



UNIVERSITÀ  
DEGLI STUDI  
DI PADOVA

Puntatori, reference, cast

Stefano Ghidoni



DIPARTIMENTO  
DI INGEGNERIA  
DELL'INFORMAZIONE



# Agenda

- void\* e cast
- Puntatori vs reference

void\* e cast  
(messyng with types)





- void\* permette di saltare qualsiasi controllo
  - È un puntatore a memoria raw
  - Devo fornire indicazioni su come deve essere usata la memoria puntata
- Attenzione: void vs void\* *Una è una parola chiave, l'altro è un puntatore*
- È possibile assegnare qualsiasi puntatore a un void\*
  - void\* rappresenta il concetto puro di "indirizzo di memoria" senza indicazioni su come usarla





# Cast

- Conversione esplicita tra i tipi (inclusi i puntatori): **static\_cast**
  - "A deliberately ugly name for an ugly and dangerous operation" (BS)
- Due conversioni "potentially even nastier":

Nome cast	Effetto
reinterpret_cast	Tipi totalmente indipendenti, es.: int e double*
const_cast	Elimina const



# Cast

- I cast servono principalmente a interfacciarsi:
  - con HW
  - con altro codice non modificabile

```
Register* in = reinterpret_cast<Register*>(0xff);  
  
void f(const Buffer* p){  
    Buffer* b = const_cast<Buffer*>(p);  
    // ...  
}
```

# Puntatori e reference



# Puntatori e reference

- Una reference è come un puntatore
  - immutabile
  - dereferenziato automaticamente
- ... oppure come un nome alternativo per un oggetto



# Puntatori vs reference

- Puntatore
  - Assegnamento: cambia il valore del puntatore, non dell'oggetto puntato
  - Per cambiare il valore dell'oggetto puntato: dereference
- Reference
  - Assegnamento: cambio valore dell'oggetto

```
int x1 = 10;  
  
int* p1 = &x1;  
  
p1 = 7;           // pericoloso!!  
  
*p1 = 7;         // corretto
```

```
int y1 = 10;  
  
int& r1 = y1;  
  
r1 = 7;
```



# Puntatori vs reference

- Puntatore
  - È acquisito con &
  - Posso spostare il puntatore assegnando un nuovo indirizzo
- Reference
  - Non posso spostarla dopo l'inizializzazione

```
int x1 = 10;
int x2 = 20;
int* p1 = &x1;
p1 = &x2;           // puntatore
                  // spostato
```

```
int y1 = 10;
int y2 = 20;
int& r1 = y1;
                  // non ho modo di spostare la
                  // reference su y2
```



# Puntatori vs reference

- Puntatore
  - Per accedere all'oggetto puntato: \* oppure []
- Reference
  - Nessun operatore per accedere al dato puntato

```
*p1 = 4;
```

```
r1 = 4;
```



# Puntatori vs reference

- Puntatore
  - Assegnamento del puntatore: due riferimenti allo stesso oggetto (**shallow copy**)
- Reference
  - Assegnamento della reference: copia dell'oggetto a cui si riferisce (**deep copy**)

```
int i, j;  
int *p1 = &i;  
int *p2 = &j;  
p2 = p1;          // copia del  
                  // puntatore
```

```
int i, j;  
int &r1 = i;  
int &r2 = j;  
r2 = r1;          // copia del  
                  // contenuto
```

- Deep copy vs shallow copy: implicazioni nell'uso della memoria con gli UDT



# Puntatori vs reference

- Puntatore
  - Esiste il null pointer
- Reference
  - Non esiste una reference non valida

```
int *p1 = nullptr;
```



# Parametri puntatori e reference

- Chiamata a funzione: abbiamo visto gli effetti di una modifica di parametri reference
- Con i puntatori si può ottenere lo stesso effetto
- Volendo cambiare il valore di una variabile ho tre opzioni:

```
int incr_v(int x) { return x + 1; }

void incr_p(int* p) { ++*p; }

void incr_r(int& r) { ++r; }
```

- Come scegliere?



# Parametri puntatori e reference

- Ritornare il valore:
  - È più chiaro e meno soggetto a errori
  - Ok per oggetti piccoli
  - Ok per oggetti grandi se hanno il **move constructor** (lo vedremo più avanti)



# Parametri puntatori e reference

- Reference vs puntatore:
  - I puntatori sono esplicativi

```
int x = 7;
```

```
incr_p(&x);
```

Esplicito

```
incr_r(x);
```

"Looks innocent"

Ricordo che in questo caso non si può sapere dalla chiamata se verrà modificato l'oggetto oppure no



# Parametri puntatori e reference

- Reference vs puntatore:
  - I puntatori possono esprimere "input non valido" con nullptr, le ref no

```
int* p = nullptr;  
  
incr_p(p);           // incr_p deve gestire questo caso  
  
void incr_p(int* p) {  
    if (p == nullptr) error("null pointer argument");  
    ++*p;  
}
```



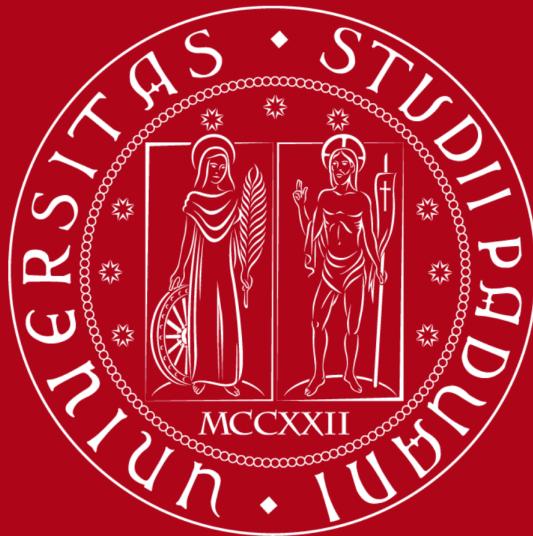
# Parametri puntatori e reference

- Reference vs puntatore – un criterio:
  - Se no-object è un valore plausibile: puntatore
  - Altrimenti: reference / const reference



## Recap

- Puntatori a void: gestire la memoria grezza
- Cast
- Puntatori vs reference
  - Differenze sintattiche
  - Differenze espressive
- Idea: deep copy vs shallow copy (sarà sviluppato più avanti)



UNIVERSITÀ  
DEGLI STUDI  
DI PADOVA

Puntatori, reference, cast

Stefano Ghidoni