



Programmazione I

Il Linguaggio C

Strutture Dati

Daniel Riccio

Università di Napoli, Federico II

18 ottobre 2021



Sommario



- Argomenti
 - Strutture astratte e concrete
 - I vettori
 - Operazioni sui vettori

Tipi di dati strutturati



I tipi considerati finora hanno la caratteristica comune di non essere **strutturati**: ogni elemento è una singola entità.

Se il programma deve trattare collezioni di dati, anche se sono dello stesso tipo, a ognuno deve essere associato un identificatore.

Supponendo di dover gestire le paghe in una ditta di 3000 dipendenti sarebbe necessario definire 3000 variabili diverse, del tipo: **operaio1**, **operaio2**, ..., **impiegato1**, **impiegato2**,, ecc.

Tipi di dati strutturati



Nella rappresentazione delle informazioni non è sufficiente considerare l'insieme dei valori, ma anche la loro **struttura**, cioè le relazioni logiche che legano tra loro i valori stessi.

Un insieme di dati e la struttura che li lega costituiscono una **struttura informativa**

La parte più piccola e indivisibile della struttura informativa si dice **elemento**.

Strutture informative



Di conseguenza è opportuno esaminare i principali tipi di aggregati dal punto di vista della struttura logica, cioè le cosiddette strutture **astratte di dati**, e i sistemi per la loro rappresentazione nella memoria di un calcolatore, cioè le possibili strutture concrete adatte a contenere le strutture astratte.

In altri termini, la formulazione di un problema è espressa tenendo conto anche del tipo di rappresentazione dei dati in memoria che si pensa di adottare

Strutture informative



Con il termine **strutture informative**, si comprendono,

- le **strutture astratte**, proprie del problema e dipendenti unicamente da questo. Esse studiano ed organizzano le relazioni logiche che intercorrono tra i dati; vengono usate per descrivere le proprietà dell'insieme dei dati indipendentemente da come questi saranno memorizzati (insieme di leggi che definiscono le relazioni esistenti fra i dati di un insieme finito);
- le **strutture concrete**, analizzano il processo dal punto di vista hardware, cioè come i dati vengono allocati nella memoria del PC. Sono le strutture interne che vengono usate per rappresentare in memoria le strutture astratte.

Strutture informative



Definire una **struttura astratta** significa stabilirne il **tipo**, precisando:

l'aspetto **statico**, ovvero

- quali sono gli elementi di base che la caratterizzano;
- quali sono le relazioni possibili tra gli elementi e come si accede ad essi

l'aspetto **dinamico**, ovvero

- le operazioni sui dati, cioè se e in quale modo è possibile operare sui dati mediante inserimenti, variazioni e cancellazioni.

Strutture informative



Un'altra classificazione prevede la distinzione tra

Struttura statica: quando il numero degli elementi che lo compongono rimane costante nel tempo. Questo significa che, una volta costruita la struttura, essa viene usata solo per operazioni di ricerca o per modificare qualche valore.

Struttura dinamica: quando il numero degli elementi può subire variazioni nel tempo (esempio: elenco telefonico, in cui si possono inserire nuovi abbonati, o cancellarne)

Strutture informative



L'**accesso** specifica il modo con cui si “raggiunge” l'informazione all'interno della struttura (attraverso operazioni di lettura e/o scrittura).

Può essere:

- **Sequenziale**: quando per raggiungere un elemento è necessario consultare tutti gli elementi che lo precedono;
- **Diretto**: quando è possibile raggiungere direttamente l'informazione desiderata, o mediante la sua posizione all'interno della struttura (es.: nei vettori tramite l'indice) o mediante altre informazioni.

Tipi di dati strutturati



I linguaggi ad alto livello mettono a disposizione dei tipi strutturati, caratterizzati sia dal tipo dei loro componenti che dai legami strutturali tra i componenti stessi, cioè dal metodo di strutturazione.

Il linguaggio C, anche in questo caso, si presenta ambivalente: permette di creare dati aggregati, senza peraltro che questi costituiscano dei tipi nell'accezione classica.

Infatti l'organizzazione strutturale dei dati e le modalità di accesso ai singoli elementi che costituiscono la struttura non vengono nascoste all'utente, che invece può interagire con esse in piena libertà.

