

Corso “Programmazione 1”

Capitolo 04: Riferimenti e Puntatori

Docente: **Roberto Sebastiani** - roberto.sebastiani@unitn.it
Esercitori: **Mario Passamani** - mario.passamani@unitn.it
Alessandro Tomasi - alessandro.tomasi@unitn.it
C.D.L.: Informatica (INF)
Ing. Informatica, delle Comunicazioni ed Elettronica (ICE)
Studenti con numero di matricola pari
A.A.: 2019-2020
Luogo: DISI, Università di Trento
URL: disi.unitn.it/rseba/DIDATTICA/progl_2020/

I Tipi Derivati

- Dai tipi fondamentali, attraverso vari meccanismi, si possono derivare tipi più complessi
- I principali costrutti per costruire tipi derivati sono:
 - i riferimenti
 - i puntatori
 - gli array
 - le strutture
 - le unioni
 - le classi

Il Tipo “Riferimento a”

- Il meccanismo dei riferimenti (reference) consente di **dare nomi multipli** a una variabile (o a un'espressione dotata di indirizzo)
 - Un riferimento è un **sinonimo** dell'espressione a cui fa riferimento
⇒ modificando l'una, si modifica anche l'altra (“aliasing”)
 - Un riferimento è un **espressione dotata di indirizzo**
- Sintassi: `tipo & id = exp;`
dove `exp` è un'espressione dotata di indirizzo

Esempio:

```
int x=1;
int &y=x; // y e` di tipo reference, e` sinonimo di x
y = 6;   // viene modificato anche x!
```

Vincoli sull'uso dei Riferimenti

- Nelle dichiarazioni di reference, l'inizializzazione è obbligatoria!

```
int &y; // errore!
```

- Non è possibile ridefinire una variabile di tipo riferimento precedentemente definita:

```
double x1, x2;
```

```
double &y=x1; // ok
```

```
double &y=x2; // errore! già' definita!
```

- Non è (più) possibile definire un riferimento a
 - un'espressione non dotata di indirizzo,
 - o a un'espressione dotata di indirizzo ma di tipo diverso.

```
float &y=10.2; // Errore
```

```
double d=3.1; //
```

```
int &z=d; // Errore
```

- Esempio di uso di references:

```
{ RIF_PUNT/reference.cc }
```

L'Operatore address-of "&"

- L'operatore & ("address-of") **ritorna l'indirizzo (l-value) dell'espressione a cui è applicato**
- Può essere applicato solo a espressioni dotate di indirizzo!
- Differente dall'uso di "&" nella definizione di riferimenti!

Esempio

```
cout << &l << endl; // stampa l'indirizzo di l
cout << &(l*5) << endl; // errore!
```

```
int n = 10;
int& r = n;
cout << "&n = " << &n << ", &r = " << &r << endl;
```

- **Esempio di uso di address-of:**
{ RIF_PUNT/address_1.cc }
- **Riferimenti e address-of:**
{ RIF_PUNT/rifVsAddressof.cc }

Il Tipo “Puntatore a...”

- Un puntatore contiene l'indirizzo di un altro oggetto
 - l'r-value di un puntatore è un indirizzo
- Definizione di un puntatore:
 - Sintassi: `tipo *id_or_init`
 - Esempio: `int *px; // px puntatore a un intero`
 - è sempre necessario indicare il tipo di oggetto a cui punta
- Un puntatore a tipo t può contenere solo indirizzi di oggetti di tipo t
- Ad una variabile puntatore viene associata una spazio di memoria atta a contenere un indirizzo di memoria,
- ...ma non viene riservato spazio di memoria per l'oggetto puntato!
- Lo spazio allocato a una variabile di tipo puntatore è sempre uguale, indipendentemente dal tipo dell'oggetto puntato

L'Operatore di Dereference “*”

- Per accedere all'oggetto puntato da una variabile puntatore occorre applicare l'**operatore di dereference** *
- Se `px` è punta a `x`, `*px` è un **sinonimo temporaneo** di `x`
⇒ modificando `*px` modifico `x`, e vice versa
- `*px` è un'**espressione dotata di indirizzo**

Esempio

```
int x=1; // x variabile tipo int
int *px; // px variabile puntatore
px=&x;   // accede alla variabile
        // puntatore
*px=x+1; // accede alla cella di memoria puntata
        // dalla variabile puntatore
```

L'esempio di cui sopra:

```
{ RIF_PUNT/pointer.cc }
```

Assegnazioni tra Puntatori

- Assegnando a un puntatore q il valore di un altro puntatore p , q punterà allo stesso oggetto puntato da p
 - $*p$, $*q$ e l'oggetto puntato da loro sono temporaneamente sinonimi

```
int i, j;  
int *p, *q;  
p = &i; // p=indirizzo di i, *p sinonimo di i  
q = &j; // q=indirizzo di j, *q sinonimo di j  
*q = *p; // equivale a j=i  
q = p; // equivale a q=indirizzo di i
```

- L'esempio di cui sopra espanso:
{ RIF_PUNT/pointer1.cc }
- L'esempio di cui sopra espanso (2):
{ RIF_PUNT/pointer2.cc }

Esempi su puntatori e riferimenti

- Esempio: riferimento ad un oggetto puntato da un puntatore:
il riferimento “segue” il puntatore?:
{ RIF_PUNT/rif_deref.cc }

Puntatori a void (cenni)

