

Linguaggio C: Tipi strutturati

Array

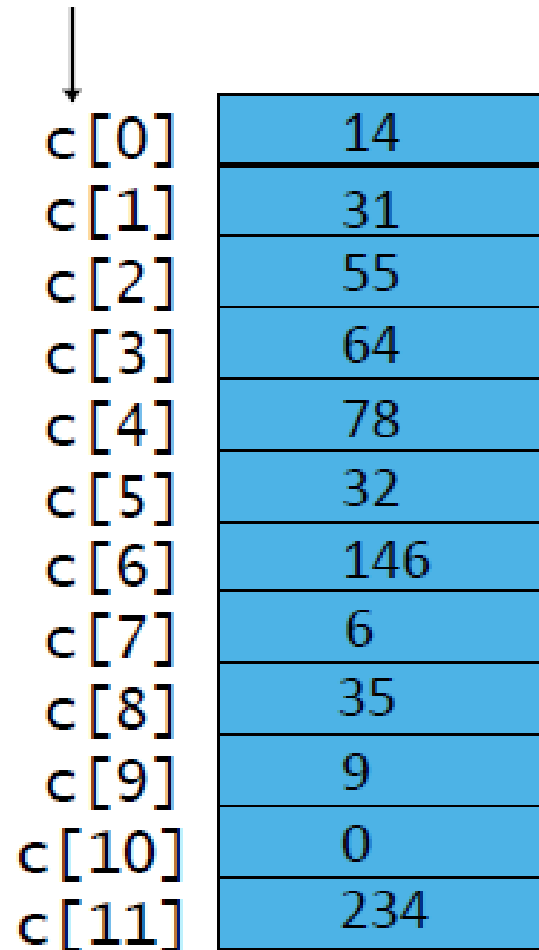
- Collezioni di locazioni di memoria consecutive
- Stesso nome e tipo
- Modella elementi di tipo **omogeneo**

Accesso:

- Nome array + posizione (per le variabili scalari, solo nome)
- nome[position number]
- Array di N elementi di nome c c[0],c[1]...,c[N-1]
- Elemento in [] è **indice** dell'array

Array

Stesso nome



c[0]	14
c[1]	31
c[2]	55
c[3]	64
c[4]	78
c[5]	32
c[6]	146
c[7]	6
c[8]	35
c[9]	9
c[10]	0
c[11]	234

↑ Posizione dell'elemento nella
collezione c

Array

- Usate come variabili scalari, ad esempio in istruzioni di lettura/scrittura:

```
printf("%d", c[0])      scanf("%d", c[0])
```

- Gli indici possono essere sostituiti da espressioni a valore intero:

```
c[5-2]      c[k*2]      c[i-j]
```

- Dichirazione

Tipo di dato + nome+numero elementi

```
arraType arrayName[numberElements]
```

```
int c[10];
```

```
float myArray[3284];
```

Array

Inizializzazione con la **dichiarazione**

- `int n[5] = { 1, 2, 3, 4, 5 };`
- `int n[5] = {0} // tutti i valori impostati a zero`
- `int n[5] = {1, 2, 3} // i valori mancanti impostati a zero`
- `int n[] = { 1, 2, 3, 4, 5 };`

Quando inizializzati, la dimensione può essere omessa

Array

Inizializzazione **lazy**

- Se un array non è inizializzato in fase di dichiarazione, occorre farlo successivamente e procedere a un ciclo di inizializzazione
- Più generalmente, gli array sono correlati ai cicli poiché le operazioni di accesso sono istruzioni iterate nella collezione.
- Ricordarsi che il linguaggio C **non effettua alcun controllo** sugli indici degli array. In fase di codifica del programma bisognerà controllare che il ciclo non vada oltre l'array.

Array

```

1  /* Fig. 6.3: fig06_03.c
2     initializing an array */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main()
7  {
8     int n[ 10 ]; /* n is an array of 10 integers */
9     int i;       /* counter */
10
11     /* initialize elements of array n to 0 */
12     for ( i = 0; i < 10; i++ ) {
13         n[ i ] = 0; /* set element at location i to 0 */
14     } /* end for */
15
16     printf( "%s%13s\n", "Element", "Value" );
17
18     /* output contents of array n in tabular format */
19     for ( i = 0; i < 10; i++ ) {
20         printf( "%7d%13d\n", i, n[ i ] );
21     } /* end for */
22
23     return 0; /* indicates successful termination */
24
25 } /* end main */

```

Element	Value
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0

Array

Esempio: somma di elementi

dichiarazione costante
della dimensione

```
1  /* Fig. 6.6: fig06_06.c
2      Compute the sum of the elements of the array */
3  #include <stdio.h>
4  #define SIZE 12
5
6  /* function main begins program execution */
7  int main()
8  {
9      /* use initializer list to initialize array */
10     int a[ SIZE ] = { 1, 3, 5, 4, 7, 2, 99, 16, 45, 67, 89, 45 };
11     int i;          /* counter */
12     int total = 0; /* sum of array */
13
14     /* sum contents of array a */
15     for ( i = 0; i < SIZE; i++ ) {
16         total += a[ i ];
17     } /* end for */
18
19     printf( "Total of array element values is %d\n", total );
20
21     return 0; /* indicates successful termination */
22
23 } /* end main */
```

inizializzazione con la
dichiarazione di variabili

accesso agli elementi
attraverso un ciclo

Esempio

- *Scrivere un programma che calcoli la media delle calorie assunte in una settimana, usando un array.*
 - Input?
Collezione di valori. Quanti?
 - Output?
Media dei valori
 - Quale struttura di controllo usare?
Struttura di iterazione
per avvalorare l'array e per accedere agli elementi per il calcolo

