Vettori e matrici

Vettori

- Dichiarazione di un vettore di oggetti del tipo Tipo:

  Tipo identificatore[dimensione];

- Uso di un elemento del vettore:

  identificatore[elemento]

- Dove dimensione e elemento sono degli oggetti di tipo int

NB

- Diversamente da altri linguaggi in C e C++ il primo elemento di un vettore ha indice 0 e l'ultimo, se n è la dimensione del vettore, ha indice n-1.

- Come in tutti i linguaggi la dichiarazione del vettore ed il suo dimensionamento comportano l'allocazione della memoria necessaria per la dimensione dichiarata: se si prova ad usare un elemento con indice superiore a n-1 si ha uno sfondamento della memoria, errore che comporta problemi in punti imprevedibili del programma!

Esempi di dichiarazioni

```
int V[25];  //dichiarazione di un vettore V
            //con 25 elementi integer
            //V[0],V[1]…V[24].

double vector[3]; //dichiarazione di un vettore
                  //vector con 3 elementi double
                  //vector[0],vector[1],vector[2].

char word[50];  //dichiarazione di un vettore word
                //con 50 elementi character
                //word[0],…,word[49].
```

Stringhe

Una stringa è un vettore di caratteri

```
#include <string>      //definizione degli oggetti
                       // stringa di caratteri

string studenti[54]; // dichiarazione del vettore
                     //studenti con 54 elementi
                     // di tipo stringa di
                     //caratteri
                     // studenti[0]…studenti[53].
```

NB

una stringa contenente n caratteri visibili e' un vettore di caratteri di dimensione n+1, infatti l'ultimo carattere, '\0', indica la fine della stringa.

Inizializzazione

```
int V[25] = {3,   5,   6,   1} ;

double vector[3] = {-1.2,   4.7,   5.9} ;

char word[50] = {'s' , 'p', 'o', 't'};

string student[54] = {"Chris Berkley",
"Kevin Chao", "Missy Mesfin","Joel Triemstra"};

int M[ ] = {1, 4, 15, 2};

char Name[ ] = "Michael Bird";
```

Inizializzazione

Queste tre dichiarazioni sono equivalenti:

```
char Name[ ] = "Michael Bird";

char Name[ ] = {'M','i','c','h','a','e','l',' ',
                                    'B','i','r','d','\0'};

char Name[13] = {'M','i','c','h','a','e', 'l',' ',
                                    'B','i','r','d','\0'};
```

Ovvero

Name[0] = 'M' , Name[1] = 'i' … Name[11] = 'd' , Name[12] = '\0'

Esempi di inizializzazione e uso

```
double A[20] ={4.1,5.3,10.6,100.9,23,-12};

double sum = 0;

for(int j =0; j<20; j = j + 1)  sum = sum + A[j];

cout << "la somma vale  "<<sum<<"."<<endl;
```

Esempi: uso di ++

```
int  A[  ] ={0,10,20,30,40,50,60,70,80,90};

int  j = 5;

cout<< "j vale  "<<j<<endl;

cout<<  "A[j++] vale " << A[j++]<<endl;

cout << "j ora vale "<<j <<endl;
```

Esempi: uso di --

```
int  A[  ] ={0,10,20,30,40,50,60,70,80,90};

int  j = 5;

cout<< "j vale "<<j<<endl;

cout<<  "A[j--] vale " << A[j--]<<endl;

cout << "j ora vale "<< j <<endl;
```

Esempi: uso di ++

```
int  A[  ] ={0,10,20,30,40,50,60,70,80,90};

int  j = 5;

cout<< "j vale  "<<j<<endl;

cout<<  "A[++j] vale "<< A[++j]<<endl;

cout << "j ora vale "<<j <<endl;
```

Esempi: uso di --

```
int  A[ ] ={0,10,20,30,40,50,60,70,80,90};

int  j = 5;

cout<< "j vale "<<j<<endl;

cout<<  "A[--j] vale "<< A[--j]<<endl;

cout << "j ora vale "<<j <<endl;
```

