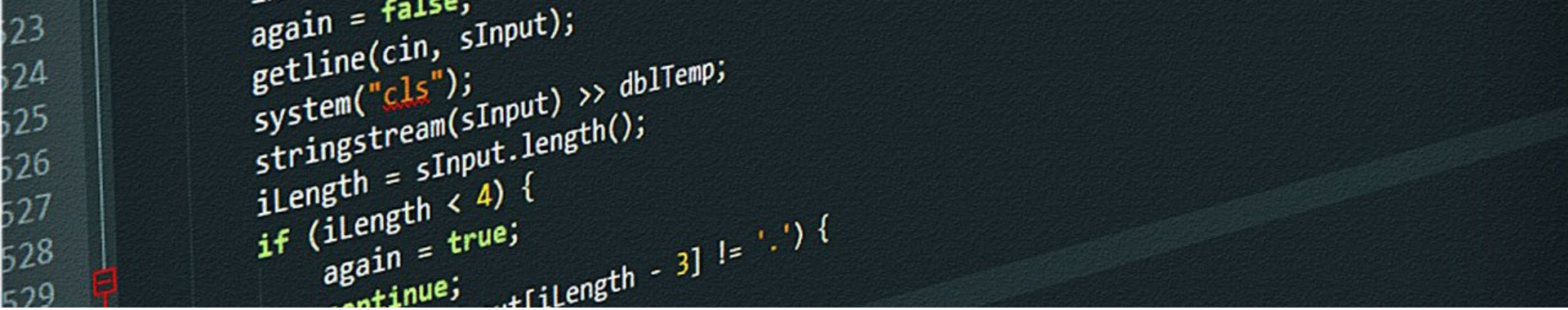

Thomas

C23-06.2 - Qualifiers

Advanced algorithms and programming

v5

Hochschule für Technik und Wirtschaft Berlin
University of Applied Sciences

Constant member functions

Example with compiler errors

- Constant Member functions were introduced to solve the problem of using constant objects:

```
#include <iostream>
#include <string>

class Person
{
public:
    Person(std::string firstname, std::string lastname)
        : m_firstname(firstname), m_lastname(lastname) { }

    std::string getName()
    {
        return m_firstname + " " + m_lastname;
    }

private:
    std::string m_firstname, m_lastname;
};
```

```
void printPersonName(const Person& rPerson) {
    std::string name = rPerson.getName(); // Error
    std::cout << "The person is: " << name << std::endl;
}

void printPersonName(const Person* pPerson) {
    std::string name = pPerson->getName(); // Error
    std::cout << "The person is: " << name << std::endl;
}

int main()
{
    Person p1("Bill", "Gates");
    printPersonName(p1);
    printPersonName(&p1);
}
```

```
1>C:\Users\admin\source\repos\C23-06\08 Const_problem\Main.cpp(20,40): error C2662: "std::string Person::getName(void)": this-Zeiger kann nicht von "const Person" in "Person &" konvertiert werden
1>C:\Users\admin\source\repos\C23-06\08 Const_problem\Main.cpp(20,24): message : Durch die Konvertierung gehen Qualifizierer verloren
1>C:\Users\admin\source\repos\C23-06\08 Const_problem\Main.cpp(10,17): message : Siehe Deklaration von "Person::getName"
1>C:\Users\admin\source\repos\C23-06\08 Const_problem\Main.cpp(25,41): error C2662: "std::string Person::getName(void)": this-Zeiger kann nicht von "const Person" in "Person &" konvertiert werden
1>C:\Users\admin\source\repos\C23-06\08 Const_problem\Main.cpp(25,24): message : Durch die Konvertierung gehen Qualifizierer verloren
1>C:\Users\admin\source\repos\C23-06\08 Const_problem\Main.cpp(10,17): message : Siehe Deklaration von "Person::getName"
```