Esempio

```
1  #include <stdio.h>
2  #define SETTIMANA 7
3
4  int main() {
5
6      int calorie[SETTIMANA] = {0}; // dichiarazione e inizializzazione del vettore
7      int somma_calorie = 0; // serve a memorizzare la somma delle calorie
8      float media_calorie = 0.0; // serve a memorizza la media delle calorie
9
10     // ciclo di inizializzazione
11     for(int i=0; i<SETTIMANA; i++) {
12         printf("Inserire il numero di calorie assunte il giorno %d:",i+1);
13         scanf("%d",&calorie[i]); // acquisizione valori di input
14     }
15
16     // ciclo di elaborazione
17     for(int i=0; i<SETTIMANA; i++) {
18         somma_calorie += calorie[i]; // aggiunge il valore alla somma
19     }
20
21     media_calorie = (float) somma_calorie / SETTIMANA; // calcolo media
22
23     //visualizzazione output
24     printf("Il numero medio di calorie assunte nella settimana è: %.2f",media_calorie);
25
26     return 0;
27 }
```

Esempio

```
1  #include <stdio.h>
2  #define SETTIMANA 7
3
4  int main() {
5
6      int calorie[SETTIMANA] = {0}; // dichiarazione e inizializzazione del vettore
7      int somma_calorie = 0; // serve a memorizzare la somma delle calorie
8      float media_calorie = 0.0; // serve a memorizza la media delle calorie
9
10     // ciclo di inizializzazione
11     for(int i=0; i<SETTIMANA; i++) {
12         printf("Inserire il numero di calorie assunte il giorno %d:",i+1);
13         scanf("%d",&calorie[i]); // acquisizione valori di input
14     }
15
16     // ciclo di elaborazione
17     for(int i=0; i<SETTIMANA; i++) {
18         somma_calorie += calorie[i]; // aggiunge il valore alla somma
19     }
20
21     media_calorie = (float) somma_calorie / SETTIMANA; // calcolo media
22
23     //visualizzazione output
24     printf("Il numero medio di calorie assunte nella settimana è: %.2f",media_calorie);
25
26     return 0;
27 }
```

dichiarazione

immissione dati

elaborazione dati

presentazione/visualizzazione risultati

Esercizio

- *Scrivere un programma che conteggi il costo totale dei prodotti in un carrello che contiene 5 prodotti*
 - Input?
Collezione di valori.
 - Output?
Costo della spesa
 - Quale struttura di controllo usare?
Struttura di iterazione
per avvalorare l'array e per accedere agli elementi per
il calcolo

Esercizio

- *Scrivere un programma che, usando un array, calcoli la media delle calorie assunte in una settimana, specificando il giorno della settimana all'atto della immissione.*
 - Input?
Collezione di valori.
 - Output?
Costo della spesa
 - Quale struttura di controllo usare?
Struttura di iterazione
per avvalorare l'array e per accedere agli elementi per il calcolo

```
Inserire il numero di calorie assunte il giorno 1: 1810
Inserire il numero di calorie assunte il giorno 2: 1620
Inserire il numero di calorie assunte il giorno 3: 2140
Inserire il numero di calorie assunte il giorno 4: 1930
Inserire il numero di calorie assunte il giorno 5: 2070
Inserire il numero di calorie assunte il giorno 6: 2400
Inserire il numero di calorie assunte il giorno 7: 1860
```

Non c'è alcuna modifica alla soluzione del problema, ma c'è una estensione alla immissione di dati

Array

Gli indici possono essere sostituiti da espressioni a valore intero:

$c[5-2]$ $c[k*2]$ $c[i-j]$

Vediamone un esempio. Conteggio della frequenza delle risposte date in un questionario:

Domanda A:

Risposta i....

Risposta ii:....

Risposta iii:....

Domanda B:

Risposta i:....

Risposta ii:....

Risposta iii:....

A

B

<i>i</i>	5
<i>iii</i>	7
....

i

ii

Array

*Dimensioni
costanti di due
array:
uno conterrà le
risposte a un
questionario e uno
conteggerà quante
volte una
determinata
risposta è stata
scelta*

```
1  /* Fig. 6.7: fig06_07.c
2      Student poll program */
3  #include <stdio.h>
4  #define RESPONSE_SIZE 40 /* define array sizes */
5  #define FREQUENCY_SIZE 11
6
7  /* function main begins program execution */
8  int main()
9  {
10     int answer; /* counter */
11     int rating; /* counter */
12
13     /* initialize frequency counters to 0 */
14     int frequency[ FREQUENCY_SIZE ] = { 0 };
15
16     /* place survey responses in array responses */
17     int responses[ RESPONSE_SIZE ] = { 1, 2, 6, 4, 8, 5, 9, 7, 8, 10,
18         1, 6, 3, 8, 6, 10, 3, 8, 2, 7, 6, 5, 7, 6, 8, 6, 7, 5, 6, 6,
19         5, 6, 7, 5, 6, 4, 8, 6, 8, 10 };
20
```

