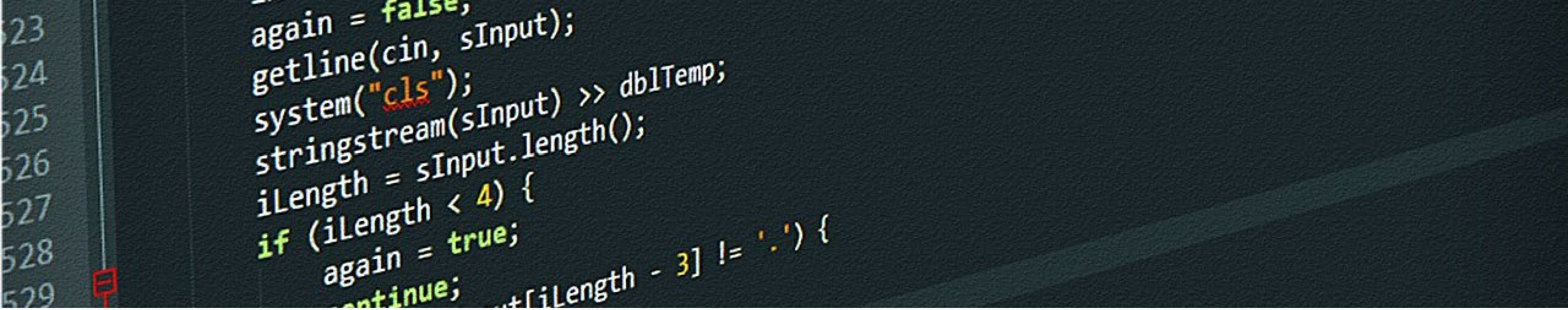


```
again = false;
getline(cin, sInput);
system("cls");
stringstream(sInput) >> dblTemp;
iLength = sInput.length();
if (iLength < 4) {
    again = true;
    continue;
```

Thomas

C23-10.1 Multiple inheritance

C23 - Advanced Algorithms and Programming

v5

Hochschule für Technik
und Wirtschaft Berlin
University of Applied Sciences

Multiple inheritance

Concept

- In C++ a class can inherit from several classes:

```
class MyClass : public MyBase1, protected MyBase2, public Mybase3 // ...
{
    // ...
};
```

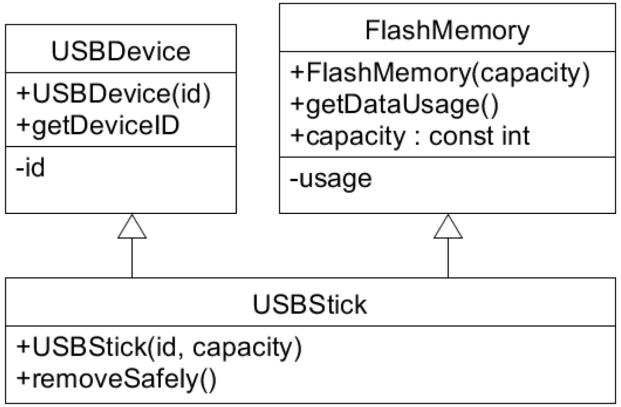
<u>In practice, multiple inheritance is controversial:</u>

- It can happen that two base classes have the same member functions
  Which function is meant when calling a function in the derived class?
- Multiple inheritance is problematic if two base classes have inherited from a common class
- Multiple inheritance may increase the complexity of the code

Multiple inheritance

Concept – USB stick example

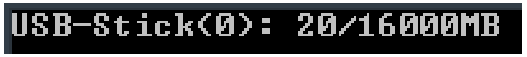
```cpp
#pragma once

class USBDevice {
public:
    USBDevice(int id) : m_id(id) { }
    int getDeviceId() const { return m_id; }
private:
    int m_id;
};

class FlashMemory {
public:
    FlashMemory(int capacityMB)
        : capacity(capacityMB), m_dataUsage(0) { }

    int getDataUsage() const { return m_dataUsage; }

    void writeData(int numMBytes)
    {
        if (numMBytes + m_dataUsage < capacity)
            m_dataUsage += numMBytes;
    }
    const int capacity;
private:
    int m_dataUsage;
};
```



```cpp
// USBStick inherits all properties of USBDevice and FlashMemory
class USBStick : public USBDevice, public FlashMemory
{
public:
    USBStick(int id, int capacity)
        : USBDevice(id), FlashMemory(capacity) { }
};
```

