Puntatori, allocazione dinamica della memoria

András Horváth
horvath@di.unito.it

Dichiarazione di un puntatore

- un puntatore è una variabile che contiene l'indirizzo in memoria di un'altra variabile
- si dice che punta (o fa riferimento) ad un'altra variabile
- la dichiarazione di un puntatore include il tipo della variabile a cui il puntatore punta e il simbolo `*`
- per esempio, puntatore ad una variabile di tipo `int`:

```
6   int *ip;
```

- la riga precedente non crea una variabile di tipo `int` ma una variabile che può contenere l'indirizzo di una variabile di tipo `int`
- come con le variabili non puntatori, una variabile alla quale non è stato assegnato un valore è da ritenere casuale
- quindi dopo la riga 6 `ip` punta ad un indirizzo a caso

Operatori

- l' operatore `&` fornisce l'indirizzo di una variabile
- l' operatore `*` dà accesso al contenuto della variabile a cui punta un puntatore
- esempio:

```
1  #include<iostream>
2  using namespace std;
3
4  int main(){
5    float *fp, fn=3.5; // fp punta a qualche parte a caso
6    fp=&fn; // dopo questa riga fp punta a fn
7    cout << *fp << endl; // stampa il valore (3.5) della variabile
8                         // a cui punta fp
9    *fp=2.4; // assegna un valora (2.4) alla variabile a cui punta fp
10   cout << fn << endl; // stampa il valore (2.4) di fn
11   cout << fp << endl; // stampa l'indirizzo contenuto in fp
12   return 0;
13 }
```


Controllo del tipo

- un puntatore di tipo `type` può puntare solo ad una variabile di tipo `type`
- il compilatore controlla
- per il programma

```
1  #include<iostream>
2  using namespace std;
3
4  int main(){
5    int *ip;
6    double d;
7    ip=&d;
8    return 0;
9  }
```

il compilatore segnala l'errore "cannot convert 'double*' to 'int*' in assignment"

Errori con puntatori

- non assegnare un indirizzo di memoria a un puntatore prima di usarlo:

```
1  #include<iostream>
2  using namespace std;
3
4  int main(){
5    long int *pointer; // pointer punta a qualche parte a caso
6    *pointer=100; // si tenta di assegnare 100 alla variabile
7                  // a cui punta pointer
8    return 0;
9  }
```

il programma si compila (è sintatticamente corretto) ma genera un errore durante l'esecuzione

Relazione fra puntatori e array statici

- avendo dichiarato un array con

```
6    int iv[10];
```

il nome, cioè **iv**, fornisce l'indirizzo del primo elemento

- questo può essere assegnato ad un vero e proprio puntatore ed essere usato per accedere agli elementi:

```
1  #include<iostream>
2  using namespace std;
3
4  int main(){
5    int *ip;
6    int iv[10];
7    ip=iv;
8    for(int i=0;i<10;i++) cin >> ip[i];
9    for(int i=0;i<10;i++) cout << iv[i] << "␣";
10   return 0;
11 }
```


Aritmetica dei puntatori

- con i puntatori si possono utilizzare gli operatori **++** (incrementare) e **−−** (decrementare)
- l'effetto è spostarsi di tanto byte quanto byte il tipo puntato occupa nella memoria
- esempio:

```
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++;    // uguale a ip=ip+1
10   cout << *ip << endl; // stampa 1
11   ip+=3;   // uguale a ip=ip+3
12   cout << *ip << endl; // stampa 4
13   ip--;    // uguale a ip=ip-1
14   cout << *ip << endl; // stampa 3
15   ip-=-2;  // uguale a ip=ip-(-2)
16   cout << *ip << endl; // stampa 5
```


Allocazione dinamica della memoria

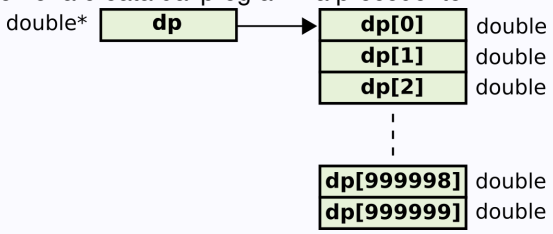
- permette di usare al massimo in maniera flessibile la memoria disponibile
- si fa con i puntatori e con gli operatori **new** e **delete**
- esempio:

```
9    double *dp;
10   dp=new double[1000000]; // alloca memoria per 1000000 double
11
12   for(int i=0;i<1000000;i++)
13     dp[i]=(double)rand()/RAND_MAX; // casuale in [0,1]
14
15   // qua si potrebbe fare qualcosa con questi 1000000 numeri casuali
16
17   // fra new e delete l'array dinamico e' disponibile
18   // e si utilizza come se fosse un array "normale"
19
20   delete[] dp; // libera la memoria (dealloca)
```