Array

Inizializzazione dei due vettori. Quello delle frequenze (frequency) con elementi impostati a zero e quello delle risposte (responses) con il valore numerico di ciascuna risposta.

```
1  /* Fig. 6.7: fig06_07.c
2      Student poll program */
3  #include <stdio.h>
4  #define RESPONSE_SIZE 40 /* define array sizes */
5  #define FREQUENCY_SIZE 11
6
7  /* function main begins program execution */
8  int main()
9  {
10     int answer; /* counter */
11     int rating; /* counter */
12
13     /* initialize frequency counters to 0 */
14     int frequency[ FREQUENCY_SIZE ] = { 0 };
15
16     /* place survey responses in array responses */
17     int responses[ RESPONSE_SIZE ] = { 1, 2, 6, 4, 8, 5, 9, 7, 8, 10,
18         1, 6, 3, 8, 6, 10, 3, 8, 2, 7, 6, 5, 7, 6, 8, 6, 7, 5, 6, 6,
19         5, 6, 7, 5, 6, 4, 8, 6, 8, 10 };
20
```


Array

	<i>responses</i>	<i>frequency</i>	
<i>A</i>	<i>i</i>	5	<i>i</i>
<i>B</i>	<i>iii</i>	7	<i>ii</i>
	

Array

$Answer = 0 \rightarrow Responses[0] = 1 \rightarrow frequency[1] = frequency[1]++$

*Scandiamo l'array delle risposte, prendiamo la risposta, come intero, accediamo all'array delle frequenze ed incrementiamo la frequenza.
Il valore letto da un array è indice per l'altro array.*

```
21  /* for each answer, select value of an element of array responses
22      and use that value as subscript in array frequency to
23      determine element to increment */
24  for ( answer = 0; answer < RESPONSE_SIZE; answer++ ) {
25      ++frequency[ responses [ answer ] ];
26  } /* end for */
27
28  /* display results */
29  printf( "%s%17s\n", "Rating", "Frequency" );
30
31  /* output frequencies in tabular format */
32  for ( rating = 1; rating < FREQUENCY_SIZE; rating++ ) {
33      printf( "%6d%17d\n", rating, frequency[ rating ] );
34  } /* end for */
35
36  return 0; /* indicates successful termination */
37
38 } /* end main */
```

Matrici

- Trattasi di array multidimensionale in forma tabellare righeX.colonne (mXn)
- Accesso
 - effettuato con due indici, il primo si riferisce alle righe, il secondo alle colonne
 - Nome array+riga+colonna
`arrayname[riga][colonna]`
 - dato un array *a* di dimensione mXn
 - `a[0][0], a[0][1], a[m-1][0], ..., a[m-1][1], ..., a[m-1][n-1]`
 - `a[2][2]=3;`
 - `printf("%d", b[0][1]);`

Matrici

- `int m[2][2]; // matrice 2 per 2`
- `int m[2][2]={ {1, 2} {3,4} }; //dichiara matrice 2 per 2 e la
inizializza in modo che m[0][0]=1, m[0][1]=2, m[1][0]=3, m[1]
[1]=4`
- `int m[2][2]={ {1} {3,4} }; //dichiara matrice 2 per 2 e la inizializza
in modo che m[0][0]=1, m[0][1]=0, m[1][0]=3, m[1][1]=4`
- `int m[2][2]={1,2,3,4}; //dichiara matrice 2 per 2 e la inizializza in
modo che m[0][0]=1, m[0][1]=2, m[1][0]=3, m[1][1]=4`
- `int m[2][2]={1,2,3}; //dichiara matrice 2 per 2 e la inizializza in
modo che m[0][0]=1, m[0][1]=2, m[1][0]=3, m[1][1]=0`

Matrici

	Column 0	Column 1	Column 2	Column 3
Row 0	<code>a[0][0]</code>	<code>a[0][1]</code>	<code>a[0][2]</code>	<code>a[0][3]</code>
Row 1	<code>a[1][0]</code>	<code>a[1][1]</code>	<code>a[1][2]</code>	<code>a[1][3]</code>
Row 2	<code>a[2][0]</code>	<code>a[2][1]</code>	<code>a[2][2]</code>	<code>a[2][3]</code>

riga

colonna

Matrici

- Inizializzazione
 - segue la sintassi degli array monodimensionali `int b[2][2]={ {1,2},{3,4} };`
 - inizializzazione con raggruppamenti per righe:
prima riga `1,2` seconda riga `{3,4}`
 - Se righe non sufficienti, gli elementi mancanti saranno impostati automaticamente a zero
- `int b[2][2]={ {1},{3,4} } → { {1,0},{3, 4 } }`
- Accesso
 - effettuato con due indici, il primo si riferisce alle righe, il secondo alle colonne-
 - Nome array+riga+colonna
`arrayname[riga][colonna]`
 - dato un array *a* di dimensione mXn
 - `a[0][0], a[0][1], a[m-1][0], ..., a[m-1][1], ..., a[m-1][n-1]`

