



# Laurea triennale in Informatica

*modulo (CFU 6) di*

## Programmazione II e Lab.

**prof. Mariarosaria Rizzardi**

Centro Direzionale di Napoli – Isola C4

stanza: n. 423 – IV piano Lato Nord

tel.: 081 547 6545

email: [mariarosaria.rizzardi@uniparthenope.it](mailto:mariarosaria.rizzardi@uniparthenope.it)

The background features a large, faint, circular seal of the University of Naples Federico II. The seal includes the text "1920 - 2020" at the top, "DEGLI STUDI DI NAPOLI" around the top inner edge, "UNIVERSITA DI NAPOLI" around the bottom inner edge, and "100° ANNIVERSARIO" at the bottom. In the center is a shield with a seated figure and a book.

# ➤ **Programmazione in C++:** **Member Initializer List**

# Member Initializer List

La **lista di inizializzazione dei membri** è una caratteristica propria dei **costruttori di classe**; essa appare sempre tra la lista di argomenti del costruttore e il suo corpo ed è preceduta da `::`.

## Esempio: costruttore con parametri

```
class Complex {  
private:  
    double Re;  
    double Im;  
public:  
    Complex(double, double);  
    /* ... */  
};  
  
Complex::Complex(double a, double b) : Re(a), Im(b) { ... }
```

lista di inizializzazione

equivale al costruttore  
con parametri

```
Complex::Complex(double a, double b)  
{  
    Re = a; inizializzazione dei membri  
    Im = b; prima dell'esecuzione del  
    ... corpo del costruttore  
}
```

Nel costruttore con parametri della classe **Complex** la notazione **Attributo(<espressione>)** indica al compilatore che **Attributo** deve memorizzare il valore fornito da **<espressione>**; **<espressione>** può anche essere qualcosa di complesso come la chiamata ad una funzione.

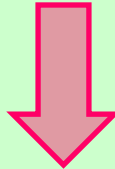
# Member Initializer List

Analogamente per il costruttore di copia:

**Esempio:** costruttore di copia

```
class Complex {  
private:  
    double Re;  
    double Im;  
public:  
    Complex(Complex &z);  
    /* ... */  
};
```

lista di inizializzazione



```
Complex::Complex(Complex &z) : Re(z.Re), Im(z.Im) { }
```

equivale al costruttore  
di copia

```
Complex::Complex(Complex &z)  
{  
    this->Re = z.Re;  
    this->Im = z.Im;  
}
```

# Esempio completo: senza Lista di inizializzazione

```
class Complex {  
private:  
    double Re;  
    double Im;  
  
public:  
    Complex() {}                costruttore default  
    Complex(double, double);   costruttore con parametri  
    Complex(Complex &z);        costruttore di copia  
  
    double getReal() { return this->Re; }  
    double getImag() { return this->Im; }  
};
```

```
#include <iostream>  
using namespace std;
```

```
int main() {  
    Complex Z1;  
    cout << "Z1 = " << Z1.getReal() << " + i*" << Z1.getImag() << endl;  
    Complex Z2(1.5, 2.5);  
    cout << "Z2 = " << Z2.getReal() << " + i*" << Z2.getImag() << endl;  
    Complex Z3(Z2);  
    cout << "Z3 = " << Z3.getReal() << " + i*" << Z3.getImag() << endl;  
    return 0;  
}
```

```
Complex::Complex(double a, double b) { Re=a; Im=b; }  
Complex::Complex(Complex &z) { Re=z.Re; Im=z.Im; }
```

```
Z1 = 3.95253e-323 + i*5.92879e-322  
Z2 = 1.5 + i*2.5  
Z3 = 1.5 + i*2.5
```

# Esempio completo: con Lista di inizializzazione

```
class Complex {  
private:  
    double Re;  
    double Im;  
  
public:  
    Complex() : Re(0), Im(0) {}      costruttore default  
    Complex(double, double);        costruttore con parametri  
    Complex(Complex &z);             costruttore di copia  
  
    double getReal() { return this->Re; }  
    double getImag() { return this->Im; }  
};
```

```
#include <iostream>  
using namespace std;
```

```
int main() {  
    Complex Z1;  
    cout << "Z1 = " << Z1.getReal() << " + i*" << Z1.getImag() << endl;  
    Complex Z2(1.5, 2.5);  
    cout << "Z2 = " << Z2.getReal() << " + i*" << Z2.getImag() << endl;  
    Complex Z3(Z2);  
    cout << "Z3 = " << Z3.getReal() << " + i*" << Z3.getImag() << endl;  
    return 0;  
}
```

```
Complex::Complex(double a, double b) : Re(a), Im(b) {}  
Complex::Complex(Complex &z) : Re(z.Re), Im(z.Im) {}
```

```
Z1 = 0 + i*0  
Z2 = 1.5 + i*2.5  
Z3 = 1.5 + i*2.5
```



# Quando usare Member Initializer List

La **lista di inizializzazione** va usata:

1. per inizializzare i dati membro costanti non statici;
2. per inizializzare i dati membro reference;
3. per inizializzare gli oggetti membro che non hanno un costruttore default;
4. per inizializzare i dati membro della classe base;
5. quando il nome del parametro del costruttore è lo stesso del dato membro (in alternativa si può usare il puntatore **this**);

