

Modernes C++

...für Programmierer

Unit 02: Ein- und Ausgabe

by

Dr. Günter Kolousek

Überblick

- ▶ Einfache imperative Programme
- ▶ Eine Quelldatei
- ▶ Ausgabe auf `stdout`
- ▶ Eingabe von `stdin`
- ▶ Erste Schritte mit ganzen Zahlen, `string`, `char`
- ▶ `if`, `switch`, `while`, `do` und `while`
- ▶ Behandlung fehlerhafter Eingaben

Minimales C++ Programm

```
int main() {  
}
```

Minimales C++ Programm – 2

```
// minimum2.cpp
// filenames for C headers start with "c"
#include <cstdlib> // preprocessor statement

int main() {
    return EXIT_SUCCESS; // vs. EXIT_FAILURE
    // for most systems this is equivalent to
    // return 0;
}
```

"Hello, World!"

```
// hello.cpp
#include <iostream>

int main() {
    // namespace "std"
    // scope resolution operator "::"
    // global object "std::cout" -> stdout
    // overloaded operator "<<"
    // output manipulator "std::endl"
    std::cout << "Hello, World!" << std::endl;
}
```

Hello, World!

”Hello, World!” – 2

```
// hello2.cpp
#include <iostream>

/*
    now, no "std::" is necessary any more,
    but don't write it in a header file at all!
*/
using namespace std;

int main() {
    // C-string literal
    cout << "Hello, World!" << endl;
}
```

Hello, World!

”Hello, World!” – 3

```
// hello3.cpp
#include <iostream>

using namespace std;

int main() {
    // \n ... plattform independent new line
    cout << "Hello, World!\n" << flush;
}
```

Hello, World!

”Hello, World!” – 4

```
// hello4.cpp
#include <iostream>

using namespace std;

int main() {
    cout << "Hello, World!";
}
```

Hello, World!

”Hello, World!” – 4

```
// hello4.cpp
#include <iostream>

using namespace std;

int main() {
    cout << "Hello, World!";
}
```

Hello, World!

Am Prozessende werden immer die Buffer geleert (außer u.U. wenn Prozess abstürzt)!

”Hello, World!” – 5

```
// hello5.cpp
#include <iostream>

using namespace std;

int main() {
    cout << "Hello, World!";
    while (true) {}
}
```

”Hello, World!” – 5

```
// hello5.cpp
#include <iostream>

using namespace std;

int main() {
    cout << "Hello, World!";
    while (true) {}
}
```

Keine Ausgabe... Prozess ist abubrechen... Auch dann keine Ausgabe!

”Hello, World!” – 6

```
// hello6.cpp
#include <iostream>
using namespace std;
int main() {
    cout << "Hello, World!\n";
    while (true) {}
}
```

”Hello, World!” – 6

```
// hello6.cpp
#include <iostream>
using namespace std;
int main() {
    cout << "Hello, World!\n";
    while (true) {}
}
```

Hello, World!

- ▶ Die C++ Standard-Streams (cin, cout,...) sind per default mit den C-Streams synchronisiert...
 - ▶ ”interaktive” Streams in C → line buffered!
 - ▶ damit leichte Interoperabilität zwischen C und C++
- ▶ → Verknüpfung lösen

”Hello, World!” – 7

```
// hello7.cpp
#include <iostream>
#include <chrono>
#include <thread>
using namespace std;
using namespace std::chrono_literals;
int main() {
    // to be called *before* any I/O!
    std::ios_base::sync_with_stdio(false);
    cout << "Hello, World!\n"; // no output at all!
    this_thread::sleep_for(2s); // chrono literal!
    // here the process is terminating normally
    // therefore the buffer will be flushed
    // hence the message will appear
}
```

→ bessere Performace, aber nicht (mehr) thread-safe!