Esempio

- Stampa dei valori

Dichiarazione

*Dichiarazione e
inizializzazione della
matrice*

*Presentazione dei
valori tramite
invocazione di
funzioni*

```
1  /* Fig. 6.21: fig06_21.c
2     Initializing multidimensional arrays */
3  #include <stdio.h>
4
5  void printArray( const int a[][ 3 ] ); /* function prototype */
6
7  /* function main begins program execution */
8  int main()
9  {
10     /* initialize array1, array2, array3 */
11     int array1[ 2 ][ 3 ] = { { 1, 2, 3 }, { 4, 5, 6 } };
12     int array2[ 2 ][ 3 ] = { 1, 2, 3, 4, 5 };
13     int array3[ 2 ][ 3 ] = { { 1, 2 }, { 4 } };
14
15     printf( "Values in array1 by row are:\n" );
16     printArray( array1 );
17
18     printf( "Values in array2 by row are:\n" );
19     printArray( array2 );
20
21     printf( "Values in array3 by row are:\n" );
22     printArray( array3 );
23
24     return 0; /* indicates successful termination */
25
26 } /* end main */
27
```

Esempio

- Stampa dei valori

-Elaborazione dei valori di matrici tramite due cicli iterativi annidati

-Uno scandisce le righe, l'altro le colonne (o viceversa.)

-Quando si denota una nuova riga, si scandiscono tutti gli elementi sulle colonne

- Il ciclo più interno esegue realmente le operazioni sui singoli elementi

```

28 /* function to output array with two rows and three columns */
29 void printArray( const int a[][ 3 ] )
30 {
31     int i; /* counter */
32     int j; /* counter */
33
34     /* loop through rows */
35     for ( i = 0; i <= 1; i++ ) {
36
37         /* output column values */
38         for ( j = 0; j <= 2; j++ ) {
39             printf( "%d ", a[ i ][ j ] );
40         } /* end inner for */
41
42         printf( "\n" ); /* start new line of output */
43     } /* end outer for */
44
45 } /* end function printArray */

```

```

Values in array1 by row are:
1 2 3
4 5 6
Values in array2 by row are:
1 2 3
4 5 0
Values in array3 by row are:
1 2 0
4 0 0

```


Esercizio

Scrivere un programma che, usando una matrice, acquisisca le calorie assunte in una settimana da un insieme di 5 individui e ne calcoli la media (per individuo e complessiva)

- Input?
 - Due collezioni (7x5) di valori di calorie assunte
- Output?
 - Media per ogni individuo e media complessiva
- Quale struttura di controllo usare?
 - Struttura di iterazione
 - per avvalorare la matrice e per accedere agli elementi
 - per il calcolo

Una soluzione

```
1  #include <stdio.h>
2  #define PERSONE 5
3  #define SETTIMANA 7
4
5- int main() {
6
7     int calorie[PERSONE][SETTIMANA] = {0}; // dichiarazione e inizializzazione del vettore
8     int somma_calorie = 0; // serve a memorizzare la somma delle calorie
9     int somma_calorie_complessive = 0; // serve a memorizzare la somma delle calorie complessive
10    float media_calorie = 0.0; // serve a memorizzare la media delle calorie di un individuo
11    float media_calorie_complessive = 0.0; // serve a memorizzare la media delle calorie complessive
12
13    // ciclo di inizializzazione
14-    for(unsigned int i=0; i<PERSONE; i++) {
15-        for(unsigned int j=0; j<SETTIMANA; j++) {
16            printf("Inserire il numero di calorie assunte dall'individuo %d il giorno %d:", i+1, j+1);
17            scanf("%d", &calorie[i][j]); // acquisizione valori di input
18        }
19    }
20
21    puts(""); //formattazione
```

Una soluzione

```
22
23 // ciclo di elaborazione
24 - for(unsigned int i=0; i<PERSONE; i++) {
25     somma_calorie = 0;
26
27 -     for(unsigned int j=0; j<SETTIMANA; j++) {
28         somma_calorie += calorie[i][j];
29     }
30
31     media_calorie = (float) somma_calorie / SETTIMANA; // calcolo media
32
33     printf("Il numero medio di calorie assunte nella settimana dall'individuo %d è: %.2f\n", i+1, media_calorie);
34     somma_calorie_complessive += somma_calorie;
35
36 }
37
38 media_calorie_complessive = (float) somma_calorie_complessive / (SETTIMANA*PERSONE); // calcolo media complessiva
39
40 //visualizzazione output
41 printf("Il numero medio di calorie assunte nella settimana è: %.2f",media_calorie_complessive);
42
43 return 0;
44 }
```

Esercizi

- *Scrivere un programma che, usando una matrice, acquisisca le calorie assunte in una settimana da un insieme di 5 individui e ne identifichi il massimo per individuo e complessivo.*
- *Scrivere un programma che, usando una matrice, acquisisca le calorie assunte in una settimana da un insieme di 5 individui e ne identifichi il giorno (lunedì, martedì,...) in cui vi è il massimo per individuo e il giorno in cui vi è il massimo complessivo.*

Variabili Strutturate

- Gli array permettono di definire variabili strutturate in forma di collezione di valori dello **stesso** tipo.
- Tuttavia, questo non consente di definire variabili di tipo più complesso. Ad esempio, Data composta da Giorno-Mese-Anno (1-January-2000), Libro composto da Autore-Prezzo-Anno (Poe-30,00-1990).
- Una **Struct** è una aggregazione che può contenere elementi di tipo **eterogeneo**, cioè elementi diversi tra di loro.
- Diversamente dagli array non contiene una collezione di valori omogenei, ma ugualmente agli array fornisce una aggregazione di valori.

