VETTORI

Finora abbiamo trattato variabili come contenitori individuali di valori, se invece volessimo contenere più valori in una variabile, dobbiamo ricorrere ai vettori, detto in gergo *array*. Un vettore è semplicemente l'indirizzo in memoria del primo elemento contenuto nel vettore stesso. Per dichiarare un vettore bisogna indicare il tipo di dati che dovrà contenere, un nome ed una dimensione tra parensi quadre. Come per le variabili, esso può essere inizializzato o meno e, se acceduto non inizializzato, conterrà valori insignificanti.
Ad esempio: `int anni[10];` significa *"alloca spazio in memoria per 10 interi e associa il nome anni* ***anni*** *all'indirizzo del primo blocco "*.
Per riempire il vettore potremmo scrivere: `anni[0] = 1997` che il compilatore interpreterà come "metti 1997 all'indirizzo di anni", nel caso di `anni[1] = 1998` il compilatore capirà "*metti 1998 in indirizzo di anni + 1, ovvero anni + 4 byte più in là*". In questo modo `1997` e `1998` si trovano esattamente affiancati a blocchi di 4 byte ciascuno (come detto prima, un intero occupa 4 byte).
Nel caso di un carattere char avremo 1 byte per carattere: `char nome[10] = "Andrea"` quindi occuperà esattamente 10 byte di memoria.
Questo vettore di tipo `char` viene anche detto stringa, perché contiene una stringa di caratteri. Quando inizializzato in questo modo, il resto del vettore viene riempito di speciali caratteri ASCII "null bytes" (`\0`), che sostanzialmente servono per indicare carateri vuoti; quindi `nome[]` in realtà contiene "`Andrea\0\0\0\0\0`" e `printf()` prontamente omette gli `\0` nella stampa.

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

int main(){

  //dichiarazione del vettore di float (4 byte) da 3 posti
  float numeri[3];
  int i;

  //scrivo nelle posizoni 0,1,2
  numeri[0] = 1.5;
  numeri[1] = 3.5;
  numeri[2] = 0.5;

  //accedo alle posizioni 0,1,2 con un bel for
  for(i=0;i<3;i++)
    printf("numeri[%d]: %.1f\n", i , numeri[i]);

  //dimensione del vettore:
  printf("numeri e' grande %ld byte\n", sizeof(numeri));

  // Inizializzo stringa
  char nome[25] = "Alessandro";
  char nome2[25];
  printf("nome contiene: %s\n",nome);

  /* nome = "Alessandro"; e' errato perche' cerco di assegnare dei caratteri
   ad un indirizzo in memoria; occorre invece usare strcpy(destinaz,sorgente)
    di string.h */

  strcpy(nome2,"Francesco");
  printf("nome2 contiene: %s\n",nome2);

  return 0; }
```

Funzioni

Una delle idee fondamentali della programmazione nonché dell'ingegneria in generale è il concetto della modularità, cioè la possibilità che un sistema complesso possa essere diviso in componenti più piccoli e semplici, in programmazione dovrebbero essere riutilizzabili.
In ogni programma scritto finora abbiamo implicitamente utilizzato il concetto di modulo, cioè la *funzione* `main()`, che è la funzione *principale* del nostro programma. Quindi a livello di Sistema Operativo il nostro programma non è altro che un'altra funzione, un modulo ri-eseguibile facente parte di un complesso sistema che è il S.O. stesso. Anche una nostra applicazione è generalmente abbastanza complessa da richiedere un approccio modulare.

Per creare una funzione, ci rifacciamo alle funzioni matematiche: sia $y = f(x)$, abbiamo uno o più argomenti, x, un nome di convenienza $f()$, ed un risultato "di ritorno", y.
Andiamo quindi a definire una funzione tutta nostra, **fuori e prima del main**, come:

