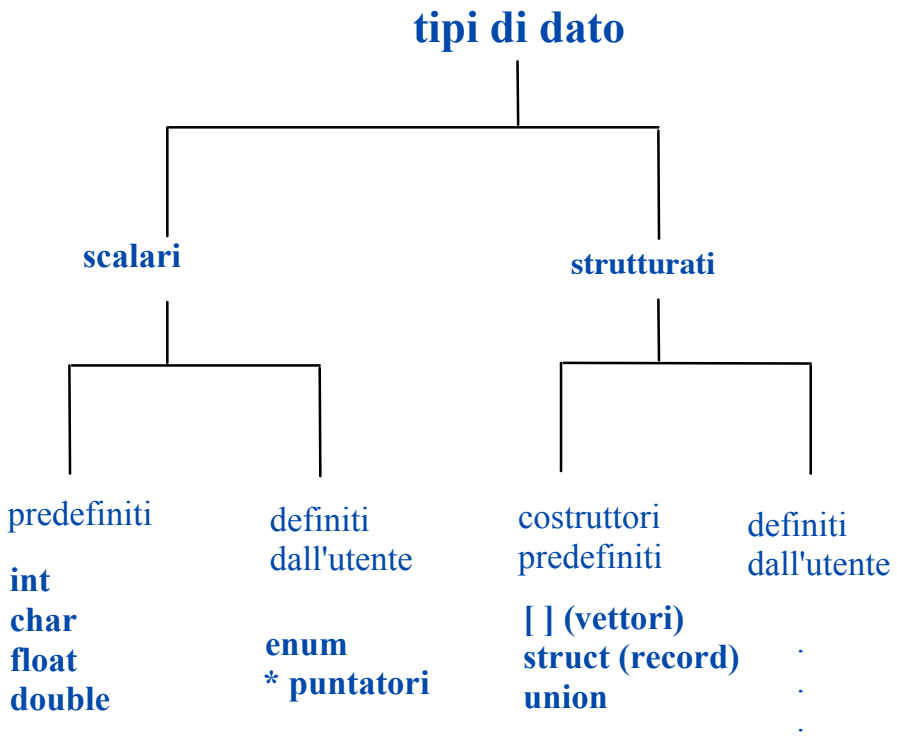
Linguaggio C:
Tipi Strutturati

Classificazione dei tipi di dato in C



Tipi di dato strutturati

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

Tipi strutturati in *C*:

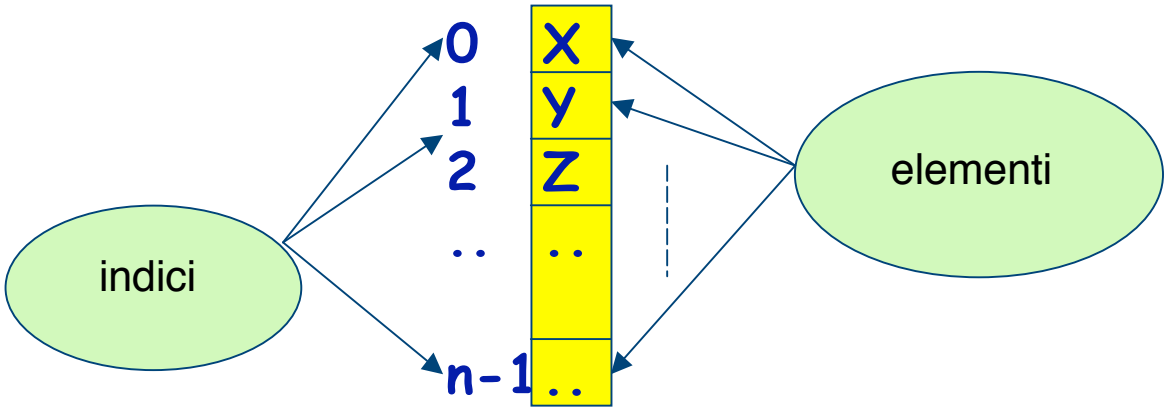
- **vettori** (o ***array***)
- **record** (***struct***)
- [record varianti (*union*)]

Vettori

Un **vettore** (o ***array***) e` un insieme ordinato di elementi tutti dello stesso tipo.

