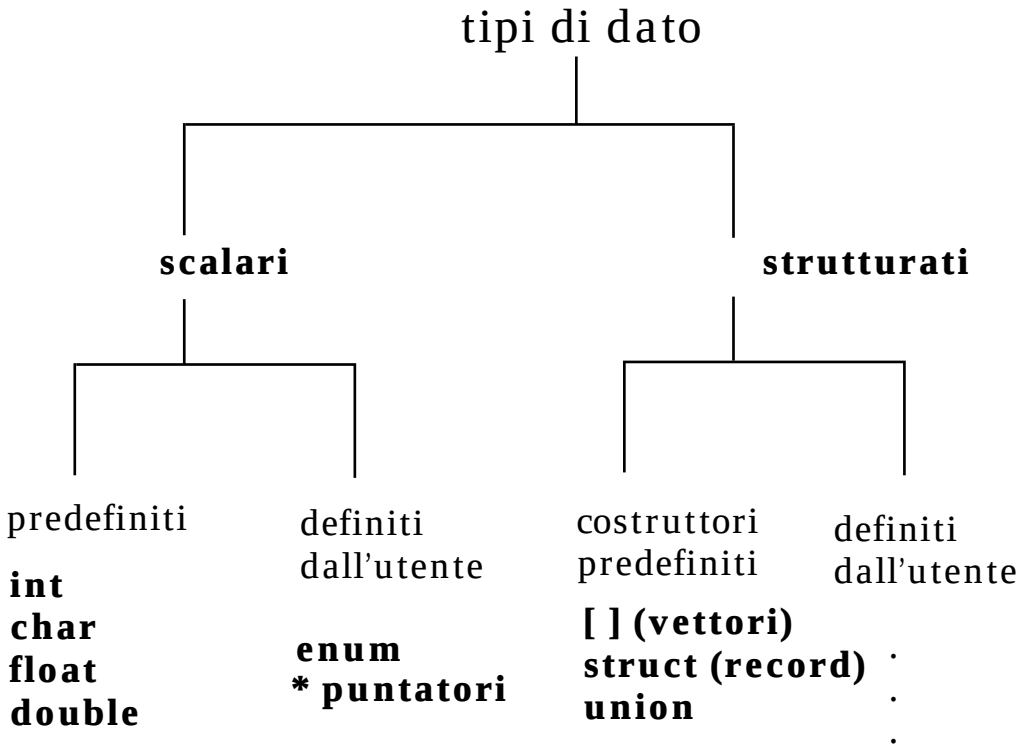
Tipi di Dato Strutturati



Ci occuperemo dei tipi strutturati:

I dati strutturati (o strutture di dati) sono ottenuti mediante composizione di altri dati (di tipo semplice, oppure strutturato).

Tipi strutturati in C che analizzeremo:

- **vettori** (o array)
- **record** (struct)

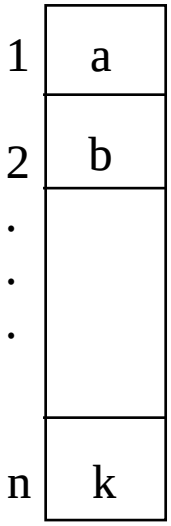
In generale, si definiscono mediante opportuni costruttori.

Vettori

Un vettore e‘ un insieme **ordinato** di elementi **tutti dello stesso tipo**.

Caratteristiche:

- omogeneità
- ordinamento ottenuto mediante dei valori interi (***indici***) che consentono di accedere ad ogni elemento della struttura.



Vettori in C

Nel linguaggio C per definire vettori, si usa il costruttore di tipo [ ].

Definizione di vettori:

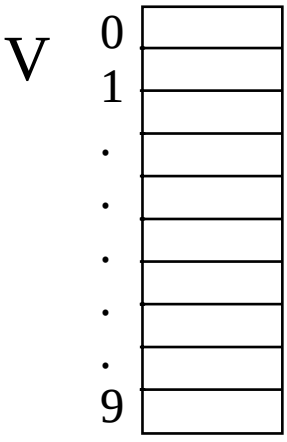
<id-tipo> <id-variable> [<**dimensione**>];

dove:

- **<id-tipo>** e` l'identificatore di tipo degli elementi componenti
- **<dimensione>** rappresenta il numero degli elementi componenti (e` una costante intera)
- **<id-variabile>** e` l'identificatore della variabile strutturata cosi` definita

Esempio:

int V[10]; /* vettore di dieci elementi interi */



Vettori

- La ***dimensione*** (numero di elementi del vettore) deve essere una costante intera, nota al momento della dichiarazione.

int N;
char V [N] ; **---> e‘ sbagliato!!!**

Accesso alle singole componenti:

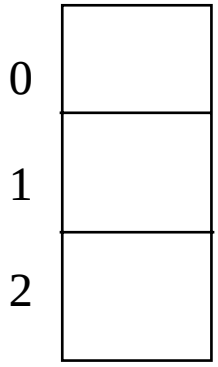
- e` possibile riferire una singola componente specificando l'indice corrispondente

V[i]

- se N e` la dimensione, il dominio di variazione degli indici e` [0,N-1].

Quindi:

int A[3];



A[0] è la prima componente
A[1] è la prima componente

A[2] è la prima componente

- Le singole componenti di un vettore possono essere manipolate coerentemente con il tipo ad esse associato:

int A[100];

› agli elementi di A e` possibile applicare tutti gli operatori definiti per gli interi.

Quindi:

```
 A[i] =n%i;
...
 scanf("%d", &A[i]);
...
```

Vettori

☞ L'indice deve essere int o char:

```
#include <stdio.h>
#define N 3

main()
{ int k;
int i=0;
int A[N];

for(k=0; k<=2; k++)
   {
   printf("Dammi elemento %d: ", k);
   scanf("%d", &A[k]);
   }
printf("Valore %d:%d\n",0,A[0]);
printf("Valore%d:%d\n",1,A[1]);
printf("Valore %d:%d\n",2,A[2]);
}
```

Dichiarazione di tipo per vettori

Dichiarazione di un identificatore di tipo:

typedef **<tipo-componente> <tipo-vettore>** [**<dim>**]

- <tipo-componente> e` l'identificatore di tipo di ogni singola componente
- <tipo-vettore> e` l'identificatore che si attribuisce al nuovo tipo
- <dim> e` il numero di elementi che costituiscono il vettore

Esempio:

```
typedef intVettori[30]; /* dich. di
                           tipo */
Vettori V1,V2;

V1[i]= ...;

/*i deve essere  IntegralType */
```

Vettori

Riassumendo:

Variabili di tipo vettore:

<tipo-componente> <nome>[<dim>];

Vettore come costruttore di tipo:

typedef <tipo-componente> <tipo-vettore> [<dim>];

Vincoli:

- <dim> e` una **costante intera**.
- <tipo-componente> e' un **qualsiasi** tipo, semplice o strutturato.

