Gli array statici

1 Dichiarazione e utilizzo di un array

Gli array sono **sequenze di variabili dello stesso tipo** cui elementi vengono situate consecutivamente nella memoria. Per dichiarare un array bisogna **specificare** il **tipo** dei suoi elementi, il **nome** e la **dimensione**. Per esempio, con `int v[5];` si crea nella memoria un array di 5 `int`:

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

Gli array dichiarati in questo modo sono **statici** perché **la loro dimensione e il loro tipo non sono modificabili** e la loro posizione nella memoria non si può cambiare. Inoltre vengono gestiti in maniera automatica: la memoria che occupano viene allocata quando si entra nel ambito della loro visibilità e deallocata quando l'esecuzione del loro ambito di visibilità termina.

C e C++ **non verificano gli indici dell'array**, cioè non viene controllato se gli indici siano validi (stiano dentro la dimensione). Per esempio,

```
double v[2]={1.2,0.8,3.4};
cout << v[2];
```

è un errore perché l'indice dell'ultimo elemento è 1. Al momento della compilazione non viene segnalato nessun errore. Durante l'esecuzione

- può succedere che il sistema operativo ferma il programma quando il programma tenta di eseguire il commando `cout << v[2];`,
- oppure il programma esegue ma il suo comportamento è casuale (stampa ciò che si trova nella memoria dopo `v[1]`).

Nel programma che segue nella riga 6 viene dichiarata un array i cui elementi sono `double`, il cui nome è `n` e la cui dimensione è 10. L'array `n` può essere utilizzato per rappresentare un vettore di 10 numeri reali. Per fare riferimento ad un elemento bisogna fornire il nome dell'array e l'indice dell'elemento fra parentesi quadrati. L'indice del primo elemento è 0. Per esempio, l'istruzione `n[2]=3.2;` assegna il valore 3.2 al terzo elemento di `n`. Il programma prende 10 numeri dalla tastiera, li mette nell'array `n` e poi ne calcola la media.

```
1  // utilizzo di un array
2  #include<iostream>
3  using namespace std;
4
5  int main(){
6    double n[10],m;
7    for(int i=0;i<=9;i=i+1)
8      cin >> n[i];
9    m=0;
10   for(int i=0;i<=9;i=i+1)
11     m=m+n[i];
12   cout << m/10 << endl;
13   return 0;
14 }
```

È possibile inizializzare i valori degli elementi durante la dichiarazione. L'istruzione `int x[4]={2,4,6,19};` crea un'array di 4 interi i cui valori sono 2, 4, 6 e 19. Quando i valori iniziali sono forniti, non si è costretti a specificare la dimensione. Per esempio, `double x[]={2.1,4.2,6.3,19.4,0.1};` crea un'array di 5 elementi.

2 Operatori ++, --, +=, -=, *= e /=

Il seguente programma illustra fra i commenti come funzionano gli operatori `++`, `--`, `+=`, `-=`, `*=` e `/=`.

```
1  // ++ -- += -= *= /=
2  #include<iostream>
3  using namespace std;
4
5  int main(){
6    int k=0;
7    k++;                  // incrementa k di 1
8    cout << k << endl;
9    ++k;                  // incrementa k di 1
10   cout << k << endl;
11   k--;                  // decrementa k di 1
12   cout << k << endl;
13   --k;                  // decrementa k di 1
14   cout << k << endl;
15   cout << k++ << endl; // stampa k e poi la incrementa di 1
16   cout << k << endl;
17   cout << ++k << endl; // incrementa k di 1 e poi la stampa
18   cout << k << endl;
19
20   k+=10;                // corrisponde a k=k+10;
21   cout << k << endl;
22   k*=10;                // corrisponde a k=k*10;
23   cout << k << endl;
24   k-=10;                // corrisponde a k=k-10;
25   cout << k << endl;
26   k/=5;                 // corrisponde a k=k/5;
27   cout << k << endl;
28
29   return 0;
30 }
```

3 Array come parametro

Nel programma che segue, nella riga 5 comincia la definizione di una funzione che prende un array come primo parametro. Maggior parte delle volte viene passato alla funzione che prende un array anche il numero di elementi (in questo caso il secondo parametro della funzione) perché all'interno della funzione non c'è modo di determinarlo. La funzione viene chiamata nella riga 19 dove bisogna indicare il nome dell'array che serve da parametro.

```
1  // cerca elemento minimo di un vettore
2  #include<iostream>
3  using namespace std;
4
5  int smallest(int v[], int n){
6    int m=v[0];
7    for(int i=1;i<n;i++)
```

```
    if(v[i]<m)
      m=v[i];
  return m;
}

int main(){
  int k[10];

  for(int i=0;i<10;i++)
    cin >> k[i];

  cout << smallest(k,10) << endl;

  return 0;
}
```

Gli array vengono passati alle funzioni utilizzando il meccanismo passaggio di parametro per riferimento. Per evitare che una funzione possa modificare gli elementi di un array che viene passato come parametro bisogna aggiungere la parola chiave **const** nell'elenco di parametri. Nel programma che segue la funzione **print** stampa un array a video senza modificarlo e quindi si utilizza **const** per garantire che l'array passato come parametro non viene modificato.

