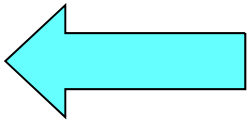
16. Array statici
Accesso fuori dagli array

Andrea Marongiu
(andrea.marongiu@unimore.it)
Paolo Valente

UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

Contenuto lezione

- Array statici
  - Passaggio alle funzioni
  - Vettori dinamici
  - Accesso fuori dall'array



Esercizio

- Riprendiamo l'esercizio precedente.
- Scrivere un programma che legge *M* temperature da tastiera, ne calcola la media e stampa i valori che stanno sopra la media.
- L'utente non comunica il numero *M* di temperature da memorizzare, ma segnala la fine della lettura inserendo un valore negativo.
- Implementare il vettore utilizzando un valore terminatore
  - Attenzione a cosa accade nel caso in cui si inseriscono *max_M* elementi !!!

Proposta programma

```
main()
{
  const int max_M = 100 ; double vett_temper[max_M] ;


  for (int i = 0; i < max_M; i++) {
    cin>>vett_temper[i];
    if (vett_temper[i] < 0) {
     vett_temper[i] = -1 ; // inseriamo il terminatore
       break ;
    }
  }


  double media = 0. ;
  // in uscita dal ciclo conterra' il num di valori letti
  int num_val_letti ;
  for (num_val_letti = 0; vett_temper[num_val_letti] >= 0; num_val_letti++)
    media += vett_temper[num_val_letti];


  media /= num_val_letti ;


  cout<<"Media: "<<media<<endl<<endl ;
```

IL PROGRAMMA E' CORRETTO?

Commento aggiuntivo

```
main()
{
  const int max_M = 100 ; double vett_temper[max_M] ;


  for (int i = 0; i < max_M; i++) {
    cin>>vett_temper[i];
    if (vett_temper[i] < 0) {
     vett_temper[i] = -1 ; // inseriamo il terminatore
       break ;
    }
  } // nota: se inseriamo max_M valori non vi sarà alcun terminatore!


  double media = 0. ;
  // in uscita dal ciclo conterra' il num di valori letti
  int num_val_letti ;
  for (num_val_letti = 0; vett_temper[num_val_letti] >= 0; num_val_letti++)
    media += vett_temper[num_val_letti];


  media /= num_val_letti ;


  cout<<"Media: "<<media<<endl<<endl ;
```

IL PROGRAMMA E' CORRETTO?

Risposta

- Errore:
  - Non si controlla di essere sempre dentro l'array nella fase di calcolo della somma delle temperature
  - Se nell'array fossero stati inseriti *max_M* elementi, allora non vi sarebbe alcun terminatore, e si rischierebbe poi di leggere fuori dall'array
    - Errore logico
    - Possibile terminazione forzata del programma, oppure lettura di valori casuali
      - Come mai terminazione forzata o lettura di valori casuali?

Contenuto memoria 1/3

- Sappiamo già che la memoria di un programma è utilizzata per memorizzare le variabili e le costanti con nome definite nel programma (e quelle allocate dinamicamente, come vedremo nelle prossime lezioni)
- Ma non solo, la memoria contiene anche delle informazioni non manipolabili direttamente dal programmatore
  - Codice delle funzioni (tradotto in linguaggio macchina)
  - Codice e strutture dati aggiuntivi, di supporto all'esecuzione del programma stesso
- In merito alle parti aggiuntive, senza entrare nei dettagli consideriamo solo che, quando il compilatore traduce il programma in linguaggio macchina, non si limita a creare solo il codice macchina che esegue ciascuna delle istruzioni esplicitamente inserite nel codice sorgente