Uso:

- il vettore e' una sequenza di dimensione fissata di componenti dello stesso tipo <tipo_componente>.
- la singola componente i-esima di un vettore V e' individuata dall'indice i-esimo, secondo la notazione V[i].
- sui singoli elementi e` possibile operare secondo le modalita` previste dal tipo <tipo_componente>.

☞ per intere strutture di dati di tipo vettore **non e' possibile** l'assegnamento diretto.

Esempi:

- Definizione di due vettori di lunghezza 6:

```
int a[6], b[6];     /*  Indici da 0 a 5  */
```

- Definizione di un vettore di lunghezza 100 ed assegnamento di un valore alle sue componenti:

```
typedef int VETT_INT [100];

VETT_INT V;
int  i=1;

while (i<100)
    {
    V[i]=i*i;  /*gli elementi del
                vettore sono 1,4,9,..*/
     i++;
    }
```

- Definizione di un vettore di lunghezza 30 ed assegnamento di un valore alle sue componenti:

```
typedef int vettore [30];
vettore v;
int i;
...
for(i=0;v[i]>0 && i<30;i++)
   v[i]= 10;
i = 0;
while  (v[i] > 0  && i < 30)
  {  v[i] = 10;    i++; }
...
```

Esempi:

- Definizione di un vettore di lunghezza N (macro-sostituzione di N con il valore 2 prima della compilazione):

```
#define   N  2
main()
{
typedef int  Vet[N];
Vet     V;
int     i;
...
   for  ( i = 0; i < N; i++)
   V[i] = ...;

}
```

☞ La #define rende il programma più facilmente modificabile.

✐ Esercizio:

Leggere da input alcuni caratteri alfabetici maiuscoli (per ipotesi al massimo 10) e riscriverli in uscita evitando di ripetere caratteri gia' stampati.

Soluzione:

while <ci sono caratteri da leggere>
{
<leggi carattere>;
if <non gia' memorizzato>
<memorizzalo in una struttura dati>;
};
while <ci sono elementi della struttura dati>
<stampa elemento>;

Occorre una struttura dati in cui memorizzare (senza ripetizioni) gli elementi letti in ingresso.

char A[10];

Codifica:

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

main()
{
char A[10], c;
int i, j, inseriti, trovato;

inseriti=0;
printf("\n Dammi 10 caratteri: ");
for (i=0; (i<10); i++)
 {
 scanf("%c", &c);
 printf("letto %c\n", c);
 /* verifica unicita' */
 trovato=0;
 for(j=0;(j<inseriti)&&!trovato; j++)
  { if (c==A[j])
    trovato=1;
  }
 if (!trovato)
  { A[j]=c;
    inseriti++;
    printf("inserito %c\n", c);
   }
 trovato=0;
 }
 printf("Inseriti %d caratteri \n",
           inseriti);
}
```

Vettori

Lettura e stampa:

☞ Non e' possibile leggere/scrivere un intero vettore (a parte come vedremo ***stringhe***); occorre leggere/scrivere le sue componenti:

```
unsigned int i,F[25],frequenza[25];

for (i=0; i<25; i++)
      {     scanf("%d",&frequenza[i]);
            frequenza[i]=frequenza[i]+1;
      }           /*    legge a terminale le componenti del
                        vettore frequenza e le incrementa
                  */
```

Assegnamento tra vettori:

Anche se due variabili vettore sono dello ***stesso tipo***, non e` possibile l'assegnamento diretto:

```
F=frequenza;          /* NO */
```

occorre copiare componente per componente:

```
for (i=0; i<25; i++)     F[i]=frequenza[i];
```

Vettori multi-dimensionali

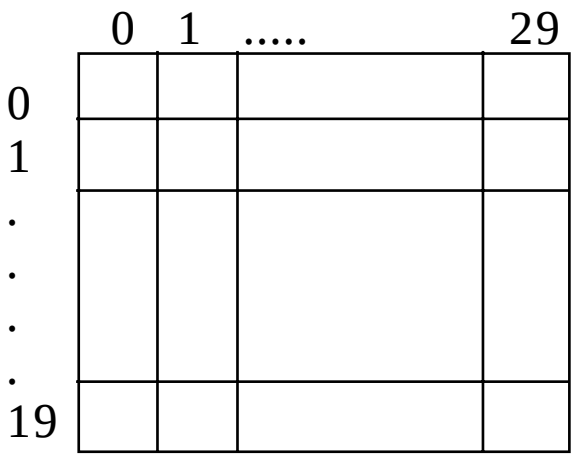
Non vi sono vincoli sul tipo degli elementi di un vettore:

- ☞ Gli elementi di un vettore possono essere a loro volta di tipo vettore (vettori multidimensionali o ***matrici***)

Definizione di vett. multidimensionali (matrici):

<id-tipo> <id-variable> [**dim$_1$**] [**dim$_2$**] ... [**dim$_n$**]

float M[20] [30];



☞ Memorizzazione **per righe**: M[0][0]
M[0][1]
....
M[0][29]
M[1][0]
....

Dichiarazione di tipi matrice:

typedef <id-tipo> **<id-tipo-vettore>** [dim_1] [dim_2] ... [dim_n]

Esempi:

```
typedef  float MatReali [20] [30];
MatReali Mat;
/*Mat e' un vettore di venti elementi,
ognuno dei quali e' un vettore di
trenta reali; quindi, matrice di 20x30 di
reali*/
```

Altro esempio:

```
typedef float VetReali[30];
typedef VetReali MatReali[20];
MatReali Mat;
```

Esempi:

```
typedef    char       tipo_vet[3];

typedef    unsigned char memoria[1024];

tipo_vet   V, V1 = {'a','b','c'};
                  /* inizializzazione - vettore di caratteri*/
memoria    m;
int        ContaColori[100][100][2];
```

V[2], denota il valore associato alla terza componente del vettore V

m[7], ottava componente del vettore m.

ContaColori[1], seconda componente (matrice)

ContaColori[1][1], vettore

m[ind+offset], componente del vettore m

Valore dell'espressione (ind+offset) compresa tra 0 e (dimensione - 1).

