INFORMATICA GENERALE II

Ingegneria delle Telecomunicazioni
Università di Trento
A.A. 2003/2004
II Bimestre

Marco Roveri
roveri@irst.itc.it

Array e Puntatori

Array statici

- Gli array la cui dimensione viene definita all'interno del programma come una costante vengono detti *array statici*.
- Molte volte non è possibile determinare a priori la dimensione di array, in quanto non si sa quanti elementi è necessario gestire.
- In C++ è possibile dichiarare degli array la cui dimensione viene determinata solo all'atto dell'esecuzione del programma. Questo genere di array vengono detti *array dinamici*.

2

Array Statici

- Gli array statici sono quelli la cui *dimensione* (numero di elementi) viene specificata nel codice, e una volta compilati, la loro dimensione non cambia.
- Si dichiarano in due modi:
  - Specificando la dimensione:
    int a[10]; // array di 10 elementi
  - Specificando gli elementi iniziali costanti:
    double b[] = {0.4, 2.20, -4.0, 1.11};
    // array di 4 elementi primo elemento 0.4 …

3

Array Statici

- Un array può essere pensato come un insieme di variabili dello stesso tipo.
- Ogni variabile di un vettore è identificata univocamente dal *nome* dell'array e da un *numero intero positivo.*
- Esempio:

  int a[10];

  - Crea un insieme di variabili a[0], a[1], …, a[9]

4

Array Statici

- Ognuna delle variabili di un array è caratterizzata da un indirizzo, un numero di byte occupati e un tipo.
  - il tipo è lo stesso per tutte le variabili di uno stesso vettore, e quindi anche lo spazio occupato da ognuna di esse.
  - L'indirizzo associato è invece differente.

5

Array statici

- Esempio:

```
int main () {
  int a[10];
  for(int i = 0; i < 9; i++)
     cout << "Ind. a[" << i << "] = " << &(a[i]);
     cout << " occupa " << sizeof(a[i]) << " byte\n";
 }
```

6

Array statici

- Il seguente frammento contiene un errore tipico:

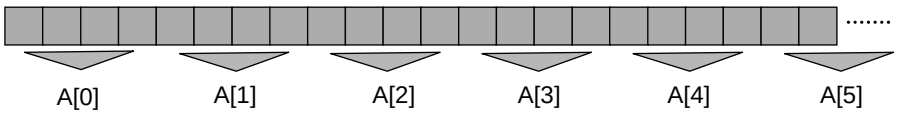
```
int A[10];
for (int j = 0; j <= 10; j++)
    A[i] = 10;
```

- Se eseguito può generare un core-dump: infatti A[10] non esiste.

7

Array Statici

- Se eseguiamo programma precedente risulta evidente che le aree di memoria delle variabili sono consecutive.



- Indirizzo, numero di byte occupati, numero di elementi di un array statico si possono trovare usando le solite primitive & e sizeof.

8

Array Statici

- Esempio (da aggiungere all'esempio precedente):

```
cout << "Ind. a = " << &a;
cout << " dimensione di a = " << sizeof(a);
cout  << " numero elementi = " << sizeof(a)/sizeof(int);
cout  << endl;
```

- Notare che &a e &(a[0]) coincidono.

9

Array Allocati Dinamicamente

- Quando non si conosce a priori la dimensione massima dell'array si ricorre ad una dichiarazione dinamica.
- Esistono due modi per dichiarare un array dinamico:
  - int x = 10; int A[x];
  - int x = 10; int *A; A = new int[x];

10

Array Allocati Dinamicamente

```
int main () {
   int dim;
   // int * A;                  // modo 2
   cout << "Inserire dimensione: " ; cin >> dim;
   int A[dim];                  // modo 1
   // A = new int [dim];   // modo 2
   cout << "Ind. A = " << &A;
   cout << " dimensione di a = " << sizeof(A);
   cout << " numero elementi = " << sizeof(A)/sizeof(int) << endl;
   for(int i = 0; i < 9; i++)
      cout << "Ind. A[" << i << "] = " << &(A[i]);
      cout << " occupa " << sizeof(A[i]) << " byte\n";
  }
```