Contenuto memoria 2/3

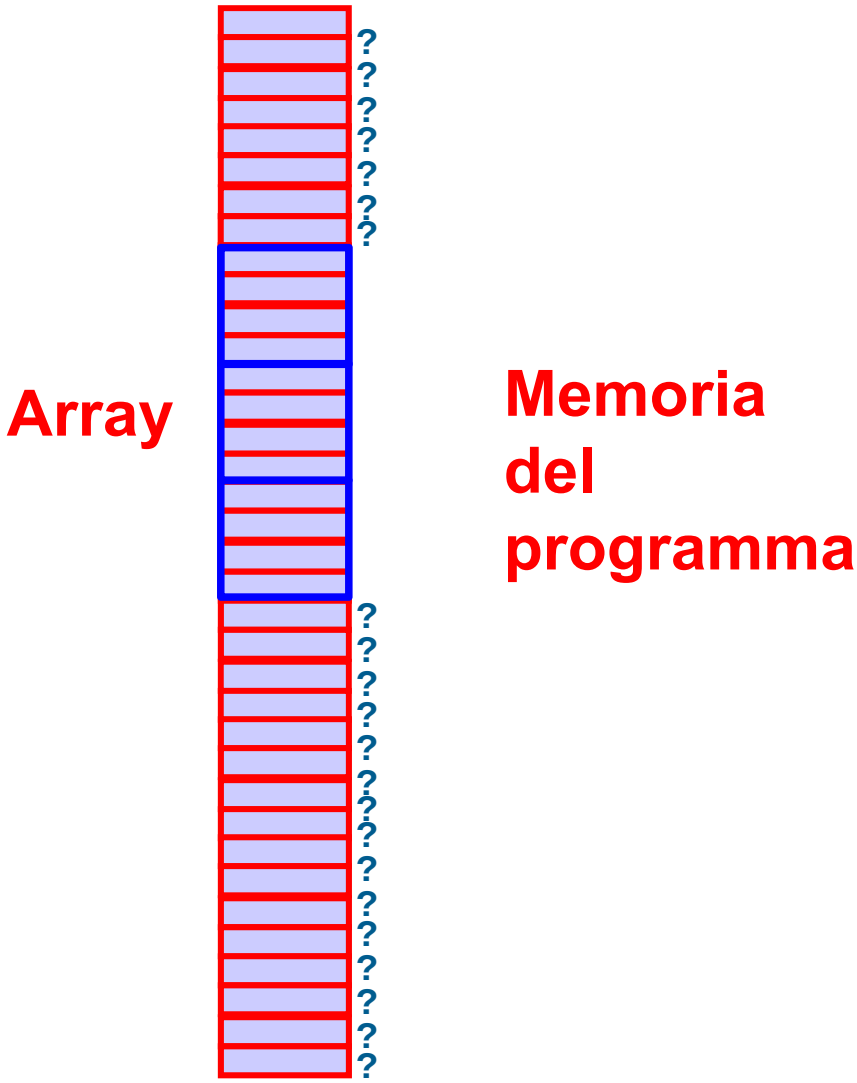
- Il compilatore di fatto aggiunge codice ulteriore, di cui vedremo qualche dettaglio in alcune delle prossime lezioni
- Tale codice effettua operazioni di supporto all'esecuzione del programma stesso (e che non sono e non possono essere esplicitamente inserite nel codice sorgente in base alle regole del linguaggio)
  - Inizializzare i parametri formali con i parametri attuali all'atto della chiamata di una funzione
  - Allocare spazio in memoria quando viene eseguita la definizione di una variabile/costante con nome
  - Deallocare spazio dalla memoria quando finisce il tempo di vita di una variabile/costante con nome
  - ...

Contenuto memoria 3/3

- Per realizzare alcune di queste operazioni il compilatore inserisce nella versione in linguaggio macchina del programma le strutture dati aggiuntive precedentemente menzionate
- In definitiva, se si accede a celle di memoria al di fuori di quelle dedicate ad oggetti correttamente definiti, si rischia di accedere a
  - Porzioni della memoria non ancora utilizzate per alcuno scopo, tipicamente contenenti valori casuali
  - Memoria occupata da altri oggetti definiti dal programmatore
  - Memoria occupata da codice o strutture dati aggiunte dal compilatore