Caratteristiche del vettore:

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



Definizione di Vettori in C

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

Sintassi:

<id-tipo> <id-variabile> [<dimensione>];

dove:

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

Esempio:

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

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

```
int    N;
char   V [N] ;        /* def. sbagliata!!!*/
```

Vettori in C

```
int V[25]; /*def. di un vettore di 25 elementi interi*/
```

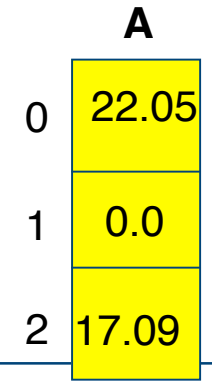
Indici:

- ad ogni elemento e` associato univocamente un ***indice***.
- e` possibile riferire una singola componente specificando l'indice `i` corrispondente, utilizzando la notazione `V[i]`.
- L'indice deve essere di tipo enumerabile (ad esempio: `int`, `char`).
- se **N** e` la **dimensione** del vettore (numero degli elementi), il dominio degli indici e` l'intervallo

[0,N-1]

Esempio:

```
float  A[3], i; /* A e` un vettore di 3 reali*/
A[0]=22.05; /*ass. di un valore al primo elemento */
A[2]=17.9; /*ass. di un valore al terzo elemento */
A[1]=0; /*ass. di un valore al secondo elemento */
```



Vettori in C

Operatori:

In C **non esistono operatori specifici** per i vettori.

Per operare sui vettori è necessario operare singolarmente sugli elementi componenti (coerentemente con il tipo ad essi associato).

Pertanto:

➔ **non e` possibile l'assegnamento diretto** tra vettori:

```
int V[10], W[10];
…
V=W;    /* e` scorretto! */
```

➔ **non e` possibile leggere (o scrivere)** un intero vettore (fanno eccezione, come vedremo, le stringhe); occorre leggere/scrivere le sue componenti;

Esempio: lettura di un vettore:

```
int V[100];
int  i;

for(i=0; i<100; i++)
   {            printf("valore %d-simo? ", i);
                scanf("%d", &V[i]);
   }
```

Gestione degli elementi di un vettore

Le singole componenti di un vettore possono essere elaborate con gli operatori del tipo ad esse associato.

Esempio: agli elementi di un vettore di interi e` possibile applicare tutti gli operatori definiti per gli interi:

```
int A[10], n=7 i=2;
A[i] =n%i; /* A[2]=1 */
A[9]=A[i++]+7; /* A[9]=1+7=8, i=3 */
scanf("%d",&A[i]); /*ass. ad A[3] il valore letto */
```

Vettore come costruttore di tipo

In C e` possibile utilizzare il costruttore **[ ]** per ***dichiarare*** tipi di dato non primitivi rappresentati da vettori.

Sintassi della dichiarazione:

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

dove:

- **<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 (deve essere una costante).

Esempio:

```
typedef int Vettore[30];
Vettore  V1,V2;+  /* V1 e V2 sono variabili di tipo
                     Vettore; ognuno rappresenta un vettore
                     di 30 elementi interi. */
```

➔ **V1 e V2 possono essere utilizzati come vettori di interi.**

Vettori in sintesi

Riassumendo:

- **Definizione di variabili di tipo vettore**:

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

- **Dichiarazione di tipi** non primitivi rappresentati da vettori:

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

Caratteristiche:

- `<dim>` e` una costante enumerabile.
- `<tipo-componente>` e' un qualsiasi tipo, semplice o strutturato.
- il vettore e' una sequenza di dimensione fissata `<dim>` 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]`.
- +L'intervallo di variazione degli indici e` `[0,..dim-1]`
- sui singoli elementi e` possibile operare secondo le modalita` previste dal tipo `<tipo_componente>`.

Inizializzazione di un vettore

Come attribuire un valore iniziale ad ogni elemento di un vettore?

Mediante un ciclo:

Per attribuire un valore iniziale agli elementi di un vettore, si puo` attuare con una sequenza di assegnamenti alle N componenti del vettore.

Esempio:

```
#define N 30
typedef int vettore [N];
vettore v;
int i;
...
for(i=0; i<N;i++)
v[i]=0;
```

`#define`: La **#define** e` una direttiva del *preprocessore C* che serve ad associare un identificatore (nell'esempio N) ad una costante (nell'esempio, 30). Rende il programma piu` facilmente modificabile.

Inizializzazione di un vettore

Come attribuire un valore iniziale ad ogni elemento di un vettore?