Variabili Strutturate con Struct

dichiarazione di una
variabile di tipo persona

```
struct persona {
```

```
    char nome[DIMENSIONE];  
    char cognome[DIMENSIONE];  
    int eta;
```

membri/
campi della
struttura

```
} p
```

variabile strutturata che
prende la struttura
definita

Struct

- Accesso:
 - Operatore di membro struttura (o operatore punto)
“.”
 - Operatore puntatore a struttura “->”
 - `p.eta=43` // accesso ai singoli membri

Struct e Tipi strutturato con Typedef

La definizione di una variabile strutturata può essere generalizzata ed usata per definire un **tipo di dato strutturato** per dichiarare innumerevoli variabili.

```
typedef struct {
```

```
    int giorno
```

```
    char mese;
```

```
    int anno;
```

```
} data
```

→ dichiarazione di un tipo
di dato strutturato

→ membri del tipo

→ nome del tipo

```
data p1;
```

```
data p2;
```

→ dichiarazione di variabili

Struct e Typedef

```
// Inizializzo e stampo una nuova variabile  
data d1 = {22, "Marzo", 2017};
```

inizializzazione

```
printf("Oggi: %d %s %d", d1.giorno, d1.mese, d1.anno);
```

visualizzazione

```
// Dichiaro una nuova variabile e acquisisco i valori  
data d2;
```

```
// Memorizzo i valori della seconda variabile
```

```
printf("\nInserisci una nuova data: ");
```

```
scanf("%d %s %d", &d2.giorno, &d2.mese, &d2.anno);
```

immissione

```
// Stampa i valori: IMPORTANTE, non c'è nessun controllo
```

```
printf("Oggi: %d %s %d", d2.giorno, d2.mese, d2.anno);
```

```
}
```

- Per la immissione e visualizzazione le funzioni scanf() e printf() continueranno ad usare tipi built-in a valori scalari.

Struct e Typedef

```
// Inizializzo e stampo una nuova variabile  
data d1 = {22, "Marzo", 2017};
```

inizializzazione

```
printf("Oggi: %d %s %d", d1.giorno, d1.mese, d1.anno);
```

visualizzazione

```
// Dichiaro una nuova variabile e acquisisco i valori  
data d2;
```

```
// Memorizzo i valori della seconda variabile
```

```
printf("\nInserisci una nuova data: ");
```

```
scanf("%d %s %d", &d2.giorno, &d2.mese, &d2.anno);
```

immissione

```
// Stampa i valori: IMPORTANTE, non c'è nessun controllo
```

```
printf("Oggi: %d %s %d", d2.giorno, d2.mese, d2.anno);
```

```
}
```

- Per la assegnazione l'operatore "=" è consentito su valori di una struct. Ad esempio:
 - `data day1={22,"Marzo",2017}`
 - `day2=day1`

Struct e Typedef

```
#include <stdio.h>

int main() {

    typedef struct{
        int x;
        int y;
    } point;

    point a = {10, 20};
    point b = {10, 20};

    if(a==b)
        puts("Equals");
    else
        puts("Not equals");

}
```

- Per la uguaglianza l'operatore “==” applicato su valori di una struct **non** è consentito, cioè il compilatore non lo ammette.

Struct e Typedef

```
#include <stdio.h>

int main() {

    typedef struct{
        int x;
        int y;
    } point;

    point a = {10, 20};
    point b = {10, 20};

    if((a.x == b.x) && (a.y == b.y))
        puts("Equals");
    else
        puts("Not equals");

}
```

- Per la uguaglianza si userà l'operatore “==” applicato su valori dei membri di una struct.

Struct e Typedef

La definizione di una variabile strutturata può essere generalizzata ed usata per definire un **tipo di dato strutturato** per dichiarare innumerevoli variabili.

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

→ membri del tipo

I membri possono essere di tre tipi: *built-in* con valori scalari(primitivi), *collezione* (array) e *user-defined* (struct a loro volta).

Struct e Typedef

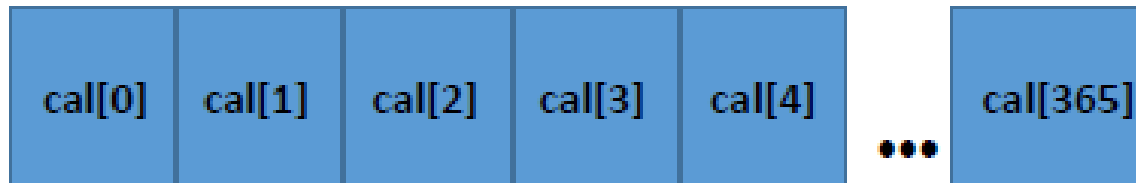
Si potrebbero costruire generare valori arbitrariamente complessi da tipi arbitrariamente strutturati. Ad esempio,

```
typedef struct {  
    int esami  
    data nascita;  
    int matricola;  
    char [20] nome;  
} studente
```

