Informatik II

Oliver Jack

Fachhochschule Jena
Fachbereich Elektrotechnik und Informationstechnik

Sommersemester 2010

Inhalt

Vorlesung 2. OOP und die Programmiersprache C++

Lernziele dieser Vorlesung

Programmierparadigmen und OOP

- Programmierparadigmen
- Konzepte ojektorientierter Programmierung

Programmgestalt und Sprachkonstrukte

- Ein Programmierproblem
- Implementierung der Problemlösung
- Programmstruktur und Programmelemente

Zusammenfassung

Lernziele

- Prinzipien, nach denen Programmierung erfolgen kann
- Prinzip der Objektorientierung
- Erste Schritte in der Programmiersprache C++
- Lösung eines Programmierproblems in C++

Programmierparadigmen

- Programmiersprachen sind Ergebnis von Denkweisen und Erfahrungen über den Programmierprozess
- Durch Generalisierung entstehen aus Denkprozessen Modellmethoden für den Entwurf und die Implementierung von Programmen (Paradigma)
- Paradigmen beziehen sich auf Programmentwurf, auf Verwendung von Abstraktion und Strukturierung

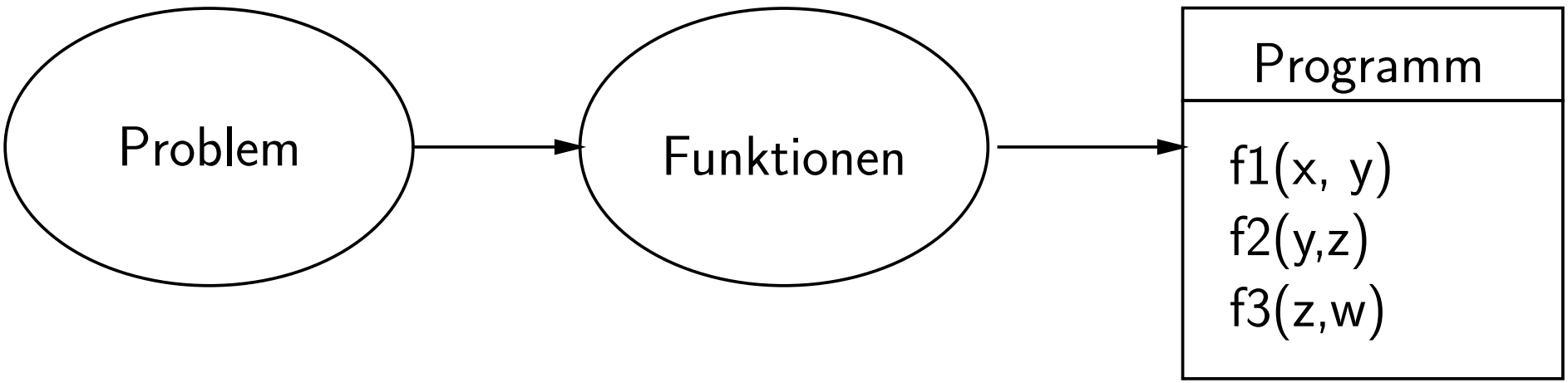
- Programmiersprache unterstützt ein Paradigma, wenn ihre Leistungsmerkmale die Anwendung der Methode erleichtern
- Gemeinsamkeit von Programmierparadigmen: auf Abstraktion beruhender Entwurf, der mit mit den Elementen des Programmierproblems korrespondiert

Programmierparadigmen (Forts.)

Es gibt nicht das “beste Paradigma” !

- Jedes Paradigma hat seine Berechtigung und Anwendungsgebiete, in denen es herausragende Eigenschaften (angemessen einfach, sicher und effizient) zeigt.
- Die Qualität eines Paradigmas zeigt sich nicht so sehr in kleinen Beispielen von wenigen Zeilen Code, sondern in großen Projekten mit vielen Entwicklern und einigen hunderttausend Zeilen Code
- Wesentlich ist, mit allen Paradigmen (Sprachmitteln, s.o.!) vertraut zu sein und diese je nach Anforderung zielstrebig einsetzen zu können