In alternativa al ciclo, e` possibile l'inizializzazione 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'inizializzazione.

Esempio

Problema:

Somma di due vettori: si realizzi un programma che, dati da standard input gli elementi di due vettori *A* e *B*, entrambi di 10 interi, calcoli e stampi gli elementi del vettore *C* (ancora di 10 interi), ottenuto come somma di *A* e *B*.

- Dichiariamo il nuovo tipo di dato vettint:

```
typedef int vettint[10];
```

- Definiamo i due vettori dati *A* e *B*, e il vettore *C* risultante dalla somma :

```
vettint A,B,C;
```

Esempio: soluzione

```
#include <stdio.h>
typedef int vettint[10];

main()
{  vettint A,B,C;
   int i;
   for(i=0; i<10; i++) /* lettura del vettore A */
   {   printf("valore di A[%d] ?, i);
       scanf("%d", &A[i]);
   }
   for(i=0; i<10; i++) /* lettura del vettore B */
   {   printf("valore di B[%d] ?, i);
       scanf("%d", &B[i]);
   }

   for(i=0; i<10; i++) /* calcolo del risultato */
       C[i]=A[i]+B[i];

   for(i=0; i<10; i++) /* stampa del risultato C */
       printf("C[%d]=%d\n",i, C[i]);
}
```

Esempio

Si consideri una scuola media costituita da 4 sezioni (da 'A' a 'D'), ognuna costituita da tre classi.

Dati gli iscritti ad ogni classe di ogni sezione, si vuole calcolare il numero medio di studenti per classe relativo ad ogni sezione.

Definiamo un nuovo tipo che rappresenta il livello:

```
typedef int livello[4]; /*un elemento per ogni sezione */
livello prime, seconde, terze;
```

NB: L'indice di un vettore deve essere di tipo ***enumerabile*** ➔ puo` essere un **char**:

```
typedef char sezione;
sezione s; /*indice per accedere ai vettori delle classi */
```

Soluzione

```
#include <stdio.h>
typedef int livello[4]; /*un elemento per ogni sezione */
typedef char sezione;
main()
{  livello prime, seconde, terze;
   sezione s;
   float  media=0;
   for(s='A'; s<='D'; s++)
   {    printf("Iscritti alla prima %c: ", s);
        scanf("%d", &prime[s-'A']);
   }
   for(s='A'; s<='D'; s++)
   {    printf("Iscritti alla seconda %c: ", s);
        scanf("%d", &seconde[s-'A']);
   }
   for(s='A'; s<='D'; s++)
   {    printf("Iscritti alla terza %c: ", s);
        scanf("%d", &terze[s-'A']);
   }
   for(s='A'; s<='D'; s++, media=0)
   {    media=prime[s-'A']+seconde[s-'A']+terze[s-'A'];
        printf("\nLa media degli iscritti per  classe nella sezione %c
                e`: %f\n", s, media/3);
   }
}
```

Esempio

Si leggano da input alcuni caratteri alfabetici maiuscoli (hp: al massimo, 10) e si riscrivano 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 variabile dove memorizzare (senza ripetizioni) gli elementi letti in ingresso: il ***vettore***

```
char A[10];
```

➔ il vettore A verra` riempito con al piu` 10 valori: definiamo la variabile inseriti, che rappresentera` la dimensione ***logica*** di A (cioe` , il numero di elementi significativi):

```
int inseriti=0;
```

➔ *la dimensione **fisica** del vettore e` 10, mentre la dimensione **logica** sara` data dal numero effettivo di caratteri diversi tra loro*

Soluzione

```
#include <stdio.h>
main()
{  char A[10], c;
   int i, j, inseriti=0, trovato;
   printf("\n Dammi 10 caratteri: ");
   for (i=0; (i<10); i++)
   {    scanf("%c", &c);
        /* verifica unicita`: */
        trovato=0;
        for(j=0;(j<inseriti)&&!trovato;j++)
                if (c==A[j])
                        trovato=1;
        /* se c non e` ancora presente in A, lo inserisco: */
        if (!trovato)
        {       A[inseriti]=c;
                inseriti++;
        }
   }
   printf("Inseriti %d caratteri \n",inseriti);
   for (i=0; i<inseriti; i++)
        printf("%c\n", A[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: in questo caso si parla di vettori multidimensionali

Definizione di vett. multidimensionali:

