ARRAY E STRINGHE

Programmando in C abbiamo imparato a utilizzare variabili e operatori. Facciamo un ulteriore passo avanti scoprendo un nuovo modo per rappresentare i dati: i vettori.

In informatica è normale aver la necessità di elaborare grandi moli di dati, che vengono acquisite nel tempo in modo non necessariamente prevedibile. Pensiamo, ad esempio, a un software finanziario che produce statistiche sull'andamento dei prezzi azionari: esso deve essere in grado di archiviare e memorizzare i dati in modo da poter fornire agli operatori tutte le informazioni necessarie in qualsiasi momento vengano richieste. Se tralasciamo le soluzioni avanzate basate sull'archiviazione fisica in basi dati più o meno strutturate (dal semplice file di testo memorizzato su hard disk alle complesse basi dati relazionali e a oggetti dei moderni database), gli strumenti attualmente a nostra disposizione non ci permettono di affrontare un problema del genere in modo automatizzato. Ciò che potremmo invece fare è crare un numero finito di variabili da gestire come entità indipendenti, ma tale soluzione è estremamente scomoda e di poca utilità. Servono, invece, strumenti che permettano di automatizzare i processi di elaborazione ciclica dei dati. Due esempi di questi strumenti sono gli **array** (detti anche '**vettori**') e le **strutture dati dinamiche**, che però non affronteremo per via della loro difficoltà.

COS'È UN ARRAY? ›››

Un array (vedi schema sotto) **è una struttura dati complessa costituita da un insieme sequenziale finito di dati elementari o strutturati, aventi tutti lo stesso tipo.** Ogni array è caratterizzato da una **dimensione**, che definisce il **numero di celle** di cui è costituito. **Ogni cella dell'array corrisponde a una singola variabile**, accessibile attraverso l'uso di un **indice**, che permette di indicare al programma la cella della struttura dati che si intende utilizzare. Nel linguaggio C la convenzione è quella di **indicizzare la prima cella di ogni array di N elementi con il valore '0' e l'ultima con il valore 'N-1'**. Nota bene che l'uso di valori di indice superiori a 'N-1' non è segnalato come errore dal compilatore, tuttavia può produrre errori in fase di esecuzione e arresti del programma. Un uso corretto degli indici, quindi, è di importanza fondamentale per il funzionamento del software. Ma passiamo alla pratica.

A sinistra, una rappresentazione schematica di un vettore di N elementi. In linguaggio C, la prima cella di ogni vettore viene indicizzata con il valore 0, mentre l'ultima è accessibile con indice 'N-1'.

DICHIARARE UN ARRAY

La dichiarazione di un array è molto simile a quella di una variabile. Inizia, come abbiamo visto finora, esplicitando il **tipo dei dati memorizzati**, seguito, come sempre, dal **nome mnemonico** che vogliamo assegnare. Come ultima cosa va indicata la **dimensione** (ossia il numero di celle che costituiscono il vettore che stiamo inizializzando) dimensione che deve essere **espressa per mezzo di una costante numerica racchiusa tra parentesi quadre**. La sintassi generica della dichiarazione è mostrata qui di seguito.

```
tipo nome_vettore
[dimensione];
```

Come tutte le variabili, anche **i vettori in C devono essere dichiarati prima di passare alla scrittura del codice operativo.** La dimensione del vettore, inoltre, deve essere un **valore costante intero.** È accettato quindi sia l'utilizzo di valori interi numerici, sia il ricorso a costanti. Non è invece consentito definire la dimensione facendo uso di una variabile (vedi il box soprastante).