Prozedurale Programmierung



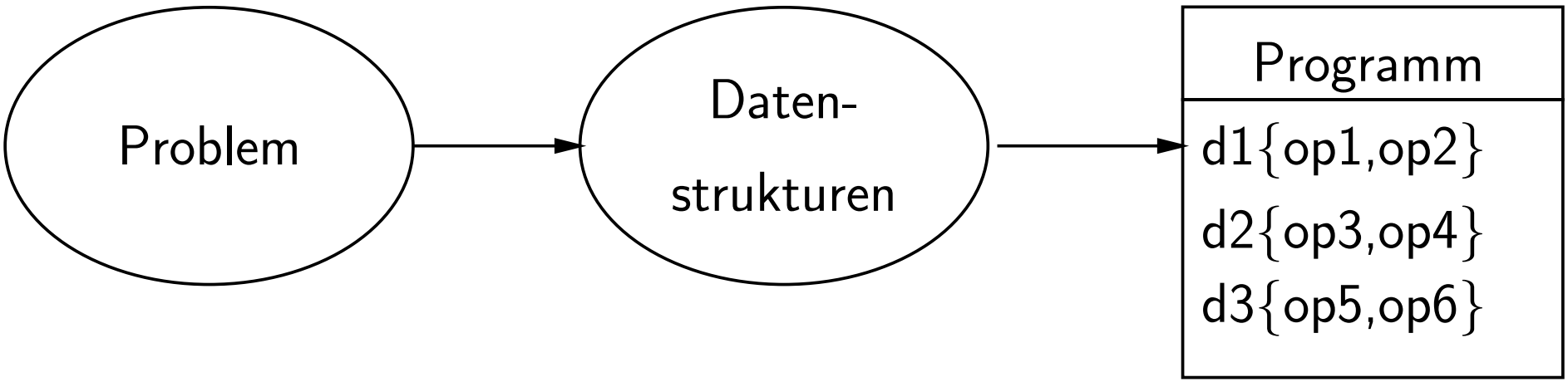
Prozedurale Programmierung (Forts.)

- Weitverbreitetes Modell
- Programm besteht aus einer Reihe von Funktionen
- Entwurf, Strukturierung und Implementierung erfolgt nach funktionaler Zerlegung

Methode

1. Identifizierung von Funktionen zur Lösung des Programmierproblems (abstrakte Operatoren)
2. Aufteilung von Funktionen in Moduln (Strukturierung)