Struct e Typedef

Oltre a definire variabili strutturate, possiamo usare un typedef per definire una collezione di valori omogenei strutturati, cioè array, ad esempio:

```
data [365]calendario
```

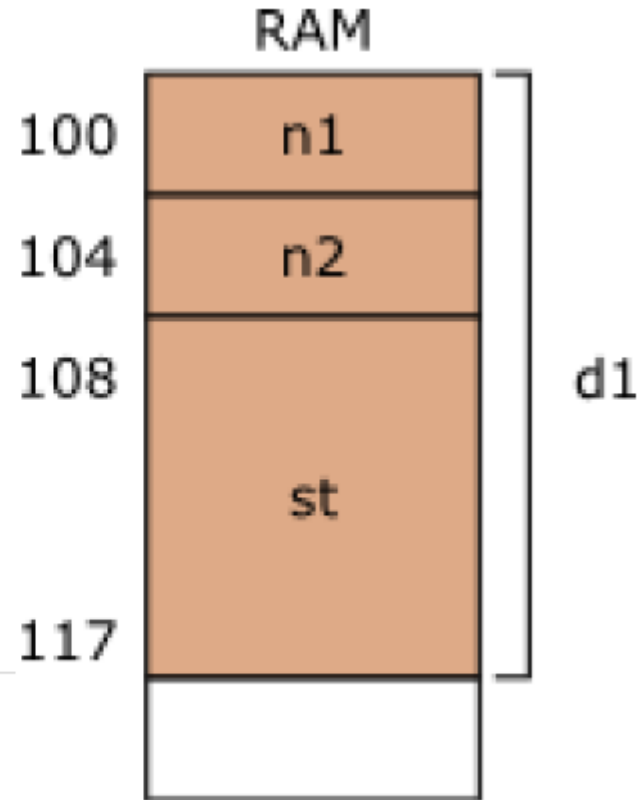


dove, ciascun `cal[i]` individua uno di valori strutturati `data` di una collezione omogenea.

Quindi, si può accedere ai membri di un valore strutturato:
`cal[0].mese="gennaio",cal[0].anno="2018"`

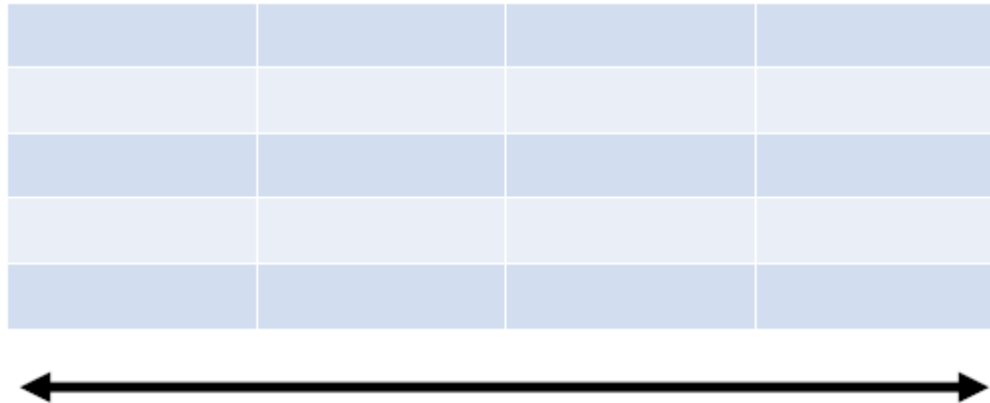
Struct in memoria

```
struct contact
{
    int n1;
    float n2;
    char st[10];
} d1;
```



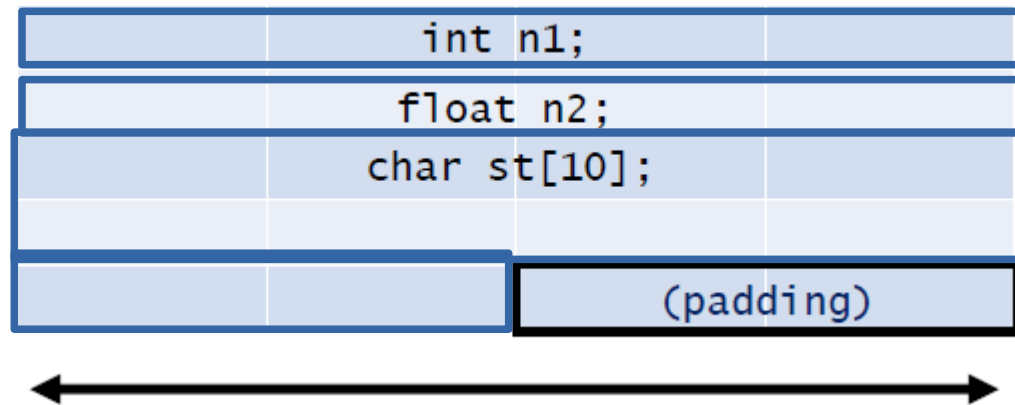
- I membri di una struct sono memorizzati in locazioni consecutive.
- La dimensione di *d1* è $2+4+10*1= \dots 20!$

Struct in memoria



- Nelle attuali architetture vi è un accesso ottimizzato alla memoria che prevede la lettura di 4 byte per volta.

