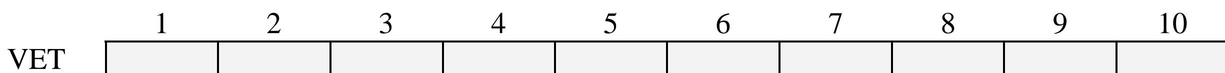
Gli Array

Un array rappresenta una variabile indicizzata (ovvero contenente un indice) che viene utilizzata per contenere più elementi dello stesso tipo. Ogni array ha un nome al quale viene associato un indice che individua i singoli elementi dell'aray. In C++ è possibile creare array di qualunque tipo: caratteri, interi, float, double, puntatori e anche array (ovvero array di array, detti array multidimensionali).

Le proprietà fondamentali di un array
Possiamo dire che un array ha quattro proprietà fondamentali:

- Gli oggetti che compongono l'array sono denominati elementi;
- Tutti gli elementi di un array devono essere dello stesso tipo;
- Tutti gli elementi di un array vengono memorizzati uno di seguito all'altro nella memoria del calcolatore;
- Il nome dell'array è un valore costante che rappresenta l'indirizzo di memoria del primo elemento dell'array stesso.

L'array ad una dimensione viene comunemente chiamato "vattore" e viene rappresentato nel modo seguente:



Per individuare un elemento del vettore si indicherà il nome della struttura e la posizione in cui si trova l'elemento. Ad esempio, per indicare il contenuto dell'elemento della posizione 4 si scriverà VET[4].

Dichiarazione di un array

Per dichiarare un array, come per ogni altra dichiarazione in C++, si deve scrivere il tipo, seguito da un nome valido e da una coppia di parentesi quadre che racchiudono un'espressione costante. Tale espressione costante definisce le dimensioni dell'array. Ad esempio:

```
int array_uno[10]; // Un array di 10 interi
char array_due[20]; // Un array di 20 caratteri
string array_tre[20];// Una array di 20 stringhe
```

Si noti che all'interno delle parentesi quadre non è possibile, in fase di dichiarazione dell'array, utilizzare nomi di variabili . E' possibile, per specificare le dimensioni di un array, usare delle costanti definite, come ad esempio:

```
#define MAX_1 10  //viene creata una costante alla quale viene assegnato il valore 10
#define MAX_2 20  //viene creata una costante alla quale viene assegnato il valore 20

int array_uno[MAX_1];// è l'equivalente di int array_uno[10];
char array_due[MAX_2];
```

L'utilizzo di costanti per definire le dimensioni di un array è una pratica altamente consigliata. Infatti, uno degli errori che spesso accadono nella manipolazione degli elementi di un array è quello di far riferimento ad elementi che vanno oltre il limite della dimensione dell'array precedentemente definita.
Il C++ (e lo stesso C), come magari si sarebbe portati a pensare, non effettua nessun controllo sugli indici che si utilizzano quando si eseguono operazioni sugli array. Per cui, ad esempio, se eseguiamo il seguente codice:

```
int array_tre[10];
......
...... // L'array viene inizializzato
int i;
for(i=1; i < 14; i++)
{
cout << array_tre[i] << endl;
}
```

il risultato sarà corretto fino a quando la variabile i sarà minore o uguale a 9 (infatti la dimensione dell'array è stata definita essere uguale a 10). Quello che avverrà, quando il programma cercherà di andare a leggere i valori array_tre[11], array_tre[12] e array_tre[13] è assolutamente casuale. Infatti, il programma andrà a leggere delle aree di memoria che non sono state inizializzate o, addirittura, che contengono dei valori relativi ad altre variabili, stampando quindi sullo schermo risultati inaspettati. Se si fosse usata una costante per definire la dimensione dell'array questo errore sarebbe stato facilmente evitato. Infatti:

```
#define MAX_3 10

int array_tre[MAX_3];
......
...... // L'array viene inizializzato
int i;
for (i=0; i < MAX_3; i++)
{
cout << array_tre[i] << endl;
}
```

Come si può notare facilmente in questo modo il programmatore non deve preoccuparsi di ricordarsi la dimensione dell'array “array_tre” poichè la costante ARRAY_trE_MAX viene utilizzata proprio per tale scopo, evitando i cosiddetti errori di "array out of bounds".

Inizializzazione di un array

Un array può essere inizializzato in due modi:

- Esplicitamente, al momento della creazione, fornendo le costanti di inizializzazione dei dati.
- Durante l'esecuzione del programma, assegnando o copiando dati nell'array.