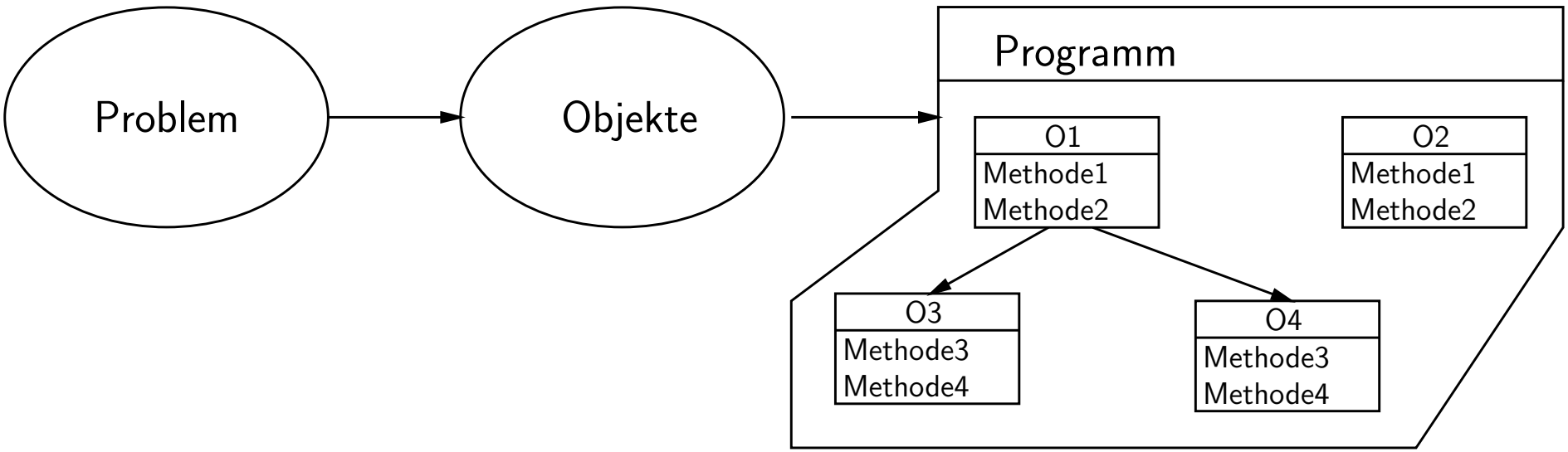
Datenabstraktion



Datenabstraktion (Forts.)

- Daten, nicht Funktionen stehen im Mittelpunkt des Interesses
- Datenstruktur ist definiert durch die darauf ausführbaren Operationen, nicht durch die Struktur der Implementierung
- Umgebung einer Datenstruktur durch einen abstrakten Datentyp (Datenkapselung)
- Zugriff auf die Datenstruktur durch Operationen (Methoden), die Teil des Datentyps sind
- Datenabstraktion ergänzt die Sichtweise prozeduraler Programmierung

Objektorientierte Programmierung (OOP)



Objektorientierte Programmierung (OOP) (Forts.)

- Modell der Programmerstellung als Reihe abstrakter Datentypinstanzen
- Typen, die im Programmierproblem die Objekte darstellen, stehen im Mittelpunkt der Betrachtung

- Operationen in den Objekttypen sind abstrakte Operatoren zur Lösung des Problems
- Objekttyp dient als wiederverwendbares Modul zur Lösung gleicher und ähnlicher Probleme

Objektorientierte Programmierung (OOP) (Forts.)

- Starke Kopplung von Daten und Operationen (Verhalten)
- Berechnung wird als Simulation von Verhalten betrachtet
- Simuliert werden Objekte durch Abstraktion

- Erste objektorientierte Programmiersprache war eine Simulationsprache (SIMULA67, 1967)
- Gemeinsamkeit aller objektorientierten Programmiersprachen
  - Datenkapselung
  - Polymorphie
  - Vererbung

Datenkapselung

Beispiel Kartenspiel

- Spielfarben: ♣, ♡, ♢, ♠
- Dies sind Typen, Teilaspekt des Kartenspiels
- Repräsentation im Programm z. B. durch Zahlen

♣	=	0	♢	=	2
♡	=	1	♠	=	3

Datenkapselung (Forts.)

Beispiel Kartenspiel

- Repräsentation soll vor der Außenwelt (Benutzer) verborgen werden und die Benutzung nicht beeinflussen
- Eigenschaften von Daten und Operationen eines Objekts
  - gekapselt (private): interne Repräsentation
  - offen (public): Schnittstelle (Interface) zu gekapselten Daten und Operationen

Polymorphie

Polymorphie (griechisch): Vielgestaltigkeit, Verschiedengestaltigkeit

- Erlaubt Konstruktion allgemeiner Schnittstellen und Operatoren
- Lokalisiert die Verantwortlichkeit für das Verhalten (Bindung der Operation an einen Datentyp)

Polymorphie (Forts.)

Beispiel arithmetische Operationen

ganze, reelle, komplexe Zahlen

Beispiel Graphiksystem

- Darstellung verschiedener geometrischer Objekte auf dem Bildschirm
- Zeichnen der Objekte kann intern verschieden sein (Kreise, Polygone, etc.)
- Allgemeine Operation (`draw`) kann objektunabhängig benutzt werden

Vererbung

- Ermöglicht hierarchische Klassifizierung von Objekten
- Methode zur Reduzierung der Komplexität
- Spezialisierung von Objekten: Vererbung aller Eigenschaften der Eltern, Definition von Daten und Operationen, die zu den Eltern abgrenzen

- Fördert Wiederverwendung von Programm-Code: aus genereller Beschreibung (Klasse) durch Hierarchisierung zu Spezialisierung (Objekt) für das aktuelle Programmierproblem

Vererbung (Forts.)