Struct in memoria



- Per garantire la ottimizzazione, alla allocazione si *st* sarebbero forniti 2 byte in più in modo da ottenere una occupazione ugualmente ad un multiplo di 4.

Esempio

- Scrivere un programma che acquisisca e visualizzi i dati di una collezione di 5 studenti, ciascuno descritto da *<nome, cognome, data di nascita, voti di esami>*
- Considerare casi limite per data di nascita e voti esami.

Esempio

Dichiarazioni di costanti

```
#include <stdio.h>
#define ESAMI 3 // numero degli esami
#define N 5 // numero degli studenti
```

```
int main() {
```

Dichiarazioni di tipo

```
// Dichiaro una struct di tipo "data"
typedef struct {
    int giorno;
    char mese[15];
    int anno;
} data ; // campi = giorno, mese, anno
```

Esempio

Dichiarazioni di tipo

```
// Dichiaro una struct di tipo "studente"
typedef struct {
    char nome[10];
    char cognome[15];
    data nascita;
    int  voti[ESAMI]; // array dei voti degli esami
} studente; // campo = nome, cognome, data di nascita, array di voti
```

```
studente corso[N]; // un corso è un array di N studenti
```

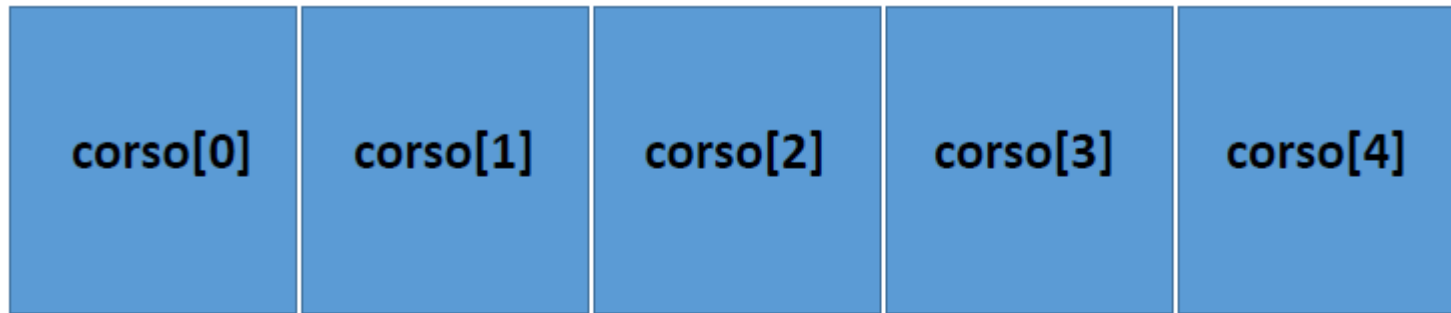
Dichiarazioni di variabili

```
// Ciclo di Acquisizione Dati
for(unsigned int i=0; i<N; i++) {
    // Leggo nome e cognome dello studente
    printf("Studente n.%d. \nInserisci nome e cognome, separati da uno spazio:", i+1);
    scanf("%s %s", &corso[i].nome, &corso[i].cognome);

    // Leggo la data di nascita dello studente
    printf("Inserisci data di nascita, separando i campi con uno spazio: ");
    // Ciclo di lettura con struct innestate una dentro l'altro
```

Immissione

Esempio



```
// Ciclo di Acquisizione Dati
for(unsigned int i=0; i<N; i++) {
    // Leggo nome e cognome dello studente
    printf("Studente n.%d. \nInserisci nome e cognome, separati da uno spazio:", i+1);
    scanf("%s %s", &corso[i].nome, &corso[i].cognome);
}
```

Immissione

Esempio

```
// Dichiaro una struct  
typedef struct {  
    char nome[10];  
    char cognome[15];  
    data nascita;  
    int voti[ESAMI];  
} studente; // campo :
```



```
// Ciclo di Acquisizione Dati  
for(unsigned int i=0; i<N; i++) {  
    // Leggo nome e cognome dello studente  
    printf("Studente n.%d. \nInserisci nome e cognome, separati da uno spazio:", i+1);  
    scanf("%s %s", &corso[i].nome, &corso[i].cognome);  
}
```

Se $i=0$, mi riferisco a `corso[0]`, che è una variabile di tipo `studente`

Immissione

Esempio

```
// Leggo la data di nascita dello studente
printf("Inserisci data di nascita, separando i campi con uno spazio: ");
// Ciclo di lettura con struct innestate una dentro l'altro
scanf("%d %s %d", &corso[i].nascita.giorno, &corso[i].nascita.mese, &corso[i].nascita
    .anno);

// Secondo ciclo del vettore, per acquisire i voti
printf("Inserisci i voti dei tuoi esami:\n");
for(unsigned int j=0; j<ESAMI; j++) {
    printf("Esame n.%d:",j+1);

    // Legge il voto del j-esimo esame per l'i-esimo studente
    scanf("%d", &corso[i].voti[j]);
}
}
```

Immissione

Esempio

```
// Ciclo di Visualizzazione Dati
for(unsigned int i=0; i<N; i++) {
    // Stampo Nome e Cognome
    printf("\nStudente n.%d: %s %s\n", i+1, corso[i].nome, corso[i].cognome);
    // Stampo data di nascita
    printf("Nato il: %d %s %d\n", corso[i].nascita.giorno, corso[i].nascita.mese, corso[i].nascita.anno);

    // Stampo voti:
    printf("Voti:");
    for(unsigned int j=0; j<ESAMI; j++) {
        printf(" %d:", corso[i].voti[j]);
    }

    puts(""); // miglioramento output
}
```

