Corso sul linguaggio C++

Modulo 3

1 - Struttura dati array

Introduzione

In molte situazioni reali può capitare di elaborare elenchi di dati di vario tipo presenti i memoria.

Quali strumenti offre C++ per trattare queste situazioni ?
Come si usano questi strumenti ?

In questa Unità vediamo uno di questi strumenti, detto **array**, che è un primo esempio di variabile strutturata; impariamo come si dichiara un array e come si utilizza in vari problemi.

Argomenti

- Struttura di un array
- Dichiarazione di un vettore
- Inizializzazione di un vettore
- Lettura e stampa di vettori
- Stringhe e vettori di caratteri
- Dichiarazione di matrici
- Inizializzazione di matrici
- Utilizzo di matrici
- Il costrutto **typedef**
- Array come parametri

Informazioni generali

Attraverso l'array è possibile rappresentare molte situazioni reali.

L'array è una **variabile strutturata** perché è fatta in modo da contenere nello stesso tempo una molteplicità di valori, a differenza delle variabili semplici (**int**, **float**, **char**, **boolean**) che ne possono contenere uno alla volta.

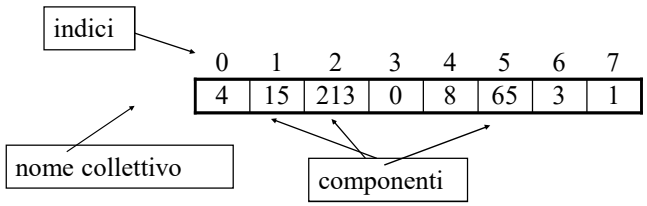
L'array, come tutte le variabili, può essere **dichiarato**, **inizializzato** ed **utilizzato**

Struttura di un array

Un array, in quanto variabile strutturata, contiene nel suo interno variabili di tipo più semplice dette **componenti.**

L'accesso a ciascuna componente avviene mediante una variabile intera associata all'array detta **indice**.

Il nome della variabile array è un **nome collettivo** di tutta la struttura ed è assegnato dal programmatore



Dichiarazione di un vettore

Un **vettore** è un array monodimensionale, ossia tale che per individuare una componente è sufficiente un solo indice.

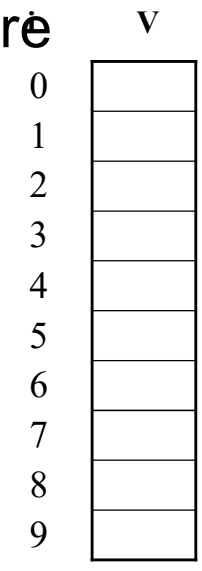
Nel linguaggio C++ la dichiarazione di un vettore assume la forma:

tipo ident [*intero*];

- dove:
- *tipo* è il tipo delle componenti;
- *ident* è il nome collettivo
- *intero* indica il numero delle componenti.

Per l'esempio a fianco la dichiarazione è:

int v[10];



Dichiarazione di un vettore

Si noti che la dichiarazione del vettore:

- va posta nel **blocco dichiarativo** delle variabili
- assegna al vettore una **dimensione prestabilita** che **NON è possibile modificare** nel corso del programma
- prevede che l'indice del vettore dichiarato può assumere solo **valori tra 0 e *intero* - 1**
- riserva uno **spazio in RAM** atto a contenere il numero ***intero*** stabilito di componenti in base al loro tipo di appartenenza ***tipo***;

i	**V**
0	5
1	2
2	1
3	12
4	54
5	1
6	0
7	23
8	135
9	4

Inizializzazione di un vettore

Un vettore può essere inizializzato da programma mediante la sintassi seguente:

tipo ident = { listavalori };

dove

- *tipo* è il tipo delle componenti;
- *ident* è il nome collettivo
- *listavalori* indica l'elenco dei valori da porre nelle varie componenti

Ad esempio
int mesi[10]= {10,20,50,89,1,34,2, 9, 0, 11};
definisce ed inizializza un vettore di interi.

Lettura e stampa di vettori

ATTIVITA': scrivere un algoritmo che, dopo aver acquisito il numero *n* di elementi da registrare in un vettore, ne esegue la lettura da input e successivamente li stampa a video.

Ovviamente è necessario un ciclo for per il caricamento, ed un altro ciclo for per la stampa.

```
Inizio
  Costante Intero MAXN=…;
   Intero v[MAXN],        // dichiara il vettore con dimensione opportuna
          i;              // indice per la scansione del vettore
  Leggi (n);
  Per i=0 a n-1 fai
    Leggi (v[i]);
  Per i=0 a n-1 fai
    Stampa (v[i]);
Fine
```


Lettura e stampa di vettori

ATTIVITA': scrivere un algoritmo che, dopo aver acquisito il numero *n* di elementi da registrare in un vettore, ne esegue la lettura da input e successivamente li stampa a video.