- In alcuni casi è utile avere una variabile puntatore che possa puntare ad entità di tipo diverso;
- tale variabile viene dichiarata di tipo “puntatore a void” cioè a tipo non specificato

```
int i; int *pi=&i;
char c; char *pc=&c;
void *tp;
tp = pi;          // punta a int
*(int*)tp=3;
tp = pc;          // punta a char
*(char*)tp='C';
```

Esempio di cui sopra:

```
{ RIF_PUNT/punt_a_void.cc }
```

Puntatori a costante (cenni)

- Definizione
 - Sintassi: `const tipo *id_or_init;`
 - Esempio: `const int *pc1 = &c1;`
- Idea: **non permettono di modificare l'oggetto puntato tramite dereference del puntatore stesso**
- Nota: non rendono l'oggetto puntato una costante

```
const int c1 = 3; int c2 = 5;
const int *pc1 = &c1; // ok
const int *pc2 = &c2; // ok
pc2 = pc1; // ok
pc1 = &c2; // ok
*pc1 = 2; // errore
c2 = 2; // ok
```

Esempio di cui sopra:

```
{ RIF_PUNT/punt_a_cost.cc }
```

Costanti puntatore (cenni)

- Definizione
 - Sintassi: `tipo *const id=exp;`
 - Esempio: `int *const pa = &a;`
- Idea: **non permettono di puntare ad un altro oggetto**
- L'oggetto puntato può essere modificato tramite dereference del puntatore stesso

```
int a, b;  
int *const pa = &a;  
*pa = 3; // ok  
pa = &b // errore: pa e' costante
```

Esempio di cui sopra:

```
{ RIF_PUNT/const_punt.cc }
```

Costanti puntatore a costante (cenni)

- Definizione
 - Sintassi: `const tipo *const id=exp;`
 - Esempio: `const int *const a = &c;`
- Idea: **non permettono di puntare ad un altro oggetto**
- L'oggetto puntato **non** può essere modificato tramite dereference del puntatore stesso

```
const int b = 2;
const int c = 3;
const int *const a = &c;
a = &b; // errore
*a= 2; // errore
c = 5; // errore
```

Esempio di cui sopra:

```
{ RIF_PUNT/const_punt_const.cc }
```

Utilizzo pratico di puntatori

Dear Santa,
How are you? I'm good.
Here is what I want for
Christmas.

A http://www.amazon.com/gp/product/B0032HF60M/ref=59_hps_bw_g21_1r03?pf_rd_m=ATVPDKIKXODER&pf_rd_s=center-3&pf_rd_f=1XW442FH2K03Y7BMWQNM&pf_rd_t=101&pf_rd_p=1328901542&pf_rd_i=16579

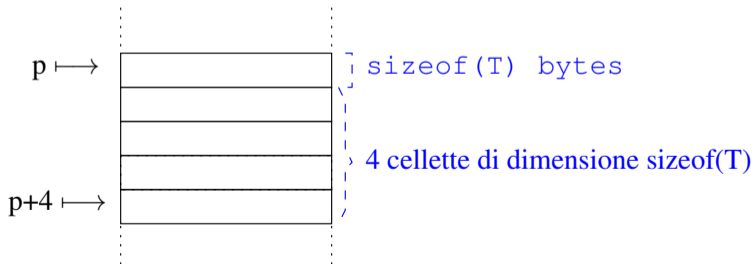
Aritmetica di Puntatori ed Indirizzi

Gli indirizzi e i puntatori hanno un'aritmetica:

se p è di tipo T^* e i è un intero, allora:

- $p+i$ è di tipo T^* ed è l'indirizzo di un oggetto di tipo T che si trova in memoria dopo i posizioni di dimensione `sizeof(T)`
- analogo discorso vale per $p++$, $++p$, $p--$, $--p$, $p+=i$, ecc.

⇒ i viene implicitamente moltiplicato per `sizeof(T)`

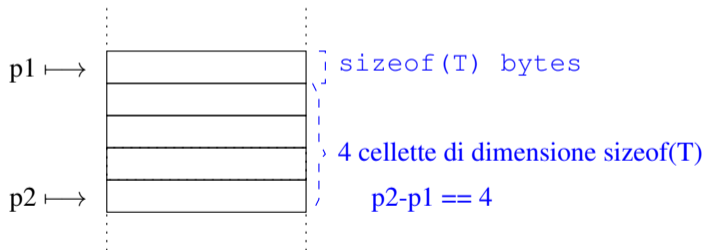


Aritmetica di Puntatori ed Indirizzi II

se p_1 , p_2 sono di tipo T^* , allora:

- $p_2 - p_1$ è un intero ed è il numero di posizioni di dimensione `sizeof(T)` per cui p_1 precede p_2 (negativo se p_2 precede p_1)
- si possono applicare operatori di confronto $p_1 < p_2$, $p_1 >= p_2$, ecc.

$\implies p_2 - p_1$ viene implicitamente diviso per `sizeof(T)`



Esempio di operazioni aritmetiche su puntatori:

```
{ RIF_PUNT/aritmetica_punt.cc }
```


Priorità tra dereference e operatori aritmetici

Nota

Attenzione alle priorità tra l'operatore dereference “*” e gli operatori aritmetici:

- $*pv+1$ è equivalente a $(*pv)+1$, non a $*(pv+1)$
- $*pv++$ è equivalente a $*(pv++)$, non a $(*pv)++$

⇒ è consigliabile usare le parentesi per non confondersi.

- Esempio di cui sopra:
{ `RIF_PUNT/priorita.cc` }