Puntatori, allocazione dinamica della memoria

1 Puntatori

Un puntatore è una variabile che come valore contiene un indirizzo della memoria. Quando un puntatore **p** contiene l'indirizzo di una variabile **v** allora il puntatore può essere utilizzato per accedere alla variabile **v** tramite il puntatore **p** e si dice che il puntatore **p** punta alla variabile **v**.

Per dichiarare un puntatore bisogna specificare il tipo della variabile alla quale il puntatore punterà, indicare con ***** che si tratta di un puntatore e fornire il nome del puntatore. Per esempio, `int *p1;` dichiara un puntatore che può essere utilizzato per puntare ad una variabile di tipo **int**.

I due operatori di base che si usano con puntatori sono ***** e **&**. Dato un puntatore **p**, con l'espressione ***p** si può accedere al valore della variabile puntata da **p** per leggerlo o modificarlo. Data una variabile **v**, l'espressione **&v** restituisce l'indirizzo di **v**.

Il seguente programma illustra l'utilizzo di ***** e **&**. Nella riga 6 vengono dichiarati un puntatore a **float** senza inizializzarlo (quindi non si sa dove punta **fp**) e una variabile di tipo **float** inizializzata a 3.5. Nella riga 7 viene assegnata a **fp** l'indirizzo di **fn** grazie all'utilizzo dell'operatore **&**. Quindi l'effetto della riga 7 è che **fp** punta (si dice anche che fa riferimento) alla variabile **fn**. Nella riga 8 viene stampato il valore della variabile **fn** (cioè 3.5) utilizzando il puntatore **fp** con l'operatore *****. Nella riga 9 il valore della variabile **fn** viene modificato utilizzando il puntatore **fp** con l'operatore *****. Nella riga viene stampato il valore di **fn**, cioè 2.4.

```
1  // operatori * e &
2  #include<iostream>
3  using namespace std;
4
5  int main(){
6    float *fp, fn=3.5;
7    fp=&fn;
8    cout << *fp << endl;
9    *fp=2.4;
10   cout << fn << endl;
11   return 0;
12 }
```

Inizializzando un puntatore a **NULL** si può rappresentare esplicitamente il fatto che il puntatore non punta a nessuna variabile.

2 Relazione fra puntatori e array

Il nome di un array statico senza specificare un indice restituisce l'indirizzo del primo elemento. Questo indirizzo può essere assegnato ad un puntatore. Dopodiché il puntatore può essere utilizzato per accedere agli elementi dell'array con la stessa sintassi che si utilizza con gli array. Il seguente programma illustra questa possibilità. Il primo ciclo **for** utilizza il puntatore **ip** per dare valori agli elementi dell'array **iv**.

```
1  // relazione di puntatori e array statici
2  #include<iostream>
3  using namespace std;
```

```
4
5  int main(){
6    int *ip, iv[10];
7    ip=iv;
8    for(int i=0;i<10;i++)
9      cin >> ip[i];
10   for(int i=0;i<10;i++)
11     cout << iv[i] << "␣";
12   return 0;
13 }
```

3 Aritmetica dei puntatori

Con i puntatori si possono utilizzare gli operatori **++** e **--**. L'effetto non è spostare il puntatore di un byte ma di quanto byte tanto il tipo puntato occupa nella memoria. Per esempio, se **p** punta ad un intero **x** di tipo **int**, allora dopo l'operazione **p++** il puntatore **p** punterà all'intero di tipo **int** che succede **x** nella memoria. La stessa logica si applica con **+=** e **-=**. Per esempio, l'operazione **p+=3;** sposta **p** di tre **int** se **p** è un puntatore a interi di tipo **int**.