Einfache Eingabe

```
// greetme.cpp
#include <iostream>
using namespace std;
int main() {
    // class "string" opposed to C-string literal
    // declared in <string>, but <iostream>...
    string name; // constructed onto stack
    cout << "Your name: ";
    // global object "std::cin" -> stdin
    // overloaded operator ">>"
    cin >> name; // flushing cout beforehand
    cout << "Pleased to meet you, " << name << "!"
}
```

Einfache Eingabe – 2

Your name: Maxi

Pleased to meet you, Maxi!

aber:

Einfache Eingabe – 2

```
Your name: Maxi  
Pleased to meet you, Maxi!
```

aber:

```
Your name:      Maxi Muster  
Pleased to meet you, Maxi!
```

Beachte:

- ▶ >> überliest führenden Whitespace!
- ▶ >> stoppt bei erstem „ungültigen“ Zeichen!

Einfache Eingabe – 3

```
// greetme2.cpp
#include <iostream>
using namespace std;
int main() {
    string first_name;
    string last_name;
    cout << "Your name: ";
    cin >> first_name;
    cin >> last_name;
    cout << "Pleased to meet you, " << first_name
         << " " << last_name << "!" << endl;
}
```

Your name: Maxi Muster
Pleased to meet you, Maxi Muster!

Einfache Eingabe – 4

Auch mit <Enter>:

Your name: Maxi

Muster

Pleased to meet you, Maxi Muster!

Jedoch was ist wenn man nicht zwei Namen hat...

- ▶ Mittels CTRL-C abbrechen
- ▶ Mittels CTRL-D (bzw. bei Windows mit CTRL-Z) den Eingabekanal schließen

Einfache Eingabe – 5

```
// greetme3.cpp
#include <iostream>
using namespace std;

int main() {
    string name;
    cout << "Your name: ";
    getline(cin, name); // reads a whole line...
    cout << "Pleased to meet you, " << name
         << "!" << endl;
}
```

Einfache Eingabe – 6

```
#include <iostream> // greetme4.cpp
using namespace std;

int main() {
    string name;  cout << "Your name: ";
    cin >> name;  getline(cin, name);
    cout << "Pleased to meet you, " << name
         << "!" << endl;  }
```

Einfache Eingabe – 6

```
#include <iostream> // greetme4.cpp
using namespace std;

int main() {
    string name;  cout << "Your name: ";
    cin >> name;  getline(cin, name);
    cout << "Pleased to meet you, " << name
         << "!" << endl; }
```

Your name: Maxi
Pleased to meet you, !

- ▶ → cin lässt \n in Stream und getline liest bis \n!
- ▶ >> ... formatierte Eingabe, getline ... unformatiert!
- ▶ d.h. Achtung bei Mischen von formatiert & unformatiert!

Einfache Eingabe – 7

```
#include <iostream> // greetme5.cpp
#include <iomanip> // quoted
using namespace std;

int main() {
    string name;  cout << "Your name: ";
    // quoted(string&, delim='"', escape='\\')
    // mainly used for processing CSV,...
    cin >> quoted(name);
    cout << "Pleased to meet you, " << name
         << "!" << endl;
    cout << "Quoted representation: "
         << quoted(name) << endl; }
```

Einfache Eingabe – 7

```
#include <iostream>    // greetme5.cpp
#include <iomanip>      // quoted
using namespace std;

int main() {
    string name;  cout << "Your name: ";
    // quoted(string&, delim='"', escape='\\')
    // mainly used for processing CSV,...
    cin >> quoted(name);
    cout << "Pleased to meet you, " << name
         << "!" << endl;
    cout << "Quoted representation: "
         << quoted(name) << endl; }
```

Your name: "Maxi \"Muster\""

Pleased to meet you, Maxi "Muster"!

Quoted representation: "Maxi \"Muster\""

Einfache Eingabe – 8

- ▶ Eingabe:
 - ▶ Kein Delimiter am Anfang → dann wie ohne quoted
 - ▶ Delimiter am Anfang → lesen bis Delimiter (inkl. Whitespace), dann wird Whitespace wieder überlesen. Anfangs- und End delimiter werden weggelassen.
 - ▶ Escape-Zeichen ignorieren, aber nachfolgendes Zeichen anhängen
- ▶ Ausgabe:
 - ▶ Delimiter wird am Anfang und Ende hinzugefügt
 - ▶ Escape-Zeichen bei Bedarf hinzufügen
- ▶ Signatur: `quoted(xxx, char delim='\"', char escape='\\')`
 - ▶ `xxx ≈ const char* || const string& || string_view || string&`
- ▶ Anwendung: CSV oder XML Daten!