Multiple inheritance
Concept – USB stick example

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

int main()
{
USBStick myUsb(0, 16000);
myUsb.writeData(20);
std::cout << "USB-Stick(" << myUsb.getDeviceId() << "): "
<< myUsb.getDataUsage() << "/" << myUsb.capacity << "MB" << std::endl;
}
```



```
USB-Stick(0): 20/16000MB
```

Multiple inheritance

Identical members in base classes – function example

```cpp
 // ...

class USBDevice {
    //  ...
std::string toString() const {
        std::stringstream s;
        s << "USB Device ID: " << m_id;
        return s.str();
    }
    // ...
};

class FlashMemory {
public:
    // ...
    std::string toString() const {
        std::stringstream s;
        s << "Data Usage: " << m_dataUsage << "/" << capacity << "MB";
        return s.str();
    }
    // ...
};
```

Multiple inheritance

Identical members in base classes – function example

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

int main()
{
    USBStick myUsb(0, 16000);
    myUsb.writeData(20);
    // Compiler error: FlashMemory::toString or USBDevice::toString?
    std::cout << myUsb.toString() << std::endl;
}
```

```
1>Main.cpp
1>C:\Users\admin\source\repos\lectures\C23\C23-10.1\02_Identical_members\Main.cpp(10,29): error C2385: Mehrdeutiger Zugriff von "toString".
1>C:\Users\admin\source\repos\lectures\C23\C23-10.1\02_Identical_members\Main.cpp(10,29): message : könnte "toString" in Basis "USBDevice" sein
1>C:\Users\admin\source\repos\lectures\C23\C23-10.1\02_Identical_members\Main.cpp(10,29): message : oder könnte "toString" in Basis "FlashMemory" sein
1>Die Erstellung des Projekts "02_Identical_members.vcxproj" ist abgeschlossen -- FEHLER.
========== Erstellen: 0 erfolgreich, 1 fehlerhaft, 0 aktuell, 0 übersprungen ==========
```

The function **toString** exists in both base classes,
therefore, the compiler cannot know which function is meant

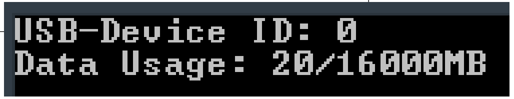
Solution: Use the Scope operator with the class name!

Multiple inheritance

Identical members in base classes – function example

- Solution: Use scope operator `::`

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

int main()
{
    USBStick myUsb(0, 16000);
    myUsb.writeData(20);

    std::cout << myUsb.USBDevice::toString() << std::endl;
    std::cout << myUsb.FlashMemory::toString() << std::endl;
}
```



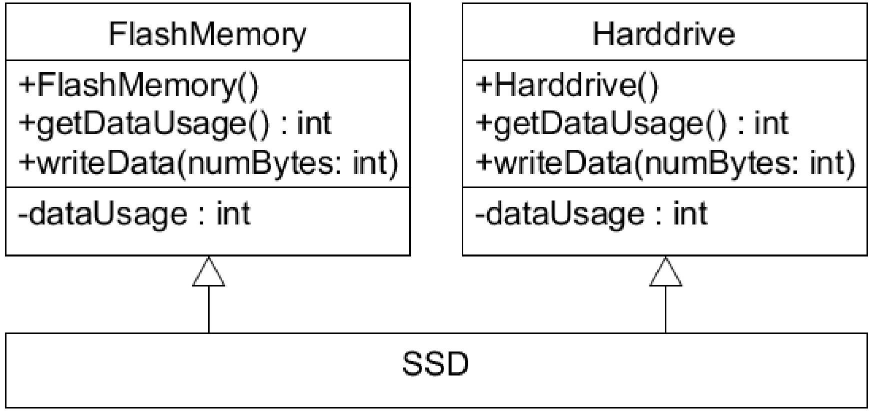
```
USB-Device ID: 0
Data Usage: 20/16000MB
```

Multiple inheritance

Identical members in base classes

Is data inherited twice?

- The following is an example to check if `dataUsage` is present twice in SSD after inheritance.



Multiple inheritance

Identical members in base classes – data example

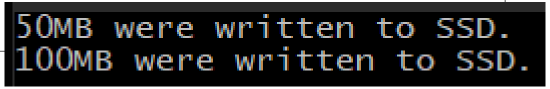
Example: Data is duplicated in both base classes