11

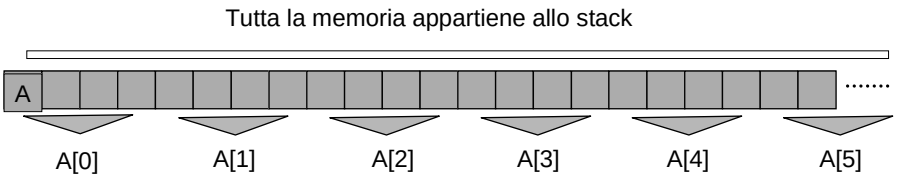
Array Allocati Dinamicamente

- L'ouptut prodotto dal programma precedente è differente a seconda che si usi il "modo 1" o il "modo 2".
  - In "modo 2":
    - L'indirizzo dell'array a non coincide con l'indirizzo di A[0].
    - Il numero di byte occupati dall'array A non coincide con il numero di byte occupati da ogni elemento moltiplicato per il numero di elementi.
    - Si può verificare che il numero di byte occupati da A non cambia al variare del numero di elementi.

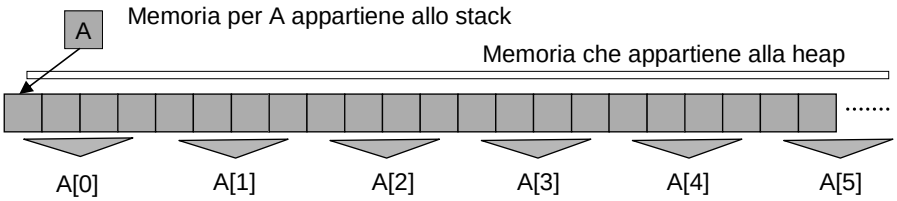
12

Array Allocati Dinamicamente

Statico (modo 1)



Dinamico (modo 2)



13

Array Allocati Dinamicamente

- Nell'esempio abbiamo visto che anche usando il "modo 2" possiamo accedere agli elementi dell'array con A[0], A[1], …
- In particolare valgono le seguenti regole:
  - se A è un puntatore a intero, allora A[0] è equivalente a *A. Questo significa che A[0] è come se fosse una variabile la cui zona di memoria associata è costituita dai byte a partire dall'indirizzo in A.
  - per A[1] vale una regola analoga. Dal punto di vista della programmazione ad alto livello, è equivalente a una variabile; la sua zona di memoria associata è quella subito sopra a A[0]. Similmente per A[2], A[3], …

14

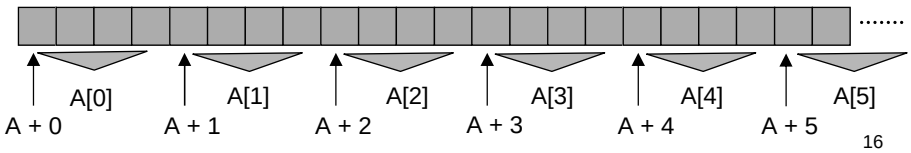
Aritmetica dei Puntatori

- In C++, è possibile effettuare delle semplici operazioni arimetiche sui puntatori, e fra puntatori e interi.
- Il caso più semplice è quello in cui si somma un intero a un puntatore: e.g. A + 3
  - Il risultato non è però quello che ci si potrebbe aspettare a prima vista.
  - Il risultato ***non*** coincide con il valore di A a cui sommo 3. L'indirizzo che si otterrebbe sarebbe poco significativo. Sarebbe l'indirizzo del terzo byte del primo intero della zona puntata da A (assumendo che A sia un array di interi).
  - Il risultato coincide con l'indirizzo del terzo elemento della sequenza puntata da A. Il suo valore numerico si ottiene sommando ad a il numero di byte occupati dall'intero, moltiplicato per 3.

15

Aritmetica dei Puntatori

- Regola generale:
  - un puntatore a un tipo è l'indirizzo iniziale di una zona di memoria in cui sono memorizzati uno o più valori di quel tipo.
  - A+i è l'indirizzo iniziale dell'i-esimo valore della sequenza puntata da A.
  - &A[i] e A + i sono equivalenti.



16

Aritmetica dei Puntatori

- Le espressioni &p[i] e p+i sono dunque equivalenti come lo sono anche le espressioni p[i] e *(p+i).
- Dati due puntatori dello stesso tipo p1 e p2 che puntano entrambi ad elementi di uno stesso array, |p2-p1| denota il numero di elementi compresi tra p1 e p2.

