COESIONE ITALIA 21-27 PIEMONTE

Cofinanziato dall'Unione europea

REGIONE PIEMONTE

WEB DEVELOPER
Fondamenti di Programmazione
Massimo PAPA

GLI ARRAY (vettori)

Intervento realizzato da ITS TECNOLOGIE DELL'INFORMAZIONE E DELLA COMUNICAZIONE

GLI ARRAY

- L'array è un importante esempio di tipo aggregato
- L'array è un insieme di elementi omogenei tra di loro
- Possiamo indicare tanti dati collettivi dello stesso tipo con un unico nome al posto di utilizzare per ogni elemento nomi differenti.
- Gli elementi vengono distinti attraverso un indice che viene assegnato a ogni elemento

GLI ARRAY

- L'array è quindi un insieme omogeneo di dati
- Dichiarazione di un array:

```
tipo nomeArray[dimensione];
```

- Esempi:

```
double temperature[10];
int naturali[1000];
string testo[30];
char frase[20];
```

GLI ARRAY

- L'elemento dell'array viene individuato dal nome dell'array con l'indice

```
double temp[10];
temp[5]; // sesto elemento
for(i=0;i<10;i++)
  cout << temp[i] << endl;
```

- La numerazione degli indici inizia da 0

GLI ARRAY

- E' possibile inizializzare un array assegnando i valori agli elementi in fase di dichiarazione:

```
double temp[3]={2.3,-4.5,3.0};
int num[]={2,-4,7,-9};
```

- I valori sono indicati tra parentesi graffe e sono separati da virgole.
- Se inizializziamo l'array in fase di dichiarazione possiamo omettere il numero di elementi tra parentesi quadre

GLI ARRAY

- Per rendere più efficace la rappresentazione del codice, si possono usare le costanti di enumerazione come indici.
- Consideriamo la gestione dell'incasso totale di una settimana di un negozio:

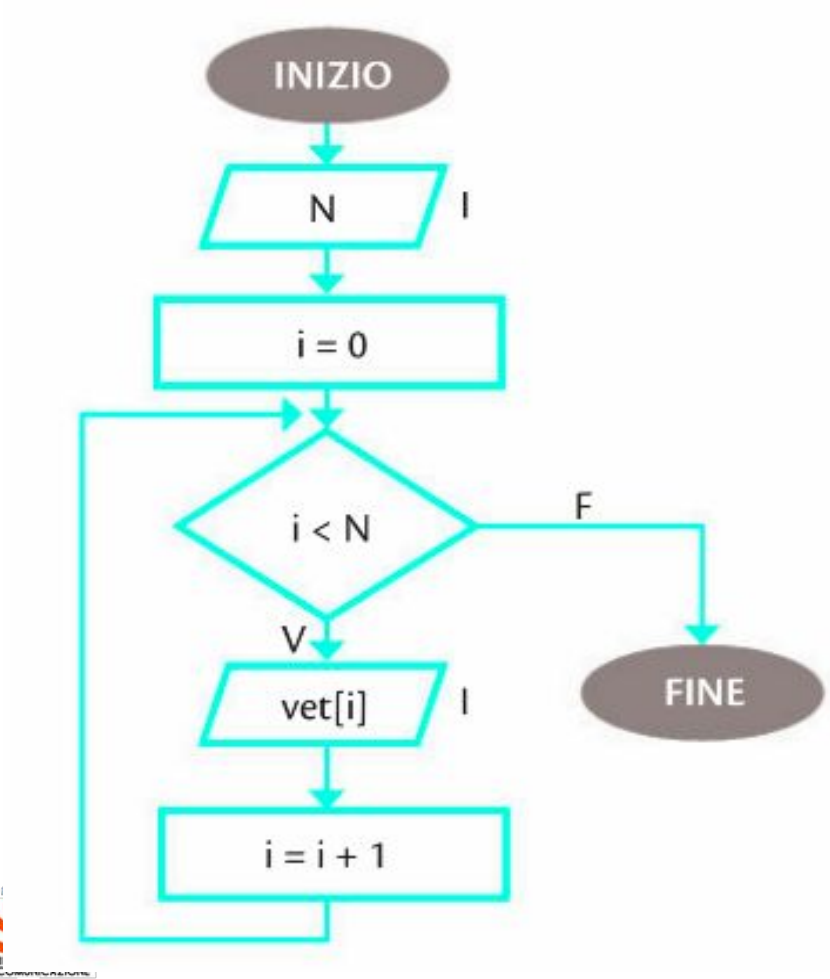
```
enum Giorni {LUN=0, MAR, MER, GIO,VEN,SAB, DOM};
int main(){
    float incasso[6];
    float totale=0;
    int i;

    for(i=LUN; i<=SAB;i++){
        cout << "Incasso del giorno "<< i <<": ";
        cin >> incasso[i];
        totale += incasso[i];
    }
    cout << "Incasso Totale = " << totale << endl;

    return 0;
}
```