I vettori

Il **vettore** è una collezione di variabili tutte dello stesso tipo (detto appunto **tipo base**) di lunghezza prefissata.

Questa collezione di variabili è individuata da un unico **nome**, il nome appunto del vettore.

Ogni elemento del vettore è detto **componente** ed è individuato dal nome del vettore seguito da un **indice** posto tra parentesi quadre.

L'**indice** può essere solo di tipo **intero** o **enumerato** e determina la posizione dell'elemento nel vettore.



I vettori

Variabili scalari: contengono un solo valore

Variabili vettoriali: contengono più valori dello stesso tipo, detti **elementi**

Se il tipo è di base (come char, int, long, double, etc.) gli elementi del vettore sono variabili scalari di quel tipo

Tutti gli elementi condividono lo stesso nome e sono contraddistinti da un indice indicato tra parentesi quadre:

Esempio: int **a**[4];

a[1] **a**[2] **a**[3] **a**[4]

in termini matematici questo si scrive:

a₁ , **a**₂ , **a**₃ , **a**₄



I vettori (definizione)

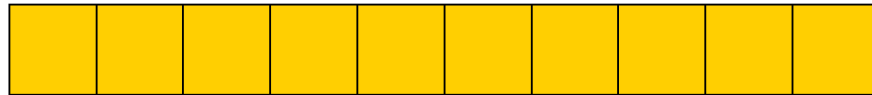
Definizione (alloca mem. per tutti gli elementi)

tipo nome[num_di_elementi];

Esempio:

int vett[10];

vett:



definisce un vettore di 10 elementi di tipo **int**

num_di_elementi deve essere una costante intera positiva nota al momento della compilazione (quindi non si può chiedere all'utente mentre il programma è in esecuzione)

La si può indicare in una **#define**

I vettori

L'indice deve essere un **valore intero** e di **tipo intero**

Il primo elemento ha indice **0**: **vett[0]**

L'ultimo elemento di un vettore di **N** elementi ha indice **N-1**, in questo esempio è **vett[9]**

Si accede ai singoli elementi indicando il nome del vettore seguito dall'indice dell'elemento tra parentesi quadre (costante o variabile):

vett[7] **vett[i]** **vett[k+1]**

I vettori



L'indice, che definisce la posizione di un elemento di un vettore, **DEVE** essere rigorosamente un intero!

Può ovviamente anche essere un'espressione, più o meno complessa, purché con risultato intero.

Esempio:

```
double x, a[30];           /* a vettore di double */
int i, j, k;
.....
x = a[2*i+j-k]; /* espressione aritmetica per l'indice */
```

```
char x, a[30];             /* a vettore di caratteri */
int i, j, k;
.....
x = a[i+j-k]; /* espressione aritmetica per l'indice */
```


I vettori

Poiché ciascun elemento del vettore è del tipo indicato nella definizione (nell'esempio è un int), può essere utilizzato in tutti i contesti in cui si può usare una variabile di quel tipo

Esempio:

```
int vett[10];  
scanf("%d", &vett[4]);  
x = vett[4] * 5;
```

N.B.

