Vorkurs C++ Programmierung

Wintersemester 2008/2009

Fachgruppe Computergrafik und Multimediasysteme

Lisa Brückbauer, Jens Orthmann
[brueckbauer, orthmann]@fb12.uni-siegen.de
Tel.: 0271/740-3452, Büros: H-A 7109, H-A 7115

Die Programmiersprache C++

- Ziele dieses Abschnittes
  - Wozu Programmierung?
  - Einführung in C++ als imperative Sprache
    - Objektorientierung kommt später!
  - Erläuterung und Übung grundlegender Sprachelemente
- Herausforderung
  - Vielzahl von Sprachelementen, insb. Zeiger
  - Erlernen der (jeder!) Programmiersprache nur durch praktisches Üben!
- Literatur
  - Online: www.cplusplus.com, www.cppreference.com
  - E-Book: Ralf Schneeweiß: Moderne C++ Programmierung http://www.springerlink.com/content/w12837/ (nur innerhalb des Uni-Netzes)
  - Ressourcen zum Kurs: www.cg.informatik.uni-siegen.de
    → Lehre → Vorkurs Linux und Programmierung

Wozu Programmierung?

- Komplexe Probleme von komplexen Systemen lösen
- Mechanismen für wiederkehrende Aufgaben „automatisieren“
  - z.B. Wetterdaten visualisieren
  - Warenbestand in einem Lager prüfen
  - Bestellung in einem Online-Shop tätigen
  - Planung der Fahrstrecke ins Urlaubsgebiet
  - Systemüberwachung im Atomkraftwerk...
- Viele verschiedene Anwendungen:
  - Web-Programmierung (HTML, PHP, Ajax, ...)
  - Datenbank-Programmierung (MySQL, Oracle, ...)
  - Mikroprozessor-Programmierung (Hardware) (Assembler, C, ...)
  - „Büroanwendungen“ (Java, C#, VB, SAP, ...)
  - ...

Wozu Programmierung?

- Viele unterschiedliche Ziele und Anforderungen:
  - Web/Netzwerkprogrammierung → Zuverlässigkeit, Benutzerfreundlichkeit (Webshops, Communitysysteme, ...)
  - Echtzeit-Industrieanwendungen → Zuverlässigkeit, Fehlertoleranz (z.B. Walzstraße in der Stahlproduktion, Fahrerassistenzssysteme im Auto)
  - Grafikprogrammierung → Ästhetik, Aussagekraft, hohe FPS (Spiele!)
  - Mathematische/Wissenschaftliche Anwendungen → Effizienz, Laufzeit, Aussagekraft, ...

Sonstige Gründe?

- Es macht Spaß!
- Es ist kreativ!
  - Es gibt viele Wege, ein gegebenes Problem zu lösen.
- Man sieht das Ergebnis!
  - Der Weg zur Lösung ist aber oft schwer und langwierig
    → Umso größer ist das Erfolgserlebnis!
- Immer eine neue Herausforderung!
  - Man lernt nie aus!
  - Technologien entwickeln sich weiter

Die Programmiersprache C++

- Eine der populärsten höheren Programmiersprachen
  - Entwicklung Anfang der 80er Jahre
  - hybride Programmiersprache:
    - Prozedurale (imperative) Programmierung
    - Maschinennahe Programmierung
    - Modulare Programmierung
    - Generische Programmierung (Templates)
    - Objektorientierte Programmierung
- Basiert auf C
- Hier und in EI-1: Konzentration auf imperativen Teil von C++
- Objektorientierung: Schwerpunkt in EI-2

Ein einführendes Programmbeispiel

- Beispiel: Berechnung der Fakultät einer Zahl
  - Definition der Fakultät einer natürlichen Zahl $n \in \mathbb{N}$:

$$n! := \prod_{i=1}^{n} i = n \cdot (n-1) \cdot \ldots \cdot 2 \cdot 1$$

- Beispiel: 7! = 7 * 6 * 5 * 4 * 3 * 2 * 1 = 5040
- Berechnungsvorschrift („Algorithmus“) zur Berechnung von n! für gegebenes n:
  1. setze Ergebnis auf 1: **`result ← 1`**
  2. solange **`n > 0`**:
     2.1 **`result ← result * n`**
     2.2 **`n ← n-1`**
- Vereinbarungen:
  - „←“ ist eine (Wert-)Zuweisung, **`result`** ist nach der Zuweisung mit **`n`** multipliziert.

