Puntatori e riferimenti

Definendo una variabile se ne determinano tre attributi

1. Nome
2. Tipo
3. Indirizzo di memoria

```
   0x4fffd34
n
   int
```

- Ad una variabile si accede tramite il suo nome

Esempio

```
cout << n

cout << &n
```


riferimenti

- un *riferimento* ad una variabile è un ulteriore nome per essa, un "alias";
- si dichiara utilizzando l'operatore **&** al tipo di dato riferito;
  - `int&` è il tipo "riferimento al tipo `int`"

```
int n;
int& r = n;
```

- una variabile di tipo "riferimento" deve essere inizializzata contestualmente alla definizione, cioè deve essere seguita dall'operatore di assegnamento e dal nome di una variabile già definita dello stesso tipo

```
void main()
{
   int n = 75;
   int& r = n;       // r è un riferimento per n
   cout << "n = " << n << ", r = " << r << endl;
   cout << "&n = " << &n << ", &r =" << &r << endl;
}
```

esecuzione:

```
n = 75, r = 75
&n = 0x4fffd34, &r = 0x4fffd34
```


Ricapitolando

Il carattere **&** ha 3 utilizzi in C++

1. Come **prefisso al nome di una variabile** ne restituisce l'indirizzo
2. Come **suffisso a un tipo nella def di un riferimento** rende quest'ultimo un alias della variabile in questione
3. Come **suffisso a un tipo nella dichiarazione dei parametri di una funzione**, rende questi parametri dei riferimenti allecorrispondenti variabili passate alla funzione.

puntatori

- Quando si definisce una variabile il compilatore riserva un'area di memoria per tale variabile
- Lo spazio è preso in maniera contigua partire da una cella, il cui indirizzo è preso come indirizzo di memoria della variabile
- Un puntatore è una variabile che contiene l'indirizzo di memoria di un'altra variabile.
- Il tipo di dato **puntatore al tipo** si dichiara usando * come suffisso al tipo di dato puntato
  - Non devono per forze essere inizializzate all'atto della definizione.
  - Può essere inizializzata a NULL (o 0)

```
int n;
int* p;
p=&n ; // p puntatore a int è inizializzato
       // all'indirizzo di n
```

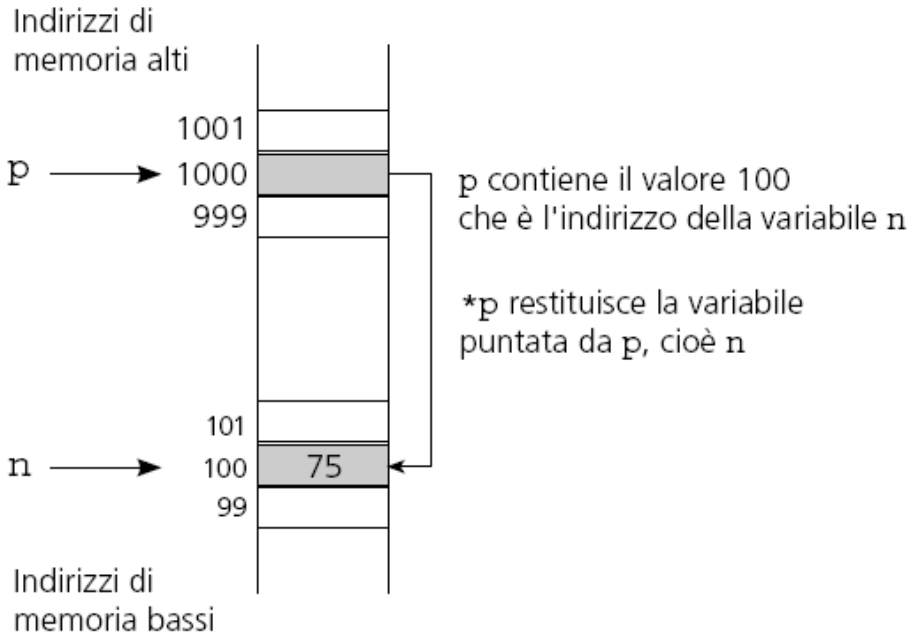

puntatori - II

una variabile di tipo *puntatore al tipo x* contiene l'indirizzo di memoria di una variabile di *tipo x*;

```
int n;
int* p; // p è variabile di tipo puntatore ad int
p = &n; // p contiene il valore dell'indirizzo di n
```

dereferenziare il puntatore significa andare alla variabile puntata partendo da un suo puntatore; l'operazione si fa con l'operatore di indirezione `*` prefisso alla variabile puntatore:

```
*p = 75;
// scrive 75 nella
// variabile n
```