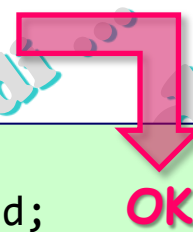
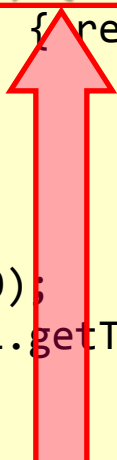
# Quando usare Member Initializer List

1. per inizializzare dati membro costanti non statici;

```
#include<iostream>
using namespace std;

class Test {
    const int t;
public:
    Test(int t) {this->t=t; }
    int getT() { return t; }
};

int main()
{
    Test t1(10);
    cout << t1.getT() << endl;
    return 0;
}
```



```
#include<iostream>
using namespace std;

class Test {
    const int t;
public:
    Test(int t) : t(t) {}
    int getT() { return t; }
};

int main()
{
    Test t1(10);
    cout << t1.getT() << endl;
    return 0;
}
```

10

**errore di compilazione:** In constructor 'Test::Test(int)':  
example1a.cpp|21|error: uninitialized const member in 'const int'  
example1a.cpp|19|note: 'const int Test::t' should be initialized  
example1a.cpp|21|error: assignment of read-only member 'Test::t'



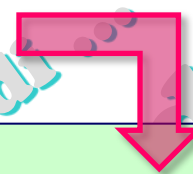
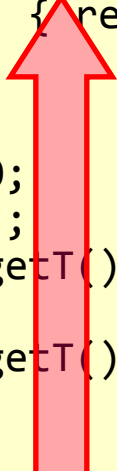
# Quando usare Member Initializer List

## 2. per inizializzare dati membro reference;

```
#include<iostream>
using namespace std;

class Test {
    int &t; // reference
public:
    Test(int &t) {this->t=t; }
    int getT() { return t; }
};

int main() {
    int x = 20;
    Test t1(x);
    cout<<t1.getT()<<endl;
    x = 30;
    cout<<t1.getT()<<endl;
    return 0;
}
```



```
#include<iostream>
using namespace std;

class Test {
    int &t; // reference
public:
    Test(int &t) : t(t) {}
    int getT() { return t; }
};

int main() {
    int x = 20;
    Test t1(x);
    cout<<t1.getT()<<endl;
    x = 30;
    cout<<t1.getT()<<endl;
    return 0;
}
```

OK!

20  
30

**errore di compilazione:** In constructor 'Test::Test(int&)':  
example2a.cpp|16|error: uninitialized reference member in 'int&'  
example2a.cpp|14|note: 'int& Test::t' should be initialized|

# Quando usare Member Initializer List

3. per inizializzare oggetti membro che non hanno un costruttore default;

OK!

```
#include<iostream>
using namespace std;
class A {
    int i;
public:
    A(int ); // costruttore con parametro di A
};
A::A(int arg) {
    i = arg;
    cout<<"costruttore con parametro di A: i="<<i<<endl;
}
class B {
    A a; // l'oggetto a è membro della classe B
public:
    B(int ); // costruttore con parametro di B
};
B::B(int x) {
    a = (A)x;
    cout<<"costruttore con parametro di B"<<endl;
}
int main()
{
    B obj(10);
    return 0;
}
```

**errore di compilazione**

```
#include<iostream>
using namespace std;
class A {
    int i;
public:
    A(int ); // costruttore con parametro di A
};
A::A(int arg) {
    i = arg;
    cout<<"costruttore con parametro di A: i="
        <<i<<endl;
}
class B {
    A a; // l'oggetto a è membro della classe B
public:
    B(int ); // costruttore con parametro di B
};
B::B(int x) : a(x) {
    cout<<"costruttore con parametro di B"<<endl;
}
int main()
{
    B obj(10);
    return 0;
}
```

costruttore con parametro di A: i = 10  
costruttore con parametro di B

# Quando usare Member Initializer List

## 4. per inizializzare i dati membro della classe base;

OK!

```
#include<iostream>
using namespace std;
class A {
    int i;
public:
    A(int ); // costruttore con parametro di A
};
A::A(int arg) {
    i = arg;
    cout<<"costruttore con parametro di A: i="<<i<<endl;
}
class B : public A { // classe B derivata da A
public:
    B(int ); // costruttore con parametro di B
};
B::B(int x) {
    A(x);
    cout<<"costruttore con parametro di B"<<endl;
}
int main()
{
    B obj(10);
    return 0;
}
```

**errore di compilazione**

```
#include<iostream>
using namespace std;
class A {
    int i;
public:
    A(int ); // costruttore con parametro di A
};
A::A(int arg) {
    i = arg;
    cout<<"costruttore con parametro di A: i="
        <<i<<endl;
}
class B : public A { // classe B derivata da A
public:
    B(int ); // costruttore con parametro di B
};
B::B(int x) : A(x) { // chiama il costruttore di A
    cout<<"costruttore con parametro di B"<<endl;
}
int main()
{
    B obj(10);
    return 0;
}
costruttore con parametro di A: i = 10
costruttore con parametro di B
```

5. quando il nome del parametro del costruttore è lo stesso del dato membro (in alternativa si può usare il puntatore `this`);

[illegible]

0

???

```
#include<iostream>
using namespace std;

class A {
    int i;
public:
    A(int );// costruttore con parametro di A
    int getI() { return i; }
};

A::A(int i) : i(i) {}
/* oppure in alternativa
A::A(int i) { this->i = i; } */

int main()
{
    A a(15);
    cout << a.getI() << endl;
}
```

15