LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

mobile and distributed systems group

Grundlagen C und C++

Einheit 04: Weitere Grundlagen in C++

Lorenz Schauer
Lehrstuhl für Mobile und Verteilte Systeme

Heutige Agenda

Teil 1: Weitere Grundlagen von C++

- Das `assert` Statement
- File Input und Output
- Pointer & Referenzvariablen

Teil 2: Zur Objektorientierung

- Einführung in Klassen & Objekten

Übungen

- Dateien lesen und schreiben
- Swap
- Personenverwaltung

Lernziele

- Weitere Grundlagen zu C++ kennenlernen
- Einführung in die Objektorientierung mit C++
- C++ weiter einüben

Das `assert` Statement

Mittels assert Statement lassen sich Eingaben oder aktuelle Variablenbelegungen auf eine Bedingung hin überprüfen

- Erleichtert das spätere Debugging
- V.a. bei größeren Programmen essentiell, um Fehler zu finden!
  - Bsp.: Teilen durch 0, oder Wurzel einer negativen Zahl

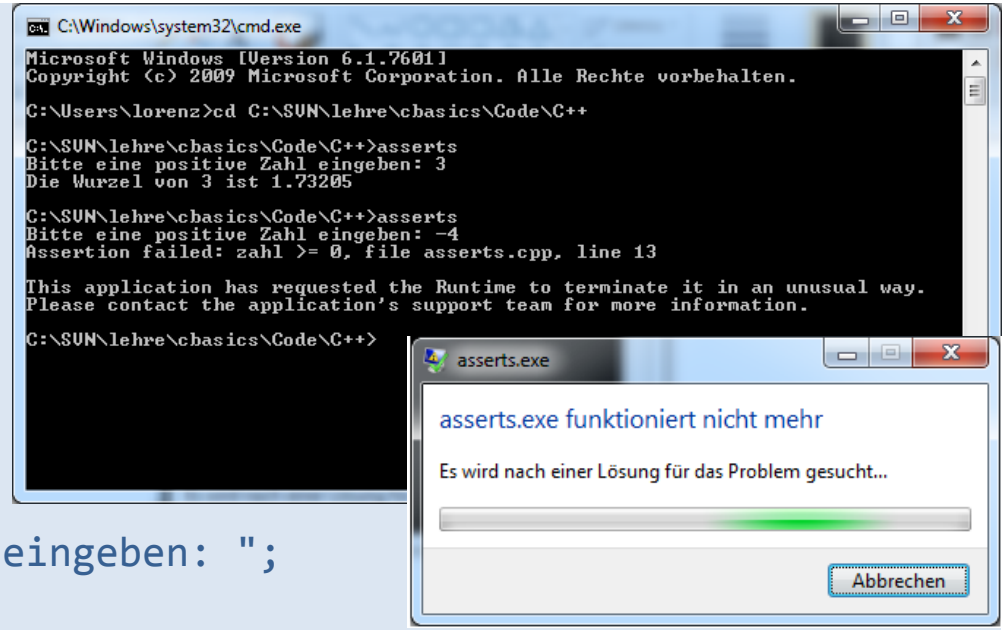
Allgemeine Verwendung:

```
// Header-File wird benötigt!
#include <cassert>

//Im Progamm:

assert(Bedingung);
```

Falls Bedingung verletzt, wird die Ausführung an der Stelle von assert beendet und eine Fehleranzeige auf der Konsole ausgegeben

Beispiel für assert Statement in einem einfachen Programm

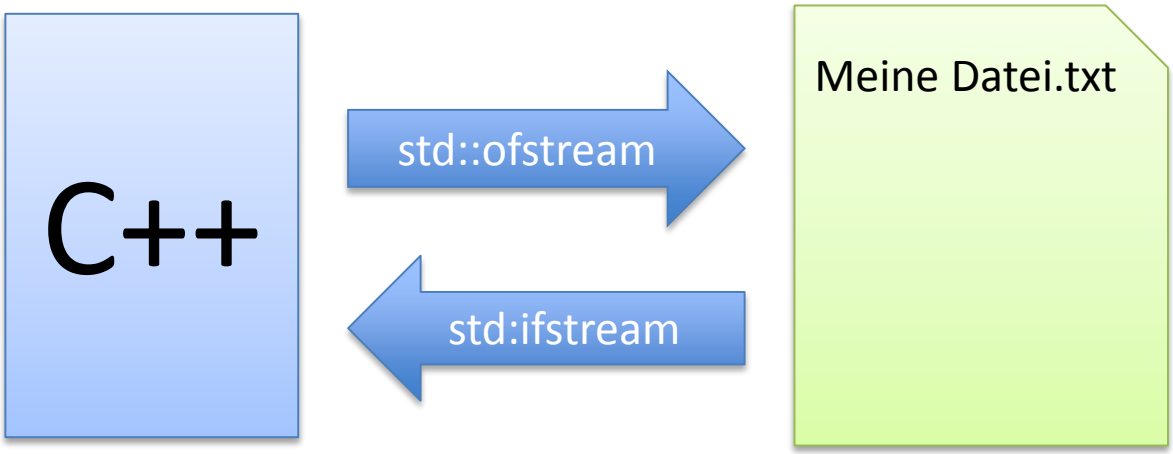

```
#include <iostream>
#include <cmath>

#include <cassert>

using namespace std;

int main(){

   int zahl;
   cout<< "Bitte eine positive Zahl eingeben: ";
   cin >> zahl;
   assert(zahl >= 0);
   cout << "Die Wurzel von "<<zahl<<" ist "<<sqrt(zahl)<<"\n";

   return 0;
}
```