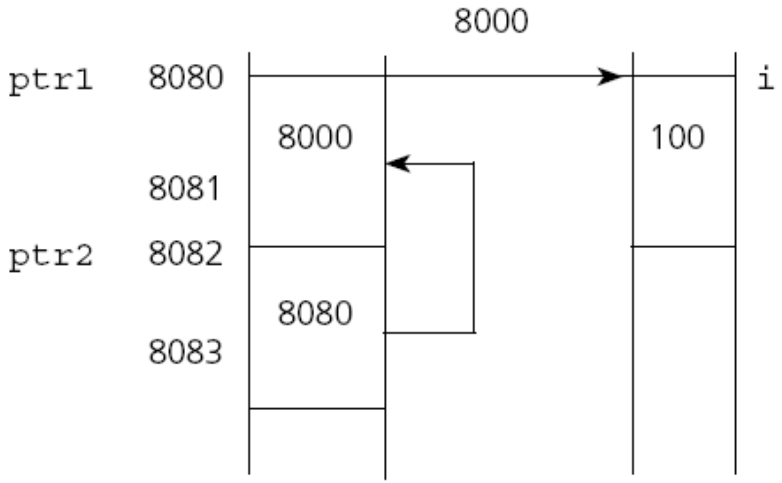


puntatori a puntatori

- un tipo puntatore può puntare qualunque tipo, anche un altro puntatore

```
int i = 100;
int* ptr1 = &i;
int** ptr2 = &ptr1;
```

`ptr1` è puntatore ad interi e punta la variabile `i` di tipo `int`;
`ptr2` è puntatore a puntatori di interi e punta la variabile `ptr1`



puntatori e array

- gli arrays sono implementati mediante puntatori; il nome di un vettore è un puntatore al primo elemento dell'array; per esempio:

```
int gradi[5] = {10, 20, 30, 40, 50};
gradi[0] == *gradi;          // 10
gradi[1] == *(gradi + 1); // 20
gradi[2] == *(gradi + 2); // 30
gradi[3] == *(gradi + 3); // 40
gradi[4] == *(gradi + 4); // 50
```

- il nome di un array è però una *costante* puntatore, non una variabile puntatore; non si può cambiarne il valore

aritmetica dei puntatori

- se si incrementa un puntatore a un tipo T di `n`, il suo valore aumenta in realtà di `n` moltiplicato per la dimensione del tipo T

```
int gradi[5] = {10, 20, 30, 40, 50};
p = gradi;
```

`p` punta al primo intero `10` in `gradi`; ma dopo l'istruzione

```
p++;
```

`p` punterà al secondo intero `20`; poiché ogni elemento di `gradi` occupa 4 bytes, il valore di `p` è stato incrementato di 4 (e non di 1)

- non si possono sommare due puntatori.
- non si possono moltiplicare due puntatori.
- non si possono dividere due puntatori.

puntatori costanti e puntatori a costanti

Puntatore costante (il nome di un array è un puntatore costante)
- il suo valore non può cambiare (possono cambiare i valori delle variabili puntate)

<tipo_dato> **const** *<nome_puntatore>* **=** *<indirizzo_variabile>*;*

```
int x, e;
int* const p = &x;
*p = e; // corretto
p = &e // scorretto; p non puo' cambiar valore
```

Puntatore a costante: il suo valore può cambiare, ma non cambia il valore a cui punta.

const *<tipo_dato>* ***** *<nome_puntatore>* **=** *<indirizzo_const>*;*

```
const int x = 25;
const int e = 50;
const int* p = &x;
*p = 15; // scorretto; ciò che p punta non puo' cambiar valore
p = &e // corretto
```


puntatori a funzioni

- non solo i dati ma anche le istruzioni stanno in memoria a certi indirizzi;
- è possibile creare puntatori che puntino a funzioni, cioè al nome della funzione, che altro non è che un puntatore alla prima istruzione della funzione:

tipo_restituito **(***_PuntatoreFunzione_**)** **(***_argomenti_**);**

```
int f(int) { ... }; // definisce la funzione f
int (*pf)(int);     // definisce il puntatore pf alla funzione
pf = &f;            // assegna l'indirizzo di f a pf
```