GLI ARRAY - Caricamento

- Il flow-chart per il caricamento di tutti gli elementi di un array è il seguente:




```
int main(){
    int N;
    float vet[100];
    int i;

    do{
        cin >> N;
    }while(N<1 || N> 100);

    for(i=0; i<N; i++){
        cout<<"\nInserisci: ";
        cin >> vet[i];
    }

    return 0;
}
```

GLI ARRAY di CHAR

- Abbiamo visto il tipo di dati complesso **`string`**
- Possiamo considerare la **stringa** come un **array di char**
- Esaminiamo il seguente codice:

```
char stringa[7]="abcdef";
int main(){
    int i;

    for(i=0; i<6;i++)
        cout << "stringa["<< i <<"] = "<<stringa[i] <<"\n";

    return 0;
}
```

- Lanciando il codice si ottiene il seguente output:

```
stringa[0] = a
stringa[1] = b
stringa[2] = c
stringa[3] = d
stringa[4] = e
stringa[5] = f
```

GLI ARRAY di CHAR

- L'array ha dimensione 7 ma la stringa è formata da 6 caratteri
- La settima posizione occupata dal carattere di fine stringa, cioè dal carattere NULL che corrisponde a '\0' o a 0
- E' importante considerare questo carattere, altrimenti si rischia di introdurre bachi nell'elaborazione.
- A differenza del tipo string, l'array di char si serve di tutta una serie di funzioni standard per effettuare le elaborazioni sull' array stesso.
- Alcune di queste sono:
  - `strcmp(const char *str1, const char *str2)` che effettua il confronto tra la string str1 e la stringa str2
  - `strcpy(const char *str1, const char *str2)` che copia la string str1 nella stringa str2
- Non approfondiremo questa libreria in quanto nei nostri algoritmi andremo a utilizzare la classe string del C++

ARRAY - Passaggio di parametri

- L'array viene sempre passato per riferimento ad una funzione
- Quello che viene passato alla funzione è l'indirizzo del primo elemento dell'array che rappresenta di fatto l'indirizzo dell'array

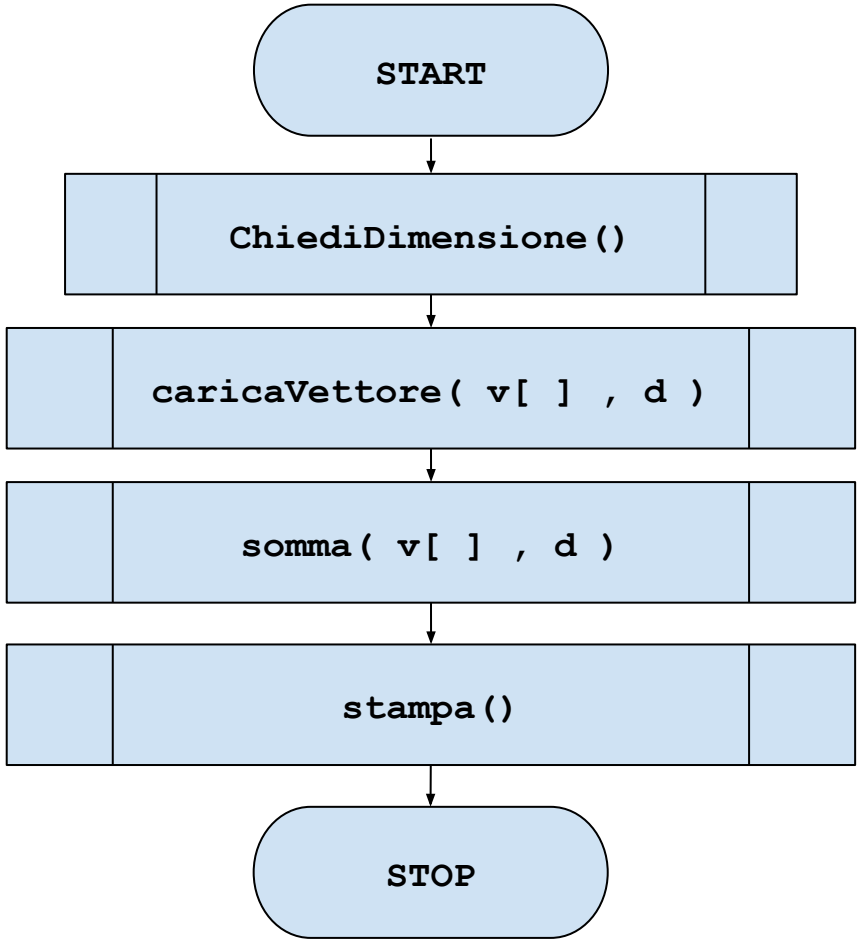
```
void fun(int v[]){ //oppure fun(int v[10])
    . . .
    . . .
}

int main(){
    int x[10];
    . . .
    fun(x);
    . . .
    return 0;
}
```