```
<id-tipo> <id-variabile>[dim1] [dim2]..[dimN];
```

Significato:

- **<id-variabile>** e` il nome di una variabile di tipo vettore di **dim1** componenti, ognuna delle quali e`
  - un vettore di **dim2** componenti, ognuna delle quali e`
    - un vettore di …., ognuna delle quali e`
      - un vettore di **dimn** componenti di tipo **<id-tipo>**.

➔ Se le dimensioni sono 2, il vettore si dice ***matrice***.

Vettori multidimensionali

Ad esempio: *matrice* di float

```
float M[20][30];
```

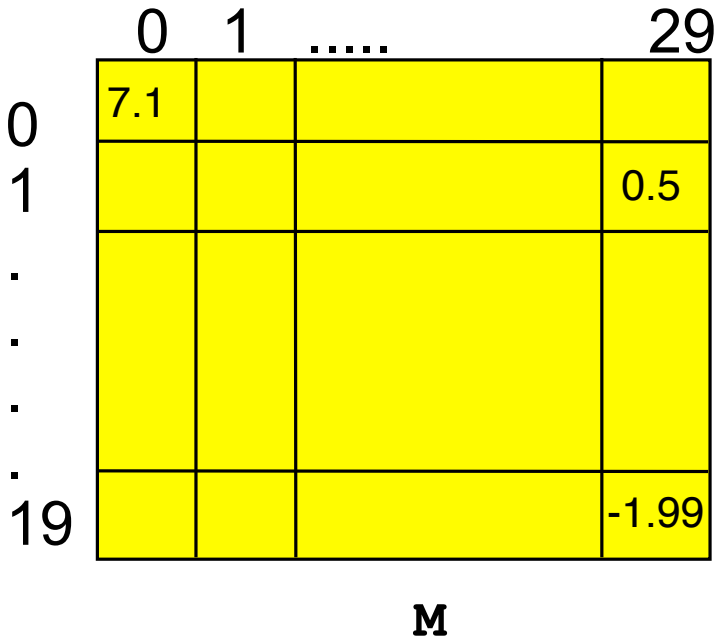
➔ M e` un vettore di 20 elementi, ognuno dei quali e` un vettore di 30 elementi, ognuno dei quali e` un float.

Accesso alle componenti:

il primo indice denota la ***riga***, il secondo la ***colonna***

```
M[0][0]=7.1;
M[1][29]=0.5;
M[19][29]=-1.99;
```

➔ **`M[0]` rappresenta il primo vettore di 30 float (la prima *riga*)**



Dichiarazione di tipo per vettori multi-dimensionali:

```
typedef  <id-tipo>  <id-tipo-vettore>[dim1][dim2].. [dimN];
```

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:Mat e` una matrice di 20
   X 30 di reali*/
```

In alternativa si puo` fare:

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

Inizializzazione di matrici

Anche nel caso di vettori multi-dimensionali l'inizializzazione si puo` effettuare in fase di definizione, tenendo conto che, in questo caso, gli elementi sono a loro volta vettori:

Esempio:

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

➔ La memorizzazione avviene "per righe":

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

Si puo` anche omettere il numero di righe:

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

Esempio: lettura e stampa di matrici

```
#include <stdio.h>
#define R 10
#define C 25

typedef float matrice[R][C];
main()
{       matrice M;
        int i, j;
        /* lettura: */
        for(i=0; i<R; i++)
                for(j=0; j<C; j++)
                {       printf("M[%d][%d]? ", i, j);
                        scanf("%f", &M[i][j]);
                }
        /*stampa: */
        for(i=0; i<R; i++)
        {       for(j=0; j<C; j++)
                        printf("%f\t", M[i][j);
                printf("\n");
        }
}
```

Esempio: prodotto di due matrici

Si realizzi un programma che esegua il prodotto (righe x colonne) di 2 matrici matrici a valori reali.

