



UNIVERSITÀ
degli STUDI
di CATANIA

Riferimenti (reference) nel C++

Corso di programmazione I AA 2019/20

Corso di Laurea Triennale in Informatica

Prof. Giovanni Maria Farinella

Web: <http://www.dmi.unict.it/farinella>

Email: gfarinella@dm.unict.it

Dipartimento di Matematica e Informatica

1. Introduzione ai riferimenti
2. Sintassi e uso dei riferimenti C++
3. Tipi di riferimenti
4. Riepilogo: puntatori vs reference

Introduzione ai riferimenti

Uso dei puntatori (C/C++): pro e contro

È noto che i puntatori consentono il passaggio di dati di grandi dimensioni mediante messaggi (invocazioni di metodi) con la massima **efficienza**.

```
1  Matrice3D *sum(Matrice3D *m1, Matrice3D *m2){  
2      //.. somma le matrici e restituisce il  
3      //puntatore alla matrice somma..  
4  }
```

Nella invocazione del metodo `sum` avverrà una **copia** (passaggio per valore) solo dell'indirizzo in memoria delle matrici di input.


Puntatore ad oggetto vs nome oggetto:

- (-) sintassi puntatori può risultare **tediosa**:

```
1      ClasseX  obj ;  
2      ClasseX  *p = &obj ;  
3      p->metodo ( ) ;  
4      //oppure ..  
5      (*p) . metodo ( ) ;
```

Uso dei puntatori (C/C++): pro e contro

- (-) Il puntatore all'oggetto potrebbe essere nullptr..

```
1      ClasseX obj1 , obj2 ;  
2      ClasseX *p = &obj1 ;  
3      p->metodo ( ) ;  
4      p = nullptr ;  
5      // ...  
6       p->metodo ( ) ; // !!! ???
```

Uso dei puntatori (C/C++): pro e contro

- (+) D'altro canto è sempre possibile **riassegnare valore a ptr** affinché punti a differenti oggetti in differenti istanti di tempo:

```
1      ClasseX obj1, obj2 ;  
2      ClasseX *p = &obj1 ;  
3      p->metodo ( ) ;  
4      p = &obj2 ;  
5      p->metodo ( ) ;
```


Dato che esiste valore `nullptr` il programmatore è **obbligato a controllarne il valore**

```
Matrice3D *sum(Matrice3D *m1, Matrice3D *m2){  
    if(m1!=nullptr || m2!=nullptr){  
        // ...  
    }
```


Uso dei puntatori (C/C++): pro e contro

Anche il **passaggio dei parametri attuali** può risultare tedioso:

```
Matrice3D m1, m2;  
// ...  
Matrice3D *result = sum(&m1, &m2);
```



Le reference furono introdotte per

- **mantenere i vantaggi derivanti dall'uso dei puntatori** (overhead pressochè nullo nel passaggio di parametri)
- **eliminare le complicazioni** derivanti dal loro uso (rischio nullptr e sintassi tediosa).

Sintassi e uso dei riferimenti C++


La notazione <type> & indica **una reference ad un oggetto** di tipo type.

```
1   ClasseX obj;  
2   ClasseX &objR = obj; //reference a obj  
3   //...
```

Dalla **linea 2** in poi, la reference objR sarà alias di obj.

Uso dei riferimenti

```
1   ClasseX obj;  
2   ClasseX &objR = obj; //reference a obj  
3   → objR.metodo(); //OK  
4   ClasseX *objP = &objR; // OK
```

*ClasseX *objP = &obj;* 

Istruzioni alle linee 3 e 4 sono “lecite”:

- accesso alle funzionalità di obj mediante objR
avviene con sintassi identica a quella usata per
accesso mediante obj.

Uso dei riferimenti

La reference va inizializzata contestualmente alla sua dichiarazione con oggetto del tipo specificato.

- vs puntatori, che possono assumere valore nullptr oppure non essere inizializzati.

Quindi, per “costruzione”, reference **sempre “valide”**.

```
1   ClasseX obj;  
2   //..  
3   ClasseX &someObjR; //NO! Errore di comp!  
4   ClasseX &objR = obj; // OK  
5   //..  
6   ClasseX *ptr; // ptr non inizializzato..
```

Uso dei riferimenti

Inoltre, per costruzione, la **reference non può essere riassegnata ad altro oggetto**

- vs puntatori, che possono essere riassegnati in qualunque momento..

```
1  int anInteger = 10, aSecondInt = 20;
2  int &intR = anInteger;
3  intR = aSecondInt; // anInteger == 20;
4  int *ptr = &anInteger;
5  ptr = &aSecondInt;
```

Istruzione alla linea 3 è come: anInteger=aSecondInt;

Uso dei riferimenti

Nel caso dei puntatori, gli **operatori** come “++” e “--” **operano sui valori** della variabile puntatore, ovvero sugli indirizzi.


Nel caso delle **reference**, che sono alias di oggetti, operano sugli oggetti di cui sono alias.

```
1  int anInteger = 10;
2  //..
3  int &intR = anInteger;
4  intR++; //anInteger==11
```

Linea 6 equivalente a istruzione: `anInteger++`.