Constant member functions

Compiler error:

```
1>C:\Users\admin\source\repos\C23-06\08 Const_problem\Main.cpp(20,40): error C2662: "std::string Person::getName(void)": this-Zeiger kann nicht von "const Person" in "Person &" konvertiert werden
1>C:\Users\admin\source\repos\C23-06\08 Const_problem\Main.cpp(20,24): message : Durch die Konvertierung gehen Qualifizierer verloren
1>C:\Users\admin\source\repos\C23-06\08 Const_problem\Main.cpp(10,17): message : Siehe Deklaration von "Person::getName"
1>C:\Users\admin\source\repos\C23-06\08 Const_problem\Main.cpp(25,41): error C2662: "std::string Person::getName(void)": this-Zeiger kann nicht von "const Person" in "Person &" konvertiert werden
1>C:\Users\admin\source\repos\C23-06\08 Const_problem\Main.cpp(25,24): message : Durch die Konvertierung gehen Qualifizierer verloren
1>C:\Users\admin\source\repos\C23-06\08 Const_problem\Main.cpp(10,17): message : Siehe Deklaration von "Person::getName"
```

Explanation:

1. No data may be changed via const references or const pointers

   printPersonName is not allowed to change the Person

2. Each member function of a class can change the data of an object

   The compiler assumes that *Person* ***could*** *be changed with the member function getName*

   getName is called from printPersonName

Constant member functions

- The compiler requires an indication that (constant) member functions do not change class member variables

- For this purpose, the keyword `const` is added as postfix after a declaration or definition of a member function.
  The compiler thus knows that the function does not change any member variables.

```
type member_function() const
{

}
```

Constant member functions

Working example


```cpp
#include <iostream>
#include <string>

class Person
{
public:
    Person(std::string firstname, std::string lastname)
        : m_firstname(firstname), m_lastname(lastname) { }

    std::string getName() const
    {
        return m_firstname + " " + m_lastname;
    }

private:
    std::string m_firstname, m_lastname;
};
```

```cpp
void printPersonName(const Person& rPerson) {
    std::string name = rPerson.getName(); // Error
    std::cout << "The person is: " << name << std::endl;
}

void printPersonName(const Person* pPerson) {
    std::string name = pPerson->getName(); // Error
    std::cout << "The person is: " << name << std::endl;
}

int main()
{
    Person p1("Bill", "Gates");
    printPersonName(p1);
    printPersonName(&p1);
}
```

```
The person is: Bill Gates
The person is: Bill Gates
```

Inline member functions

- Inline function calls are directly replaced by the content of the function!
  - Faster, but more memory usage (especially with frequent function calls)
  - 'inline' is a recommendation to the compiler. It does not have to follow this recommendation.

- Inline functions must be implemented in the same file as the class (usually in the header file)

- Functions that are implemented directly in the class are automatically inline
  The keyword 'inline' can be used optionally to increase readability
- All functions implemented in a header file must be 'inline'
  The keyword 'inline' for implementation or declaration is mandatory

Inline member functions

Example with compiler/linker errors

Inline.h

```
#pragma once

class InlineExampleClass
{
public:
    // implicitly inline
    InlineExampleClass() : a(1), b(2), c(3), d(4), e(5), f(6), g(7) { }

    int getA() { return a; } // implicitly inline
    inline int getB() { return b; } // explicitly inline
    int getC(); // inline (see implementation)
    inline int getD(); // explicitly inline

    // Non-inline functions must be implemented in a *.cpp file
    int getE(); // linker error (see below)!

    int getF(); // not inline! (not implemented in the same file)
    int getG(); // not inline! (not implemented in the same file)

private:
    int a, b, c, d, e, f, g;

};

inline int InlineExampleClass::getC() { return c; } // inline
int InlineExampleClass::getD() { return d; } // inline
int InlineExampleClass::getE() { return e; } // linker error!
```

Inline member functions

Example with compiler/linker errors

Inline.cpp

```cpp
#include "Inline.h"

// not inline
int InlineExampleClass::getF()
{
    return f;
}

// Does not work! Inline functions must be in the same
// file be like the class!
// => linker error
inline int InlineExampleClass::getG()
{
    return g;
}
```

Main.cpp

```cpp
#include <iostream>
#include "Inline.h"

int main()
{
InlineExampleClass ex;

std::cout << ex.getA() << "\n";
std::cout << ex.getB() << "\n";
std::cout << ex.getC() << "\n";
std::cout << ex.getD() << "\n";
std::cout << ex.getE() << "\n";
std::cout << ex.getF() << "\n";
std::cout << ex.getG() << "\n";
}
```

Hochschule für Technik
und Wirtschaft Berlin

University of Applied Sciences

www.htw-berlin.de