```
1 tipo_di_ritorno nome_funzione(tipo_argomento argomento, ...) {
2   return variabile/valore;
3 }
```

questa si chiama firma della funzione, ovvero il suo nome ed argomenti, la identifica univocamente all'interno del programma. Si potrebbe anche solo dichiarare una funzione prima del main con tipo di ritorno e firma, chiamato prototipo, ed implementarla dopo il main.

Modifichiamo ora l'esempio precedente dei vettori con l'uso delle funzioni, ad esempio, deleghiamo il riempimento dei numeri di un vettore di float ad una apposita funzione:

```
1 void riempiVettore(float vett[], int dim) {
2
3 }
```

In questo caso particolare, notiamo che il tipo di ritorno è `void`, vuol dire che la funzione non ritorna nulla, infatti non è presente l'istruzione `return`.
Vogliamo ora implementare ciò che si evince dal nome dato alla funzone; per svolgere tale compito sono necessari un vettore e la sua dimensione. Questi vengono *passati* alla funzione alla riga 33.

```
1 void riempiVettore(float vett[], int dim) {
2     vett[0] = 1.52; vett[1] = 3.55; vett[2] = 0.59;
3 }
```

Ci sono due modi per passare parametri ad una funzione: passaggio per riferimento e passaggio per valore.
Nel primo caso ciò che viene dato in pasto alla funzione è l'indirizzo della variabile che si intende modificare; si dice che *la funzione chiamata può scrivere nella variabile passata dal chiamante.* In questo modo il chiamato può modifcare una variabile appartenente al chiamante.
Nel nostro esempio è il caso di `numeri[]`, infatti ricordiamo che un vettore è rappresentato dall'indirizzo della prima variabile, ergo è un puntatore, perciò stiamo effettuando un passaggio per riferimento: infatti stampando l'indirizzo di `numeri[]` (riga 35) esso coinciderà con quello di `vett[]` (riga 20) che si trova dentro la funzione `stampaNumeri`.
Per quanto riguarda il passaggio per valore, viene semplicemente effettuata una copia del valore della variabile data in pasto al chiamato e l'originale non può essere modificato esternamente. Nel caso della variabile `dim` abbiamo un passaggio per valore, ovvero il valore di `dim` (3) viene letteralemente copiato in un'altra variabile locale della funzione `riempiVettore()`, nonostante si chiami anche essa `dim`.
Come prova vengono stampati gli indirizzi delle due variabili `dim`, e stavolta notiamo che sono diversi.

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

void riempiVettore(float vett[], int dim) {

  vett[0] = 1.52;
  vett[1] = 3.55;
  vett[2] = 0.59;

}

void stampaNumeri(float vett[], int dim) {

  int i;

  printf("-- stampo da dentro stampaNumeri() --\n");
  for(i=0;i<dim;i++)
    printf("vett[%d]: %.1f\n", i , vett[i]);

  printf("Indirizzo di vett[]: %p\n", vett);
  printf("Indirizzo di dim: %p\n", &dim);
    printf("-- fine da dentro stampaNumeri() --\n");

}

int main(){

  //dichiarazione del vettore di float (4 byte) da 3 posti
  int dim  =3;
  float numeri[dim];

  //scrivo nelle posizoni 0,1,2
  riempiVettore(numeri,dim);

  printf("Indirizzo nel main di numeri[]: %p\n", numeri);
  printf("Indirizzo nel main di dim: %p\n", &dim);

  //accedo alle posizioni 0,1,2 con un for
  stampaNumeri(numeri, dim);

  //dimensione del vettore:
  printf("numeri e' grande %ld byte\n", sizeof(numeri));

  // Inizializzo stringa
  char nome[25] = "Alessandro";
  char nome2[25];
  printf("nome contiene: %s\n",nome);

  /*
    nome = "Irene"; e' errato perche' cerco di assegnare dei caratteri ad un indirizzo in memoria; nel caso occorre usare strcpy(destinaz,sorgente) di string.h
  */

  strcpy(nome2,"Francesco");
    printf("nome2 contiene: %s\n",nome2);

  return 0;
}
```