File Input/Output

Wir können neben den direkten Nutzereingaben/-Ausgaben über die Konsole natürlich auch Werte über Dateien einlesen, bzw. in Dateien Werte schreiben.

- Dafür benötigen wir die header-File: `fstream`
- Und erzeugen uns einen Input- bzw. Outputstream:
  - `std::ofstream` -> Um auf Dateien zu schreiben
  - `std::ifstream` -> Um von Dateien zu lesen



File Input/Output

In Dateien Schreiben

```
#include <fstream>

//Variable für den outputstream erzeugen:
std::ofstream mein_output_stream(„der/pfad/zur/datei.txt“);

//Outputstream nutzen, um auf Datei zu schreiben:
mein_output_stream << „Hallo hallo\n“;

//Nicht vergessen! Outputstream wieder schließen:
mein_output_stream.close();
```

Aus Dateien lesen

```
#include <fstream>

//Variable für den inputstream erzeugen:
std::ifstream mein_input_stream(„der/pfad/zur/datei.txt“);

//Inputstream nutzen, um von Datei zu lesen:
mein_input_stream >> ankommenden_Daten_pro_splate >> und_zeile;

//Nicht vergessen! Inputstream wieder schließen:
mein_input_stream.close();
```

Beispiel: Daten in eine Datei schreiben


```
#include <iostream>
#include <cassert>
#include <fstream>

int main(){

   double x[4] = {4.0, 3.4, 2.2, 7.7};
   double y[4] = {1.2, 9.3, 4.5, 5.6};

   std::ofstream mein_output_stream("meineDatei.txt");

   // Oder falls Daten angehängt werden sollen:
   std::ofstream mein_output_stream("meineDatei.txt“, std::ios::app);

   assert(mein_output_stream.is_open());

   for(int i=0; i<4;i++){
      mein_output_stream << x[i] << " " << y[i] <<"\n";
   }
   mein_output_stream.close();
   return 0;
}
```

Beispiel: Daten aus einer Datei lesen


```
#include <iostream>
#include <cassert>
#include <fstream>

int main(){

   double xx[4];
   double yy[4];

   std::ifstream mein_input_stream("meineDatei.txt");
   assert(mein_input_stream.is_open());

   for(int i=0; i<4;i++){
      mein_input_stream >> xx[i] >> yy[i];
   }

   mein_input_stream.close();
   return 0;
}
```

Programmieraufgabe zu Datei Input/Output

Legen Sie sich in Ihrem Code-Verzeichnis eine normale Textdatei an und speichern Sie folgenden Inhalt:

```
4.5  10.2
3.4  2.9
1.3  5.2
13.4 19.2
34.2 58.7
```

Schreiben Sie nun ein Programm, welches den Inhalt der Datei zeilenweise einliest, und jeweils die größere der beiden Zahlen in einer weiterer Textdatei ausgibt.

Überprüfen Sie die Ausgabe in der neuen Datei.

Pointer und Referenzvariablen

Normalerweise können Funktionen nur einen festen Rückgabewert zurückliefern

Bsp.:

- `double getMean(double sum, int length)`
  `{return sum/length;}`
  -> liefert einen Double-Wert zurück

Aber: Mittels Pointer oder Referenzvariablen können wir Variablen außerhalb der Funktion verändern und dadurch (effektiv) mehrere „Rückgabewerte“ erzeugen.

```
// Beispiel mit Pointer:

void calcRealAndImaginary(double r, double theta, double* pReal, double*
pImaginary)
{

   *pReal = r*cos(theta);
   *pImaginary = r*sin(theta);

}
```

Pointer und Referenzvariablen


```
// Komplettes Beispiel mit Pointer:

#include <iostream>
#include <cmath>

void calcRealAndImaginray(double,double,double*,double*);

int main(){

   double r = 4.4;
   double theta = 1.7;
   double x,y;

   calcRealAndImaginray(r,theta,&x,&y);

   std::cout << "Real-Teil: "<<x<<"\n";
   std::cout << "Imaginaer-Teil: "<<y<<"\n";

   return 0;
}

void calcRealAndImaginray(double r, double theta, double* pReal, double*
pImaginary){
   *pReal = r*cos(theta);
   *pImaginary = r*sin(theta);
}
```