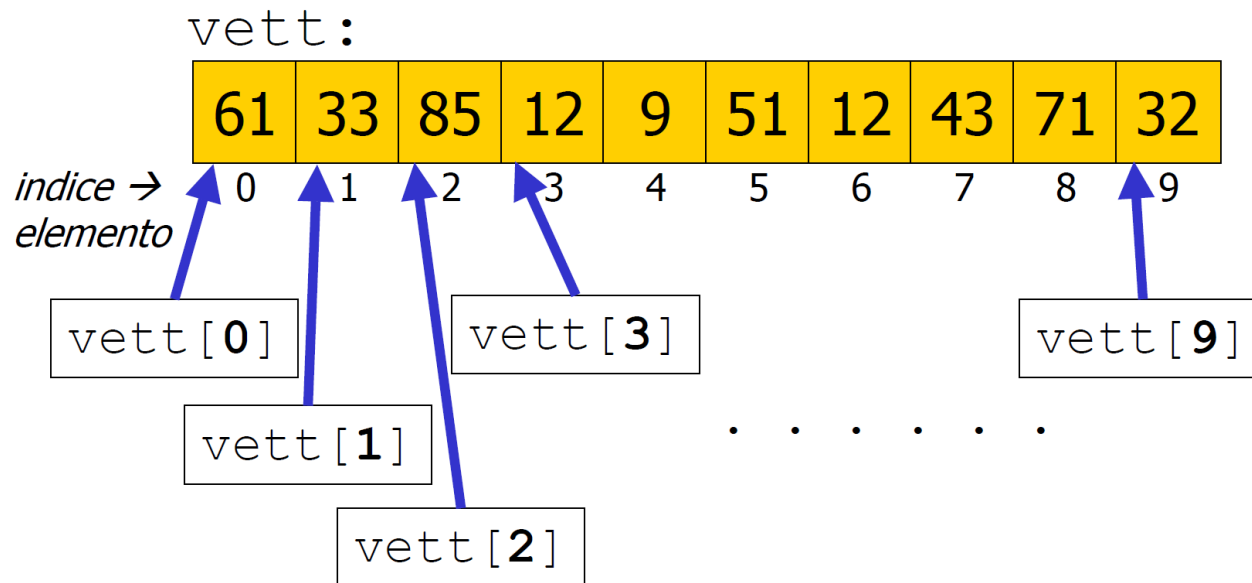


infatti **vett[4]** è un valore (scalare) di tipo **int**

I vettori

Gli elementi del vettore sono allocati in locazioni di memoria contigue e successive

```
int vett[10];
```





I vettori

Il **nome** di un vettore è usato dal compilatore come sinonimo dell'indirizzo di memoria del primo elemento del vettore (non è una variabile, per cui non si può assegnarle un valore)

Supponendo che il vettore **vett** sia di **10** elementi interi da **32 bit** e inizi in memoria dall'indirizzo A00, **vett**[0] avrà indirizzo **A00**, **vett**[1] → **A04**, ..., **vett**[9] → **A24**, l'ultimo byte → **A27**

vett:

61	33	85	12	9	51	12	43	71	32
0	1	2	3	4	5	6	7	8	9
A00	A04	A08	A0C	A10	A14	A18	A1C	A20	A24

Indirizzi crescenti →

I vettori

La presenza di un indice suggerisce che sia possibile efficacemente scandire tutti i valori di un vettore mediante un ciclo **for**

```
#define N 10
```

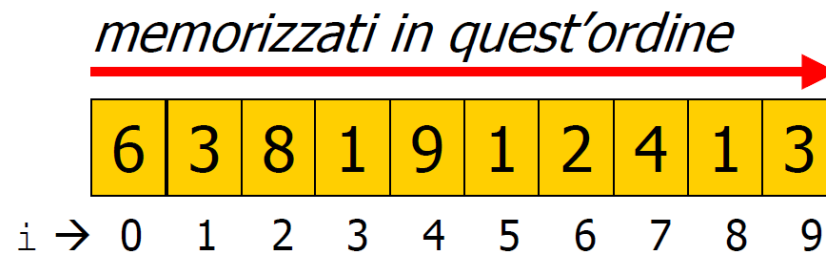
```
int vett[N];
```

```
for (i=0; i<N; i++)
```

```
    scanf("%d", &vett[i]);
```

```
for (i=N-1; i>=0; i--)
```

```
    printf("%d\n", vett[i]);
```





I vettori

Si “**sfora**” il vettore quando si accede a elementi oltre i limiti del vettore

Esempio

```
int vett[4]={1,2,3,4}, x=9, y=8;
```

```
vett[5]=12;
```

Si accede ad una porzione di memoria che è possibile sia adibita ad altro, ad esempio ad altre variabili, modificandole involontariamente

vett:				x:	y:				
1	2	3	4	9	12				
0	1	2	3						
A00	A04	A08	A0C	A10	A14	A18	A1C	A20	A24

I vettori

Esempio:

```
int vett[N];  
for (i=1; i<=N; i++)  
    scanf("%d", vett[i]);
```

Si salta l'elemento **0** e si accede all'elemento **N** che non esiste.