Definizione:

date due matrici rettangolari A[N][L] e B[L][M], il risultato di AxB e` una matrice C[N][M] tale che:
∀ i ∈[0,N-1], ∀ j ∈[0,M -1]:

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

Rappresentazione dei dati:

```
/*costanti per le dimensioni delle matrici:*/
#define N 2
#define M 3
#define L 4

float A[N][L];
float B[L][M];
float C[N][M];
```

Soluzione

```
#include <stdio.h>
#define N 2
#define M 3
#define L 4
main()
{  float A[N][L];
   float B[L][M];
   float C[N][M];
   int i, j, k;
/* inizializzazione di A e B */
printf("Elementi di A?\n");
for (i=0; i<N; i++)
{  printf("\nRiga %d (%d elementi): ", i, L);
   for (j=0; j<L; j++)
        scanf("%f",&A[i][j]);
}

printf("Elementi di B?\n");
for (i=0; i<L; i++)
{  printf("\nRiga %d (%d elementi): ", i, M);
   for (j=0; j<M; j++)
        scanf("%f",&B[i][j]);
} /* continua.. */
```

Soluzione

```
/* ...prodotto matriciale:  */
for (i=0; i<N; i++)
   for (j=0; j<M; j++)
   {    C[i][j]=0;
        for (k=0; k<L; k++)
               C[i][j]+= A[i][k]*B[k][j];
   }

/* stampa  del risultato: */
   for (i=0; i<N; i++)
   {    printf("\nC[%d]:\t", i);
        for (j=0; j<M; j++)
               printf("%f\t",C[i][j]);
   }
}
```

Esempio: ordinamento di un vettore

Dati n valori interi forniti in ordine casuale, 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.

Soluzione:

Esistono vari procedimenti risolutivi per questo problema (v. algoritmi di ordinamento, o ***sorting***):

- **naïve-sort**
- bubble sort
- shell sort
- merge sort
- ...

Risolveremo il problema utilizzando il Metodo dei Massimi successivi (*naïve-sort*).

Ordinamento di un vettore mediante *naïve sort*

Descrizione dell' algoritmo:

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, 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.

Soluzione

```
#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=dim-1; i>1; i--)
   {   max=i;
       for( j=0; j<i; j++)
       if (V[j]>V[max])
              max=j;
       if (max!=i) /*scambio */
       {      tmp=V[i];
              V[i]=V[max];
              V[max]=tmp;
       }
}
```

Soluzione

```
/* ..stampa il vettore ordinato */
   for (i=0; i<dim; i++)
       printf("\n%d", V[i]);
}
```

Vettori di Caratteri: le Stringhe

Una stringa e` un vettore di caratteri, manipolato e gestito secondo la **convenzione**:

l'ultimo elemento significativo di ogni stringa e` seguito dal carattere nullo '\0' (codice zero).

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

Ad esempio:

```
char A[10]={'b','o','l','o','g','n','a','\0'};
```

'b'	'o'	'l'	'o'	'g'	'n'	'a'	'\0'	?	?

oppure:

```
char A[10]="bologna"; /*  il terminatore  '\0' e` aggiunto
                                automaticamente */
```

Lettura di stringhe (formato %s):

```
char B[25];

scanf("%s", &B); /*      la sequenza di caratteri letta viene assegnata a B; il
                         terminatore  '\0' e` aggiunto automaticamente */
```

Esempio:lunghezza di una stringa

Programma che calcola la ***lunghezza*** di una stringa (cioe` , il numero di caratteri significativi).

```
/* lunghezza di una stringa */
#include <stdio.h>
main()
{  char str[81]; /* str ha al max. 80 caratteri*/
   int i;
   printf("\nImmettere una stringa:");
   scanf("%s",&str); /* %s formato
                        stringa */
   for (i=0; str[i]!='\0'; i++);
        printf("\nLunghezza: \t %d\n",i);
}
```

➔ La `scanf` legge i caratteri in ingresso fino al primo separatore (bianco, newline, ecc.) e li assegna al vettore `str` aggiungendo il terminatore di stringa.

NB: La lunghezza di una stringa si puo` anche calcolare utilizzando la funzione standard di libreria `strlen()` (previa inclusione di `string.h`)

Esempio:concatenamento di stringhe

Programma che concatena due stringhe s1 e s2 date: il risultato viene inserito in s1.

```
#include <stdio.h>
/* concatenamento di due stringhe */
main()
{
   char  s1[81], s2[81];
   int   j,i;
printf("\nPrima stringa: \t");
scanf("%s",&s1);
printf("\nSeconda stringa: \t");
scanf("%s",&s2);
for (j=0; s1[j]!='\0'; j++);
for (i=0; s2[i]!='\0' && i+j<79; i++)
        s1[i+j]=s2[i];
s1[i+j]='\0';    /* fine stringa */
printf("\nStringa:\t%s\n",s1);
}
```