Vettori multidimensionali (matrici)

- Dichiarazione di una matrice con
  - rDim righe
  - cDim colonne

```
Tipo  identificatore[rDim][cDim];
```

- Esempio:

```
double  tabella[3] [4];
```

Dichiarazione ed inizializzazione

Per l'inizializzazione si elencano gli elementi della prima riga, poi della seconda etc…

```
int  M[3][4] = {{1,2,3,4},
                {5,6,7,8},
                {9,10,11,12}};
```

oppure

```
int  M[3][4] = {1,2,3,4,5,6,7,8,9,10,11,12};
```

Esempio: prodotto scalare

```
#include <iostream.h>

 int main( ) {
     double A[3],B[3];
     double pScal=0.;
     cout  <<"inserisci gli elementi del vettore A"<< endl;
     cin >> A[0]>> A[1]>>A[2];
     cout  <<"inserisci gli elementi del vettore B"<< endl;
     cin >> B[0]>> B[1]>>B[2];
     for(int j=0; j<3; j++) {     pScal += A[j]*B[j];     }
     cout <<" A scalare B vale "<< pScal<<endl;
 return 1;}
```

Esempio: prodotto vettoriale

```
#include <iostream.h>
 int main( ) {
     double A[3]= {1.,0.,0. };
     double B[3]= {0.,1.,0. };
     double C[3];

     int permutazioni[3][2]={   {1,2},    {2,0},    {0,1}};
     for(int j=0; j<3; j++) {  //loop sulle 3 componenti di C
        int k=permutazioni[j][0];
        int m=permutazioni[j][1];
        C[j]=A[k]*B[m]-A[m]*B[k];
        cout<<"C["<<j<<"]="<<C[j]<<endl;
     }
 return 1;}
```

Esempio: modulo di un vettore

```
#include <iostream.h>
#include <math.h>        // funzioni matematiche (libreria C)
 int main( ) {
     double A[3];
     double mod=0.;
     cout  <<"inserisci gli elementi del vettore A"<< endl;
     cin >> A[0]>> A[1]>>A[2];

     for(int j=0; j<3; j++) {   mod += A[j]*A[j];    }

     cout <<" il modulo quadro di A vale "<< mod<<endl;
     if( mod > 0 ) {
         mod = sqrt(mod);
         cout <<" il modulo di A vale "<< mod<<endl;
     }
     return 1;}
```

Esempio: prodotto di matrici

```
#include <iostream.h>
// calcolo di M3=M1*M2
//////////////////////////
// | 1 0 |
// | 2 1 | * | 0 1 2 |
// | 0 2 |   | 1 2 1 |
//////////////////////////
int main( ) {
    int M1[3][2]= {   {1,0},   {2,1},    {0,2}};
    int M2[2][3]= {   {0,1,2},   {1,2,1}};
    int M3[3][3];

    for(int j=0; j<3; j++) {      //loop sulle righe di M1
     for(int k=0; k<3; k++) {     //loop sulle colonne di M2
        M3[j][k]=0;
        for(int l=0; l<2; l++ ) {//loop sulle colonne di M1
                M3[j][k]+= M1[j][l]*M2[l][k];
    }}}
    return 1;}
```

Dimensionamento dinamico

```
int * punt_int = new int[50];
```

è equivalente a `int punt_int[50];`

ma ci sono dei casi, qualora la dimensione di un vettore non sia nota a priori, in cui si può usare solo la prima forma. Possiamo infatti scrivere

```
int n;
int * punt_int;
cin << n;
punt_int = new int[n];
```

ma non

```
cin << n;
int punt_int[n];   // errore!
```

Allocazione dinamica della memoria

L'operatore new alloca la quantità di memoria richiesta per un oggetto di tipo o classe tipo e restituisce l'indirizzo dell'area di memoria riservata (puntatore).

Forma generale:

new tipo

Delete

Per liberare memoria allocata dinamicamente si usa la keyword delete, ovvero

```
int * punt_int;
punt_int = new int[50];
        :
delete[ ]  punt_int;
```