```
#define    M 3
main()
{     int              Mat[M*2][M*2];
      ...
}
```

Inizializzazione di vettori

E‘ possibile inizializzare un vettore in fase di definizione.

Esempio:

```
int v[10] = {1,2,3,4,5,6,7,8,9,10};
    /*  v[0] = 1;  v[1] = 2; ...  v[9] = 10;  */
```

Addirittura e‘ possibile fare:

```
    int v[ ] = {1,2,3,4,5,6,7,8,9,10};
```

☞ La dimensione e‘ determinata sulla base dell’inzializzazione.

Inizializzazione di una matrice:

```
    int matrix[4][4] = {{1,0,0,0},{0,1,0,0},{0,0,1,0},{0,0,0,1}};
```

Memorizzazione **per righe**:

matrix

	0	1	2	3
0	1	0	0	0
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1

```
    int matrix[ ][4] ={{1,0,0,0},{0,1,0,0},{0,0,1,0},{0,0,0,1}};
```

Esercizio:

Programma che esegue il prodotto (righe x colonne) di matrici quadrate NxN a valori interi:

$$C[i,j] = \sum_{(k=1..N)} A[i][k]*B[k][j]$$

```
#include <stdio.h>
#define N 2
main()
{
typedef int Matrici[N][N];
int Somma;
int i,j,k;
Matrici A,B,C;
/* inizializzazione di A e B */
  for (i=0; i<N; i++)
    for (j=0; j<N; j++)
       scanf("%d",&A[i][j]);
  for (i=0; i<N; i++)
    for (j=0; j<N; j++)
       scanf("%d",&B[i][j]);
/* prodotto matriciale */
  for (i=0; i<N; i++)
    for (j=0; j<N; j++)
          {  Somma=0;
             for (k=0; k<N; k++)
              Somma=Somma+A[i][k]*B[k][j];
C[i][j]=Somma;}
/* stampa */
  for (i=0; i<N; i++)
    for (j=0; j<N; j++)
       printf("%d",C[i][j]);
}
```

✍ **Esercizio:**

Dati n valori interi forniti in ordine qualunque, stampare in uscita l'elenco dei valori dati in ordine crescente.

☞ E' necessario mantenere in memoria tutti i valori dati per poter effettuare i confronti necessari.

› Utilizziamo i vettori

Ordinamento di un vettore:

Esistono vari procedimenti risolutivi (v. algoritmi di ordinamento).

Metodo dei Massimi successivi:

Dato un vettore: int V[dim];

1. eleggi un elemento come massimo temporaneo (V[max])
2. confronta il valore di V[max] con tutti gli altri elementi del vettore (V[i]):
   - se V[i]>V[max] , max=i
3. quando hai finito i confronti (se max!=dim-1) scambia V[max] con V[dim-1] › il massimo ottenuto dalla scansione va in ultima posizione.
4. riduci il vettore di un elemento (dim=dim-1) e, se dim>1, torna a 1.

Codifica:

- Primo livello di specifica:

```
#include <stdio.h>
#define dim 10

main()
{
int V[dim], i,j, max, tmp, quanti;

/* lettura dei dati */

/*ordinamento */

for(i=0; i<dim; i++)
  {quanti=dim-i;

  /*ciclo di scansione
  del vettore i-simo
  (di dimensione=quanti) */
   }
  }

/*stampa del vettore V ordinato */
}
```

Codifica:

```
#include <stdio.h>
#define dim 10

main()
{
int V[dim], i,j, max, tmp, quanti;

/* lettura dei dati */
for (i=0; i<dim; i++)
  { printf("valore n. %d: ",i);
    scanf("%d", &V[i]);
   }


/*ordinamento */

for(i=0; i<dim; i++)
  {quanti=dim-i;
  max=quanti-1;
  for( j=0; j<quanti; j++)
   { if (V[j]>V[max])
        max=j;
   }
  if (max<quanti-1)
   { /*scambio */
   tmp=V[quanti-1];
   V[quanti-1]=V[max];
   V[max]=tmp;
   }
  }
/*stampa */
for(i=0; i<dim; i++)
  printf("Valore di V[%d]=%d\n", i, V[i]);
}
```

Vettori di Caratteri: le Stringhe

Una *stringa* e' un vettore di caratteri, manipolato e gestito secondo alcune *convenzioni*:

- rappresentazione come ***vettori di caratteri***.
- Ogni stringa e` terminata dal ***carattere nullo*** '\0' (valore decimale zero).

☞ E` responsabilita` del programmatore gestire tale struttura in modo consistente con il concetto di stringa (ad esempio, terminatore '\0').

Stringhe costanti:

Sono sequenze di caratteri racchiuse tra doppi apici:

```
"Esempio di stringa"
""                                  /* stringa vuota*/
"Carattere  speciale\" "


"p"             /* stringa - lunghezza 2      */
'p'             /* carattere                  */
```

Esempio:

Programma che calcola la lunghezza di una stringa.