```
#pragma once

class FlashMemory {
public:
    FlashMemory() : m_dataUsage(0) { }
    int getDataUsage() const { return m_dataUsage; }
    void writeData(int numMBytes) { m_dataUsage += numMBytes; }
private:
    int m_dataUsage;
};

class Harddrive {
public:
    Harddrive() : m_dataUsage(0) { }
    int getDataUsage() const { return m_dataUsage; }
    void writeData(int numMBytes) { m_dataUsage += numMBytes; }
private:
    int m_dataUsage;
};

// Inherits all features of FlashMemory and Harddrive
class SSD : public Harddrive, public FlashMemory { };
```

Multiple inheritance

Identical members in base classes – data example

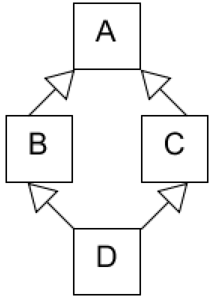
```cpp
#include <iostream>
#include "SSD.h"

int main()
{
    SSD ssd;
    ssd.Harddrive::writeData(50);
    std::cout << ssd.Harddrive::getDataUsage()
        << "MB were written to SSD." << std::endl;

    ssd.FlashMemory::writeData(100);
    std::cout << ssd.FlashMemory::getDataUsage()
        << "MB were written to SSD." << std::endl;
}
```



```
50MB were written to SSD.
100MB were written to SSD.
```

- The data is duplicated; the SSD inherits two separate memories from the two base classes
  Each base class keeps its own data during inheritance

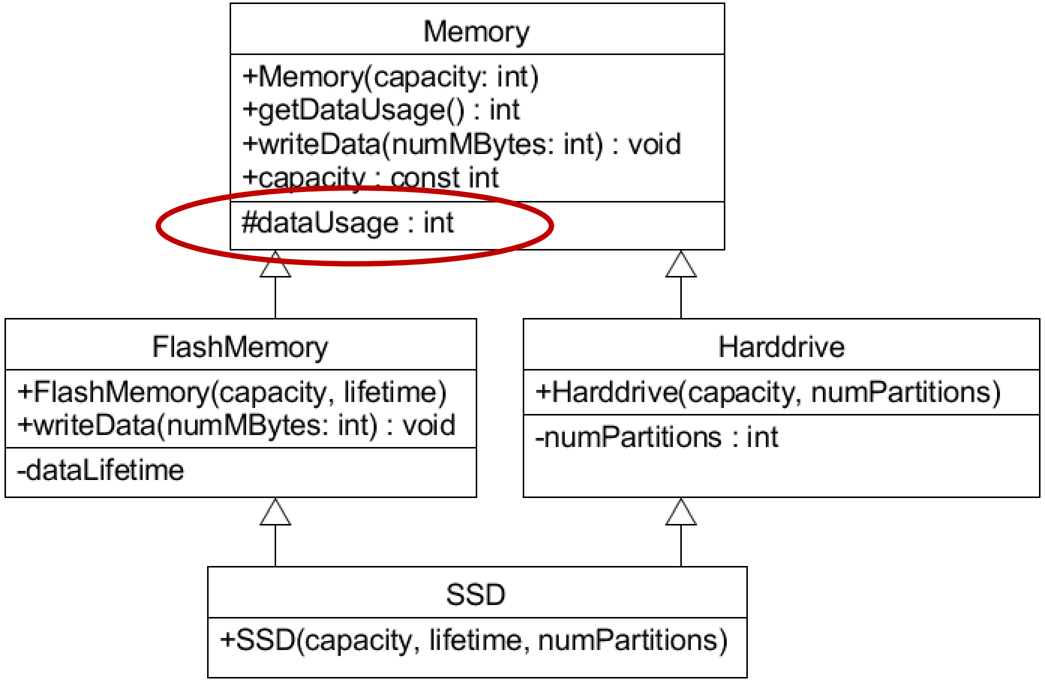
Multiple inheritance
Diamond Problem



- B and C inherit all properties of A.
- D indirectly inherits all properties twice (once each from B and from C).

Multiple inheritance
Diamond Problem

Example


Memory
+Memory(capacity: int)
+getDataUsage() : int
+writeData(numMBytes: int) : void
+capacity : const int
#dataUsage : int
FlashMemory
+FlashMemory(capacity, lifetime)
+writeData(numMBytes: int) : void
-dataLifetime
Harddrive
+Harddrive(capacity, numPartitions)
-numPartitions : int
SSD
+SSD(capacity, lifetime, numPartitions)

Multiple inheritance

Diamond Problem – example