Regole di dichiarazione dei vettori

SI

Dimensione tramite valore esplicito

```
int vettore [10];
```

Dimensione tramite costante

```
const int dimensione = 10;
int vettore [dimensione];
```

NO

Dimensione tramite variabile

```
int dimensione = 10;
int vettore [dimensione];
```

ACCEDERE A UN ARRAY

Con la dichiarazione di un array, come abbiamo visto, abbiamo a disposizione una serie sequenziale di variabili, a cui è, ovviamente, necessario poter accedere. L'accesso alle singole celle dell'array è consentito attraverso l'uso delle parentesi quadre, che hanno funzione di **operatore di indicizzazione.** Per poter lavorare su una ben precisa cella del vettore è sufficiente **far seguire al nome dell'array la posizione della cella racchiusa tra due parentesi quadre.** A differenza della fase dichiarativa del vettore, durante l'utilizzo dell'array tra le parentesi quadre può essere utilizzata una variabile

Lettura e scrittura di un vettore

ACCESSO IN LETTURA

Uso di una costante

```
var =  vettore[4];    /* in var viene copiato il valore della 5° cella del vettore */
```

Uso di una variabile

```
int i = 2;
var = vettore [i];    /*i è inizializzato a 2, quindi in var viene copiato il valore
                        della terza cella dell'array */
```

ACCESSO IN SCRITTURA

Uso di una costante

```
vettore[3] = var;    /* nella 4° cella dell'array viene copiato il valore della
                        variabile var */
```

Uso di una variabile

```
int i = 10;
vettore[i] = var;     /* nell'undicesima cella del vettore viene copiato il valore
                         della variabile var */
```

o una qualsiasi espressione matematica che produca un risultato intero. Come già accennato nei paragrafi precedenti, l'indice racchiuso tra le parentesi quadre può assumere solo valori compresi tra 0 e N-1 (inclusi), dove N è la dimensione dell'array. Nel box in fondo alla pagina precedente puoi vedere mostrati alcuni esempi di accesso in lettura e in scrittura da un array pre-dichiarato (il codice per la dichiarazione è stato omesso).

LE STRINGHE ›››

Seguendo i Workshop che si occupano argomenti di programmazione avrai sicuramente notato che non abbiamo mai utilizzato variabili che ci permettessero di rappresentare testi o parole. Non esiste, infatti, un tipo di dato primitivo che consenta di dichiarare una '**stringa**' (come sono chiamate le sequenze di caratteri). La gestione di questo formato di dati, al contrario, è affidata ai vettori e, in particolare, agli **array di caratteri**. Convenzionalmente, in C **una stringa è un array di caratteri terminato da un 'carattere tappo'**. Tale carattere corrisponde al **valore numerico '0'** (carattere **NULL** della tavola ASCII) e nel linguaggio C viene indicato con il simbolo '**\0**'. Tieni presente che **una stringa non deve essere necessariamente lunga quanto il vettore che la ospita.** Un vettore da 100 caratteri può, quindi, ospitare qualsiasi stringa alfanumerica di dimensione inferiore o uguale ai 99 caratteri (il centesimo, infatti, verrebbe occupato dal terminatore). Ricorda, comunque, che **lo spazio occupato in memoria è legato esclusivamente alla dimensione del vettore.** Quindi se è dichiarato un vettore di caratteri con dimensione uguale a 100, esso occuperà 100 byte (ogni variabile char richiede un byte) indipendentemente dalla stringa contenuta. Nello schema sotto puoi osservare una rappresentazione della stringa "Ciao" all'interno di un vettore di caratteri. Negli StepbyStep che seguiranno, potrai sperimentare il funzionamento degli array applicandoli sia a sequenze di numeri interi, sia alla manipolazione delle stringhe.

La stringa "Ciao" rappresentata in C. È composta da quattro caratteri più uno di terminazione ('\0'). Per memorizzare una stringa di N caratteri è necessario un vettore di dimensione pari ad almeno N+1.

LA COPIA DEI VETTORI ›››

Una nota particolare deve essere fatta in merito alla copia dei vettori. Lavorando con le variabili abbiamo sempre effettuato la copia dei valori attraverso l'operatore di assegnamento '='. Per i vettori questo non è fattibile, o quanto meno, non produce i risultati che ci si potrebbe aspettare. Un array è, infatti, sostanzialmente una variabile che non contiene fisicamente i dati su cui stiamo lavorando, bensì l'indirizzo della porzione di memoria da cui ha inizio lo spazio di memorizzazione (questa variabile particolare prende il nome di 'puntatore'). L'operazione di copia 'classica' vista finora non crea, quindi, una copia del vettore, in quanto replica solo l'indirizzo di riferimento in memoria e non il contenuto. Dà invece origine a un alias, ossia a una variabile apparentemente distinta dalla sorgente di copia, ma che in realtà condivide con essa lo stesso spazio di memoria e gli stessi dati. La copia di due vettori deve essere effettuata, invece, cella per cella in modo iterativo, ricorrendo all'uso delle strutture cicliche. Un esempio di copia che fa uso del ciclo for è il seguente:

```
for(i=0; i<lunghezza_vettori; i++) {
   vettore_dest[i] = vettore_sorg[i];
   }
```