(mentre sul mio computer con **double dp[1000000];** il programma genera un errore durante l'esecuzione)

Allocazione dinamica della memoria

struttura nella memoria creata dal programma precedente:



Allocazione dinamica, funzioni con vettori

```
1  // funzione che restituisce un array "dinamico"
2  #include<iostream>
3  using namespace std;
4
5  // legge un vettore dalla tastiera
6  int *readarray(int &c){ // numero di numeri restituito in c
7    cout << "Dimensione:␣";
8    cin >> c; // numero di elementi dalla tastiera
9    int *p=new int[c]; // alloca memoria per c numeri reali
10   cout << "Gli␣elementi:␣";
11   for(int i=0;i<c;i++)
12     cin >> p[i]; // inserisce un numero reale nella posizione i (fatta c volte)
13   return p; // restituisce l'indirizzo dove si trovano i numeri
14 }
```


Allocazione dinamica, funzioni con vettori

```
16 void printarray(int *p, int c){ // p punta ai numeri da stampare
17                                 // c e' il numero di numeri
18   cout << "Gli␣elementi:␣";
19   for(int i=0;i<c;i++)
20     cout << p[i] << "␣";
21   cout << endl;
22 }
23
24 int main(){
25   int *v, n; // un puntatore e un intero
26   v=readarray(n); // in n si riceve il numero di numeri
27                   // in v il loro indirizzo
28   printarray(v,n); // stampa l'array dinamico puntato da v
29   delete[] v; // libera la memoria
30   return 0;
31 }
```


Allocazione dinamica, senza puntatori non si può

- la precedente funzione **readarray** con un array statico produce un errore durante l'esecuzione (sintatticamente non presenta problemi):

```
5  int *readarray(int &c){
6    cout << "Dimensione:␣";
7    cin >> c;
8    int p[c];
9    cout << "Gli␣elementi:␣";
10   for(int i=0;i<c;i++)
11     cin >> p[i];
12   return p;
13 }
```

- l'array statico **p** è "in vita" soltanto all'interno della funzione, restituire il suo indirizzo è sbagliato

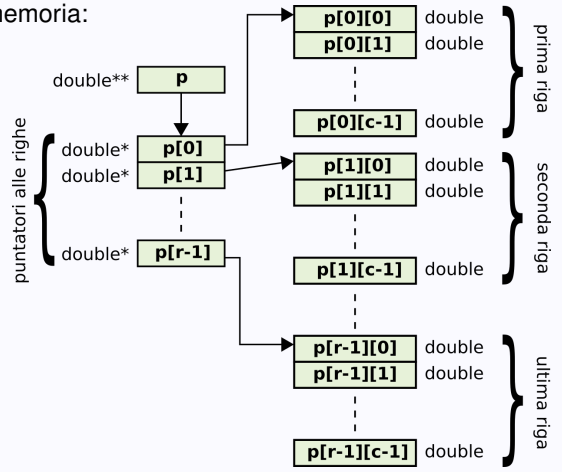
Allocazione dinamica, matrici

- per **rappresentare una matrice** possiamo utilizzare un puntatore `p` di tipo `double**` che punta ad un array dinamico
- usiamo `p` per allocare memoria per un vettore di `r` puntatori di tipo `double*` dove `r` è il numero di righe
- ciascuno di questi puntatori `double*` si usa per allocare memoria per `c` numeri reali dove `c` è il numero di colonne

```
7   int r,c;
8   cin >> r >> c; // numero di righe e colonne dalla tastiera
9   double **p; // puntatore utilizzabile per accedere ad una matrice
10  p=new double*[r]; // alloca memoria per r puntatori
11  for(int i=0;i<r;i++)
12    p[i]=new double[c]; // alloca memoria per c double r volte
```


Allocazione dinamica, matrici

struttura nella memoria:



Allocazione dinamica, matrici

- avendo allocato la memoria, la matrice (un array bidimensionale dinamico) si usa, dal punto di vista della sintassi, come se fosse un array statico

```
14  double a,b;
15  cin >> a >> b; // intervallo per i numeri casuali
16  for(int i=0;i<r;i++)
17    for(int j=0;j<c;j++)
18      p[i][j]=(double)rand()/RAND_MAX/(b-a)+a;
19
20  // stampa matrice
21  for(int i=0;i<r;i++){
22    for(int j=0;j<c;j++)
23      cout << p[i][j] << "␣";
24    cout << endl;
25  }
```


Allocazione dinamica, matrici

- quando la matrice non serve più bisogna liberare (deallocare) la memoria

```
28  for(int i=0;i<r;i++)
29    delete[] p[i]; // libera una riga (eseguita per ogni riga)
30  delete[] p; // libera memoria occupata da r puntatori
```