```
#include <stdio.h>

/* lunghezza di una stringa */

main()
{
 char   str[81]; /* stringa di al max.
                    80 caratteri */
 int    i;

printf("\nImmettere una stringa: \t");
scanf("%s",&str); /* %s formato
                     stringa */

for (i=0; str[i]!='\0'; i++);

printf("\nLunghezza: \t %d\n",i);

}
```

Vengono acquisiti i caratteri in ingresso fino al primo carattere di spaziatura (bianco, newline, tabulazione, salti pagina).

Esempi:

```
char  string[81]; /*  max 80 caratteri
                      significativi
                      (+ il terminatore'\0')*/
```

Inizializzazione di una variabile stringa:

```
char  text[6] = {'P','l','u','t','o','\0'};
char  text[ ] = {'P','l','u','t','o','\0'};
char  text[ ] = "Pluto";
```

text	P	l	u	t	o	\0

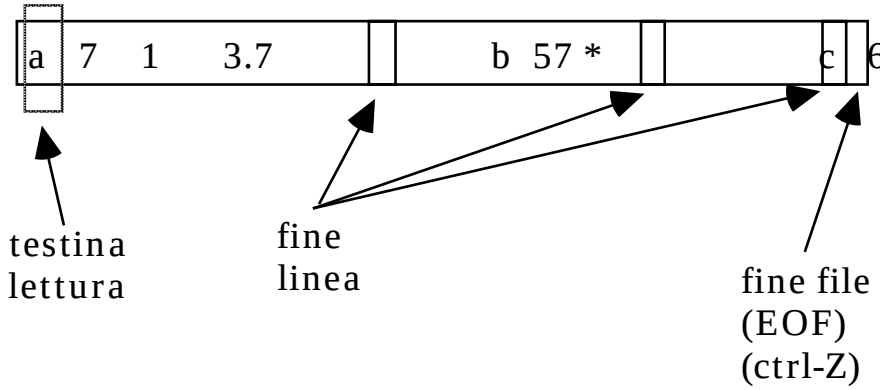
Esempio:

Programma che concatena due stringhe date.

```
#include <stdio.h>

/* concatenamento di due stringhe */

main()
{
  char   s1[81], s2[81];
  int    l,i;

printf("\nPrima stringa: \t");
scanf("%s",&s1);
printf("\nSeconda stringa: \t");
scanf("%s",&s2);

for (l=0; s1[l]!='\0'; l++);

for (i=0; s2[i]!='\0' && i+l<79; i++)
                    s1[i+l]=s2[i];

s1[i+l]='\0';   /* fine stringa */

printf("\nStringa:\t%s\n",s1);

}
```

Input/Output a caratteri

I dispositivi di ingresso ed uscita sono visti come file di caratteri (file ***testo***) terminati da una marca speciale di ***end-of-file*** (EOF) ed organizzati eventualmente su piu‘ linee (***newline*** ’\n’).



Esistono funzioni (libreria standard) per leggere e scrivere **singoli caratteri:**

- **getchar**: restituisce il prossimo carattere disponibile in ingresso;
- **putchar**: stampa un carattere su output.

<variabile> = getchar**()**

putchar**(<variabile>)**

---

dove <variabile> e‘ di tipo intero (o carattere).

putchar/getchar

getchar:

int **getchar**(void);

- non richiede argomenti e restituisce come risultato il carattere letto **convertito in int** (o **EOF** in caso di end-of-file o errore).

putchar:

int **putchar**(int c);

- richiede un argomento (il carattere da scrivere), e restituisce come risultato il carattere scritto (o **EOF** in caso di errore).

Esempio:

Programma che copia da input (la tastiera) su output (il video):

```
#include <stdio.h>
main()
{
  int c;
  c = getchar();
  while (c != EOF)
      {  putchar(c);
         c = getchar();
      }
}
```

☞ L'uso di una variabile intera anziche' un carattere e' dovuto al fatto che il valore speciale EOF e' spesso negativo (-1) e l'uso di un char e' corretto solo se nella versione di C utilizzata il tipo **char** e' ***signed***. Altrimenti il test c!=EOF e' sempre vero (si avrebbe un ciclo infinito).

☞ La funzione getchar **comincia** a restituire caratteri solo quando è stato battuto il tasto di invio (***input "bufferizzato"***)

Versione piu‘ sintetica:

```
#include <stdio.h>
main()
{
  int c;
  while ((c = getchar()) != EOF)
                    putchar(c);
}
```

Esempio:

Ricopiatura dell'input sull'output convertendo minuscole in maiuscole.

```
#include <stdio.h>
#define scostamento 'a'-'A'

main()
{
  int c;
  while ((c = getchar()) != EOF)
    if (c>='a' && c<='z')
          putchar(c-(scostamento));
    else putchar(c);
}
```

☞ **putchar(c-(scostamento))** viene espanso in:

putchar(c-('a'-'A'))

Input/Output a linee di caratteri

Poiche‘ scanf legge stringhe fino al primo carattere di spaziatura non e‘ adatta a leggere intere linee (che possono contenere spazi bianchi, caratteri di tabulazione, etc.).

Per fare I/O di linee:

- gets
- puts

gets:

e` una funzione standard, che legge una intera riga da input, fino al primo carattere di fine linea ('\n', ***newline***) e l'assegna ad una stringa.

```
char str[80];
gets(str);
```

- assegna alla stringa str i caratteri letti
- e` una funzione; ritorna come risultato
    - indirizzo del primo carattere (se OK)
    - NULL, in caso di fine file o errore

- Il carattere '\n' viene sostituito (nella stringa di destinazione) da '\0'.

puts:

E‘funzione standard che scrive una stringa sull’output aggiungendo un carattere di fine linea (’\n’, ***newline***).

```
char str[80];
puts(str);
```

- in caso di errore restituisce **EOF**

Esempio:

Concatenamento di stringhe.

```
#include <stdio.h>

/* concatenamento di due stringhe */

main()
{
 char   s1[81], s2[81];
 int    l,i;

printf("\nPrima stringa: \t");
gets(s1);
printf("\nSeconda stringa: \t");
gets(s2);

for (l=0; s1[l]!='\0'; l++);

for (i=0; s2[i]!='\0' && i+l<79; i++)
                    s1[i+l]=s2[i];

s1[i+l]='\0';   /* fine stringa */

printf("\nStringa:\t%s\n",s1);

}
```

Esempi:

- Ricopia l'input nell'output (a linee):

```
#include <stdio.h>
main()
{
char s[81];

while (gets(s))
  puts(s);
}
```

- Uso di puts e putchar:

