Flavia Cocca
giovedì 3 giugno 2021 14:30

1-Puntatori

I **puntatori** possono essere utilizzati per riferire oggetti di ogni tipo :

- Allo stesso modo degli array, si possono allocare e deallocare oggetti dinamici di ogni tipo mediante gli operatori **new** e **delete**

`int *p = new int` -> **p** è un punatore ad un intero

Esempio:

```
int *p ;        -> puntatore ad un oggetto di tipo int
p = new int ;   -> allocazione di un oggetto dinamico di tipo int (NON è un array!)
delete p ;      -> deallocazione di un oggetto puntato da p
```

Esempio:

```
struct s {int a, b ;} ;
s *p2 ;         -> puntatore ad un oggetto di tipo s
p2 = new s ;    -> allocazione di un oggetto  dinamico di tipo s
delete p2 ;     -> deallocazione oggetto puntato da p2
```

Esempio:

```
const int *p    -> puntatore ad oggetto di tipo int, non modificabile tramite p
int * const p   -> puntatore costante ad oggetto di tipo int
int * p[10]     -> array di 10 puntatori ad int
int (*p)[10]    -> puntatore ad array di 10 interi
```

Operatore di indirizzo &

- Restituisce l'indirizzo dell'oggetto a cui viene applicato, è:
  - unario;
  - prefisso;

`&x` -> indirizzo di **x**

Esempio:

```
int v[10];
int(*p)[10];
p = &v;      -> equivalente a:
             int(*p)[10]=&v; -> un puntatore ad un array di 10 int(che corrispondono all'area di memoria di v)

int *p1;
p1 = v;      -> equivalente a:
             int *p = v; -> perché *p=p1=v
```

Esempio:

```
main() {

int i, j,k;             -> p = indirizzo di i
int *p = &i;            -> p2 costante:            p = indirizzo di j
int * const p2 = &j;    -> cambio il valore di p:  p = indirizzo di j
p = p2;                 -> cambio il valore di p2: p2 = indirizzo di k
p2 = &k;                -> genera un errore a tempo di compilazione -> p2 costante

}
```

Operatore di deferenziazione *

- Ritorna un riferimento all'oggetto puntato, è:
  - unario;
  - prefisso;

  `*p` -> l'oggetto puntato da **p**

Esempio:

```
main(){

int i,j;
int *p = &i;          p -> i[ ]
int * const p2 = &j;  p2 -> j[ ]
*p = 3;               -> va nell'area di memoria puntata da p : i[3]
*p2=4;                -> va nell'area di memoria puntata da j: j[4]
int x = *p;           cioè x = i
i = *p2;              cioè i = J

}
```

Puntatori a puntatori

- Un puntatore può puntare a (contenere l'indirizzo di) un altro puntatore.

Esempio:

```
main() {

int i, *p;
int **q;    -> puntatore a puntatore a int
q = &p;     -> q = indirizzo di p
p = &i;     -> p = indirizzo di i
**q = 3;    -> equivale a i = 3

}
```

Selettori di campo

`struct s {int a, b ;};` -> Oggetto di tipo struttura indirizzato da un puntatore **p**
`s *p;`

Due modi per riferire ad un campo della struttura indirizzata da p:
- `(*p).a`
- `p -> a`

Esempio:

```
main() {

struct s {int a, b;} s1;
s *p2;          -> puntatore ad un oggetto di tipo s
p2 = &s1;
(*p2).a = 3;    -> equivalente a s1.a = 3
```

```
p2->a = 3;      -> equivalente a s1.a = 3

}
```

Riferimenti

I riferimenti sono dichiarati usando l'operatore **&** :

- **A livello di utilizzo**, un riferimento ad una variabile è un ulteriore nome per essa, in pratica un alias.
- **A livello di implementazione**, un riferimento contiene l'indirizzo di un oggetto puntato, come un puntatore.

- **Passaggio per valore:** impedisce i cambiamenti e spreca memoria per le copie
- **Passaggio per riferimento:** consente la modifica del parametro attuale attraverso la modifica al corrispondente parametro formale

Esempio:

```
int main() {

int a = 20, b=15, c=12, d=8;     a=[20]  b=[15]  c=[12]  d=[8]
                                            ↑o              ↑o
                                          punt            prp

int *punt;
punt = &b;
int *prp;
prp = &d;
f(a, punt, c, prp);
cout << a << " " << *punt << " " << c << " " << *prp << " " << endl;

}
                          c=[10]
                            ↑o
      i=[10]    p=q=10    ri=[10]    rp=prp
void f(int i, int *p, int &ri, int *&rp) {

int *q = new int;
i = 10;
p = q;
*p = 10;
ri = 10;
rp = q;        rp=10 -> prp=10 -> d=10
*rp = 30;      rp=30 -> prp=30 -> d=30

}

SOLUZIONE: 20 15 10 30
```

Puntatori ed array

- Il nome di un array corrisponde ad un puntatore al primo elemento dell'array stesso

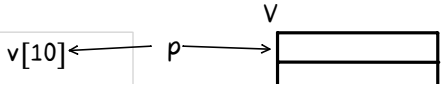
  SE:
  - x[N] è un array di N elementi di tipo T

  ALLORA:
  - x equivale a &x[0] (riferimento)

- Se dichiaro un puntatore **p** di tipo **T** allora posso assegnare a **p** l'indirizzo del primo elemento di **x** nel seguente modo:
  - T *p = x;

Esempio:

```
main() {
```

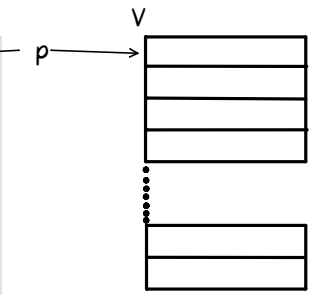


Esempio:

```
main() {

const int N = 10;
int v[N];
int *p = v;        -> assegna a p l'indirizzo (del primo elemento) di v
*(p + 2) = 7;      -> equivale a v[2]=7

}
```



Esempio:

```
main() {

const int N = 10;
int v[N], z[N];
*v = *z;           -> equivale a v[0] = z[0]

}
```