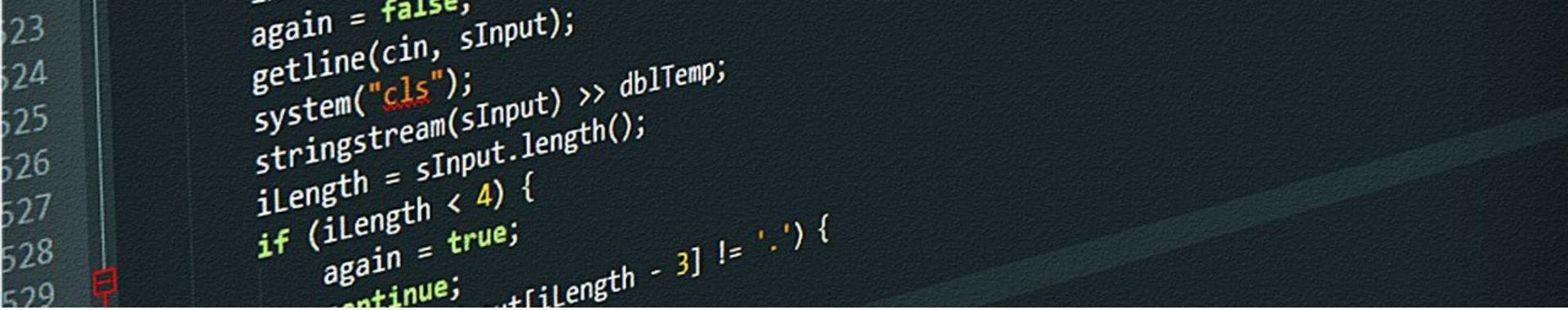

Thomas

C23-09.3 Smart Pointer

C23 – Fortgeschrittene Algorithmen und Programmierung

v4

Hochschule für Technik und Wirtschaft Berlin
University of Applied Sciences

Standardbibliothek

Smart Pointer

Standardbibliothek	
auto_ptr (veraltet)	Entspricht ungefähr dem unique_ptr
Standardbibliothek (seit C++11)	
unique_ptr	• Kein anderer Smartpointer darf den selben Zeiger verwalten • Endet die Lebensdauer oder wird dem unique_ptr NULL zugewiesen, dann wird der Speicher des Zeigers automatisch freigegeben. • Zuweisung an einen anderen unique_ptr ist nicht erlaubt.
shared_ptr	• shared_ptr teilen sich den Besitz eines Zeigers • Intern wird gezählt, wie viele shared_ptr einen bestimmten Zeiger teilen • der Speicher des Zeigers wird freigegeben, wenn alle shared_ptr, die sich den Besitz teilen zerstört werden. • Durch Zuweisung an einen anderen shared_ptr wird der Besitz mit diesem geteilt
weak_ptr	• Übernimmt keine Verantwortung (Besitz) über einen Zeiger • Kann auf ein Pointer referenzieren, der eigentlich von einem shared_ptr verwaltet wird, um den Zustand abzufragen.
Boost – Smartpoiner (Kompatibilität → falls C++11 nicht verfügbar ist)	
scoped_ptr, shared_ptr, weak_ptr	Entspricht unique_ptr, shared_ptr und weak_ptr aus der Standardbibliothek!

Standardbibliothek
Smart Pointer

Beispiel – unique_ptr

```
#include <iostream>
#include <memory> // C++11 Smartpointer

int main()
{
    int* pIntPointer = new int(10);
    std::unique_ptr<int> pUniqueOwner(pIntPointer); // besitzt jetzt 'pIntPointer'

    // Dereferenzieren geht genauso wie bei normalem Pointer!
    std::cout<<*pUniqueOwner <<std::endl;

    // gibt den Speicher wieder frei!
    pUniqueOwner = NULL;
    // oder pUniqueOwner.reset();

    // passiert aber sowieso, wenn der Block (die Main-Funktion) endet!
    return 0;
}
```

Standardbibliothek
Smart Pointer

shared_ptr Beispiel (1/2)

```
#include <iostream>
#include <memory> // C++11 Smartpointer

using namespace std;

class MyClass
{
public:
    MyClass(int val) : m_value(val) { }

    ~MyClass()
    {
        cout<<"Myclass (" <<m_value <<") destructed!" <<endl;
    }

    int getValue() const { return m_value; }

private:
    int m_value;
};
```

Standardbibliothek
Smart Pointer

shared_ptr Beispiel (2/2)

```
int main()
{
    shared_ptr<MyClass> pSharedPtr1(new MyClass(12));
    shared_ptr<MyClass> pSharedPtr2 = pSharedPtr1;

    {
        shared_ptr<MyClass> pSharedPtr3 = pSharedPtr1;
        cout<<"Besitzer: " <<pSharedPtr1.use_count() <<endl;
    }

    cout<<"Besitzer: " <<pSharedPtr1.use_count() <<endl;
    pSharedPtr2.reset();
    cout<<"Besitzer: " <<pSharedPtr1.use_count() <<endl;
    std::cout<<"Wert: " << pSharedPtr1->getValue() << std::endl;
    pSharedPtr1.reset();

    if (!pSharedPtr1) {
        cout<<"Kein Besitzer mehr! Das Objekt wurde zerstoert. " <<endl;
    }
    return 0;
}
```

```
Besitzer: 3
Besitzer: 2
Besitzer: 1
Wert: 12
Myclass (12) destructed!
Kein Besitzer mehr! Das Objekt wurde zerstoert.
```

Standardbibliothek

Smart Pointer – Best Practice

- Vorzugsweise auf etablierte Bibliotheken zurück greifen, statt den Code selbst neu zu implementieren.
  In der Regel sind diese Bibliotheken gut getestet und wurden schon vielfach verwendet.
- Immer Smart Pointer verwenden, wenn Speicher dynamisch allokiert wird.

Hochschule für Technik
und Wirtschaft Berlin

University of Applied Sciences

www.htw-berlin.de