Il seguente programma illustra l'aritmetica dei puntatori.

```
1  // aritmetica di puntatori
2  #include<iostream>
3  using namespace std;
4
5  int main(){
6    int *ip, iv[]={0,1,2,3,4,5,6,7,8,9};
7    ip=iv;
8    cout << *ip << endl; // stampa 0
9    ip++;
10   cout << *ip << endl; // stampa 1
11   ip+=3;
12   cout << *ip << endl; // stampa 4
13   ip--;
14   cout << *ip << endl; // stampa 3
15   ip-=-2;
16   cout << *ip << endl; // stampa 5
17   return 0;
18 }
```

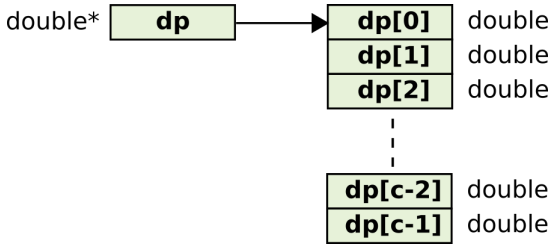
4 Allocazione dinamica della memoria

I puntatori, con le operatori **new** e **delete**, permettono di utilizzare la memoria in maniera dinamica ed efficiente. L'operatore **new** alloca memoria per uno o più variabili e ne restituisce l'indirizzo. L'operatore **delete** dealloca (libera) la memoria puntata da un puntatore. Il programma che segue alloca memoria per un array dinamico di **c** elementi dove il valore di **c** è preso dalla tastiera. Una volta allocata la memoria per l'array, l'accesso agli elementi avviene tramite il puntatore **dp** con la stessa sintassi che si utilizza con gli array statici. Nella riga 15 la memoria viene liberata.

```
1  // allocazione della memoria
2  #include<iostream>
3  using namespace std;
4
5  int main(){
```

```
6   double *dp;
7   int c;
8   cin >> c;
9   dp=new double[c]; // alloca memoria per c numeri reali
10  for(int i=0;i<c;i++)
11    cin >> dp[i];
12  for(int i=0;i<c;i++)
13    cout << dp[i] << " ";
14  cout << endl;
15  delete[] dp; // dealloca (libera) la memoria
16  return 0;
17 }
```

Il programma precedente crea nella memoria la struttura illustrata dalla figura successiva. Il puntatore **dp** (di tipo **double***) punta ad un blocco di **c** numeri reali (ognuno di tipo **double**).



Il seguente programma contiene una funzione per leggere un vettore dalla tastiera e restituirne l'indirizzo (e anche la dimensione del vettore con passaggio di parametro per riferimento). N.B.: una funzione del genere non può essere implementata utilizzando array statici (vedi lucidi per più dettaglio).

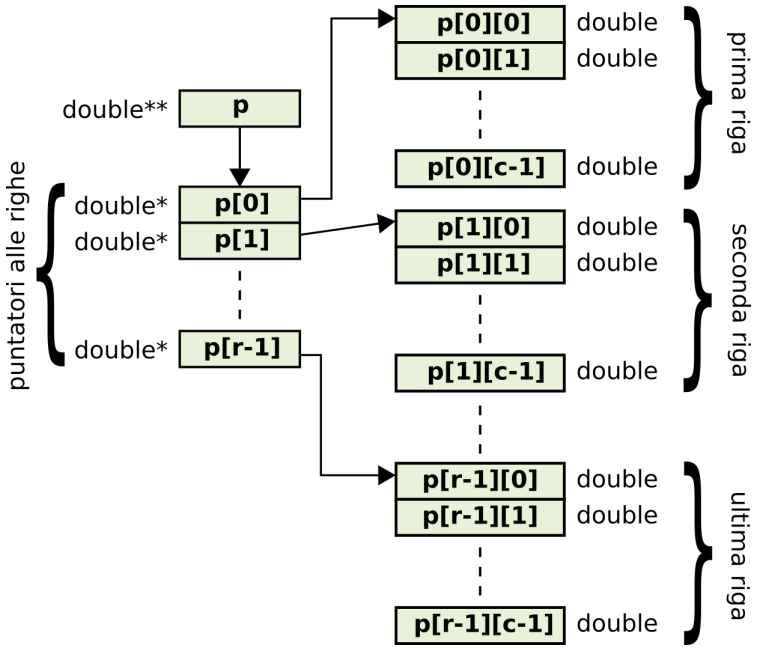
```
1  // funzione che restituisce un array "dinamico"
2  #include<iostream>
3  using namespace std;
4
5  int *readarray(int &c){
6    cout << "Dimensione: ";
7    cin >> c;
8    int *p=new int[c];
9    cout << "Gli elementi: ";
10   for(int i=0;i<c;i++)
11     cin >> p[i];
12   return p;
13 }
14
15 void printarray(int *p, int c){
16   cout << "Gli elementi: ";
17   for(int i=0;i<c;i++)
18     cout << p[i] << " ";
19   cout << endl;
20 }
21
22 int main(){
23   int *v, n; // un puntatore e un intero
24   v=readarray(n);
25   printarray(v,n);
26   delete[] v;
27   return 0;
```

```
28 }
```