Beispiel Obst

- Ein Apfel ist eine Frucht
- Eine Frucht ist Nahrung
- Früchte besitzen alle Eigenschaften von Nahrung
- Äpfel besitzen alle Eigenschaften von Früchten

Vorteile von OOP

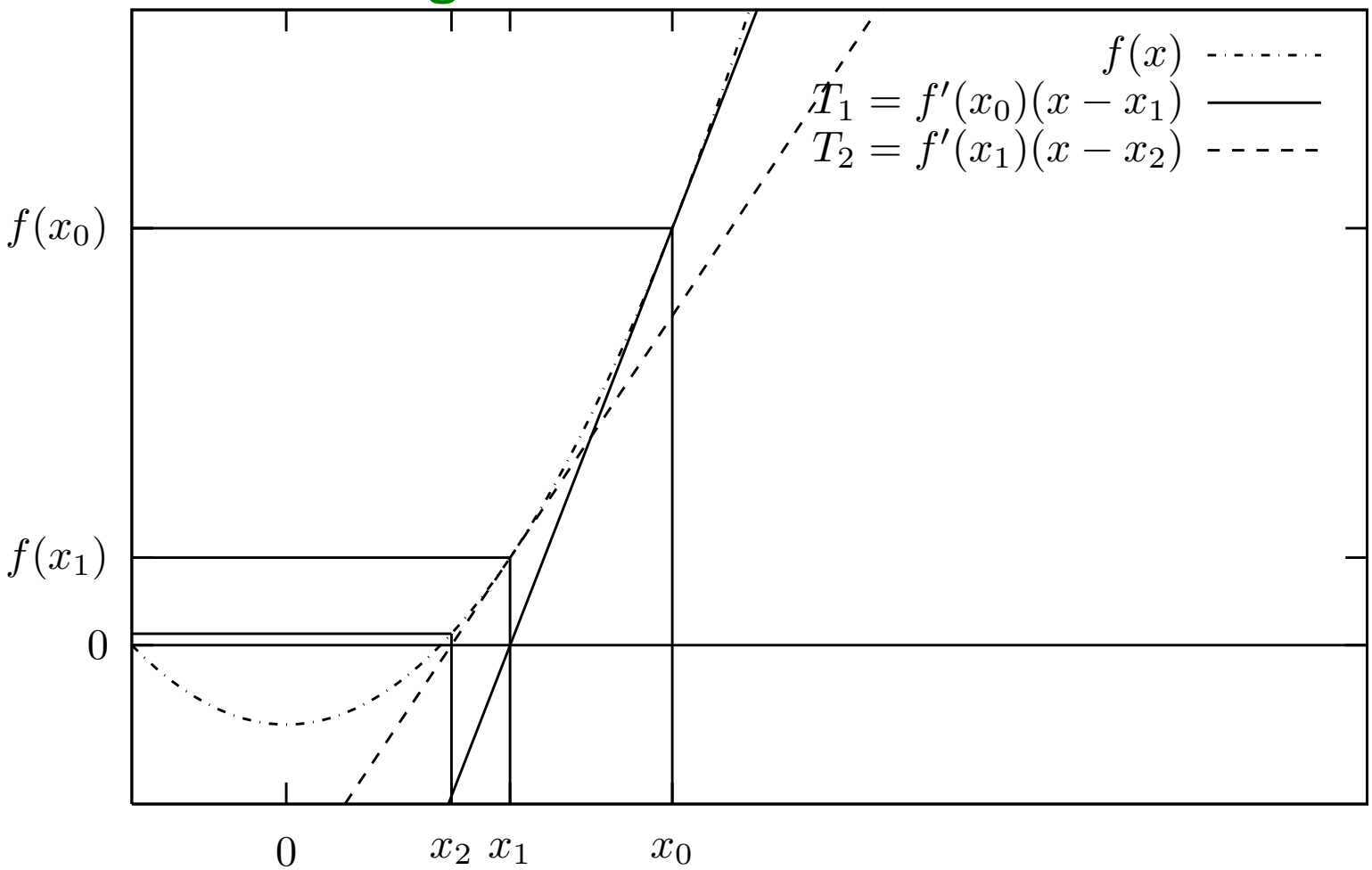
- Gut anwendbar auf sehr große Programmierprojekte
- Datenkapselung erlaubt gute Aufteilbarkeit auf mehrere Entwickler und Programmierer
- Polymorphie erlaubt leichte intuitive Benutzung von Operationen (Methoden) und reduziert die Komplexität für Entwickler und Programmierer
- Vererbung erlaubt gute Strukturierung des Programmierproblems durch Hierarchisierung von Typen und Objekten
- Objektorientierte Programme sind oft robuster und besser zu warten und zu ändern als prozedurale Programme