Array in memoria

```
int a[3] ;
```



Errore gestione della memoria

- In ogni caso, accedere al di fuori delle celle di memoria dedicate ad oggetti correttamente definiti è un **errore di gestione della memoria**
  - Se l'accesso avviene in lettura si leggono di fatto valori casuali
  - Se l'accesso avviene in scrittura si rischia la cosiddetta corruzione della memoria del programma, ossia la sovrascrittura del contenuto di altri oggetti definiti nel programma o delle strutture dati aggiuntive precedentemente menzionate
    - Se si corrompono le strutture dati aggiuntive il comportamento del programma diventa **impredicibile**

Diritti di accesso alla memoria

- I processori moderni permettono di suddividere la memoria in porzioni distinte e stabilire quali operazioni si possono o non si possono effettuare su certe porzioni della memoria, nonché quando si ha diritto di effettuarle
  - Ad esempio, il codice di un programma viene tipicamente collocato in una porzione della memoria del programma stesso che è etichettata come non modificabile
  - Alcune porzioni della memoria gestite direttamente dal sistema operativo possono non essere accessibili, neanche in lettura, dal programma

Eccezione di accesso illegale

- Se un programma tenta di scrivere/leggere in una porzione della memoria in cui non ha il diritto di effettuare tale operazione, viene tipicamente generata dal processore una eccezione hardware di accesso illegale alla memoria
  - Le eccezioni hardware sono un meccanismo del processore che fa interrompere l'esecuzione dell'istruzione in corso e saltare immediatamente all'esecuzione di codice speciale dedicato alla gestione dell'eccezione
  - Tipicamente il codice di gestione delle eccezioni hardware di accesso illegale alla memoria termina immediatamente il processo (che quindi fallisce)

Segmentation Fault

- In ambiente UNIX l'eccezione scatenata da un accesso illegale alla memoria è tipicamente chiamata *Segmentation Fault*
- Come sappiamo, le eccezioni sono poi gestite da codice dedicato e stabilito dal sistema operativo nella fase di avvio
- Il codice di gestione dell'eccezione *Segmentation Fault* tipicamente termina forzatamente il processo che ha generato l'eccezione

Accesso fuori da un array

- In conclusione, accedere fuori da un array è un errore sotto due punti di vista:
  - Errore logico
    - Non ha senso accedere ad un elemento al di fuori dell'array stesso
  - Errore di gestione della memoria
    - Corruzione della memoria nel caso di accesso in scrittura

Versione corretta

```
main()
{
  const int max_M = 100 ; double vett_temper[max_M] ;


  for (int i = 0; i < max_M; i++) {
    cin>>vett_temper[i];
    if (vett_temper[i] < 0) {
     vett_temper[i] = -1 ; // inseriamo il terminatore
       break ;
    }
  } // nota: se inseriamo max_M valori non vi sarà alcun terminatore!


  double media = 0. ;
  int num_val_letti ; // a fine ciclo conterra' il num di valori letti


  for (num_val_letti = 0; num_val_letti < max_M &&
        vett_temper[num_val_letti] >= 0; num_val_letti++)
    media += vett_temper[num_val_letti];


  media /= num_val_letti ;
  cout<<"Media: "<<media<<endl<<endl ;
  cout<<"Valori superiori alla media:"<<endl ;
  for (int i = 0 ; i < max_M && vett_temper[i] > 0 ; i++)
    if (vett_temper[i] > media) cout<<vett_temper[i]<<endl ;
}
```

Questa condizione verifica che non si acceda fuori dai limiti dell'array dinamico

Domanda

- Dato un vettore dinamico di interi rappresentato mediante
  - un *array*
  - un contatore **cont** del numero di elementi
- A quanto è uguale in ogni istante il numero di elementi del vettore dinamico?

Risposta ed altra domanda

- E' uguale ovviamente a **cont**
- Se, in un certo istante si volesse inserire un nuovo elemento in fondo al vettore
- A cosa sarebbe uguale l'indice dell'elemento dell'*array* in cui memorizzare il nuovo elemento?
- Cosa bisognerebbe fare dopo aver aggiunto il nuovo elemento?