Pointer und Referenzvariablen

Eine Alternative zu Pointers stellen **Referenzvariablen** dar:

- Sind stellvertretende Variablen mit anderem Namen innerhalb einer Funktion
- Jede Änderung dieser Variablen innerhalb der Funktion bewirkt auch eine Änderung der „echten“ Variablen außerhalb der Funktion
- Werden mittels & im Methodenkopf gekennzeichnet
- Sind einfacher zu nutzen als Pointer

```
// Beispiel der Funktion von vorher nun mit Referenzvariablen:

void calcRealAndImaginray(double r, double theta, double& real, double&
imaginary){

   real = r*cos(theta);
   imaginary = r*sin(theta);

}
```

Pointer und Referenzvariablen


```
// Komplettes Beispiel mit Referenzvariablen:

#include <iostream>
#include <cmath>

void calcRealAndImaginray(double,double,double&,double&);

int main(){

   double r = 4.4;
   double theta = 1.7;
   double x,y;

   calcRealAndImaginray(r,theta,x,y);

   std::cout << "Real-Teil: "<<x<<"\n";
   std::cout << "Imaginaer-Teil: "<<y<<"\n";

   return 0;
}

void calcRealAndImaginray(double r, double theta, double& real, double&
imaginary){
   real = r*cos(theta);
   imaginary = r*sin(theta);
}
```

Programmieraufgabe zu Methoden

Schreiben Sie ein Programm, welches eine Funktion `swap` aufruft, welche die Werte (in `double`) der übergebenen Parameter vertauscht.

Implementieren Sie diese Funktion auf 3 Arten:

- Call-by-value
- Call-by-Reference
  - Mit Pointer
  - Mit Referenzvariablen

Testen Sie Ihre Funktion mit verschiedenen Parametern!

Primäre Motivation für C++

- Add Classes to C

Prinzipien der objektorientierten Programmierung:

- **Objekte** erzeugen, die ihrerseits Attribute (Daten) und Methoden (Funktionen) bereitstellen
- **Klassen**, als Bauplan von Objekten
- **Abstraktion**: Nur die wichtigen Informationen werden zur Verfügung gestellt. Die Details bleiben verborgen.
- **Kapselung**: Daten und Funktionen sind gekapselt in einem Objekt.
  - Attribute sollen nicht direkt, sondern nur indirekt verändert werden können.
- **Vererbung**: Es gibt Unterklassen, welche weitere spezifischere Methoden und Attribute bereitstellen können
- **Polymorphismus**: Operatoren oder Funktionen lassen sich auf verschiedene Arten einsetzen
- **Überladen**: Ist ein Teilbereich des Polymorphismus und kann auf existierende Operatoren bzw. Funktionen angewendet werden, so dass diese mit neuen Daten(-typen) umgehen können.

Eine C++ Klasse wird mit dem Schlüsselwort `class` eingeleitet.

```
class Book
{
   public:
      std::string author, title, publisher;
      int price;
      int yearOfPublication;
};
```

Auch möglich:
`private`, oder `protected`

Wichtig:
`;` beachten!

Objekt erzeugen und Datenzugriff:

```
int main(){

   // Objekt erzeugen:
   Book mein_buch;

   // Datenzugriff (schreibend):
   mein_buch.titel =“Der Name der Rose“;

   // Datenzugriff lesend:
   std::cout << „Der Buchtitel lautet: “ << mein_buch.titel;
}
```

Wir können Klassendefinitionen und Programm in eine Datei schreiben:

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

class Person{

   public:
      std::string name, vorname;
      int alter;
};

int main(){

   Person pers1;

   pers1.name = "Peter";
   pers1.vorname = "Hans";
   pers1.alter = 35;

   std::cout << pers1.vorname << " "<< pers1.name << " ist "<<pers1.alter<< "
Jahre alt.\n";

   return 0;
}
```

Das wird spätestens bei größeren Programmen sehr umständlich und fehleranfällig!
Daher brauchen wir eine Struktur => Header-Files (Inhalt der nächsten Stunde)

Programmieraufgabe zu Klassen

Schreiben Sie ein Programm in C++ zur Verwaltung von Studenten.

- Ein Student soll dabei als Objekt einer Klasse implementiert sein, die folgenden Eigenschaften hat:
  - Name, Vorname
  - Alter, Matrikelnummer
  - Hauptfach
- Ein direkter Zugriff auf diese Eigenschaften von Außen soll vermieden werden. Schreiben Sie daher noch passende Getter- und Setter Methoden
- Verwenden Sie Ihre Klasse und erzeugen sie 2 Studenten:
  - Hans Maier, 22, Matrikelnummer: 123456, Hauptfach: Geschichte
  - Julia Superman, 24, Matrikelnummer: 111213, Hauptfach: Biologie

- Lassen Sie sich nun die Vornamen und das Alter der Studenten anzeigen!