Ganze Zahlen und if

```
// agetest.cpp
#include <iostream>
using namespace std;

int main() {
    int age;

    cout << "How old are you? ";
    cin >> age; // ">>" ... also for integers

    if (age < 18) {
        std::cout << "You are underage!" << endl;
    } else {
        std::cout << "You are full-aged!" << endl;
    }
}
```

Ganze Zahlen – 2

```
// add.cpp
#include <iostream>
using namespace std;
int main() {
    int num1; int num2;
    int res; // not initialized!

    cout << "The first number: "; cin >> num1;
    cout << "The second number: "; cin >> num2;
    cout << "The result: " << res << endl;
}
```

The first number: 1
The second number: 2
The result: 134514785

Ganze Zahlen – 3

- ▶ keine automatische Initialisierung von fundamentalen Typen
- ▶ Faustregel: Initialisiere immer explizit!

```
{ // scope starts
    int num1; // not initialized

    num1 = 0; // assignment!!
} // scope ends: num1 does not exist any more
{ // new scope starts
    int num1{0}; // initialized with 0
    int num2{}; // also initialized with 0
}
```

Eingabe im Detail

```
// add2.cpp
#include <iostream>
using namespace std;
int main() {
    int num1; int num2; int res;

    cout << "The first number: "; cin >> num1;
    cout << "The second number: "; cin >> num2;
    res = num1 + num2;
    cout << "The result: " << res << endl;
}
```

The first number: 1
The second number: 2a
The result: 3

Eingabe im Detail – 2

```
#include <iostream>           // add3.cpp
using namespace std;
int main() {
    int num1; int num2; int res;

    cout << "The first number: "; cin >> num1;
    cout << "The second number: "; cin >> num2;
    res = num1 + num2;
    cout << "The result: " << res << endl;

    string rest; cin >> rest;
    cout << "Remaining string: " << rest << endl;
}

// wie vorher, dann:
Remaining string: a
```

Eingabe im Detail – 3

- ▶ Starte das Programm mit ausschließlich gültigen Werten
 - ▶ es wird bei der Eingabe von `rest` hängen
 - ▶ der Eingabestrom muss mit `CTRL-D` geschlossen werden
- ▶ Jetzt, die erste Eingabe ungültig:

The first number: 1a

The second number: The result: 1

Remaining rest:

- ▶ `num1` erhält den Wert 1
- ▶ nächste Eingabe ("a") ist ungültig
 - ▶ → `num2` wird auf 0 gesetzt
 - ▶ → `cin` wechselt in Fehlermodus

Eingabe im Detail – 4

- ▶ Leerzeichen werden überlesen
 - ▶ space, tab,... (außer newline)
- ▶ Zeichen werden gelesen, sodass Eingabe gültig für den jeweiligen Datentyp ist
- ▶ Kann nicht gültig eingelesen werden, dann wird Variable auf "Nullwert" gesetzt und Eingabestrom wechselt in Fehlermodus
- ▶ "Ungültige" bleiben im Eingabestrom

Rechnen mit Zahlen

```
#include <iostream> // calc.cpp
using namespace std;
```

```
int main() {
    int num1; int num2; char op;

    cout << "First number: "; cin >> num1;
    cout << "Operator [+,-,*,/]: "; cin >> op;
    cout << "Second number: "; cin >> num2;

    if (op == '+') cout << num1 + num2 << endl;
    else if (op == '-') cout << num1-num2 << endl;
    else if (op == '*') cout << num1*num2 << endl;
    else if (op == '/') cout << num1/num2 << endl;
}
```

Rechnen mit Zahlen – 2

- ▶ Division durch 0:

First number: 1

Operator [+,-,*,/]: /

Second number: 0

...SIGFPE (Fliesskomma-Ausnahmefehler)...