Lo standard non richiede che il compilatore faccia controlli su un eventuale sforamento, molti compilatori lo forniscono opzionalmente, ma ciò riduce le prestazioni in quanto ad ogni accesso al vettore viene fatto un controllo preliminare per verificare che non si sfori

I vettori

Se il vettore non è inizializzato, i valori in esso contenuti sono **indeterminati**

La lista degli inizializzatori può essere indicata tra parentesi graffe dopo un segno '='

```
int vett[4]={12, 5, 3, 6};
```

Gli **inizializzatori** devono essere noti al compile time, quindi devono essere espressioni costanti: numeri, #define, valori enum, indirizzi di memoria di variabili statiche

Non possono essere: variabili, valori const, risultati di funzioni, indirizzi di memoria di variabili automatiche

I vettori

La dimensione può essere omessa, in questo caso viene calcolata dal compilatore contando i valori:

```
int vett[]={12, 5, 3, 6};
```

ci sono 4 elementi, hanno i valori indicati

Se la lista contiene meno valori di quelli indicati dalla dimensione ma almeno uno, quelli non specificati sono inizializzati a **0**:

```
int vett[4]={6, 2};
```

i 4 valori sono 6, 2, 0, 0

Per inizializzare a 0 tutto un vettore:

```
int vettore[10]={0};
```

Il primo valore **0** è indicato esplicitamente, i successivi sono posti a **0** automaticamente

I vettori

Sugli elementi del vettore agiscono gli operatori previsti per il **tipo_componente**. Pertanto è lecito scrivere:

```
valor_fin = vett_x[m1] + vett_y[m2];
```

purché, naturalmente, **valor_fin**, il vettore **vett_x** e il vettore **vett_y** siano dello stesso tipo.

Il tempo necessario per accedere a un elemento di un vettore è indipendente dal valore dell'indice: il vettore è pertanto una

struttura ad accesso casuale

I vettori



Non ci sono operatori che agiscono sul vettore nel suo complesso

Non è lecito pertanto l'assegnamento di un vettore ad un altro vettore.

Se **vett_x** e **vett_y** sono vettori, l'istruzione:

vett_x = vett_y;

è errata anche se **vett_x** e **vett_y** sono dello stesso tipo.

Per trasferire un vettore in un altro occorre copiare un elemento per volta.

I vettori

Quando si definisce un vettore il compilatore riserva un'area di memoria sufficiente per contenerlo e associa l'indirizzo iniziale di quell'area al nome simbolico (identificatore) da noi scelto per il vettore.

Pertanto il nome **vett_dati** non è una vera e propria variabile, ma piuttosto un **puntatore**: in pratica **vett_dati** è l'**indirizzo** di memoria del primo elemento del vettore cioè l'indirizzo di **vett_dati[0]**.

Ecco perché è errata l'istruzione:

```
vett_x = vett_y;
```

I vettori



I cicli sono particolarmente utili per implementare operazioni su un vettore.

Utilizzo tipico: applicazione iterativa di un'operazione sugli elementi di un vettore.

Schema:

```
int data[10], ind;
```

```
.....
```

```
for (ind=0; ind<10; ind++)
```

```
{
```

```
    elaborazione dell'elemento data[ind]
```

```
}
```

Ad ogni *ciclo* è interessato l'elemento individuato dall'indice **ind**.

Leggere un vettore da input

Leggere 10 valori da tastiera e memorizzarli in un vettore.
Quindi calcolarne il minimo ed il massimo.



Algoritmo

definisci un vettore **vettdati** di **10** elementi
definisci **minimo** e **massimo**

for ind che va da 0 a 9

leggi(dato)

vettdati[ind] ← dato

minimo=**massimo**=**vettdati**[0]

for ind che va da 0 a 9

 se **vettdati**[ind] > **massimo**

massimo = **vettdati**[ind]

 altrimenti se **vettdati**[ind] < **minimo**

minimo = **vettdati**[ind]

stampa(**minimo** e **massimo**)

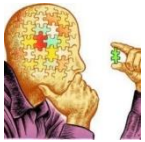


Programma

```
main.c Ctrl+B
1  #include <stdio.h>
2  #define NUMDATI 10
3
4  main ()
5  {
6      int minimo, massimo, ind;
7      int vettdati[NUMDATI];
8
9      /* lettura dei dati */
10     for (ind = 0; ind < NUMDATI; ind++){
11         printf ("Introduci vettdati[%d]: ", ind);
12         scanf ("%d", &vettdati[ind]);
13     }
14
15     /* cerca il massimo e il minimo */
16     massimo = vettdati[0];
17     minimo = vettdati[0];
18     for (ind = 1; ind < NUMDATI; ind++){
19         if (vettdati[ind] > massimo)
20             massimo = vettdati[ind];
21         else if (vettdati[ind] < minimo)
22             minimo = vettdati[ind];
23     }
24     printf ("\nIl massimo e' %d e il minimo e' %d \n ", massimo, minimo);
25 }
```