Historie von C++

- Die Programmiersprache C wurde 1970 von B. Kernighan und D. Ritchie zur Implementation des Betriebssystems UNIX entworfen (rein prozedural)
- C ist seit 1989 im ANSI Standard X3.159-1989, ISO/IEC Standard 9899:1990 normiert, aktueller Standard ist normiert ISO/IEC 9899:1999.
- C++ ist von B. Stroustrup 1980 bei den Bell Labs (Bell AT&T) als Erweiterung von C entworfen worden.
- Wichtigstes neues Sprachkonzept: Klassen (ursprünglich hieß die Sprache C with Classes)
- C++ ist eine Obermenge von C, jedes ANSI-C-Programm kann mit einem C++-Compiler übersetzt werden
- C++ ist seit 1998 im ISO/IEC Standard 14882:1998 normiert, aktueller Standard ist ISO/IEC 14882:2003.

Ein einfaches Programmierproblem

Wurzelberechnung mit dem Newton-Verfahren



Ein einfaches Programmierproblem (Forts.)

Schnittpunkt der Tangente T_1 mit der x-Achse

$$\begin{aligned} f'(x_0) &= \frac{f(x_0)}{x_0 - x_1} \\ \iff x_1 &= x_0 - \frac{f(x_0)}{f'(x_0)} \end{aligned}$$

