Università della Calabria

Corso di Laurea in Ingegneria Elettronica

Programmazione Orientata agli Oggetti

A.A. 2006/2007

Riferimenti, puntatori e memoria dinamica

Paolo Trunfio *

* DEIS, Università della Calabria – http://si.deis.unical.it/~trunfio

Riferimenti

Un riferimento può essere considerato un alias per un oggetto:

```
int a = 5;
int &b = a;  // b ed a si riferiscono alla stessa locazione di memoria
b = b + 5;  // a = a + 5
cout << a << endl;  // 10
cout << &b << endl;  // stampa l'indirizzo di b (ovvero di a), per es.: 0x22ff74
```

Un riferimento deve essere inizializzato contestualmente alla dichiarazione, e si riferisce sempre all'oggetto indicato nell'inizializzazione:

```
int &c;  // errore: riferimento non inizializzato
```

I riferimenti sono utilizzati principalmente per il passaggio dei parametri e per i valori di ritorno delle funzioni.

Puntatori

Un puntatore ad un generico tipo T è una variabile capace di contenere l'indirizzo di un oggetto di tipo T. Per dichiarare un puntatore *p* ad un oggetto di tipo T si scrive T ***p***.

```
int a = 3;

int *b = &a;  // b è un puntatore ad interi, inizializzato con l'indirizzo di a

cout << b << endl;  // stampa l'indirizzo di a, per esempio: 0x22ff74

cout << *b << endl;  // stampa il valore della cella puntata da b: 3

*b = 8;  // scrive 8 nella cella puntata da b

cout << a << endl;  // 8
```

L'accesso all'oggetto puntato dal puntatore è detto *dereferenziazione* e si ottiene mediante l'operatore prefisso *.

Puntatori

E' possibile dichiarare array di puntatori:

```
int a = 3, b = 6;

int *v[2];  // un array di 2 puntatori ad interi

v[0] = &a;

v[1] = &b;

cout << *v[0] << " " << *v[1] << endl;  // 3  6
```

E' anche possibile dichiarare puntatori a puntatori:

```
char c = 'A';

char *p1 = &c;  // puntatore a caratteri

char **p2 = &p1;  // puntatore a puntatore a caratteri

**p2 = 'B';

cout << c << endl;  // B
```

Puntatori e costanti

```
char a = '0';
char *b = &a;
const char *c = &a;  // puntatore a costante
*c = '1';  // errore
c = b;  // ok
char *const d = &a;  // puntatore costante
*d = '2';  // ok
d = b;  // errore
const char *const e = &a;  // puntatore costante a costante
*e = '3';  // errore
e = b;  // errore
cout << *e;  // ok
```


Relazione tra puntatori e array

Il nome di un array può essere usato come puntatore al primo elemento dell'array stesso:

```
int a[] = { 5, 10, 15 };
cout << *a << endl;  // 5
cout << *(a+1) << endl;  // 10
cout << *(a+2) << endl;  // 15
cout << *(a+3) << endl;  // lettura fuori dai limiti
*(a+2) = 8;  // a[2] = 8
*(a+3) = 13;  // scrittura fuori dai limiti
```

Allocazione dinamica della memoria

La memoria dinamica (*heap*) è un'area di memoria libera che può essere utilizzata dal programmatore per memorizzare dinamicamente oggetti secondo le necessità dell'applicazione. Per allocare un oggetto nell'heap si usa l'operatore *new*, mentre per deallocarlo si usa l'operatore *delete*.

```
#include <iostream>
using namespace std;

int main() {
  int *p;
  p = new int;  // alloca la memoria per un int, e ne assegna l'indirizzo a p
  *p = 6;  // assegna il valore 6 alla locazione di memoria puntata da p
  cout << "Valore memorizzato: " << *p << endl;  // 6
  delete p;  // rilascia la memoria
}
```


Allocazione dinamica della memoria

E' possibile allocare dinamicamente oggetti di qualsiasi tipo:

```
int *a;
a = new int;  // alloca un int
*a = 75;
cout << *a << endl;  // 75

string *b;
b = new string;  // alloca una stringa
*b = "UNICAL";
cout << *b << endl;  // UNICAL

double *c = new double;  // dichiara e alloca un double
*c = 3.14;
cout << *c << endl;  // 3.14
```

Inizializzazione della memoria dinamica

E' possibile inizializzare la memoria allocata dinamicamente, specificando il valore tra parentesi dopo il nome del tipo:

```
#include <iostream>
using namespace std;

int main() {
  int *p;
  p = new int(15);  // alloca un intero e lo inizializza con il valore 15
  cout << "Valore memorizzato: " << *p << endl;  // 15
  delete p;  // rilascia la memoria

  string *s = new string("UNICAL");  // dichiara, alloca e inizializza
  cout << "Valore memorizzato: " << *s << endl;  // UNICAL
  delete s;  // rilascia
}
```


Allocazione dinamica di array

Con l'operatore *new* è possibile allocare dinamicamente degli array. La dimensione dell'array si specifica tra parentesi quadre dopo il nome del tipo.

```
int main() {
  cout << "Dimensione dell'array: ";
  int dim;
  cin >> dim;
  int *v = new int[dim];  // dichiara e alloca un array di dimensione dim
  for (int i = 0; i < dim; i++) {  // legge l'array
    cout << "Elemento in posizione " << i << ": ";
    cin >> v[i];
  }
  for (int i = 0; i < dim; i++) {  // stampa l'array
    cout << "Elemento in posizione " << i << " = " << v[i] << endl;
  }
  delete[] v;  // rilascia la memoria
}
```

Si noti l'uso delle parentesi quadre dopo l'operatore *delete*.

Allocazione dinamica di array bidimensionali

```
#include <iostream>
using namespace std;

int main() {
 // legge le dimensioni
 int numRighe, numColonne;
 cout << "Numero di righe = ";
 cin >> numRighe;
 cout << "Numero di colonne = ";
 cin >> numColonne;

 // alloca la matrice
 int **m = new int*[numRighe];
 for (int i = 0; i < numRighe; i++)
  m[i] = new int[numColonne];

 // riempie la matrice
 int x = 1;
 for (int i = 0; i < numRighe; i++)
  for (int j = 0; j < numColonne; j++)
   m[i][j] = x++;

 // stampa la matrice
 for (int i = 0; i < numRighe; i++) {
  for (int j = 0; j < numColonne; j++)
   cout << m[i][j] << "\t";
  cout << endl;
 }

 // dealloca la matrice
 for (int i = 0; i < numRighe; i++)
  delete[] m[i];
 delete[] m;

} // main
```