Risposta

- L'indice del nuovo elemento sarebbe uguale a **cont**
- Dopo aver inserito il nuovo elemento **cont** va incrementato di 1

Doppia funzione contatore

- Il contatore del numero di elementi ha quindi una doppia funzione
  - Indica il numero di elementi attualmente presenti nel vettore dinamico
  - Fornisce l'indice dell'elemento dell'*array* in cui memorizzare il nuovo elemento nel caso di inserimento in coda

Esercizio

- Dato un vettore di al più *max_M*=5 elementi interi non nulli, si copino in un altro vettore solo gli elementi compresi tra 10 e 500
- Al termine, si stampi (solo) il numero di valori copiati nel secondo vettore
- Proviamo a risolverlo assieme …

Idee

- Ipotesi: implementiamo il vettore utilizzando il valore 0 come terminatore
- Bisognerà ovviamente scandire tutti gli elementi del primo vettore
  - Mentre si scandisce il vettore, si può copiare ogni valore accettabile nel nuovo vettore ed incrementare, per il secondo vettore, un contatore diverso da quello utilizzato per scandire il primo vettore
- Infine bisognerà stampare il valore finale del contatore relativo al secondo vettore

Struttura dati

- Costante (int) per denotare la dimensione massima dei due vettori: max_M
  - Probabilmente il secondo vettore avrà meno valori ammissibili del primo, ma perché il programma funzioni in tutti i casi, gli array utilizzati per rappresentare entrambi i vettori devono essere dimensionati per contenere max_M elementi
- Servono due array di double, ciascuno di dimensione pari a max_M
- Servono, poi, due variabili ausiliarie (int) come contatore della scansione del primo vettore e come contatore dei valori effettivamente copiati

Proposta programma

```
main()
{
    const int max_M = 5 ;
    int vett_uno[max_M] = { 100, 3, 200, 0, 300 } ;
    int vett_due[max_M];

    int conta=0;
    for (int i=0; vett_uno[i] != 0 && i<max_M; i++)
      if (vett_uno[i]>=10  &&  vett_uno[i]<=500) {
          vett_due[conta]=vett_uno[i];
          conta++;
      }
    if (conta < max_M) // altrimenti scriviamo "fuori"
        vett_due[conta] = 0; // inseriamo il terminatore
    cout<<"Sono stati copiati "<<conta<<" elementi"<<endl;
}
```

COSA STAMPA?
COME MAI?
E' CORRETTO?

Proposta programma

Si accede all'elemento *i*-esimo **prima** di controllare di non essere fuori dall'array

```
main()
{
    const int max_M = 5 ;
    int vett_uno[max_M] = { 100, 3, 200, 0, 300 } ;
    int vett_due[max_M];

    int conta=0;
    for (int i=0; vett_uno[i] != 0 && i<max_M; i++)
      if (vett_uno[i]>=10  &&  vett_uno[i]<=500) {
          vett_due[conta]=vett_uno[i];
          conta++;
      }
    if (conta < max_M) // altrimenti scriviamo "fuori"
        vett_due[conta] = 0; // inseriamo il terminatore
    cout<<"Sono stati copiati "<<conta<<" elementi"<<endl;
}
```

COSA STAMPA?
COME MAI?
E' CORRETTO?

Versione corretta

```
main()
{
    const int max_M = 5 ;
    int vett_uno[max_M] = { 100, 3, 200, 0, 300 } ;
    int vett_due[max_M];

    int conta=0;
    for (int i=0; i<max_M && vett_uno[i] != 0; i++)
      if (vett_uno[i]>=10  &&  vett_uno[i]<=500) {
          vett_due[conta]=vett_uno[i];
          conta++;
      }
    if (conta < max_M) // altrimenti scriviamo "fuori"
        vett_due[conta] = 0; // inseriamo il terminatore
    cout<<"Sono stati copiati "<<conta<<" elementi"<<endl;
}
```

COSA STAMPA?
COME MAI?
E' CORRETTO?