

2. Klassen in C++

- Konstruktoren können auch mit einem function try block implementiert werden, auch wenn ein passender handler vorliegt, wird die Ausnahme **immer** re-thrown !!!

```
struct Y {  
    X* p;  
    Y(int i) try : p(new X)  
    { if (i) throw "huhh"; }  
    catch(...)  
    { /* delete p; NOT ALLOWED !!! */  
      /* throw "huhh"; implicitly */}  
    ~Y() { delete p; }  
};
```

15.3 (10): Referring to any non-static member or base class of an object in the handler for a function-try-block of a constructor or destructor for that object results in undefined behavior.

2. Klassen in C++

Initialisierung vs. Zuweisung:

= im Kontext einer Objektdeklaration: **Initialisierung**

```
X x = something; // initialize
```

= nicht im Kontext einer Objektdeklaration: **Zuweisung**

```
x = something; // assign !
```

```
class X {  
    const int c;  
public:  
    X(int i): c(i) {} // ok, aber  
    // X(int i) {c=i;} // falsch  
};
```



Prefer initialization !

2. Klassen in C++

Initialisierung vs. Zuweisung:

```
#include <iostream>

class A {
public:
    A(int i){ std::cout<<"A("<<i<<")\n"; }
};

class B {
    A myA;
public:
    B (int i) { std::cout<<"B("<<i<<")\n"; }
};

int main() { A a(1); B b(2); } // valid C++ ?????
```

2. Klassen in C++

Initialisierung vs. Zuweisung:

```
#include <iostream>
```

```
class A {  
public:  
    A(int i){ std::cout<<"A("<<i<<")\n"; }  
};
```

```
class B {  
    A myA;  
public:  
    B (int i) { std::cout<<"B("<<i<<")\n"; }  
};
```

```
int main() { A a(1); B b(2); }
```



Prefer initialization !

Error init.cpp 11: Cannot find default constructor to initialize member 'B::myA' in function B::B(int)



2. Klassen in C++

Initialisierung vs. Zuweisung:

```
#include <iostream>
```

```
class A {  
public:  
    A(int i){ std::cout<<"A("<<i<<" )\n"; }  
};
```

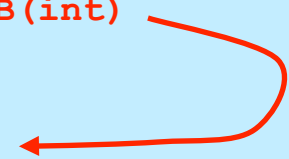
```
class B {  
    A myA;  
public:  
    B (int i) { myA = i; std::cout<<"B("<<i<<" )\n"; }  
};
```

```
int main() { A a(1); B b(2); }
```



Prefer initialization !

Error init.cpp 11: Cannot find default constructor to initialize member 'B::myA' in function B::B(int)



2. Klassen in C++

Initialisierung vs. Zuweisung:

```
#include <iostream>
```

```
class A {  
public:
```

```
    A(int i = 0 ) { std::cout << "A("<<i<<")\n"; }
```

```
};
```

```
class B {
```

```
    A myA;
```

```
public:
```

```
    B (int i) { myA = i; std::cout << "B("<<i<<")\n"; }
```

```
};
```

```
int main() { A a(1); B b(2); }
```



Prefer initialization !

```
$ init  
A(1)  
A(0) ?  
A(2) ?  
B(2)
```

2. Klassen in C++

Initialisierung vs. Zuweisung:

```
#include <iostream>
```

```
class A {  
public:  
    A(int i){ std::cout << "A("<<i<<")\n"; }  
};
```

```
class B {  
    A myA;  
public:  
    B (int i) : myA(i) { std::cout << "B("<<i<<")\n"; }  
};
```

```
int main() { A a(1); B b(2); }
```



Prefer initialization !

```
$ init  
A(1)  
A(2)  
B(2)
```

2. Klassen in C++

Initialisierung vs. Zuweisung:

```
#include <iostream>
```



even better: uniform initialization !

```
class A {  
public:
```

```
    A(int i){ std::cout << "A{"<<i<<"}\n"; }
```

```
};
```

```
class B {  
    A myA;
```

```
public:
```

```
    B (int i) : myA{i} { std::cout << „B{"<<i<<"}\n"; }
```

```
};
```

```
int main() { A a{1}; B b{2} } }
```

```
$ init  
A{1}  
A{2}  
B{2}
```


2. Klassen in C++

C++ idiom: *Resource Acquisition Is Initialization* (*)

```
void doDB() { // from Steven C. Dewhurst: C++ Gotchas (gotcha #67)
    lockDB();
    // do stuff with database ... but could throw !?
    unlockDB();
}
```

```
void doDB() {
    lockDB();
    try { // do stuff with database ...
    }
    catch ( ... ) { unlockDB(); throw; } // ugly
    unlockDB();
}
```

(of an object !*

2. Klassen in C++



C++ idiom: *Resource Acquisition Is Initialization*

```
// better:  
class DBLock {  
    public:  
        DBLock() { lockDB(); }  
        ~DBLock() { unlockDB(); }  
};
```

```
void doDB() {  
    DBLock lock;  
    // do stuff with database ...  
}
```