Tipi di riferimenti

1. Riferimenti di tipo lvalue modificabile o “non const”:
sono riferimenti ad oggetti che possono cambiare stato.

```
1      ClasseX obj;  
2   ClasseX &objR = obj; //reference a obj  
3      //metodo() potrebbe cambiare stato di obj..  
4      objR.metodo(); // OK
```

NB: Il nome **lvalue** indica un “oggetto” che risiede in memoria non temporaneamente o in altre parole **una espressione che si può usare nella parte sinistra di un assegnamento** (da qui il nome “lvalue”).

Altro esempio di riferimento *lvalue* modificabile (“non const”):

```
int anInteger = 10;  
//..  
int &intR = anInteger;
```

intR = 10 + anInteger;

Tipi di riferimenti

2. Riferimenti const (con lvalue)

```
int a = 10;  
const int &intR = a; //OK
```

```
ClasseX obj;  
const ClasseX &objRef = obj; //OK
```

Si può far uso di **lvalue** (e.g. `a` e `obj`) nella parte destra dello assegnamento, per creare reference di tipo `const`.

Il riferimento non potrà essere usato per modificare oggetto di cui è alias.

2. Riferimenti const (con rvalue)

```
const int &anIntR = {1}; //OK
```

Un **rvalue** (il letterale 1) viene usato per inizializzare il valore del riferimento. In questo caso:

- viene creato oggetto temporaneo con valore 1.
- l'oggetto temporaneo viene usato per **inizializzare la reference** anIntR.
- il ciclo di vita dell'oggetto temporaneo è identico a quello della reference anIntR, quindi sarà **distrutto insieme alla reference**.

Esempio svolto

A24_01.cpp

Riferimenti come parametri formali di funzioni

```
1      void sum(Matrice3D &m1, Matrice3D &m2, \
2          Matrice3D &result){
3          //somma le due matrici
4          //la reference result rappresenta il risultato
5      }
```

Vantaggi:

*(*m1) → metodo()*
m1.metodo()

- **sintassi semplificata** per operare sugli oggetti da modificare **all'interno del blocco** della funzione;
- passaggio di dati **efficiente**

Tipi di riferimenti

Funzioni che restituiscono reference

```
1  int& f(int v[], int i){  
2      return v[i];  
3  }  
4  // ...  
5  int arr[3] = {0};  
6  f(v, 2) = 10; // equiv. a: arr[2]=10;
```

NB: risultato invocazione della **f** **usato nella parte sinistra** di una espressione di **assegnamento**.

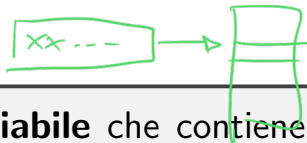
Può essere utile, ad esempio per overloading operatore “[]” (si vedrà in seguito..)

Esempio svolto

A24_02.cpp

Riepilogo: puntatori vs reference

Puntatori vs reference



Puntatore è variabile che contiene indirizzo di locazione di altra variabile.

VS

Reference non è puntatore, ma semplice alias.

Puntatori vs reference

Se oggetto del “riferimento” deve cambiare, allora **andrebbe usato un puntatore**, altrimenti una reference.

```
void fp(char *p){  
    while(*p)  
        cout << *p++;  
}
```



```
void fp(char &r){  
    while(r)  
        cout << r++; // NO!  
}
```

~~12~~

Puntatori vs reference

Per “collezioni” di oggetti (e.g. array) usare puntatori.

```
1      string x = " ciao" ;
2      string y = " pippo" ;
3      string& a1[] = {x,y}; // NO! Errore ..
4      string* a2[] = {&x,&y}; // OK!
```

“error: declaration of a1 as array of references”

Puntatori vs reference

Se si “necessita” della nozione di “non valore” o valore “null”, allora è bene usare i puntatori.

```
1      void fp(X* p){
2          if(p==nullptr){
3              //no value, ...
4          }
5          else {
6              // uso *p...
7          }
8      }
```

Puntatori vs reference

Se il “non valore” o “null” non è **contemplato**, allora usare i riferimenti.

```
1      void fr(X& r){  
2          //OK, r e' reference, quindi  
3          //sempre valida per costruzione  
4          //..codice che fa uso di r..  
5      }
```

Implementare una classe `Matrice3D`. In particolare:

- il costruttore deve permettere allo usercode di specificare le tre dimensioni della matrice, ed un ulteriore valore con cui inizializzare tutti gli elementi della matrice; specificare argomenti standard sia per le dimensioni, che per il valore di inizializzazione;
- i metodi `getDimX()`, `getDimY()`, `getDimZ()`;
- un metodo `stampa()`, che stampi tutti gli elementi della matrice;

Homework H24.1

- un metodo `sommaByPtr` che prenda in input due parametri formali di tipo puntatore a `Matrice3D` e restituisca la somma delle due matrici come puntatore al tipo `Matrice3D`;
- un metodo `sommaByReference` che prenda in input due parametri formali reference al tipo `Matrice3D` e restituisca la somma delle due matrici come reference al tipo `Matrice3D`;
- un metodo `getElement(int x, int y, int z)` che restituisca una reference all'elemento di indici `x`, `y`, e `z`.
- un metodo `getValue(int x, int y, int z)` che restituisca il valore dello elemento di indici `x`, `y`, e `z`.

FINE