```
#pragma once
#include <string>
#include <sstream>

class Memory {
public:
    Memory(int capacityMB) : capacity(capacityMB), m_dataUsage(0) { }

    int getDataUsage() const { return m_dataUsage; }

    // virtual, because FlashMemory overwrites this function!
    virtual void writeData(int numMBytes) {
        if (numMBytes + m_dataUsage < capacity) m_dataUsage += numMBytes;
    }

    const int capacity;
protected:
    int m_dataUsage;
};
```

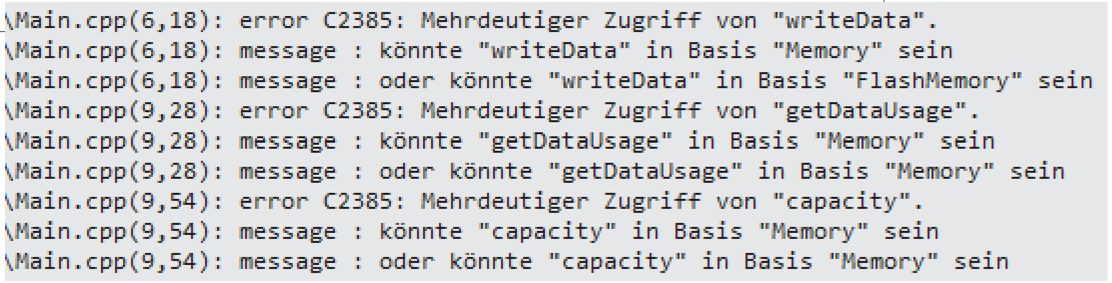
Multiple inheritance
Diamond Problem – example

```
class FlashMemory : public Memory {
public:
    FlashMemory(int capacityMB, int writeCycles)
        : Memory(capacityMB), m_wCyclesLeft(writeCycles) { }

    virtual void writeData(int numMBytes) {
        if (m_wCyclesLeft > 0) Memory::writeData(numMBytes);
        else m_dataUsage = 0;
    }
private:
    int m_wCyclesLeft;
};

class Harddrive : public Memory {
public:
    Harddrive(int capacityMB) : Memory(capacityMB), m_numPartitions(0) {}
    void createPartition() { m_numPartitions++; }
    int getNumPartitions() { return m_numPartitions; }
private:
    int m_numPartitions;
};

class SSD : public Harddrive, public FlashMemory {
public:
    SSD(int capacity, int writeCycles)
        : Harddrive(capacity), FlashMemory(capacity, writeCycles) { }
};
```

Multiple inheritance

Diamond Problem – example