Ovviamente è necessario un ciclo for per il caricamento, ed un altro ciclo for per la stampa.

```
Inizio
  Costante Intero MAXN=…;
   Intero v[MAXN],        // dichiara il vettore con dimensione opportuna
          i;              // indice per la scansione del vettore
  Leggi (n);
  Per i=0 a n-1 fai
    Leggi (v[i]);
  Per i=0 a n-1 fai
    Stampa (v[i]);
Fine
```

M.Malatesta 1 - Struttura dati array-01
10
03/11/2008

Lettura e stampa di vettori

```
#include <iostream>
#include <cstdlib>
using namespace std;
const int MAXN=10;
int main()
{   int v[MAXN],
      n;
   cout<<"Immetti il numero di valori: ";   cin>>n;
   cout<<"*** Immissione dati *** "<<endl;
   for (int i=0; i<n; i++)
   {  cout<<"Valore: ";
      cin>>v[i];
   }
   cout<<"*** Stampa dati *** "<<endl;
   for (int i=0; i<n; i++)       cout<<"Valore: "<<v[i]<<endl;
   system("Pause");
   return 0;
}
```

ATTIVITA': scrivere la codifica in C++ dell'algoritmo realizzato.

Utilizzo di matrici

ATTIVITA': scrivere la codifica in C++ dell'algoritmo realizzato

Nel **main**() si dichiara localmente la matrice *mat[][]* che verrà poi passata come parametro alle funzioni.

```
#include <iostream>
#include <cstdlib>
using namespace std;
#define RIGHE 10
#define COLONNE 10
void lettura (int mat[][COLONNE], int r, int c);
int somma (int mat[][COLONNE], int r, int c);
int main()
{
  int r, c,
     mat[RIGHE][COLONNE];
  cout<<"Numero di righe: ";   cin>>r;
  cout<<"Numero di colonne: ";   cin>>c;
  lettura (mat, r, c);
  cout<<endl<<"La somma e' "<<somma(mat, r, c)<<endl;
  cout<<endl<<"La media e' "<<(float) somma(mat, r, c)/(r*c)<<endl;
  system("pause");
  return EXIT_SUCCESS;
}
```

Dichiarazione di matrici

Si può considerare anche un array pluridimensionale detto **matrice**, che può essere a due o più dimensioni.

Una matrice a 2 dimensioni si dichiara con:

tipo ident [*righe*][*colonne*]*;*

dove :

- *tipo* è il tipo delle componenti;
- *ident* è il nome della matrice
- *righe* indica il numero delle righe
- *colonne* indica il numero delle colonne

Trattandosi di una matrice, le componenti sono individuate da **due** indici.

Inizializzazione di matrici

L'inizializzazione di una matrice si effettua con una sintassi simile quella degli array monodimensionali

Ad esempio

int v[3][2]={{2,3},{0,3}, {1,8}};

2	3
0	3
1	8

char mesi[12][10]=
{"Gennaio","Febbraio","Marzo","Aprile","Maggio",
"Giugno", "Luglio","Agosto","Settembre",
"Ottobre","Novembre", "Dicembre"};

Utilizzo di matrici

Vediamo un esempio di utilizzo di matrici.

ATTIVITA': scrivere un'applicazione che legga da input gli elementi di una matrice m[3][4] di interi e successivamente calcoli e stampi la loro media aritmetica. Utilizzare le funzioni.

Si dichiara e si alloca una matrice mat[3][4] di interi e si istanzia la funzione *lettura (m, r, c)* che, attraverso un doppio ciclo **for** legge da input, uno ad uno, i valori da caricare in essa, utilizzando *r* righe e *c* colonne.
Successivamente, alla variabile *media* si assegna il risultato della funzione *somma (m, r, c)* diviso *r*c* e si stampa *media*. La funzione *somma (m, r, c)* calcola la somma degli elementi di *m[][]* sommando in un accumulatore *somma* (azzerato) i valori di *m[][]* mediante un doppio ciclo **for**.

Utilizzo di matrici

ATTIVITA': scrivere la codifica in C++ dell'algoritmo realizzato

Nel **main**() si dichiara localmente la matrice *mat[][]* che verrà poi passata come parametro alle funzioni.

```
#include <iostream>
#include <cstdlib>
using namespace std;
#define RIGHE 10
#define COLONNE 10
void lettura (int mat[][COLONNE], int r, int c);
int somma (int mat[][COLONNE], int r, int c);
int main()
{
  int r, c,
     mat[RIGHE][COLONNE];
  cout<<"Numero di righe: ";   cin>>r;
  cout<<"Numero di colonne: ";   cin>>c;
  lettura (mat, r, c);
  cout<<endl<<"La somma e' "<<somma(mat, r, c)<<endl;
  cout<<endl<<"La media e' "<<(float) somma(mat, r, c)/(r*c)<<endl;
  system("pause");
  return EXIT_SUCCESS;
}
```

