Sequenze di elementi omogenei - Array

lunedì 19 ottobre 2020 10:56

Struttura degli array

Gli array sono un tipo di dato strutturato (nota bene, NON variabili), che ci permettono di distinguere sequenze di elementi (variabili) di stesso tipo fra quelli già visti, ovvero interi, con virgola, di caratteri ecc.

Un array è costituito da una dimensione (se la dimensione è prefissata (con una costante), si tratterà di array statici (ovvero fissata a compilatore), mentre se è stabilita durante il runtime si parla di array dinamici, in cui la dimensione non fissata può contenere uno spazio arbitrariamente grande in memoria).

esempio

int num[5];

num[0]
num[1]
num[2]
num[3]
num[4]

rappresentazione convenzionale

[0]	[1]	[2]	[3]	[4]

rappresentazione abbreviata

gli array: sintassi

- un array è formato da un numero prefissato di elementi dello stesso tipo
- In questa parte del corso ci concentreremo su **array unidimensionali** dove gli elementi saranno disposti in un elenco

```
tipodiDato  nomeArray[espressioneIntera];
```

una volta valutata è un **intero positivo** che descrive la dimensione dell'array

Altro esempio:

Per accedere ai dati degli array si usa un indice, il cui valore ha estremi compresi: da 0 a DIMENSIONE - 1. La dimensione è precedentemente fissata.

[0]	[1]	[2]	[3]	[4]
		45	10	55

```
int list[5];
list[2]=45;
list[1*2+1]=10;
list[4]=list[2]+list[3];
```

accesso agli elementi dell'array

```
nomeArray[espressioneIndice];
```

specifica una posizione nell'array a **partire da 0**

- float angle[4];
  angle[0]=4.93;
  angle[1]=-1.2;
  angle[2]=0.72;
  angle[3]=1.67;

[0]	[1]	[2]	[3]
4.93	-1.2	0.72	1.67

Codice ambiguo

dimensione degli array

- gli array possono anche essere dichiarati in questo modo
- ```
  const int ARRAY_SIZE = 10; //costante globale
  int list[ARRAY_SIZE];
  ```
- Quando si dichiarano gli array la loro dimensione deve essere **nota al tempo di compilazione**
- Il seguente codice è sconsigliato (anche se ammesso da alcuni recenti compilatori):

```
int dimensione;
cout << "inserisci dimensione" ;
cin >> dimensione;
int list[dimensione];
```

il compilatore non sa quanta porzione di memoria riservare all'array

Il concetto di Out of Bound

Indici fuori dai limiti (out of bound)

- float a[10];
  a[i]=4.5; // ha senso per tutti i valori di i tra 0 e 9
- altrimenti ho un accesso ad una locazione di memoria non riservata al programma
- In C++, con gli array, non abbiamo una protezione nei confronti di questo fenomeno
  - non riceveremo un messaggio di errore specifico dal compilatore (ne' in fase di esecuzione) ne' avremo gli strumenti (la conoscenza) per poter sollevare eccezioni
- Il programma accederà all'elemento di memoria e l'effetto di questo accesso è impredicibile
- Ricade sul programmatore l'onere di garantire il rispetto dei limiti!
- In seguito incontreremo altri strumenti che ci forniscono maggiori garanzie

ATTENZIONE!

`char x[5]`

è formato dai seguenti elementi:
`x[0], x[1], x[2], x[3], x[4]`
```

Dichiarazione e inizializzazione totale e parziale

- INIZIALIZZAZIONE TOTALE
  - double sales[5]={1.4, 2.5, 14.5, 5.3, -1.4};
  - possiamo anche scrivere
    double sales[]={1.4, 2.5, 14.5, 5.3, -1.4};
    la dimensione è implicita nell'inizializzazione

[0]	[1]	[2]	[3]	[4]
1.4	2.5	14.5	5.3	-1.4

- INIZIALIZZAZIONE PARZIALE
  - double sales[5]={1.4};

[0]	[1]	[2]	[3]	[4]
1.4				

Codice ambiguo

- int list1[4]={1,2,3,4};
  int list2[4];
- list2=list1; **// non è corretto (come fare?)**
- cin >> list2; **// non è corretto (come sopra)**
- if (list1 == list2)
  …. **// è corretto ma non fa quello che vogliamo**
  // (confronta indirizzi di memoria)

repetita iuvant:

una variabile di tipo array

non contiene i valori dell'array

ma l'indirizzo in memoria

a partire dal quale

i valori sono memorizzati

Inoltre

```
float numeri[10];
```

- `numeri` NON è una variabile di tipo float;
  `numeri[i]` (con `i` intero) lo è
- `numeri` NON È NEMMENO VARIABILE:
  contiene un indirizzo di memoria prefissato
- ~~`numeri[10]`~~ NON è un elemento dell'array!!!
  L'indice deve essere fra 0 e 9!!!
- `numeri` contiene una sola informazione: un indirizzo.
  Non contiene la dimensione!

- se `A` è l'indirizzo base dell'array `x`

indirizzo base di un array

- se `A` è l'indirizzo base dell'array `x`
- e `D` è la dimensione degli elementi di `x`

  `D = sizeof x[0];`
- allora l'indirizzo di `x[i]` si calcola così:

  `A + D*i`
- `A` è contenuta in `x`
  e `D` è la dimensione del tipo degli elementi di `x`.
- Il limite superiore per `i` invece (numero elementi) NON È SCRITTO DA NESSUNA PARTE!!!!!

indirizzo base di un array

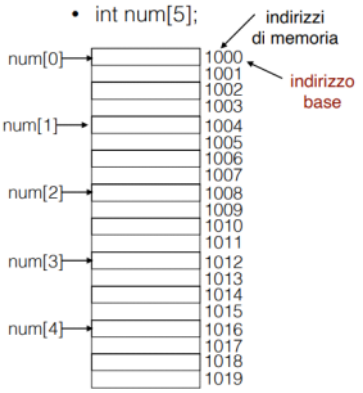
- l'indirizzo base di un array è l'indirizzo di memoria del suo primo elemento

- int num[5];
- cout << num << endl;

  stampa l'indirizzo base

(vedi N.B.)

```
Indirizzo base 0x7fff57741ae0
(in esadecimale)
```



N.B.

L'indirizzo base di un array di interi è l'indirizzo di memoria della cella che contiene il primo intero dell'array

Scegli una risposta:

- Vero ✖
- Falso

ne contiene solo una parte!

Questa affermazione è sbagliata in modo sottile, ma la differenza è importante, quindi fate attenzione!

L'indirizzo base è sì l'indirizzo di memoria occupato dalla prima cella dell'array. Ma questa cella in generale non sarà sufficiente a contenere il primo intero. Quindi la frase corretta sarebbe stata

in modo corretto

"L'indirizzo base di un array di interi è l'indirizzo di memoria della prima cella occupata dall'array" oppure "... della prima cella occupata dal primo elemento dell'array" (il quale elemento avrà bisogno di altre celle contigue per essere memorizzato)

La risposta corretta è 'Falso'.