I vettori

Scrivere un programma che legga un numero decimale positivo minore di 1024 e lo converta nella corrispondente codifica binaria



Ragionamento

Per convertire in binario puro un numero decimale occorre eseguire una sequenza di divisioni per **2** prendendo i resti (0 oppure 1): occorre dunque un vettore per memorizzare questi resti.

Poiché i numeri devono essere compresi tra **0** e **1023** sono sufficienti **10** bit: il nostro vettore sarà pertanto lungo **10** elementi e in ogni elemento memorizzeremo una cifra.

I vettori

Scrivere un programma che legga un numero decimale positivo minore di 1024 e lo converta nella corrispondente codifica binaria



Ragionamento

I resti ottenuti dalle divisioni per 2 vanno però letti al contrario, conviene pertanto riempire il vettore a partire dall'ultimo elemento.

Per eseguire le divisioni per due conviene servirsi di un ciclo il quale, ad ogni iterazione, calcola un nuovo bit (resto della divisione per 2).

for o **while**? È pressochè indifferente usare un ciclo for o un ciclo while. Occorre però che le inizializzazioni delle variabili siano adattate al ciclo prescelto.

Esempio con while



Programma

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

```
int main()
```

```
{  
    int ind, numero, num;  
    int binario[10];
```

```
/* inizializza il vettore risultato con tutti zeri */
```

```
for(ind=0; ind<10; ind++)  
    binario[ind] = 0;
```

```
printf("\nIntroduci un numero intero positivo minore di 1024: ");  
scanf("%d", &numero);
```

N.B. equivale a

```
for(ind=0; ind<10; binario[ind++]=0);
```

Esempio con while



Programma

```
if ( (numero >= 0) && (numero < 1024) ) {  
  
    num = numero;      /* num è il "dato-guida" del ciclo */  
    ind = 9;  
  
    while (num != 0) {    /* finché num è diverso da 0! */  
  
        binario[ind] = num % 2; /* calcola un nuovo bit */  
        num /= 2;             /* aggiorna num per il prossimo ciclo */  
        ind--;                /* aggiorna l'indice del vettore */  
    }  
  
    printf("\nConversione del numero %d:  ", numero);  
  
    for (ind=0; ind<10; ind++)          /* Visualizza il vettore: */  
        printf("%1d", binario[ind]); /* un bit per volta */  
} else  
    printf("\nNumero non lecito!");  
}
```


Esempio con for



Programma

```
#include <stdio.h>

int main()
{
    int ind, numero, num;
    int binario[10];

    /* non è necessario inizializzare il vettore in quanto il ciclo for deve
       scrivere comunque tutti gli elementi del vettore */

    printf("\nIntroduci un numero intero positivo minore di 1024: ");
    scanf("%d", &numero);
```

Esempio con for



Programma

```
if ((numero >= 0) && (numero < 1024))
{
    num = numero;
    for (ind = 9; ind >= 0; ind--){
        /* con un indice che va da 9 a 0 */

        binario[ind] = num % 2;      /* calcola un nuovo bit */
        num /= 2;                    /* aggiorna num per il prossimo ciclo */
    }
    printf ("\nConversione del numero %d:  ", numero);
    for (ind = 0; ind < 10; ind++)    /* Visualizza il vettore: */
        printf ("%1d",binario[ind]); /* un bit per volta! */
} else
printf ("\nNumero non lecito!");
}
```

Conversione decimale-binario

Convertire un valore decimale in binario



```
✓ ↗ 📋  
Introduci un numero intero positivo minore di 1024: 127  
Conversione del numero 127: 000111111
```

Esercizi



1. Copia di un vettore

Scrivere un programma che prenda in input N valori, li inserisca in un vettore e poi ne faccia una copia in un nuovo vettore.

2. Stampa di valori in ordine crescente

Scrivere un programma che prenda in input N valori, li inserisca in un vettore e poi li stampi in ordine crescente.

- *Ricerca e stampa del minimo N volte.*