Il programma sviluppa l'algoritmo "ordinamento per selezione". Il procedimento è il seguente: si cerca l'elemento più piccolo del vettore e si scambia questo elemento con il primo elemento, poi si cerca l'elemento più piccolo del vettore escludendo la prima posizione e si scambia questo elemento con il secondo elemento, poi si cerca l'elemento più piccolo del vettore escludendo le prime due posizioni e si scambia questo elemento con il terzo elemento, e così via fino al momento in cui si ottiene un array ordinato.

Nel programma la funzione **minindex** restituisce l'indice dell'elemento più piccolo fra gli elementi **v[i]**, **v[i+1]**, ..., **v[j]**. La funzione **exchange** effettua uno scambio. La funzione **selection** implementa l'algoritmo "ordinamento per selezione". La funzione **selection** deve poter modificare l'array preso in input e quindi l'array viene passato senza **const**.

```
// ordinamento per selezione
#include<iostream>
using namespace std;

// stampa vettore di interi
void print(const int n, int v[]){
  for(int i=0;i<n;i++)
    cout << v[i] << "␣";
  cout<<"\n";
}

// restituisce indice dell'elemento minimo in (v[i],v[i+1],...,v[j])
int minindex(const int v[], int i, int j){
  int m=i;
  for(int k=i+1;k<=j;k++)
    if(v[k]<v[m])
      m=k;
  return m;
}

// scambio
void swap(int &a, int &b){
  int temp=a;
  a=b;
```

```
25    b=temp;
26  }
27
28  // ordinamento per selezione
29  void selection(int n, int v[]){
30    int i,mi;
31    for(i=0; i<n-1; i++){
32      mi = minindex(v,i,n-1);
33      swap(v[mi],v[i]);
34    }
35  }
36
37  int main(){
38    const int x=10;
39    int v[x]={10,6,8,2,4,1,7,8,2,3};
40    print(x,v);
41    selection(x,v);
42    print(x,v);
43    return 0;
44  }
```

Nel programma che segue ci sono diversi funzioni per illustrare l'utilizzo degli array.

La funzione **read_vector** legge un vettore dalla tastiera. La funzione **print_vector** stampa un array. La funzione **ordered** restituisce **true** se l'array passato come parametro è ordinato e **false** altrimenti. La funzione controlla ogni coppia di elementi successivi nell'array e restituisce **false** appena capita che **v[k]>v[k+1]** si verifichi. Se questo non succede mai allora l'array è ordinato e la funzione restituisce **true**. La funzione **turn_around** inverte l'ordine degli elementi in un array.

La funzione **try_to_change** illustra il fatto che anche se un array intero viene passato utilizzando passaggio di parametro per riferimento, per un singolo elemento si utilizza passaggio di parametro per valore. Quindi la chiamata nella riga 97 non può modificare il primo elemento dell'array. Viceversa, la chiamata nella riga 101 cambia un elemento del vettore perché la funzione **change_int** utilizza passaggio di parametro per riferimento.

La funzione **search_it** implementa la cosiddetta ricerca binaria. La ricerca binaria è un algoritmo che permette di verificare la presenza di un elemento in un array ordinato. L'idea è la seguente. Sia **x** l'elemento da cercare e **m** l'indice dell'elemento centrale: se **x==v[m]** allora l'elemento c'è, se **x>v[m]** allora l'elemento può esserci solo fra gli elementi che hanno indice maggiore di **m**, se **x<v[m]** allora l'elemento può esserci solo fra gli elementi che hanno indice minore di **m**. La ripetuta applicazione di questa idea trova **x** oppure "svuota" la parte del vettore dove **x** può posizionarsi e quindi stabilisce che **x** non è presente nell'array. Nella funzione **f** e **l** sono gli indici del primo e dell'ultimo elemento del segmento dell'array dove **x** può essere.