➔ il concatenamento di due stringhe si puo` anche ottenere utilizzando la funzione standard di libreria `strcat()` (previa inclusione di string.h)

Libreria standard sulle stringhe

Il C fornisce una libreria standard di funzioni per la gestione di stringhe;per poterla utilizzare e` necessario includere il file header <string.h>:

```
#include <string.h>
```

Lunghezza di una stringa:

```
int strlen(char  str[ ]);
```

➔ restituisce la lunghezza (cioe` , il numero di caratteri significativi) della stringa `str` specificata come argomento.

Ad esempio:

```
char S[10]="bologna";
int k;
k=strlen(S);        /* k vale 7*/
```

Libreria standard sulle stringhe

Concatenamento di 2 stringhe:

```
strcat( char str1[], char str2[]);
```

➔ concatena le 2 stringhe date str1 e str2. Il risultato del concatenamento e` in str1.

Ad esempio:

```
char S1[10]="reggio";
char S2[10]="emilia";
strcat(S1, S2); /*S1="reggioemilia"*/
```

Copia di stringhe:

```
strcpy(char str1[], char str2[]);
```

➔ copia la stringa str2 in str1.

Ad esempio:

```
char S1[10]="Giuseppe";
char S2[10];
strcpy(S2,S1); /* S2="Giuseppe"*/
```

Libreria standard sulle stringhe

Confronto di 2 stringhe:

```
int strcmp(char str1[ ], char str2[ ]);
```

➔ esegue il confronto tra le due stringhe date str1 e str2. Restituisce:

- 0 se le due stringhe sono identiche;
- un valore negativo(ad esempio, -1), se str1 precede str2 (in ordine lessicografico);
- un valore positivo (ad esempio, +1), se str2 precede str1 (in ordine lessicografico).

Ad esempio:

```
char S1[10]="bologna";
char S2[10]="napoli";
int k;
k=strcmp(S1,S2); /* k < 0 */
k=strcmp(S1,S1); /* k = 0 */
k=strcmp(S2,S1); /* k > 0 */
```

Input/Output di linee

La scanf (con formato %s) prevede come separatore anche il blank (spazio bianco):

➔ e` possibile soltanto leggere stringhe che non contengono bianchi;

Ad esempio, se l'input e`:

Nel mezzo del cammin di nostra vita,
mi ritrovai in una selva oscura…
…

Leggendo con:

```
char s1[80], s2[80];
scanf("%s", &s1); /* s1 vale "Nel" */
scanf("%s", &s2); /*s2 vale "mezzo"*/
```

➔ la scanf non e` adatta a leggere intere linee (che possono contenere spazi bianchi, caratteri di tabulazione, etc.).

Per questo motivo, in C esistono funzioni specifiche per fare I/O di linee:

- `gets`
- `puts`

Input/Output di linee

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.

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

assegna alla stringa `str` i caratteri letti.
Il carattere '\n' viene sostituito (nella stringa di destinazione `str`) da '\0'.

Ad esempio, dato l'input:
Nel mezzo del cammin di nostra vita,
mi ritrovai in una selva oscura...

Leggendo con:

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

S vale:
"***Nel mezzo del cammin di nostra vita,***"

Input/Output di linee

puts:

e` una funzione standard che scrive una stringa sull'output aggiungendo un carattere di fine linea ('\n', newline):

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

Ad esempio:

```
char S1[80]="Dante Alighieri";
char S2[80]="La Divina Commedia";
puts(S1);
puts(S2);
```

stampa sullo standard output:

Dante Alighieri
La Divina Commedia

➔ In generale: `puts(S);` e` **equivalente** a `printf("%s\n", S);`

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 e` necessario introdurre 4 variabili:

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

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

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

Tipi strutturati: il Record

Un record e` un insieme finito di elementi, in generale 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.**

Ad esempio:

NOME	COGNOME	REDDITO	ALIQUOTA

Formalmente:

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

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

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

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

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

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

Ad esempio:

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

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 definizioni campi> } <id-variabile>;
```

dove:

- **`<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.

Il Record in C: `struct`

Esempio:

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

➔ **`Nome`, `Cognome`, `Reddito` ed `Aliquota`** sono i campi della variabile **`contribuente`** (di tipo **`struct`**).

Il tipo struct

Operatori:

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

➔ e` possibile l'assegnamento diretto tra record di tipo equivalente.

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.

Il tipo struct

Ad esempio:

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

contribuente.Reddito=2000+1500;
strcpy(contribuente.Nome,"Mario");
strcpy(contribuente.Cognome,"Rossi");
contribuente.Aliquota=40;
```

Inizializzazione di record:

E` possibile inizializzare i record in fase di definizione.