```
putchar('A');
putchar('B');
puts("C");
putchar('D');
```

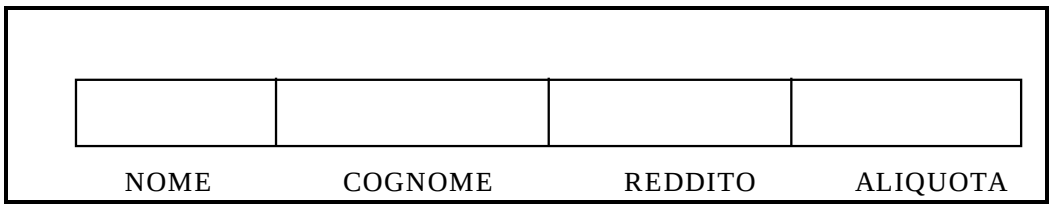
Effetto:

ABC
D

Tipi strutturati: Il Record

Un record e‘ un **insieme finito** di elementi **non omogeneo**:

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



Formalmente:

Il record e` un tipo strutturato costruito con **prodotto cartesiano**:

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

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

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

$$(a_{c1}, a_{c2}, ..., a_{cn})$$

dove $a_{ci} \times a_{ci}$.

Ad esempio: Il numero complesso e` definito attraverso il prodotto cartesiano: R x R

Esempio:

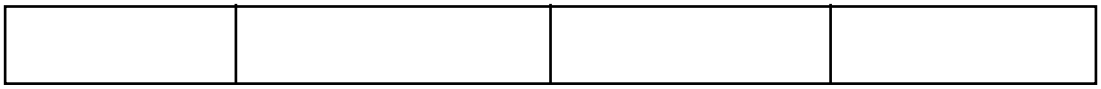
Memorizzare i dati di un certo numero di contribuenti (Nome, Cognome, Reddito, Aliquota).

Per ciascuna persona:

Nome, Cognome, Reddito, Aliquota

dove:

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



NOME COGNOME REDDITO ALIQUOTA

Il tipo struct in C

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 record:

struct { <lista definizioni campi> } <id-variabile>;

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

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

- **<id-variabile>** e` l'identificatore della variabile di tipo record cosi` definita.

Esempi:

```
struct  {    int anno;
             int mese;
             int giorno;
                        } data;
```

- **Dichiarazione di tipo strutturato record:**

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

```
typedef struct   {    int anno;
                      int mese;
                      int giorno;
                      }tipodata;

tipodata data, nuova_data;
unsigned int    anno;
```

☞ Gli identificatori di campo di un record devono essere distinti tra loro, ma non necessariamente diversi da altri identificatori (ad es., anno).

Record: Accesso ai Campi

Per indicare i campi di un record, in C si usa una 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.

- Gli unici operatori previsti per dati di tipo record sono:
  - operatore di assegnamento (**=**): e` possibile l'**assegnamento** diretto tra record di tipo equivalente.
  - operatori di uguaglianza/disuguaglianza (==, !=)

Ad esempio:

```
tipodata ieri, oggi;
int durata;

scanf("%d%d%d",&ieri.giorno,&ieri.mese,
&ieri.anno);
oggi=ieri;
oggi.giorno=oggi.giorno%durata+1;
```

Record

Riassumendo:

Sintassi:

```
[typedef] struct {
            <tipo_1> <nome_campo_1>;
            <tipo_2> <nome_campo_2>;
                ...
            <tipo_N> <nome_campo_N>;
                } <nome>;
```

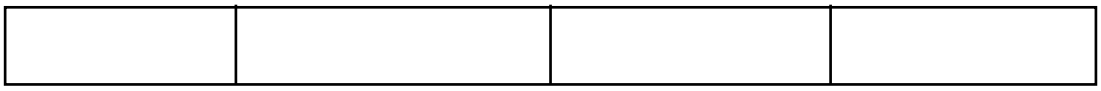
Vincoli:

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

Uso:

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

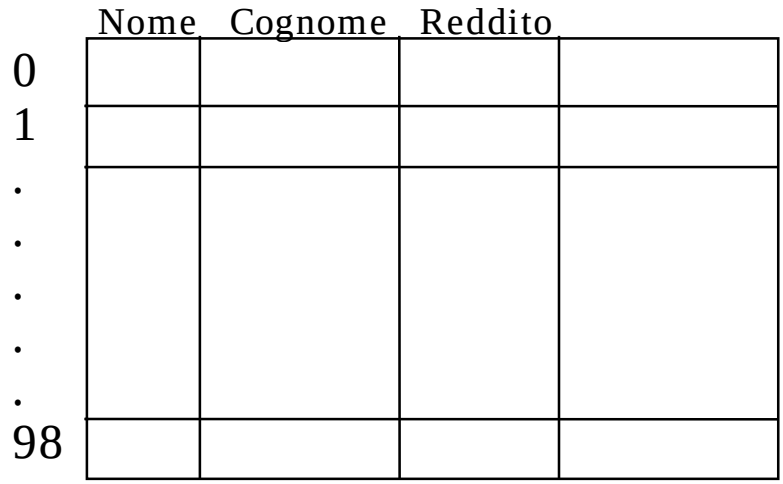
Esempi:



NOME COGNOME REDDITO ALIQUOTA

```
typedef struct {char Nome[20];
                char Cognome[20];
                int  Reddito;
                int  Aliquota;} Persone;

Persone  P[99];
```

☞ P e' un vettore di 99 elementi, ciascuno dei quali e' di tipo Persone ❯ vettore di record (struttura ***tabellare***).



P[0] --> dato strutturato: record

P[0].Nome --> nome della prima persona nella tabella.

E' possibile eseguire una inizializzazione:

```
struct { char   Nome[20];
         char Cognome[20];
         int        Reddito;
         int        Aliquota;
         } p = ("Mario","Rossi", 17000,10);
```

- **Esercizio:**

```
/* programma che legge le coordinate di
un punto in un piano e lo modifica a
seconda dell'operazione richiesta*/
#include <stdio.h>

main()
{
typedef struct{float  x,y;}punto;

punto P;
unsigned int op;
float  Dx, Dy;

/* si leggono le coordinate da input i
dati e le si memorizza in P */

  printf("ascissa? ");
  scanf("%f",&P.x);
  printf("ordinata? ");
  scanf("%f",&P.y);

/* lettura dell'operazione richiesta:
  0: termina
  1: proietta sull'asse x
  2: proietta sull'asse y
  3: trasla di Dx, Dy */
printf("%s\n","operazione(0,1,2,3)?");
scanf("%d",&op);
```