Iterationsverfahren

1. Starte mit Schätzwert x_0
2. Berechne

$$x_{n+1} := x_n - \frac{f(x_n)}{f'(x_n)}$$

bis $|x_{n+1} - x_n| \leq \varepsilon$, wobei ε Genauigkeit ist

Spezialfall $f(x) = x^2 - c$ Berechnung von $\sqrt{c}$

$$x_{n+1} = x_n - \frac{x_n^2 - c}{2x_n}$$

Ein einfaches C++-Programm

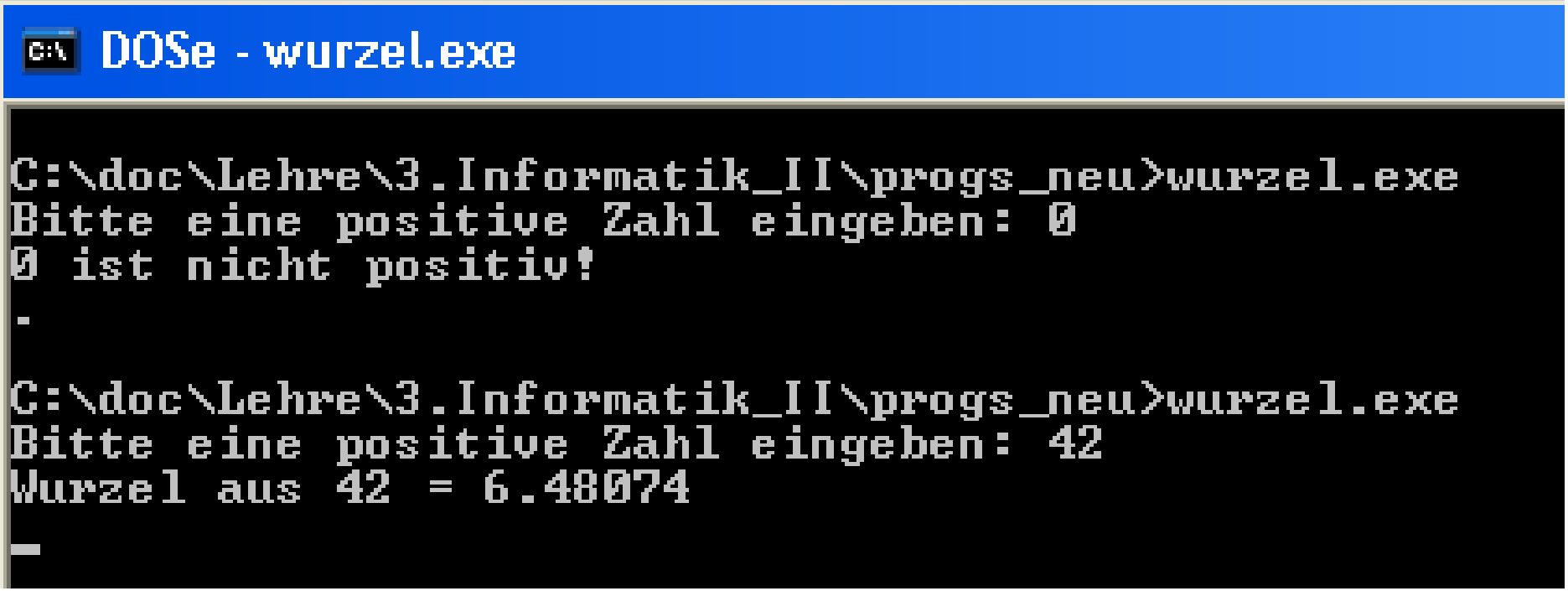
```
/* Wurzelberechnung
 * Eingabe: positive reelle Zahl x
 * Ausgabe: Wurzel aus x
 */
#include <iostream>
using namespace std
// Konstanten und Variablen
const float genauigkeit (1e-4);
float wert, wurzel, altwurzel, fehler;
int main ()
{
// Eingabe
  cout << "Bitte eine positive Zahl eingeben: ";
  cin >> wert;
  if (wert <= 0) {
    cout <<  wert << " ist nicht positiv!" << endl;
    return (1); // Rueckgabewert, 1 = falsche Eingabe
  }
```

Ein einfaches C++-Programm (Forts.)

```
// Algorithmus nach Newton
  wurzel = wert;  // Erster Schaetzwert
  do {
    altwurzel = wurzel;
    wurzel = wurzel
     - (wurzel * wurzel - wert) / (2 * wurzel);
    if (altwurzel <= wurzel) {
      fehler = wurzel - altwurzel;
    }
    else {
      fehler = altwurzel - wurzel;
    }
  } while (fehler >= genauigkeit);
// Ausgabe
  cout << "Wurzel␣aus␣" << wert << "␣=␣" << wurzel << endl;
  return (0); // Rueckgabewert, 0 = alles ok!
}
```

Ein einfaches C++-Programm (Forts.)

Programmlauf



Elemente eines C++-Programms

Ein einfaches Programm, generelle Struktur

1. Kommentar
2. Präprozessoranweisungen
3. Namensraum-Direktiven
4. Konstantendeklarationen
5. Variablendeklarationen
6. Eingabeanweisungen
7. Programmanweisungen
8. Ausgabeanweisungen
9. Rückgabewert

```
/*...*/
#include <iostream>
using namespace std;
const float genauigkeit (1e-4);
float wert, wurzel, ...
int main ()
{ ...
  cin >> ...;
  ...
  wurzel = wert;
// Erster Schaetzwert
  do {
    ...
  } while (fehler >= genauigkeit);
  ...
  cout << ...;
  return (0);
}
```

Kommentare

- Kommentare erläutern, was ein Programm oder Programmteil bewirkt
- Programmanweisungen beschreiben, wie etwas bewirkt wird
- Kommentarblock, mehrzeilig
  Benutzung zu längeren Erläuterungen, insbesondere zu Beginn des Programms
- Einzeiliger Kommentar
  Benutzung zu kurzen Erläuterungen, z. B. zur Beschreibung von Variablen

Kommentare (Forts.)