Ad esempio:

```
struct {     char    Nome[20];
                     char    Cognome[20];
                     int     Reddito;
                     int     Aliquota;
  }contribuente =("Mario","Rossi",3500,40);
```

Il costruttore di tipo struct

Il costruttore struct puo` 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>` e` l'insieme delle definizioni dei campi componenti;
- `<id_tipo>` e` l'identificatore del nuovo tipo.

Ad esempio:

```
typedef struct {     int anno;
                     int mese;
                     int giorno;
               }tipodata;
tipodata data;
unsigned int anno=1999;
data.anno=anno;
data.mese=1;
data.giorno=6;
```

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

Il tipo struct in sintesi

Riassumendo, la sintassi da adottare e` :

```
[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.

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, ognuno associato a una particolare coordinata:

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

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

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

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

Esercizio sui record

```
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("errore!");
   }
printf("%s\n","nuove coordinate: ");
printf("%f\t%f\n",P.x,P.y);
}
```

Vettori e record

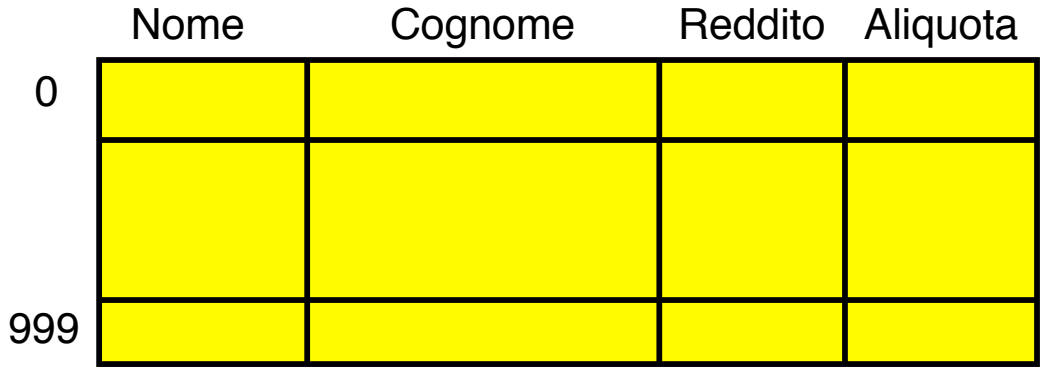
Non ci sono vincoli riguardo al tipo degli elementi di un vettore: si possono realizzare anche vettori di record (strutture tabellari).

Ad esempio:

```
typedef struct {        char    Nome[20];
                        char    Cognome[20];
                        int     Reddito;
                        int     Aliquota;
                        } Contribuente;
contribuente archivio[1000];
```

➔ `archivio` e` un vettore di 1000 elementi, ciascuno dei quali e` di tipo `contribuente`

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



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;
…
```

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 {        char nome[30];
                        char cognome[30];
                        int voto[3];
                } studente;
```

La classe e` rappresentata da un vettore di studenti:

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

Esercizio

```
#include <stdio.h>
#define ita 0
#define mat 1
#define ing 2
#define N 20
typedef struct{ char nome[30];
        char cognome[30];
        int voto[3];
        } studente;

main()
{       studente classe[N];
        float m;
        int i; int j;
        /* lettura dati */
        for(i=0;i<N; i++)
        {       fflush(stdin);
                gets(classe[i].nome);
                gets(classe[i].cognome);
                for(j=ita; j<=ing; j++)
                        scanf("%d", &classe[i].voto[j]);
        }
```

Esercizio

```
        /*continua.. stampa delle medie */
        for(i=0;i<N; i++)
        {  for(m=0, j=ita; j<=ing; j++)
               m+=classe[i].voto[j];
           printf("media di %s %s: %f\n",
               classe[i].nome, classe[i].cognome, m/3);
        }
}
```