La funzione **search_rec** implementa la ricerca binaria con ricorsione.

```
1   // funzioni con array
2   #include<iostream>
3   using namespace std;
4
5   // dimensione massima di un array durante l'esecuzione:
6   const int max_size=100;
7
8   // lettura di un array dalla tastiera
9   int read_vector(int v[]){
10    int n;
11    do{
12      cout << "Dimensione (al massimo " << max_size << "): ";
13      cin >> n;
```

```
14    }while(n>max_size);
15    for(int i=0;i<n;i++)
16      cin >> v[i];
17    return n;
18  }
19
20  // stampa array
21  void print_vector(const int v[], int n){
22    for(int i=0;i<n;i++)
23      cout << v[i] << "␣";
24    cout << endl;
25  }
26
27  // restituisce true se l'array e' ordinato
28  bool ordered(const int v[], int n){
29    for(int k=0;k<n-1;k++)
30      if(v[k]>v[k+1])
31        return false;
32    return true;
33  }
34
35  // inverte l'ordine degli elementi
36  void turn_around(int v[], int n){
37    int i=0,j=n-1,t;
38    while(i<j){
39      t=v[i];
40      v[i]=v[j];
41      v[j]=t;
42      i++;
43      j--;
44    }
45  }
46
47  // cambia parametro passato per valore
48  void try_to_change(int e){
49    cin >> e;
50  }
51
52  // cambia parametro passato per riferimento
53  void change_int(int &e){
54    cin >> e;
55  }
56
57  // ricerca binaria iterativa
58  bool search_it(const int v[], int k, int x){
59    int f=0,l=k-1,m;
60    while(f<=l){
61      m=(f+l)/2;
62      if(v[m]==x)
63        return true;
64      if(v[m]<x)
65        f=m+1;
66      else
67        l=m-1;
```

```
68    }
69    return false;
70  }
71
72  // ricerca binaria ricorsiva
73  bool search_rec(const int v[], int f, int l, int x){
74    if(f>l)
75      return false;
76    int m=(f+l)/2;
77    if(v[m]==x)
78        return true;
79    if(v[m]<x)
80      return search_rec(v,m+1,l,x);
81    else
82      return search_rec(v,f,m-1,x);
83  }
84
85  // modulo main
86  int main(){
87    int v[max_size],k;
88    k=read_vector(v);
89    if(ordered(v,k))
90      cout << "il vettore e' ordinato" << endl;
91    else
92      cout << "il vettore non e' ordinato" << endl;
93    print_vector(v,k);
94    turn_around(v,k);
95    print_vector(v,k);
96
97    try_to_change(v[0]); // non cambia v[0] perche singolo elemento e' passato per valore
98    print_vector(v,k);
99    cout << "change first entry to: ";
100
101   change_int(v[0]);  // cambia v[0] perche il singolo elemento e' passato per riferimento
102   print_vector(v,k);
103
104   if(search_it(v,k,3))
105     cout << "3 e' presente" << endl;
106   else
107     cout << "3 non e' presente" << endl;
108
109   if(search_rec(v,0,k-1,3))
110     cout << "3 e' presente" << endl;
111   else
112     cout << "3 non e' presente" << endl;
113
114   return 0;
115 }
```

4 Array multidimensionali

Gli array possono essere multidimensionali. Un array bidimensionale può essere utilizzato per rappresentare una matrice. Il seguente programma illustra l'utilizzo di array bidimensionali per lavorare con matrici. Gli array multi-

dimensionali sono simili a quelli monodimensionali. Una differenza importante è che quando una funzione prende come parametro uno o più array multidimensionali, nell'elenco di parametri della funzione **possiamo omettere dalla specifica degli array multidimensionali solo la prima dimensione e le altre devono essere fornite tramite costanti** conosciuti durante la compilazione. Nel programma sotto questo costante è **`max_size`** utilizzato per specificare il numero di colonne quando una matrice (array bidimensionali) appare come parametro (per esempio, nella riga 9).

```
1  // funzioni con array bidimensionali (matrici)
2  #include<iostream>
3  using namespace std;
4
5  // dimensione massima di un array durante l'esecuzione:
6  const int max_size=100;
7
8  // lettura di una matrice dalla tastiera
9  void read_matrix(int m[][max_size], int &r, int &c){
10   do{
11     cout << "Numero di righe (al massimo " << max_size << "): ";
12     cin >> r;
13   }while(r>max_size);
14   do{
15     cout << "Numero di colonne (al massimo " << max_size << "): ";
16     cin >> c;
17   }while(c>max_size);
18   for(int i=0;i<r;i++)
19     for(int j=0;j<c;j++)
20       cin >> m[i][j];
21 }
22
23 // stampa una matrice a video
24 void print_matrix(const int m[][max_size], int r, int c){
25   for(int i=0;i<r;i++){
26     for(int j=0;j<c;j++)
27       cout << m[i][j]<< " ";
28     cout << endl;
29   }
30 }
31
32 // true se la matrice e' simmetrica
33 bool symmetric(const int m[][max_size], int r){
34   for(int i=0;i<r;i++)
35     for(int j=i+1;j<r;j++)
36       if(m[i][j]!=m[j][i])
37         return false;
38   return true;
39 }
40
41 // restituisce in v la somma degli elementi di ciascun riga
42 void row_sum(const int m[][max_size], int r, int c, int v[]){
43   for(int i=0;i<r;i++){
44     v[i]=0;
45     for(int j=0;j<c;j++)
46       v[i]+=m[i][j];
47   }
48 }
49
```

```
50 // stampa vettore
51 void print_vector(int v[], int n){
52   for(int i=0;i<n;i++)
53     cout << v[i] << " ";
54   cout << endl;
55 }
56
57 int main(){
58   int m1[max_size][max_size],d1,d2;
59   int s[max_size];
60   read_matrix(m1,d1,d2);
61   print_matrix(m1,d1,d2);
62
63   if(d1==d2)
64     if(symmetric(m1,d1))
65       cout << "la matrice e' simmetrica" << endl;
66     else
67       cout << "la matrice non e' simmetrica" << endl;
68
69   row_sum(m1,d1,d2,s);
70   cout << "somme delle righe: ";
71   print_vector(s,d1);
72   return 0;
73 }
```