- Kommentarblock beginnt mit "/*" und endet mit "*/".

```
/* Wurzelberechnung
 * Eingabe: positive reelle Zahl x
 * Ausgabe: Wurzel aus x
 */
```

- Einzeiliger Kommentar beginnt mit "//" und endet mit dem Zeilenende.

```
// Konstanten und Variablen
// Eingabe
// Algorithmus nach Newton
wurzel = wert; // Erster Schaetzwert
// Ausgabe
// Rueckgabewert ...
```

Präprozessoranweisungen

- Präprozessor ist dem Compiler vorgeschaltet
- Führt Ergänzungen und Veränderungen des Quellprogramms durch
- Präprozessoranweisungen beginnen mit "#"
- Zunächst wichtig: Einfügen von Programmtext (include)
  `#include <Datei>`
- Eingebunden werden meist Header-Dateien, sie enthalten Daten- und Funktions-, bzw. Klassendeklarationen

Präprozessoranweisungen (Forts.)

- Zur Benutzung von Ein- und Ausgabeoperatoren wird die Bibliothek `iostream` in das Quellprogramm eingebunden

```
#include <iostream>
```

- Ein Dateiname in spitzen Klammern (<>) bezeichnet eine Systemdatei (Suche in den Systemverzeichnissen)
- Bei Bibliotheks-Header-Dateien entfällt im Unterschied zu C die Angabe `".h"`
- Ein Dateiname in Anführungszeichen ("") wird im aktuellen Verzeichnis gesucht

Datendeklaration

- Konstanten und Variablen müssen vor Ihrer Benutzung deklariert werden
- Sie durch Namen und Typ gekennzeichnet
- Datendeklarationen werden wie alle Programmkonstrukte mit Ausnahme von Kommentaren und Präprozessoranweisungen durch ein Semikolon ";" abgeschlossen

- Konstante, eine Fließkommazahl mit Wert $0.0001 = 1 \cdot 10^{-4}$

```
const float genauigkeit (1e-4);
```

- Variablen, vier Fließkommazahlen

```
float wert, wurzel, altwurzel, fehler;
```

Datendeklaration (Forts.)

Initialisierung

- Eine Konstante wird bei der Deklaration naturgemäß mit einem Wert belegt.
- `const float genauigkeit (1e-4);`
- In C++ geschieht dies durch Angabe des Wertes in Klammern (*wert*) (Unterschied zu C)
- In C: `const float genauigkeit = 1e-4;` (in C++ auch möglich).
- Gleiches gilt für Initialisierung von Variablen.

Die `main ()`-Funktion

- Ein C++-Programm besteht aus mindestens einer Funktion
- Eine Funktion besteht aus
  - Funktionskopf mit Rückgabewert-Typ, Namen und Argumentliste, eingeschlossen in Klammern (`()`)
  - Funktionsrumpf mit in geschweiften Klammern (`{}`) eingeschlossenen Programmkonstrukten
- Eine Funktion wird beendet mit der Anweisung
  `return (wert);`
  Rückgabewert ist wert
- Ist der Rückgabewert-Typ `void`, entfällt die Angabe (wert) zu der `return`-Anweisung.

Die `main ()`-Funktion (Forts.)

- Jedes C++-Programm muss die Funktion `main ()` enthalten
- ```
  int main ()
  {
  // Eingabe
     ...
  // Algorithmus nach Newton
     ...
  // Ausgabe
     ...
  // Rueckgabewert
    return (0);
  }
  ```
- Die `main ()`-Funktion muss den Rückgabewert-Typ `int` haben.
  ```

Eingabeanweisung

- Stream-orientierte Eingabe in C++ (Unterschied zu C)
- Benutzt wird die Klasse `cin` mit dem Operator >>
- Die Deklarationen zu Stream-orientierter Ein- und Ausgabe sind in `<iostream>` enthalten.
- Zur Benutzung

```
#include <iostream>
```

```
cin >> wert;
```

- Die Variable `wert` erhält den Wert, der über die Tastatur eingegeben wird

Eingabeanweisung (Forts.)

- Mehrere Eingaben:

```
cin >> wert1 >> wert2 >> wert3;
```

- "Leerzeichen" (Blank, Tabulator, Return) werden als Ende der Eingabe interpretiert!
- Eingabe

```
20 30 40 // wert1=20; wert2=30; wert3=40;
```

Ausgabeanweisung

- Analog zu Eingabe in C++ (Stream-orientiert, Unterschied zu C)
- Benutzt wird die Klasse `cout` mit dem Operator <<
- Zur Benutzung