→ Abfrage

- ▶ Division mit Rest:

First number: 7

Operator [+,-,*,/]: /

Second number: 2

3

→ double

Rechnen mit Zahlen – 3

```
#include <iostream> // inf.cpp  
using namespace std;
```

```
int main() {  
    // 1 ... "int"  
    // 0.0 ... "double"  
    cout << 1 / 0.0 << endl;  
}
```

inf

- ▶ inf...infinity
- ▶ -inf...z.B. bei: $-1 / 0.0$

switch

```
#include <iostream> // calc2.cpp
using namespace std;
int main() {
    double num1; double num2; char op;
    cout << "First number: "; cin >> num1;
    cout << "Operator [+,-,*,/]: "; cin >> op;
    cout << "Second number: "; cin >> num2;

    switch (op) {
        case '+':
            cout << num1 + num2;
            break;
        case '-': cout << num1 - num2; break;
        case '*': cout << num1 * num2; break;
        case '/': cout << num1 / num2; break;
    } }
```

switch - 2

```
#include <iostream> // calc3.cpp
using namespace std;
int main() {
    double num1; double num2; char op;
    cout << "First number: "; cin >> num1;
    cout << "Operator [+,-,*,/]: "; cin >> op;
    cout << "Second number: "; cin >> num2;
    switch (op) {
        case '+':
            cout << num1 + num2;
            break; // otherwise continue with '-'
        case '-': cout << num1 - num2; break;
        case '*': cout << num1 * num2; break;
        case '/': cout << num1 / num2; break;
        default: cout << "invalid operator";
    } }
```

while & fehlerhafte Eingaben

```
#include <iostream> // badinput.cpp
using namespace std;
int main() {
    char proceed{'y'};

    while (proceed == 'y') {
        double num;
        cout << "Number: ";
        cin >> num;

        cout << "Proceed? [y/n] ";
        cin >> proceed;
        cout << endl;
    }
}
```

funktioniert bei gültiger Eingabe, aber...

while & fehlerhafte Eingaben – 2

Number: a

Number: Proceed? [y/n]

Number: Proceed? [y/n]

Number: Proceed? [y/n]

Number: Proceed? [y/n]

Number: Proceed? [y/n]

...

while & fehlerhafte Eingaben – 2

Number: a

Number: Proceed? [y/n]

Number: Proceed? [y/n]

Number: Proceed? [y/n]

Number: Proceed? [y/n]

Number: Proceed? [y/n]

...

cin in Fehlerzustand!

while & fehlerhafte Eingaben – 3

```
#include <iostream> // badinput2.cpp
using namespace std;
int main() {
    char proceed{'y'};

    while (proceed == 'y') {
        double num; cout << "Number: ";
        if (!(cin >> num)) {
            // cerr: will be flushed automatically
            cerr << "ungültig: Abbruch\n";
            return 1;
        }
        cout << "Proceed? [y/n] ";
        cin >> proceed; cout << endl;
    }
}
```

do - while

```
#include <iostream> // badinput3.cpp
using namespace std;
int main() {
    char proceed; // init not necessary
    do {
        double num;
        cout << "Number: ";
        if (!(cin >> num)) {
            cerr << "ungültig: Abbruch" << endl;
            return 1;
        }
        cout << "Proceed? [y/n] ";
        cin >> proceed; cout << endl;
    } while (proceed == 'y'); // semicolon!
}
```

do - while

```
#include <iostream> // badinput3.cpp
using namespace std;
int main() {
    char proceed; // init not necessary
    do {
        double num;
        cout << "Number: ";
        if (!(cin >> num)) {
            cerr << "ungültig: Abbruch" << endl;
            return 1;
        }
        cout << "Proceed? [y/n] ";
        cin >> proceed; cout << endl;
    } while (proceed == 'y'); // semicolon!
}
```

Funktioniert, aber terminiert bei ungültiger Zahleneingabe!

do - while - 2

```
#include <iostream>    // badinput4.cpp
using namespace std;
int main() {
    char proceed;
    do {
        double num;
        cout << "Number: ";
        if (!(cin >> num)) {
            cerr << "ungültig: Abbruch" << endl;
            continue;
        }
        cout << "Proceed? [y/n] ";
        cin >> proceed;    cout << endl;
    } while (proceed == 'y');
}
```

Fehlerbehandlung bei Eingabe

Testen!