17

Aritmetica dei Puntatori

- Esempio:

```
int main () {
  int x[10]; int *p;
  for(int i = 0; i < 10; i++) x[i] = 0;
  p = x;            // indirizo  primo elem.
  *(p + 3) = 4;   // equivale a x[3]=4
  p = &x[5];
  p += 4;          // avanza di 4
 *p = 7;          // equivale a x[9]=7
  for(int i = 0; i < 10; i++)
     cout << "x[" << i << "] = " << x[i] << endl;
}
```

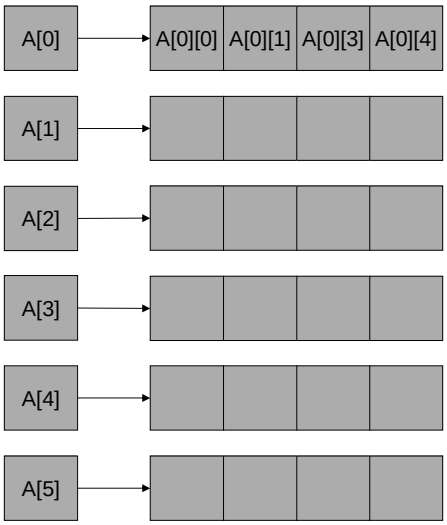
18

Array multidimensionali

- È possibile dichiarare array i cui elementi sono a loro volta degli array, generando degli array multidimensionali (matrici).
- Sintassi:
  tipo id[dim1][dim2]…[dimn];
  tipo id[dim1][dim2]…[dimn]={lista_valori};
  tipo id[][dim2]…[dimn]={lista_valori};
- Quindi in C++ un array multidimensionale dim1 x dim2 x … x dimn può essere pensato come un array di dim1 array multidimensionali dim2 x … x dimn.

19

Array multidimensionali



20

Funzioni con parametri di tipo array

- Una funzione può avere un argomento formale del tipo "array di T" che corrisponde ad un puntatore ad oggetti di tipo T
- Il corrispondente parametro attuale è costituito dal nome di un array di oggetti di tipo T che equivale all'indirizzo del suo primo elemento
- Quando viene passato un array ad una funzione, non vengono copiati i suoi elementi nel corrispondente parametro attuale ma viene ricopiato solo l'indirizzo del primo elemento

21

Funzioni con parametri di tipo array

- Esempio:

```
// funzione che somma i primi n elementi di un array.
int somma(int v[], int n) {
   int i, s=0;
   for (i=0; i<n; i++) s += v[i];
   return s;
 }
```

- Nella dichiarazione di un parametro formale di tipo array (monodimensionale) si può omettere la specifica della dimensione per poter operare su array di dimensioni diverse.

22

Esempio di passaggio di array

```
int main() {
  int a[100], n;
  cout<<"Quanti numeri (max 100)?";
  cin >> n; if ((n>100)|| (n ≤ 0)) return;
  for (int i=0; i<n; i++) {
    cout << endl << i+1 << ": ";
    cin >> a[i];
  };
  cout << "La media vale "
       << somma(a,n)/n << endl;
}
```

23

Passaggio array multimemsionali

- Dato che nelle chiamate di funzioni viene passato solo l'indirizzo alla prima locazione dell'array, un parametro array può essere specificato usando indifferentemente la notazione degli array o dei puntatori
- Le seguenti scritture sono equivalenti:
  int f(int arr[dim1][dim2]…[dimN]);
  int f(int arr[][dim2]…[dimN]);
  int f(const int *arr[dim2]…[dimN]);

24

Funzione che restituisce un array

- Una funzione non può restituire direttamente un array ma può restituire un puntatore al primo elemento dell'array
- Un errore tipico che si commette in tale situazione è la restituzione di un puntatore ad una variabile locale che viene rimossa dalla memoria al terminare della funzione:

```
int *times(int a[10], int k) {
  int b[10];
  for (int i=0; i<10; i++)
     b[i]=a[i]*k;
  return b; // errore b è locale!
}
```

25

Funzione che restituisce un array

```
// eleva al quadrato e somma indice
int * compute(int i[], int n) {
  int * r = new int [n];
  for(int j = 0; j < n; j++)
     r[j] = i[j] * i[j] + j;
  return r;
}

// La routine chiamante diventa responsabile
// della deallocazione dell'area di memoria occupata
// da r.
```

26

Esercizi

- Come mai nel programma "compute" precedente possiamo restituire "r"?
- Riscrivere la funzione "somma" usando solo l'aritmetica dei puntatori.
- Scrivere una funzione che memorizza in un array il risultato del fattoriale dei primi N numeri, e lo ritorni: A[i] = i!
- Scrivere un programma che stampa indirizzi di un array X (i.e. &X, &(X[0]), &(X[1]), …), chiami una funzione print(X, N) che a sua volta stampa le stesse informazioni e discuterne gli output prodotti.

27