```
#include <iostream>
```

- Wert des Ausdrucks wird auf dem Bildschirm ausgegeben
-

```
cout << "Bitte␣...␣eingeben:␣";
```

- Mehrere Ausgaben

```
cout << "Wurzel␣aus␣" << wert;
cout << "␣=␣" << wurzel << endl;
```

- Strings werden durch Anführungszeichen ("") eingeschlossen
- Spezielle Konstante für Zeilenumbruch `endl`

Ausdrücke und Zuweisungen

- Zunächst arithmetische Ausdrücke
- ```
  wurzel - (wurzel * wurzel - wert)
  / (2 * wurzel)
  ```
- entspricht

$$\texttt{wurzel} - \frac{\texttt{wurzel}^2 - \texttt{wert}}{2\texttt{wurzel}}$$

- Zuweisung geschieht mit dem Operator =
- *variable* = *ausdruck*
- *variable* erhält den Wert von *ausdruck*
- ```
  wurzel = wurzel - (wurzel * wurzel - wert)
  / (2 * wurzel)
  ```
- Zuweisung wird oft mit Vergleich (==) verwechselt!

Verzweigung

- Bedingte Anweisungen
- Programmkonstrukt
  1. ```
     if (ausdruck) {
         anweisungen
     }
     ```
  2. ```
     if (ausdruck) {
         anweisungen
     }
     else {
         anweisungen
     }
     ```
- *ausdruck* kann die Werte wahr oder falsch annehmen

Verzweigung (Forts.)

```
if (wert <= 0) {
  cout << wert << " ist nicht positiv!" << endl;
  return (1);
}
```

```
if (altwurzel <= wurzel) {
  fehler = wurzel - altwurzel;
}
else {
  fehler = altwurzel - wurzel;
}
```

implementiert mit altwurzel $= x_n$, wurzel $= x_{n+1}$

$$|x_{n+1} - x_n| = \begin{cases} x_n - x_{n+1} & \text{falls } x_{n+1} \leq x_n; \\ x_{n+1} - x_n & \text{andernfalls.} \end{cases}$$

Verzweigung (Forts.)

- Arithmetischer Vergleich (*wert* <= 0) entspricht $wert \leq 0$

Schleife

- Bedingte Wiederholung
- Programmkonstrukt nicht abweisende Schleife

```
do {
      anweisungen
} while (ausdruck);
```

- *anweisungen* werden solange ausgeführt, wie *ausdruck* wahr ist

Schleife (Forts.)

```
do {
  altwurzel = wurzel;
  wurzel = wurzel - (wurzel * wurzel - wert)
   / (2 * wurzel);
  if (altwurzel <= wurzel) {
    fehler = wurzel - altwurzel;
  }
  else {
    fehler = altwurzel - wurzel;
  }
} while (fehler >= genauigkeit);
```

Rückgabewert

- Für die Funktion `int main ()` gibt der Rückgabewert den Status der Programmausführung an das Betriebssystem bzw. das aufrufende Programm zurück
- Vereinbarung:
  - Wert 0 bedeutet: das Programm ist ordnungsgemäß beendet
  - Wert ungleich 0 bedeutet: das Programm ist nicht ordnungsgemäß beendet

Rückgabewert (Forts.)

- ```
  if (wert <= 0) {
    cout << wert << "␣ist␣nicht␣positiv!" << endl;
    return (1);
  }
  ```
- Nicht ordnungsgemäß, falsche Eingabe
- ```
  main () {
    ...
    return (0);
  }
  ```
- Ordnungsgemäß, Berechnung durchgeführt

Zusammenfassung

- Programmierparadigmen sind Modellmethoden für den Entwurf und die Implementierung von Programmen.
- Objektorientierung stellt Typen in den Mittelpunkt der Betrachtung.
- Objektorientierte Programmiersprachen beinhalten Datenkapselung, Polymorphie und Vererbung.
- In C++ kann wie in C auch prozedural programmiert werden.
- Das Ein-/Ausgabesystem in C++ ist verschieden von dem in C.
- Initialisierung von Konstanten und Variablen kann in C++ anders als in C erfolgen.