```
switch (op)
  {case 1:P.y= 0;break;
   case 2:P.x= 0; break;
   case 3:printf("%s","Traslazione?”);
          scanf("%f%f",&Dx,&Dy);
             P.x=P.x+Dx;
             P.y=P.y+Dy;
          break;
  default: ;
    }
printf("%s\n","nuove coordinate ");
printf("%f%s%f\n",P.x,"  ",P.y);
}
```

Esercizio: gestione di una rubrica telefonica

Scrivere un programma che legga dallo standard input i dati relativi a un archivio di numeri telefonici. Ogni elemento dell'archivio è caratterizzato dalle seguenti informazioni:

- nome
- cognome
- numero_telefono

Una volta inizializzato l'archivio, il programma deve essere in grado di attuare varie richieste dell'utente:

- *stampa*: visualizzazione sullo standard output del contenuto dell'archivio
- *ricerca*: dati in ingresso nome e cognome di una persona presente in archivio, si richiede la visualizzazione del numero telefonico relativo alla persona data;
- *aggiornamento*: modifica dei dati di un indirizzo presente nell'archivio. Vengono forniti in ingresso nome e cognome e nuovo numero di telefono della persona presente in archivio ed il programma assegna il nuovo numero alla persona
- *inserimento*: inserimento di un nuovo record nell'archivio, dati nome, cognome e numero di telefono della persona da inserire.
- *cancellazione*: eliminazione di un elemento dall'archivio, dati il nome e il cognome della persona da cancellare.
- *uscita*: termine del programma.

L'interazione tra l'utente e il programma avviene in modo ciclico: l'utente può sottoporre una richiesta ad ogni ciclo ed il programma sfruttando un meccanismo di selezione (per esempio `switch`) reagisce nel modo richiesto. L'esecuzione del programma termina quando l'utente richiede l'uscita.

Soluzione

```
#include <stdio.h>
#define N 100

typedef struct
{
      char nome[20], cognome[30];
      char tel[16];
} elemento;
main()
{
   int i,j,fine,scelta,stop,inseriti=0;
   rubrica R;
   char nome[20], cognome[30], tel[16];

   for (i=0,fine=0;i<N && !fine; i++)
   {
      printf("\nInserire nome:  ");
      gets(R[i].nome);
      printf("\nInserire cognome:  ");
      gets(R[i].cognome);
      printf("\nInserire numero:  ");
      gets(R[i].tel);
      printf("\n Ancora? (SI=0,
         NO=1)");");
      scanf("%d",&fine);
      fflush(stdin);
}
inseriti=i;

fine=0;
do
{
   printf("Scegli l'operazione:\n");
   printf("\t1\tStampa\n");
   printf("\t2\tRicerca\n");
```

```
printf("\t3\tAggiornamento\n");
printf("\t4\tInserimento\n");
printf("\t5\tCancellazione\n");
printf("\t6\tUscita\n");
printf("\n\nScelta:  ");
scanf("%d",&scelta);
fflush(stdin);
switch(scelta)
{
case 1:
     printf("\n\n");
     for(i=0;i<inseriti;i++)
     printf("%s\t%s\t%s\n",R[i].nome,
          R[i].cognome, R[i].tel);
     break;
case 2:
     printf("\nInserire nome:   ");
     gets(nome);
     printf("Inserire cognome:  ");
     gets(cognome);
     for(i=0, stop=0;i<inseriti &&
        !stop;i++)
        if (!strcmp(nome,R[i].nome &&
         !strcmp(cognome,R[i].cognome))
           stop=1;
```

```
        if (stop)
        {
           i--;
           printf("%s\t%s\t%s\n",
        R[i].nome,R[i].cognome,R[i].tel);
        }
        else printf("%s\t%s\t non
           trovato\n",nome, cognome);
        break;
case 3:
        printf("\nInserire nome: ");
        gets(nome);
        printf("Inserire cognome:  ");
        gets(cognome);
        for (i=0,stop=0;i<inseriti
           && !stop;i++)
           if (!strcmp(nome,R[i].nome)&&
            !strcmp(cognome,R[i].cognome))
              stop=1;
        if (stop)
        {
           i--;
           printf("Trovato %s %s: numero
              attuale %s",R[i].nome,
              R[i].cognome,
              R[i].tel);
           printf("\nNumero? ");
           gets(R[i].tel);
           printf("%s\t%s\t%s\n",
              R[i].nome,R[i].cognome,
              R[i].tel);
        }
        else
           printf("%s\t%s\t non
              trovato\n",nome,cognome);
           break;
```

```
case 4:
   printf("\nInserire nome:  ");
   gets(R[inseriti].nome);
   printf("\nInserire cognome:  ");
   gets(R[inseriti].cognome);
   printf("\nInserire numero:  ");
   gets(R[inseriti].tel);
   inseriti++;
   break;
case 4:
   printf("\nInserire nome:  ");
   gets(nome);
   printf("\nInserire cognome:  ");
   gets(cognome);
   printf("\nInserire numero:  ");
   for (i=0,stop=0;i<inseriti
        && !stop;i++)
      if (!strcmp(nome,R[i].nome)&&
          !strcmp(cognome,R[i].cognome))
        stop=1;
   if (stop)
   {
      i--;
      printf("Cancellazione di %s %s"
        ,R[i].nome,R[i].cognome);
      for(j=i;j<inseriti-1;j++)
        R[j]=R[j+1];
      inseriti--;
   }
   else
        printf("%s\t%s\t non
           trovato\n",nome,cognome);
   break;
   case 6: fine=1;break;
   default: printf("Scelta
     sbagliata\n");
```