```cpp
class Harddrive : public Memory {
public:
    Harddrive(int capacityMB) : Memory(capacityMB), m_numPartitions(0) {}
    void createPartition() { m_numPartitions++; }
    int getNumPartitions() { return m_numPartitions; }
private:
    int m_numPartitions;
};

class SSD : public Harddrive, public FlashMemory {
public:
    SSD(int capacity, int writeCycles)
        : Harddrive(capacity), FlashMemory(capacity, writeCycles) { }
};
```



```
\Main.cpp(6,18): error C2385: Mehrdeutiger Zugriff von "writeData".
\Main.cpp(6,18): message : könnte "writeData" in Basis "Memory" sein
\Main.cpp(6,18): message : oder könnte "writeData" in Basis "FlashMemory" sein
\Main.cpp(9,28): error C2385: Mehrdeutiger Zugriff von "getDataUsage".
\Main.cpp(9,28): message : könnte "getDataUsage" in Basis "Memory" sein
\Main.cpp(9,28): message : oder könnte "getDataUsage" in Basis "Memory" sein
\Main.cpp(9,54): error C2385: Mehrdeutiger Zugriff von "capacity".
\Main.cpp(9,54): message : könnte "capacity" in Basis "Memory" sein
\Main.cpp(9,54): message : oder könnte "capacity" in Basis "Memory" sein
```

Compiler Error: The data of the common base class is inherited twice

Multiple inheritance

Diamond Problem

Problem: SSD inherits the properties of `memory` twice (once via `FlashMemory` and once via `Harddrive`)

Solution: virtual inheritance

- `FlashMemory` and `Harddrive` must inherit *<u>virtually.</u>*
- The keyword **`virtual`** inherits additional information about the base class.
- The `SSD` class can then distinguish which member variables and functions originate from the base class or which have been inherited via `FlashMemory` or `Harddrive`. <u>Shared data from the base class is not inherited twice!</u>
- Note: Virtual inheritance consumes additional memory.

Multiple inheritance

Diamond problem – example / solution

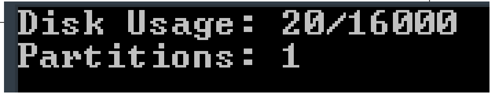
```
class FlashMemory : virtual public Memory {
public:
    FlashMemory(int capacityMB, int writeCycles)
        : Memory(capacityMB), m_wCyclesLeft(writeCycles) { }

    virtual void writeData(int numMBytes) {
        if (m_wCyclesLeft > 0) Memory::writeData(numMBytes);
        else m_dataUsage = 0;
    }
private:
    int m_wCyclesLeft;
};

class Harddrive : virtual public Memory {
public:
    Harddrive(int capacityMB) : Memory(capacityMB), m_numPartitions(0) {}
    void createPartition() { m_numPartitions++; }
    int getNumPartitions() { return m_numPartitions; }
private:
    int m_numPartitions;
};
```

Multiple inheritance

Diamond problem – example / solution

```cpp
class SSD : public Harddrive, public FlashMemory {
public:
    // The virtual base class must be explicitly initilized, if there is no default 
constructor,
    SSD(int capacity, int writeCycles)
        : Harddrive(capacity), FlashMemory(capacity, writeCycles), Memory(capacity) { }
};
```

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

int main() {
    SSD ssd(16000, 1000000);
    ssd.writeData(20);
    ssd.createPartition();
    std::cout << "Disk Usage: "
        << ssd.getDataUsage() << "/" << ssd.capacity << std::endl;
    std::cout << "Partitions: " << ssd.getNumPartitions() << std::endl;
}
```



```
Disk Usage: 20/16000
Partitions: 1
```

Multiple inheritance

Best practice

- Due to the problems described above, multiple inheritance must be used with caution
  However, it also allows to write efficient code

- Multiple inheritance is clearer with purely abstract classes (interfaces)
  At most one base class can be a concrete class. The interface classes may not have a base class (This corresponds to the keyword '`implements`' from Java)
- The Diamond problem must be considered