Fallen:

```
// NOT: DBLock lock();  
// NOT: DBLock();
```

2. Klassen in C++

C++ idiom: *Resource Acquisition Is Initialization*

```
struct X {
    X() { cout<<"X()\n"; }
    ~X() { cout<<"~X()\n"; }
};

struct Xpointer { // a (not very) smart pointer
    X* pointer;
    Xpointer(X* p): pointer(p){}
    ~Xpointer(){delete pointer;}
};

struct Y {
    Xpointer p;
    Y(int i) try : p(new X)
    { if (i) throw "huhh"; }
    catch(...)
    { cout<< "caught local\n"; }
    ~Y() {}
};

int main() try {
    cout<<"sizeof(Y)="<<sizeof(Y)<<endl;
    Y y0(0);
    Y y1(1);
}
catch(...) { cout<<"caught final\n";}
```



```
#include <iostream>
using std::whatever;
```

```
sizeof(Y)=4
X()
X()
~X()
caught local
~X()
caught final
```

2. Klassen in C++

C++ idiom: Resource Acquisition Is Initialization

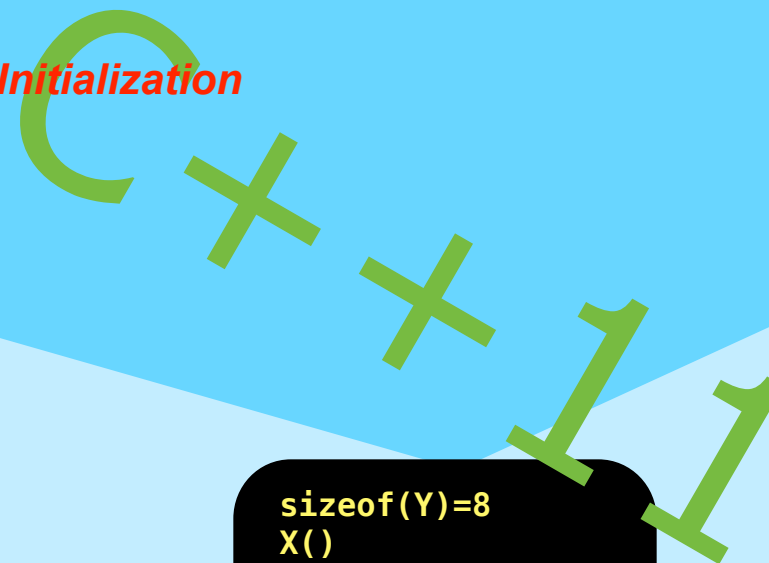
```
#include <iostream>
#include <memory>

struct X {
    X() { std::cout<<"X()\n"; }
    ~X() { std::cout<<"~X()\n"; }
};

struct Y {
    std::unique_ptr<X> p;
    Y(int i) try : p(new X)
    { if (i) throw "huhh"; }
    catch(...)
    { std::cout<< "caught local\n";}

    ~Y() {}
};

int main()
    try {
        std::cout<<"sizeof(Y)="<<sizeof(Y)<<std::endl;
        Y y0(0);
        Y y1(1);
    }
    catch(...) { std::cout<<"caught final\n"; }
```



```
sizeof(Y)=8
X()
X()
~X()
caught local
~X()
caught final
```

2. Klassen in C++



```
#include <iostream>  
using std::whatever;
```



C++ idiom: *Resource Acquisition Is Initialization*



```
class Trace { // C++ Gotchas, dito #67  
public:  
    Trace (const char* msg): m_(msg) {cout << "Entering " << m_ << endl;}  
    ~Trace() {cout << "Exiting " << m_ << endl;}  
private:  
    const char* m_;  
};  
Trace a("global");  
void foo(int i) {  
    Trace b("foo");  
    while (i--) { Trace l("loop"); /* ... */ }  
    Trace c("after loop");  
}  
int main() { foo(2); }
```

```
$ t  
Entering global  
Entering foo  
Entering loop  
Exiting loop  
Entering loop  
Exiting loop  
Entering after loop  
Exiting after loop  
Exiting foo  
Exiting global
```

2. Klassen in C++



```
#include <iostream>
#include <ctime>
using std::whatever;
```



C++ idiom: *Resource Acquisition Is Initialization*



```
class Timer {
    long start, stop;
    void report()
        {cout<<(stop-start)/1000000.0<<"s"<<endl;}
public:
    Timer():start(clock()){}
    ~Timer(){ stop=clock(); report();}
};
```

2. Klassen in C++



```
#include <iostream>
#include <chrono>
```



C++ idiom: *Resource Acquisition Is Initialization*



```
class Timer { // conforms to C++11
    std::chrono::steady_clock::time_point start;
    std::string what;
public:
    Timer(std::string s): start(std::chrono::steady_clock::now()), what(s) {}
    ~Timer() {
        auto duration = std::chrono::steady_clock::now() - start;
        std::cout << what+":\t" <<
            std::chrono::duration_cast<std::chrono::milliseconds>(duration).count()
            << " ms" << std::endl;
    }
};
```