Ein einführendes Programmbeispiel

- Programm zur Berechnung der Fakultät

```
#include <iostream>

int main ( int argc, char* argv )
{
   unsigned int result;
   result = 1;
   unsigned int n = 10;
   for ( unsigned int i = n; i >= 1; i-- )
   {
     result = result * i;
   }
   std::cout << „Fakultaet von “ << n << „: “ << result;

   return 0;
}
```

Ein einführendes Programmbeispiel

- Programm zur Berechnung der Fakultät

```
#include <iostream>

int main ( int argc, char* argv )
{
   unsigned int result;
   result = 1;
   unsigned int n = 10;
   for ( unsigned int i = n; i >= 1; i-- )
   {
     result = result * i;
   }
   std::cout << „Fakultaet von " << n << „: " << result;

   return 0;
}
```

Header-Datei: iostream beinhaltet Funktionalität für Ein- und Ausgaben

`main`-Funktion Haupteinstiegspunkt eines Programms.

Berechnung Hier: Eine Schleife mit n Durchläufen

Ausgabe auf der Konsole. Hier: Text + Ergebnis

Ein einführendes Programmbeispiel

- Programm zur Berechnung der Fakultät

```
#include <iostream>

int main ( int argc, ch
{
   unsigned int result;
   result = 1;
   unsigned int n = 10;
   for ( unsigned int i = n; i >= 1; i
   {
     result = result * i;
   }
   std::cout << „Fakultaet von " << n << „: " << result;

   return 0;
}
```

Variable **result**: Hier soll später unser Ergebnis drinstehen!

Variable **n**: Der Wert von **n** ist unsere „Eingabe“. Wir weisen der Variablen **n** den Wert **10** zu.

Ein einführendes Programmbeispiel

- Progr der Fakultät

```
#includ

int mai
{
   unsigned int result;
   result = 1;
   unsigned int n = 10;
   for ( unsigned int i = n; i >= 1; i-- )
   {
     result = result * i;
   }
   std::cout << „Fakultaet von " << n
             << „: " << result;

   return 0;
}
```

Berechnung:
In jedem Schleifendurchlauf wird die Variable **result** mit der Variablen **i** multipliziert.
i wird in jedem Schleifendurchlauf verändert.

Durchlauf	i	result
-	-	1
1	10	10
2	9	90
3	8	720
4	7	5040
5	6	30240
6	5	151200
7	4	604800
8	3	1814400
9	2	3628800
10	1	3628800
11	0	?

Programmgerüst

Das Programmgerüst für die nächsten Tage...

```
#include <iostream>
int main (int argc, char* argv[])
{
  //Hier kann man „Befehle" einfügen
  //...
  return 0;
}
```

Was braucht ein „Programm“?

- **Variablen** als „Platzhalter“ für Operanden und Ergebnis(se)/Zwischenergebnisse
  - brauchen einen Namen und einen Typ
- **Anweisungen**, was mit den Variablen geschehen soll
  - Berechnungen, Ein-/Ausgabe, ...
- Einen **Algorithmus** (Berechnungsvorschrift)
  - z.B. Algorithmus zur Berechnung von n! für gegebenes n:
    1. setze Ergebnis auf 1: `result ← 1`
    2. solange `n > 0`:
       2.1 `result ← result * n`
       2.2 `n ← n-1`

Variablen: Deklaration und Initialisierung

- Welche Namen/Bezeichner für Variablen darf man wählen?
  - Grundsätzlich jede Kombination aus Ziffern, Buchstaben und dem _ (underscore).
  - Konventionen erleichtern das arbeiten!
  - Führende Ziffern sind nicht erlaubt (z.B. `01var`)
  - C++-Schlüsselwörter können nicht als Bezeichner verwendet werden:

```
and, and_eq, asm, auto, bitand, bitor, bool, break, case,
catch, char, class, compl, const, const_cast, continue,
default, delete, do, double, dynamic_cast, else, enum,
explicit, export, extern, false, float, for, friend, goto,
if, inline, int, long, mutable, namespace, new, not,
not_eq, operator, or, or_eq, private, protected, public,
register, reinterpret_cast, return, short, signed, sizeof,
static, static_cast, struct, switch, template, this, throw,
true, try, typedef, typeid, typename, union, unsigned,
using, virtual, void, volatile, wchar_t, while, xor, xor_eq
```


Variablen: Deklaration und Initialisierung

- Deklaration: ***Typ Name;***
  - ***Typ:*** Typ der Variablen, z.B. `int, float, bool`
    Name: Name der Variablen. Sollte aussagekräftig sein!

```
int ergebnis;
bool istGeradeZahl;
int summe;
```