Visualizzazione

Esempio

```
// Ciclo di Visualizzazione Dati
for(unsigned int i=0; i<N; i++) {
    // Stampo Nome e Cognome
    printf("\nStudente n.%d: %s %s\n", i+1, corso[i].nome, corso[i].cognome);
    // Stampo data di nascita
    printf("Nato il: %d %s %d\n", corso[i].nascita.giorno, corso[i].nascita.mese, corso[i].nascita.anno);

    // Stampo voti:
    printf("Voti:");
    for(unsigned int j=0; j<ESAMI; j++) {
        printf(" %d:", corso[i].voti[j]);
    }

    puts(""); // miglioramento output
}
```

Visualizzazione

Esercizio

- *Scrivere un programma che acquisisca e visualizzi i dati di una collezione di 5 studenti, ciascuno descritto da <nome, cognome, data di nascita, voti di esami>*
- *Considerare casi limite per data di nascita e voti esami.*
- *Calcolare e visualizzare la media dei voti di esami per ciascuno studente*
- *Visualizzare lo studente con la media esami più alta*

Tipi di dato unione

- Tipo strutturato che consente di memorizzare in una stessa variabile una varietà di valori.
- Solo un valore alla volta può essere memorizzato e solo l'ultimo valore inserito può essere recuperato.
- Quindi, i membri di una unione condividono le stesse locazioni di memoria con il risultato di preservare memoria.

```
union Number {  
    int x;  
    float y;  
};  
union Number value;
```

Tipi di dato unione

- Accesso e Operatori:
 - assegnazione con operatore “=” tra valori *union*.
 - accesso a membri di una union: “.” e “→” (con puntatori)

Tipi di dato unione

- Esempio:

*Dichiarazione
di tipo*

*Dichiarazione
di variabili*

Elaborazione

Visualizzazione

```

1  /* Fig. 10.5: fig10_05.c
2     An example of a union */
3  #include <stdio.h>
4
5  /* number union definition */
6  union number {
7      int x;      /* define int x */
8      double y; /* define double y */
9  }; /* end union number */
10
11 int main()
12 {
13     union number value; /* define union value */
14
15     value.x = 100; /* put an integer into the union */
16     printf( "%s\n%s\n%s%d\n%s%f\n\n",
17             "Put a value in the integer member",
18             "and print both members.",
19             "int:  ", value.x,
20             "double:\n", value.y );
21

```

Tipi di dato unione

- Esempio:

Elaborazione

```

22     value.y = 100.0; /* put a double into the same union */
23     printf( "%s\n%s\n%s%d\n%s%f\n",
24             "Put a value in the floating member",
25             "and print both members.",
26             "int:  ", value.x,
27             "double:\n", value.y );
28
29     return 0; /* indicates successful termination */
30
31 } /* end main */

```

[illegible]

```
Put a value in the floating member
and print both members.
int:    0
double: 100.000000
```

Tipi di dato enumerativo

- Collezione di valori numerici costanti *int* rappresentati da identificatori simbolici, i cui valori numerici sono automaticamente impostati.
- Le variabili possono assumere come valori solo gli identificatori simbolici e non le loro rappresentazioni numeriche.
- Sugli identificatori vige l'ordinamento fornito dalle loro rappresentazioni numeriche.
- Esempi:

```
enum Months { JAN = 1, FEB, MAR, APR, MAY, JUN, JUL, AUG,  
              SEP, OCT, NOV, DEC};
```

Si è definito una collezione di costanti numerici che cominciano da 1 ed automaticamente terminano a 12, cioè sono incrementati di 1.

Tipi di dato enumerativo

- Esempio:

```
1  /* Fig. 10.18: fig10_18.c
2     Using an enumeration type */
3  #include <stdio.h>
4
5  /* enumeration constants represent months of the year */
6  enum months { JAN = 1, FEB, MAR, APR, MAY, JUN,
7               JUL, AUG, SEP, OCT, NOV, DEC };
8
9  int main()
10 {
11     enum months month; /* can contain any of the 12 months */
12
13     /* initialize array of pointers */
14     const char *monthName[] = { "", "January", "February", "March",
15                                "April", "May", "June", "July", "August", "September", "October",
16                                "November", "December" };
17
18     /* loop through months */
19     for ( month = JAN; month <= DEC; month++ ) {
20         printf( "%2d%11s\n", month, monthName[ month ] );
21     } /* end for */
22
23     return 0; /* indicates successful termination */
24 } /* end main */
```

Esercizi

Scrivere un programma che permette di

- *caricare un vettore di conti correnti bancari: ogni conto è descritto in termini del numero del conto, il titolare del conto (una variabile di tipo Persona) e del saldo;*
- *stampare l'intero archivio;*
- *dato il numero di conto effettuare il prelievo da tale conto*
- *dato il numero di conto effettuare l'accredito su tale conto*
- *dato un numero di conto stampare le informazioni del conto*

Considerare casi limite per prelievo, accredito e numero di conto in input.