Number: a

ungültig: Abbruch

Was ist das Problem?

Fehlerbehandlung bei Eingabe

Testen!

Number: a

ungültig: Abbruch

Was ist das Problem?

proceed ist nicht initialisiert!

Fehlerbehandlung bei Eingabe – 2

```
#include <iostream> // badinput5.cpp
using namespace std;
int main() {
    char proceed{'y'};
    do {
        double num; cout << "Number: ";
        if (!(cin >> num)) {
            cerr << "ungültig: Abbruch" << endl;
            continue;
        }
        cout << "Proceed? [y/n] ";
        cin >> proceed; cout << endl;
    } while (proceed == 'y');
}
```

Fehlerbehandlung bei Eingabe – 2

```
#include <iostream> // badinput5.cpp
using namespace std;
int main() {
    char proceed{'y'};
    do {
        double num;  cout << "Number: ";
        if (!(cin >> num)) {
            cerr << "ungültig: Abbruch" << endl;
            continue;
        }
        cout << "Proceed? [y/n] ";
        cin >> proceed;  cout << endl;
    } while (proceed == 'y');
}
```

Wieder eine Endlosschleife, da `cin` noch immer in Fehlerzustand...

Fehlerbehandlung bei Eingabe – 3

```
#include <iostream> // badinput6.cpp
using namespace std;
int main() {
    char proceed{'y'};
    do {
        double num;  cout << "Number: ";
        if (!(cin >> num)) {
            cerr << "ungültig: Abbruch" << endl;
            cin.clear(); // now not in error mode
            continue;
        }
        cout << "Proceed? [y/n] ";
        cin >> proceed;  cout << endl;
    } while (proceed == 'y');
}
```

Fehlerbehandlung bei Eingabe – 3

```
#include <iostream> // badinput6.cpp
using namespace std;
int main() {
    char proceed{'y'};
    do {
        double num;  cout << "Number: ";
        if (!(cin >> num)) {
            cerr << "ungültig: Abbruch" << endl;
            cin.clear(); // now not in error mode
            continue;
        }
        cout << "Proceed? [y/n] ";
        cin >> proceed;  cout << endl;
    } while (proceed == 'y');
}
```

Wieder eine Endlosschleife, da "a" ungelesen in Eingabe!

Fehlerbehandlung bei Eingabe – 4

```
#include <iostream> // badinput7.cpp
#include <limits>
using namespace std;
int main() { char proceed{'y'};
    do {
        double num; cout << "Number: ";
        if (!(cin >> num)) {
            cerr << "ungültig: Abbruch" << endl;
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(),
                      '\n');
            continue;
        }
        cout << "Proceed? [y/n] ";
        cin >> proceed; cout << endl;
    } while (proceed == 'y'); }
```

Fehlerbehandlung bei Eingabe – 5

Testen!

Number: a

ungültig: Abbruch

Number: b

ungültig: Abbruch

Number: 1

Proceed? [y/n] a

ok, Rest ist nur mehr Logik... ; -)

Fehlerbehandlung bei Eingabe – 6

- ▶ Fehler → Eingabestrom wechselt in Fehlerzustand
- ▶ >> liefert wieder den Eingabestrom zurück
- ▶ ! liefert die Negation des Wahrheitswertes
- ▶ Eingabestrom ist *kein* Wahrheitswert → implizite Konvertierung!
- ▶ Eingabestrom im Fehlerzustand → ! liefert true
- ▶ `clear()` löscht `failbit`
- ▶ `ignore()` verwirft Zeichen