- Il passaggio per referenza è il solo modo con cui si può passare un array come parametro di una funzione

ARRAY - Passaggio di parametri

- Esempio: acquisire un array dallo standard input e stampare sullo standard output la somma di tutte le sue componenti



ARRAY - Passaggio di parametri

```
const int MAX = 100;

// prototipi funzioni
int chiediDimensione();
void caricaVettore(int v[], int d);
int somma(int v[], int d);

int main(){
     int n;
     int v[MAX];
     int tot;
     n = chideiDimensione();
     cout << "Carica elementi vett \n";
     caricaVettore(v,n);
     tot = somma(v,n);
     cout << "Somma = " << tot << endl;
     return 0;
}

// inserimento dimensione array
int chiediDimensione(){
     do {
          cout << "Dim vett: ";
          cin >> d;
     } while (d < 1 || d > MAX);
     return d;
}

// caricamento vett
void caricaVettore (int v[] , int d){
     int i;
     for (i=0; i<d; i++){
          cout << "Ele "<<i<<" : ";
          cin >> v[i];
     }
}

//Somma componenti vett
int somma( int v[], int d){
     int s = 0;
     int i;
     for(i=0; i<d; i++)
          s += v[i];
     return s;
}
```

Matrice

- La matrice è un array che ha 2 o piu' dimensioni, in altre parole presenta più di una colonna.
- Le matrici vengono definite in modo analogo agli array:

```
tipo nomeMatrice[N][M];
```

- Per esempio supponiamo di avere una matrice di numeri reali rettangolare di 20 righe e 5 colonne:

```
float mat[20][5]; // contiene 100 numeri reali
```

- Matrici multidimensionali:

```
tipo nomeMatrice[N1][N2]...[Nn];
```

- Per esempio una matrice tridimensionale di stringhe:

```
string testo[10][10][20]; // contiene 2000 stringhe
```

Matrice - primo esempio

- Consideriamo di gestire gli incassi di 20 reparti di un supermercato durante tutta la settimana, creo la seguente struttura dati:

```
const int numReparti = 20;
const int giorniSettimana = 7;
double incassi[numeReparti][giorniSettimana];
```

- Calcoliamo l'incasso totale di tutti i reparti il giovedì:

```
double totale = 0;
int i;
for(i=0; i< numeroReparti; i++)
   somma += incassi[i][3]; // 3 -> giovedì
cout << somma << endl;
```

Matrice - secondo esempio

- Consideriamo di gestire le temperature massime registrate in 50 città nei 12 mesi dell'anno, nei 31 giorni del mese, creo la seguente struttura dati:

```
const int numCitta = 50;
const int mesiAnno = 12;
const int giorniMese = 31;
double temp[numCitta][mesiAnno][giorniMese];
```

- Calcoliamo la temperatura media nel mese di febbraio della prima quinta città:

```
double media = 0;
int i;
for(i=0; i< giorniMese; i++)
   media += temp[4][1][i];
cout << media/giorniMese << endl;
```

Gli array paralleli (o associativi)

- Se dovessimo gestire le temperature minime e massime potremo duplicare la struttura dati vista prima:

```
double tempMAX[numCitta][mesiAnno][giorniMese];
double tempMIN[numCitta][mesiAnno][giorniMese];
```

- Abbiamo definito due array paralleli, ogni elemento di pari posizione in un array è correlato al rispettivo elemento nell'altro array.
- Stampiamo la temperatura massima e minima della seconda città, nell'ultimo giorno di dicembre.

```
cout << tempMAX[1][11][30] << " " << tempMIN[1][11][30];
```

Gli array paralleli (o associativi)

- Un altro utilizzo si ha quando si hanno informazioni che non sono omogenee, p.e. un elenco di persone di cui si conosce nome (string) e età (int):

```
string nomePersona[numPersone];
int etaPersona[numPersone];
```

- Si vuole stampare il nome e l'età della terza persona dell'elenco:

```
cout << nomepersona[2] << " " << etapersona[2];
```