STEPbySTEP

UN SEMPLICE ANALISI STATISTICA DELLE TEMPERATURE DELLA SETTIMANA

In questo esempio all'utente verrà chiesto di inserire 7 numeri decimali, rappresentanti le temperature lette durante i sette giorni della settimana (mostrati in forma numerica da '1', lunedì, a '7', domenica). I dati inseriti dall'utente vengono memorizzati in un array e, successivamente, utilizzati per calcolare la media.

La media di un vettore numerico

```
#include <stdio.h>

int main ()
{
    int i=0;  /* variabile indice */
    float temperature [7];  /* vettore di memorizzazione delle
                               temperature */
    float media=0;  /* media delle temperature */
    float valoreletto;  /* il valore letto da letteratura */

    for(i=0; i<7; i ++)  /* memorizza le 7 temperature nel
                            vettore */
    {
        printf("\nInserisci la temperatura del giorno %d: ", i+1);
        scanf("%f", &valoreletto);     /* acquisisce la temperatura */
        temperature[i] = valoreletto;       /* memorizza la temperatura
                                               nel vettore */
     }


     /* effettua il calcolo della media sui dati del vettore */
     for(i=0; i<7; i++)
     {
        media+=temperature[i]; /* somma i valori del vettore */
     }

     media /= 7; /* divide la variabile per 7 in modo da ottenere la
        media matematica dei valori */

     /* stampa il risultato a video */
     printf("\n\nLa media della settimana è stata :%3.3f", media);

     /* attende la pressione di un tasto */
     fflush(stdin);
     getchar();
    return 1;
}
```

UN CLONE DELLA FUNZIONE STRLEN

In C esiste una libreria dedicata alla manipolazione delle stringhe chiamata string.h. Tra le funzioni più note di questa libreria troviamo la strlen, una funzione che calcola la lunghezza della stringa 's' passata come parametro (la lunghezza è intesa come numero di caratteri della stringa, escluso il carattere finale '\0'). In questo esempio definiremo la funzione equivalente personalizzata MyStrlen e la utilizzeremo per contare i caratteri delle stringhe inserite dall'utente.

La funzione MyStrlen

```
#include <stdio.h>

/* la funzione MyStrlen */
int MyStrlen (char * str) /* 'char *' serve per indicare che la
      funzione riceverà come parametro un vettore di caratteri*/
{
    int len;  /* la variabile */

       /* incrementa len finchè non viene individuato il terminatore
        della stringa (carattere '\0') */
    for (len = 0 ; str[len] != '\0' ; len++);

       /* restituisce il valore len che rappresenta la lunghezza
       della stringa */
    return len;
}

/* programma principale */
int main()
{
/* definiamo un array di 100 caratteri */
char stringa [100];
int lunghezza; /* lunghezza della stringa letta */

printf("\nScrivi una stringa di caratteri e premi invio (massimo 100):
");

/* lettura della stringa dall'utente. Da notare l'utilizzo del segnaposto
%s che indica alla scanf la lettura di una stringa
e l'assenza dell'operatore & nella seconda parte della scanf.
I vettori, infatti, sono rappresentati tramite il loro indirizzo
di memoria e, quindi, la lettura delle stringhe non necessita
dell'uso di tale operatore.*/

scanf("%s", stringa);

/* chiamata della funzione definita in precedenza passando
la stringa letta come parametro. Il valore restituito viene
memorizzato nella variabile "lunghezza" */
lunghezza = MyStrlen(stringa);
/*stampa a video della stringa letta */
printf("\nLa stringa inserita è lunga %d caratteri", lunghezza);
fflush(stdin);  /* attende la pressione di un tasto */
getchar();
return 1;
}
```