Corso di Programmazione I

Puntatori

Riferimenti

- Da C++ a UML
  - Capitolo 8; §1,§2 (Richiamo)
  - Capitolo 8; §8.3.1, §8.3.5, §8.4
- Da Fondamenti d'Inf. II
  - Parte II, Capitolo VIII; §2 (Richiamo)

I puntatori

- Il C++ prevede puntatori a funzione e puntatori a dati di qualsiasi natura, semplici o strutturati.

In particolare il puntatore viene utilizzato :

- per il riferimento a liste di variabili dinamiche;
- per il riferimento a funzioni;
- per gli array, in particolare per l'elaborazione di stringhe;
- per i parametri formali di funzione, in alternativa allo scambio per riferimento;
- per il riferimento a locazioni specifiche della memoria, associate a dispositivi hardware (ad esempio per l'ingresso-uscita)

Tipo Puntatore

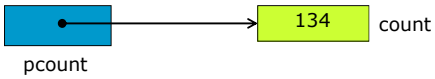
- Un tipo che può assumere come insieme dei valori tutti gli indirizzi di memoria
- Un puntatore ad un variabile di tipo T identifica l'indirizzo di memoria della variabile.

Quindi…

*una variabile di tipo puntatore è una variabile che contiene l'indirizzo di memoria di un'altra variabile (si chiama così appunto perché **punta** ad un'altra variabile)*

Tipo Puntatore

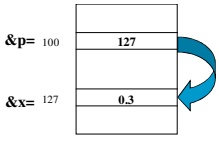
- typedef int *pint;
- pint pcount;
- int count=134;
- pcount=&count;



I puntatori: notazioni in C++

- **p** è un variabile di tipo puntatore
- **p=&x** al puntatore p viene assegnato l'indirizzo di x
- ***p** denota la variabile puntata da p (**deferenziazione**)

Esempio:
Sia x una variabile di indirizzo 127 e valore 0,3;
p è un puntatore ad x;



- char* pc; //pc è un puntatore a variabile di tipo carattere
- int* pi; //pi è un puntatore a variabile di tipo intero
- float* pf; //pf è un puntatore a variabile di tipo float
- double* pd; //pd è un puntatore a variabile di tipo doppia precisione

Esempio

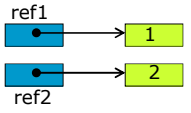
```
int main()
{
    int i=10;
    int* p=&i; //P contiene l'indirizzo di n
    *p=20;


    return 0;
}
```

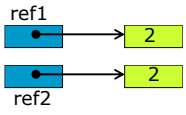
NB:La dimensione del puntatore (sizeof()) è costante ed è indipendente dal tipo puntato (tipicamente è di 4 byte su architetture a 32 bit)

Esempi di assegnazione

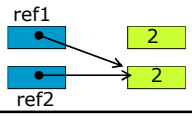
```
int i=1, j=2;
pint ref1=&i,ref2=&j;
```




```
*ref1=*ref2;
```




```
ref1=ref2;
```



Inizializzazione dei puntatori

- Un puntatore non inizializzato, come qualsiasi variabile, ha un valore iniziale aleatorio (valore dei registri di memoria all'atto della sua definizione)
- E' importante dunque assegnare un valore iniziale.
- Un puntatore a cui viene assegnato il valore 0 o NULL non punta ad alcun oggetto, cioè non indirizza alcun dato valido in memoria (è detto **puntatore nullo**)

```
char *p=0; //oppure
char *p=NULL;
//si può testare il valore del puntatore
if (p==0) …
//oppure
if (p==NULL)
```

Puntatore a puntatore

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

- ptr1 e ptr2 sono due puntatori diversi. Il primo è un puntatore ad un intero. Il secondo, invece, è un puntatore a puntatore.

```
i= 95;
*ptr1=95;
**ptr2=95;
```

Puntatori costanti

- Un puntatore costante ha un valore che non può cambiare

T * const p=<indirizzo var>;

Es:

int * const p1= &x;

p1 è un puntatore costante che punta ad x.

Attenzione:

p1 è costante, ma *p1(cioè x) è una variabile a cui può essere cambiato il valore.