Il seguente programma illustra la possibilità di rappresentare matrici con allocazione dinamica della memoria. La funzione **allocmatrix** ha il compito di allocare memoria per una matrice di **r** righe e **c** colonne e restituirne l'indirizzo.

```
1  // matrici con allocazione dinamica
2  #include<iostream>
3  #include<cstdlib>
4  using namespace std;
5
6  // alloca memoria per una matrice
7  double **allocmatrix(int r, int c){
8    double **p;
9    p=new double*[r];
10   for(int i=0;i<r;i++)
11     p[i]=new double[c];
12   return p;
13 }
```

La figura che segue illustra la struttura che **allocmatrix** crea nella memoria. Il puntatore **p** (di tipo **double ****) e quindi un puntatore a puntatore) punta ad un vettore di puntatori **p[0],p[1],...,p[r-1]** (ognuno di tipo **double ***) e ognuno di questi punta ad un blocco di numeri (di tipo **double**) che corrisponde ad una riga della matrice.



La funzione **deallocmatrix** libera la memoria occupata da una matrice. Con un ciclo **for** libera la memoria occupata dai numeri stessi riga per riga e poi libera la memoria occupata dai puntatori che puntano alle righe. La funzione **printmatrix** stampa una matrice a video, **randomentries** riempie una matrice con numeri casuali. Il modulo **main**

- stabilisce il numero di righe (**x**) e colonne (**y**) tramite la tastiera,
- alloca memoria per una matrice di **x** righe e **y** colonne chiamando **allocmatrix**,
- chiama **randomentries** per riempire la matrice con numeri casuali fra 2 e 5,
- stampa la matrice a video con **printmatrix**,

- libera la memoria occupata dalla matrice con **deallocmatrix**.

```
14
15 // dealloca la memoria occupata da una matrice
16 void deallocmatrix(double **m, int r){
17   for(int i=0;i<r;i++)
18     delete[] m[i];
19   delete[] m;
20 }
21
22 // stampa matrice a video
23 void printmatrix(double **m, int r, int c){
24   for(int i=0;i<r;i++){
25     for(int j=0;j<c;j++)
26       cout << m[i][j] << "␣";
27     cout << endl;
28   }
29 }
30
31 // riempie una matric con elementi casuali fra a e b
32 // la memoria per la matrice deve essere allacata precedentemente
33 void randomentries(double **m, int r, int c, int a, int b){
34   for(int i=0;i<r;i++)
35     for(int j=0;j<c;j++)
36       m[i][j]=(double)rand()/RAND_MAX*(b-a)+a;
37 }
38
39 int main(){
40   double **m;
41   int x,y;
42   cin >> x >> y;
43   m=allocmatrice(x,y);
44   randomentries(m,x,y,2,5);
45   printmatrix(m,x,y);
46   deallocmatrice(m,x);
47   return 0;
48 }
```