Vediamo un esempio di inizializzazione esplicita al momento della creazione:

```
int array_quattro[3] = {12, 0, 4};
char vocali[5] = {'a','e','i','o','u'};
float decimali[2] = {1.329, 3.34};
```

Per inizializzare un array durante l'esecuzione del programma occorre accedere, generalmente con un ciclo, ad ogni elemento dell'array stesso ed assegnargli un valore. L'accesso ad un elemento di un array avviene indicando il nome dell'array e, tra parentesi quadre, l'indice corrispondente all'elemento voluto. Così, VET[3] indicherà il terzo elemento dell'array VET .
Vediamo ora un esempio di inizializzazione eseguita durante l'esecuzione del programma:

```
int numeri_pari[10];
int i;

for(i =1; i < 10; i++)
{
numeri_pari[i] = i * 2 ;
}
```

Il programma precedente non fa altro che assegnare all'array i numeri pari da 2 ad 18.

Esempio:

Inserire N numeri positivi in un vettore:

```
K=1;
do
{
cout<<”Inserisci il “<<K<<”^ elemento”;
cin>>NUM;
vet[K}=NUM;
K=K+1;
}
while (K<=N);
```

Il programma non è corretto. Infatti nella iterazione viene richiesto l’inserimento di un valore numerico e lo si inserisce comunque nel vettore. La versione corretta potrebbe essere la seguente:

```
K=0;
do
{
cout<<”Inserisci il “<<K<<”^ elemento”;
cin>>NUM;
if (NUM>0)
  {
     K=K+1;
    Vet[K] = NUM;
  }
}
while(K<=N);
```

Vettori e sottoprogrammi

L'utilizzo di un vettore in un sottoprogramma pone un vincolo: non possono essere "passati" per valore. Infatti C++ per ottimizzare l'uso della RAM non consente il passaggio di parametri per valore se si tratta di strutture dati.
Sarebbe quindi errato scrivere:

```
void CARICA(int &vetx[], int N);
```

si deve invece scrivere:

```
void CARICA(int vetx[], int N)
```

In questo caso la varibile N viene passata per valore, l'array per riferimento.

Array bidimensionali (Matrici)

Un array bidimensionale, detta anche matrice, ha le stesse caratteristiche di un array ad una dimensione. L'unica differenza è che occorrono due indici per identificare un elemento della struttura, il primo indice si chiama indice di riga, il secondo indice di colonna.
Si può pensare ad una tabella a due entrate (tipo battaglia navale) dove per individuare un punto nella griglia si devono fornire la riga e la colonna all'incrocio delle quali si viene a trovare.
Avendo compreso l'utilizzo dei vettori, una matrice può essere vista come un vettore di vettori, in cui ogni elemento è un vettore di un certo numero di elementi.
Esempio: 3 vettori di 10 elementi ciascuno:

	1	2	3	4	5	6	7	8	9	10
VET1										

	1	2	3	4	5	6	7	8	9	10
VET2										

	1	2	3	4	5	6	7	8	9	10
VET3										

Se fossero separati, per ogni vettore dovremmo specificare il nome, mantenendo invariata la dimensione (10 elementi).
Risulterebbe complicato specificare quale vettore considerare per poter individuare l'elemento interessato.
Inoltre il programma che utilizzasse questa modalità di lavoro non risponderebbe ai requisiti di propri di una applicazione; infatti la soluzione proposta non deve essere utilizzabile solo per risolvere "il problema" ma una classe di problemi.
La soluzione è quella di unire i vettori in un'unica struttura (appunto la matrice), in cui con un indice indichiamo il vettore da dove prelevare il dato, con un altro invece indichiamo la posizione all'interno del vettore dove si trova l'elemento interessato. In questo modo all'avvio dell'applicazione, sarà sufficiente indicare quanti vettori (numero di righe) e quanti elementi in ciascun vettore (numero di colonne). Il programma creerà in memoria la struttura ad hoc per quella sessione di lavoro.

Nel nostro caso:

	1	2	3	4	5	6	7	8	9	10
1										
2				12						
3										

MAT

La sua dichiarazione potrebbe essere:
int MAT[3][10]; //matrice con 3 righe e 10 colonne : ogni elemento è un numero intero
char MATX[3][10] ; // matrice con 3 righe e 10 colonne : ogni elemento è un carattere.

Ad esempio l'istruzione MAT[2][4] = 12 assegna all'elemento che si trova nella seconda riga e quarta colonna il valore 12.

Matrici e sottoprogrammi

Per poter inviare come parametro una matrice ad un sottoprogramma, è necessario definire un "tipo" di dato ad hoc in modo che possa essere considerato come una variabile anche se particolare. Per fare questo si utilizza il comando TYPEDEF. Esempio: definire una matrice 10x10 di numeri interi.
typedef int MATRICE[10][10]:
MATRICE numeri;
in questo modo è come se si fosse definita la matrice: int numeri[10][10];
E' necessario farlo perché nella chiamata alla procedura INSERISCI si passa come parametri la matrice e le dimensioni logiche M (righe) ed N(colonne):

INSERISCI(numeri, M,N)

Il sottoprogramma invece dovrà essere definito nel modo seguente:

void INSERISCI(MATRICE numx, int MX, int NX)

Ovviamente vale anche per le matrici il vincolo di non poter passare la struttura per valore. Viene infatti sempre passata per riferimento.