`*p1= 10; //OK`

`p1=&y; //ERRORE !!!!!`

Puntatori a costanti

const T * p=<indirizzo const_o_stringa>

È un puntatore ad una variabile const. Il valore del puntatore può essere modificato ma non il valore puntato.

```
const int x=25;
cont int *p1=&x;
```

`*p1= &e; //OK, qualunque tentativo di modificare il valore di e comporterà un errore di compilazione`

`*p1=15; //ERRORE !!!!!`

Puntatori costanti a costanti

- const T * const p=<indirizzo const_o_stringa>

const int x=25;

const int * const p1=&x;

Qualsiasi tentativo di modificare p1 o *p1 produrrà un errore di compilazione

I puntatori: operazioni

•Il C++ tratta esplicitamente le espressioni di tipo puntatore;

•Sono previsti i seguenti operatori:

- **•Assegnazione** tra puntatori che puntano allo stesso tipo T*;
- •Operatori unari unitari (forma prefissa e postfissa)
  - **•incremento** "++"
  - **•decremento** "- - "
  - **•somma e differenza** tra un puntatore ed un intero
    - •"p+i" punta all'elemento che segue di i posizioni quello puntato da p;

I puntatori: operazioni

T * p;

p=p+1; → **P=p+sizeof(T);**

- l'incremento a p è tale che p+1 punta all'area di memoria immediatamente successiva a quella impegnata dall'elemento puntato da p.
- Questa tecnica è particolarmente utile per i puntatori ad array

I puntatori a dati strutturati

Un puntatore può puntare a variabili strutturate (di tipo array o record). Esamineremo i seguenti casi:

- puntatore ad array;
- puntatore a record;

Tra i casi elencati, il puntatore ad array può utilmente essere sostituito dalla notazione a[i] tipica degli array

I puntatori ad array

Un puntatore ad un array a è una variabile che punta alla prima locazione dell'array (a[0] in C++);

considerando che i successivi elementi dell'array sono allocati in posizioni contigue, esso consente di puntare anche a tutti gli altri elementi dell'array.

```
T  a [dim]; // a è un array di dim  elementi di tipo T
T* p  ;     // p è un puntatore a T
```

```
p=a;
```

```
p=&a[0];
```

Un puntatore ad array è dunque di per sé un puntatore (costante) al tipo T degli elementi dell'array

I puntatori ad array: esempio

```
const int dim=100;
float  a [dim]; // a è un array di dim  elementi di tipo float
float* p  ;     // p è un puntatore a float

//azzeramento di tutti gli elementi di un array di 100 elementi
for (p = a, int i=0; i<dim; i++, p++)
        *p= 0;
```

Siano p e q sono due puntatori allo stesso array A
-p punta all'elemento A[i]
-q all'elemento A[j],
la differenza p-q fornisce la "distanza" tra i due elementi

Puntatori a Stringhe

Puntatori a record

Un puntatore ad un record è una variabile che punta all'indirizzo di memoria ove il record è allocato;
esso è molto utile nella realizzazione dei tipi dinamici

```
// Dichiarazioni di un tipo strutturato Ts
struct  Ts { // Ts è un tipo strutturato
                    D1;
                    .....;
                    Dn ;  } ;

Ts r ;  // r è variabile di tipo Ts

// Dichiarazione di un tipo puntatore a Ts
typedef  Ts* Punt;
// Dichiarazione  di variabile di tipo puntatore a Ts
Punt p;
```

Puntatori a record

```
Punt p = &r; // p è una  variabile puntatore inizializzata ad r
```

p può essere ridefinito nel corso del programma

```
p= &r1; //ora p punta alla variabile r1 di tipo Ts
```

Accesso alla singola componente:

`(*p).Di;` ⟷ `p->Di;`

Puntatori a record: un esempio

```
// Definizione di tipo struttura:
struct Abbonati {  // definisce struttura di nome Abbonati
                   Stringa nominativo;
                   Stringa indirizzo;
                   int numTel ; };

typedef Abbonati* Pabb;

// Definizione di variabili
Abbonati a;        // a una variabile di tipo Abbonati
Pabb p= &a;        // p, di tipo Pabb, è inizializzato ad a

// Accesso alla componente nominativo mediante puntatore
p->nominativo;
```