```
      }; /* fine switch */}
    } while (!fine); /* fine do */
} /* fine main */
```

✐ Esercizio Proposto:

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

- **nome**
- **eta**
- **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.

Il tipo puntatore

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

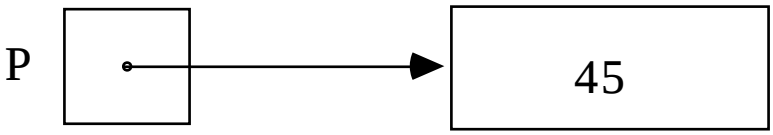
- Una variabile di tipo puntatore puo` avere come valore l’indirizzo di un’altra variabile (variabile *puntata*).

Definizione di una variabile puntatore:

<TipoElementoPuntato> *<NomePuntatore>;

- <TipoElementoPuntato> e` il tipo della variabile puntata
- **<**NomePuntatore> e` il nome della variabile di tipo puntatore
- il simbolo * e` un costruttore di tipo:

int *P;



P e` una variabile di tipo puntatore ad intero.

Dichiarazione di un tipo puntatore:

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

- <TipoElementoPuntato> e` il tipo della variabile puntata
- **<**NomePuntatore> e` il nome del tipo cosi` dichiarato.

Puntatori

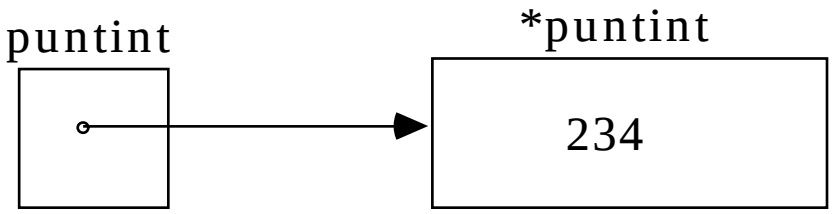
Accesso alla variabile puntata:

* e‘ un operatore di *dereferencing* (oltre che un costruttore di tipo): si applica a un indirizzo e restituisce il valore memorizzato a quell’indirizzo.

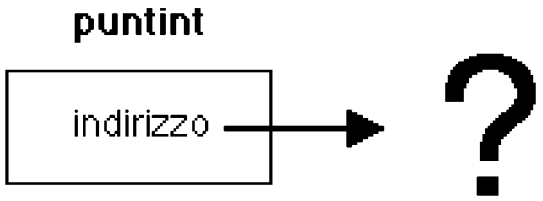
Quindi:

int *puntint;

punt_int	è il **puntatore** ❯ contiene l’indirizzo dell'elemento puntato
***punt_int**	è la **variabile puntata** ❯ contiene il valore (intero) dell'elemento puntato



Puntatori



Come si instaura il legame tra puntatore e variabile puntata ?

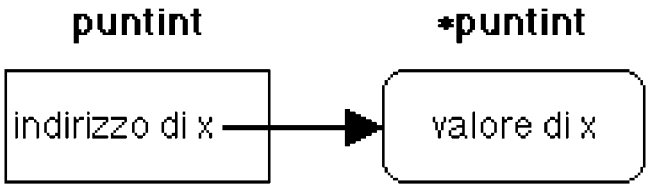
Operatore Indirizzo &:

E‘ un’operatore unario che restituisce l’indirizzo dell’operando:

int x;
&x e` l’indirizzo di x;

Quindi:
Se voglio che punt_int punti alla variabile x (cioe` che il valore di puntint sia l'indirizzo di x):

punt_int = **&**x;



- ☞ **&** si applica solo ad ***oggetti che esistono in memoria*** (quindi, gia` definiti).
- ☞ & non e` applicabile ad espressioni.

Puntatori

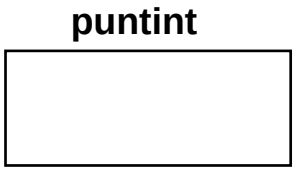
Per consentire controlli statici di tipo, un puntatore deve **puntare** (*referenziare*) dati di un **tipo prefissato**.

- La variabile punt_int conterra' solo riferimenti a celle che contengono dati interi.

*punt_int, ***variabile puntata*** (di tipo intero).

A qualunque variabile di tipo pointer puo' essere assegnato il valore **NULL**.

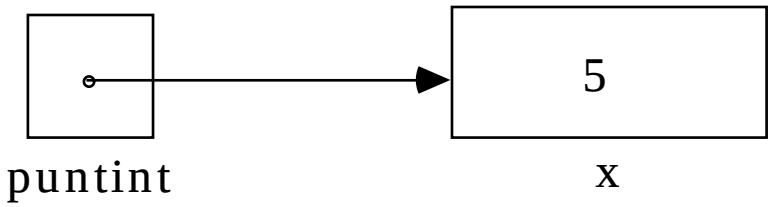
punt_int=**NULL**;



Esempio:

```
int  k,*punt_int, x;
x = 5;

punt_int = &x;  /* &x e‘ l’indirizzo
                   di x  */
```

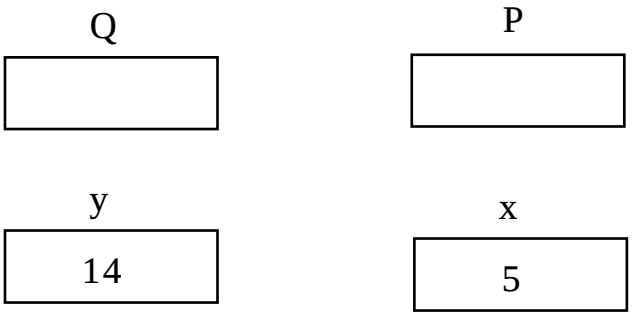
A questo punto *punt_int e‘ un ***alias*** per x:



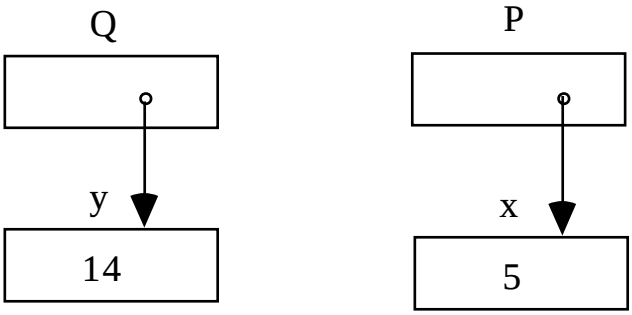