Utilizzo di matrici

```
void lettura (int mat[][COLONNE], int r, int c)
{
  int i, j;
  cout<<"*** Immissione valori ***"<<endl;
  for (i=0;i<r;i++)
  {
     for (j=0;j<c;j++)
     {
        cout<<"Valore["<<i<<", "<<j<<"]: ";
        cin>>mat[i][j];
     }
  }
}
int somma (int mat[][COLONNE], int r, int c)
{
  int i, j, s=0;
  for (i=0;i<r;i++)
    for (j=0;j<c;j++)
              s+=mat[i][j];
  return s;
}
```

Le funzioni *lettura*() e *somma*() ricevono come parametro la matrice ed operano rispettivamente, il caricamento e il calcolo della sommma degli elementi

Il costrutto typedef

Con il costrutto **typedef** si può associare a qualunque struttura dati un **identificatore di tipo** , in modo da abbreviare le dichiarazioni successive.

Ad es:

typedef anno **int** mese[12];

……..

anno a1, a2, a3;

In tal modo è più pratico associare un tipo alle variabili.

analogamente:
typedef int mese[12] anno;

Array come parametri

Gli array (*mono o multidimensionali*) possono essere passati come parametri alle funzioni, rendendo quindi il codice più leggibile e più compatto. Quando si passa ad una funzione un array come parametro:

- nell'istanza della funzione:
  - il parametro attuale è il nome dell'array, *senza le dimensioni*
- nel prototipo:
  - il parametro formale è il tipo dell'array, il nome e le parentesi vuote.
  - viene considerato automaticamente il *passaggio per indirizzo* (*by reference*) e pertanto occorre ricordare che la funzione **opera sui valori originali dell'array**

Array come parametri

Vediamo un esempio di uso di vettori come parametri di funzioni.

> **ATTIVITA**': scrivere un'applicazione contenente una funzione con parametri *v[]* ed *n* per leggere *n* elementi di un vettore di dimensione 10 ed un'altra funzione con parametri *v[]* ed *n* che successivamente li stampi a video nella forma:
>
> v[0] = …
>
> v[1] = …

Si dichiara un vettore *v[ ]* di 10 componenti intere e, dopo ave immesso il numero *n* di elementi da utilizzare, attraverso una funzione *lettura (v, n)* si leggono da input, uno ad uno, i valori da caricare in esso.
Successivamente, mediante una funzione *stampa (v, n)* si stampano i singoli elementi del vettore.

Array come parametri

```
#include <iostream>
#include <conio.h>
using namespace std;
const int MAXN=10;
void lettura (int v[], int n);          /* caricamento valori */
void stampa (int v[], int n);           /* stampa valori */
int main()
{  int v[MAXN],
       n;
   cout<<"Immetti il numero di valori: ";    cin>>n;
   lettura(v, n);
   stampa(v, n);
   system("Pause");
   return 0;
}
```

Questo è il programma principale

Array come parametri

```
void lettura (int v[], int n)
{
   cout<<"*** Immissione dati *** "<<endl;
   for (int i=0; i<n; i++)
   {
      cout<<"Valore: ";
      cin>>v[i];
   }
}
void stampa (int v[], int n)
{
   cout<<"*** Stampa dati *** "<<endl;
   for (int i=0; i<n; i++)
      cout<<"Valore: "<<v[i]<<endl;
}
```

Si osservi il secondo parametro *n*, che consente di stabilire il numero di elementi del vettoreda utilizzare di volta in volta

Il vettore viene passato per indirizzo e qualunque funzione può modificarne il contenuto

Cosa ho imparato

- Come si rappresenta logicamente un vettore e quali sono le sue caratteristiche
- Cosa vuol dire e come si effettua la dichiarazione e l'inizializzazione di un vettore
- Qual è la rappresentazione logica di una matrice e quali sono le sue caratteristiche
- Cosa vuol dire e come si effettua la dichiarazione e l'inizializzazione di una matrice

Cosa ho imparato a fare

- Scrivere applicazioni che facciano uso di vettori e matrici.
- Scrivere funzioni che facciano uso di vettori e matrici come parametri.
- Usare il costrutto **typedef** per rappresentare tipi di dato specifici per il programma

Terminologia

- Array
- Vettore
- Matrice
- Indice
- Componenti
- Indirizzo base
- Nome collettivo
- Struttura dati
- **typedef**

Altre fonti di informazione

- J. Purdum, C – ed. Jackson
- Romagnoli-Ventura, C/C++ - Ed. Petrini
- A.Lorenzi-D.Rossi, Il linguaggio C++ - ed. ATLAS