```
k = *punt_int;    /*  k = 5  */

*punt_int = k + 1;/*  x = 6  */
```

Esempio:

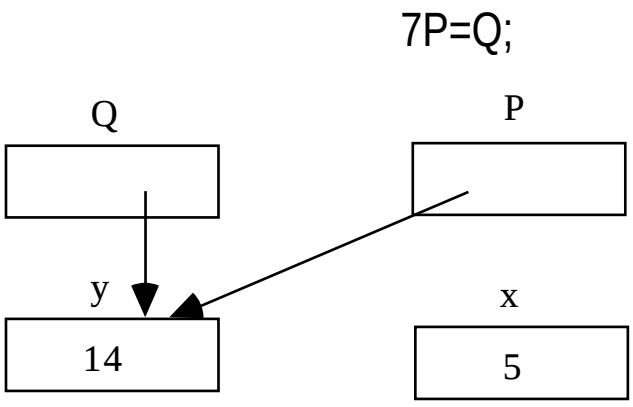
```
int  *P, *Q, x, y;

x = 5;
y = 14;
```



P = &x;
Q = &y;



☞ E‘ possibile l’**assegnamento diretto** tra puntatori:



Vettori & Puntatori

Vettori:

- in C, i vettori vengono allocati in memoria in **parole consecutive** (cioe` parole fisicamente adiacenti), la cui ***dimensione*** dipende dal tipo degli elementi del vettore.

- Il ***nome*** di una variabile di tipo vettore viene considerato dal C come *l'**indirizzo*** della prima parola di memoria occupata dal vettore.

Esempio:

int V[10];

V ➢ puntatore al primo elemento di V (ovvero V[0]).

☞ V è una costante:
- come nome equivale a **&**V[0]
- come tipo equivale a puntatore ad intero:

```
int *p, V[10];
p=V;
```

☞ Non sono ammesse operazioni del tipo:

```
V = p;     /* NO!  */
```

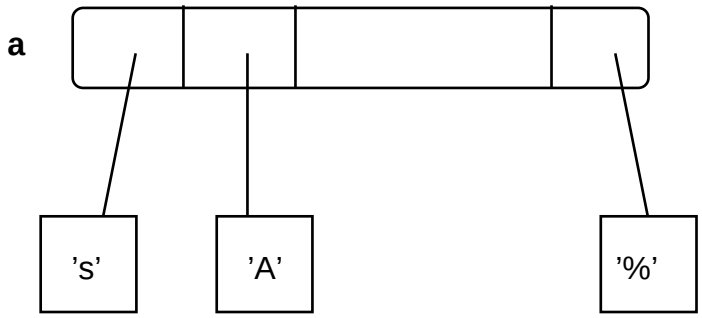
Vettori & Puntatori

☞ **[ ]** ha precedenza rispetto a *

Quindi:

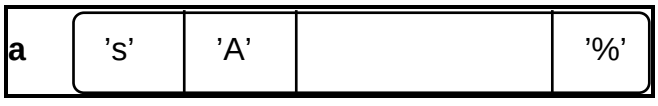
char *a[ ]; ===> equivale a **char** *(a[ ]);

› a è un **vettore di puntatori** a carattere.



☞ Per un puntatore ad un vettore di caratteri è necessario forzare la precedenza (con le parentesi)

char (* a) [ ];



Vettori & Puntatori

In C, ogni riferimento ad un elemento di un vettore è espanso come un *puntatore dereferenziato*:

V[0]	equivale a	*(V)
V[1]	equivale a	*(V + 1)
V[i]	equivale a	*(V+i)
V[expr]	equivale a	*(V + expr)

*(V+i) rappresenta l'(i+1)-esimo elemento di V

Aritmetica degli Indirizzi

Il C consente di eseguire operazioni di somma e sottrazione sui puntatori (a vettori).

Somma tra puntatori:

p = V	equivale a	p = &V[0]
p = V + 1	equivale a	p = &V[1]
p = V + expr	equivale a	p = &V[expr]

Differenza tra puntatori:

```
int  V[10],  *p, *q;
p = V;
q = &V[5];
```

L'espressione:

p - q

restituisce un valore intero pari al numero di elementi esistenti tra l'elemento a cui punta p (V[0]) e quello a cui punta q (V[5]).

Per l'esempio: -5

Esempio:

```
main ()
{
char a[] = "0123456789";/*a e‘ un
                          vettore di
                           caratteri */
int i = 5;

printf("%c%c%c%c\n",a[i],a[5],i[a],5[a]);
}
```

Stampa:

5 5 5 5

☞ Per il compilatore V[i] e i[V] sono lo stesso elemento, perche‘viene sempre eseguita la conversione:

V[i] ❯ *(V+i)

senza eseguire alcun controllo ne‘ su V ne‘ su i.

*(V + i) == *(i + V)

Puntatori a strutture:

```
typedef  struct   {   int  Campo_1,
                           Campo_2;
                      ... } TipoDato;

TipoDato  S, *P;
```

P = &S;

si accede alle componenti della struttura referenziata da P:

(*P).Campo_1 = ...;

Operatore ->:

L'operatore **->** consente di accedere ad un campo di una struttura referenziata da un puntatore.

P **->** Campo_1 = 75;

- Notazione piu' compatta.

Conversione esplicita di tipo: operatore di cast

int i;
float f;
double d;

In generale, sono automatiche le conversioni di tipo che non provocano perdita di informazione.

f + i (**int** convertito in **float**)

Espressioni che possono provocare perdita di informazioni non sono pero' illegali (***warning***):

```
i = f /f          /*  troncamento */

f = d        /* il double puo' essere arrotondato
                    o troncato */
```

In qualunque espressione e' possibile forzare una particolare conversione utilizzando l'***operatore di cast***:

(<tipo>) <espressione>

Esempio:

```
int   i;
long double  x;
double y;

i = (int) sqrt(384);
x = (long double) y*y;
```

Esempio:

```
#include <stdio.h>

main()
{
  int    *ip, dato=13;
  char   *p;

  ip=&dato;      /* ip punta a dato */
  p = (char *) ip;

  printf("Intero:\t%d\n",*ip);
  printf("Carattere:\t%c\n",*p);
}
```

```
#include <stdio.h>

main()
{
  int    *ip, dato=13;
  char   *p;
  ip=&dato;      /* ip punta a dato */
  printf("Intero:\t%d\n",*ip);
  printf("Carattere:\t%c\n",(char) *ip);
}
```