  - Achtung: Der Wert einer auf diese Weise deklarierten Variablen ist zunächst undefiniert, d.h. sie hat einen unbekannten Wert!
  - C++ reserviert benötigten Speicher für die Variable, in Abhängigkeit vom Datentyp (s. übernächste Folie).
- Zuweisung: ***Name = Wert***
  - ***Wert*** ist der neue Wert (=Inhalt) der Variablen
  - Die Variable ***Name*** muss zuvor deklariert worden sein!

```
istGeradeZahl = true;
summe = 13 + 15;
```


Variablen: Deklaration und Initialisierung

- Deklaration mit Initialisierung: ***Typ Name = Wert;***
  - ***Wert*** ist der initiale Wert (=Inhalt) der Variablen

```
float f = 14.31; //Achtung: Punkt, kein Komma!
int operand = 17;
char zeichen = 'a';
```

Variablen: Deklaration und Initialisierung

- Beispiele:

```
#include <iostream>
int main ( int argc, char* argv[] )
{
   //Deklarationen:
   int ergebnis;
   bool istGeradeZahl;
   int summe;
   //Zuweisungen:
   istGeradeZahl = true;
   summe = 13 + 15;
   //Deklarationen mit Zuweisung
   float f = 14.31;      //Achtung: Punkt, kein Komma!
   int operand = 17;
   char zeichen = 'a';

   return 0;
}
```


Einfache Datentypen

Typ	Größe im Speicher	Funktion
Ganze Zahlen		
`int`	32 bit	Vorzeichenbehaftete ganze Zahl
`short`	16 bit	Vorzeichenbehaftete ganze Zahl
`long`	64 bit	Vorzeichenbehaftete ganze Zahl
`unsigned int`	32 bit	Vorzeichenlose ganze Zahl
`unsigned short`	16 bit	Vorzeichenlose ganze Zahl
`unsigned long`	64 bit	Vorzeichenlose ganze Zahl
Gleitkommazahlen		
`float`	32 bit	reelle Zahlen
`double`	64 bit	reelle Zahlen (double precision)
Wahrheitswerte		
`bool`	8 bit	Wahrheitswert (true, false)
Zeichen		
`char`	8 bit	ASCII-Zeichen („Buchstaben, Zahlen und Sonderzeichen“)

Variablen: Deklaration und Initialisierung

- Gleitkommazahlen (floating point)
  - Hintergrund: Siehe EI-1 Vorlesung!
  - Wichtig hier: Benutzung zur Darstellung reeller Zahlen.
  - Verwendung analog zu `int`-Typen:

```
#include <iostream>
int main ( int argc, char* argv[] )
{
   float pi = 3.1415;
   int k = 4;
   //Achtung: Ergebnisvariable muss vom passenden Typ sein!
   float ergebnis = pi * k;
   std::cout << ergebnis;
   return 0;
}
```


Operatoren

- Arithmetische Operatoren
  - schon gesehen: `+ - * /`
  - Modulo-Operator (%): liefert den Rest der ganzzahligen Division

    `7 % 3 = 1`

    Benutzung: `int rest = 7 % 3;  //rest hat den //Wert 1`
- Zuweisungsoperator (=)
  - kein Gleichheitszeichen im Sinne der Mathematik
  - Ergebnis einer Berechnung wird einer Variablen zugewiesen

Operatoren...

- Inkrement (`++`) und Dekrement (`--`)
  - Erhöhen oder Verringern einer Ganzzahl um 1

```
int i = 3;          //Initialisierung mit 3
i++;                //Erhöhung um 1
```

  - z.B. bei **`for`**-Schleifen!

```
for ( unsigned int i = n; i >= 1; i-- )
{
   result = result * i;
}
```


Operatoren

- Vergleichsoperatoren
  - Umsetzung von logischen Vergleichen
  - Symbole: **`== , != , > , < , >= , <=`**
  - Liefern als Ergebnistyp immer **`bool`** (also **`true`** oder **`false`**)
- Beispiel:

```
int zahl1 = 15;
int zahl2 = 3;
bool groesser = (zahl1 >= zahl2);
```

→ **`groesser`** ist **`true`**

Prioritäten von Operatoren

- Punkt- vor Strichrechnung
  - **`int ergebnis = 1-2*3+4;`**
    `//ergebnis hat den Wert -1`
- Operatoren haben verschiedene Prioritäten
  - Hoch:
    - +, - als Vorzeichen: → **`int k = -3;`**
    - ! (Negation) → **`bool b = !false;`**
  - Weniger hoch:
    - *, /, % (Multiplikation, Division, Modulo)
  - Weniger hoch:
    - +, - (Addition, Subtraktion)
  - Noch weniger...
    - <=, <, >=, > (kleiner/gleich, kleiner, größer/gleich, größer)
  - Noch weniger...
    - ==, != (gleich, ungleich)

Prioritäten von Operatoren

- Es gibt noch mehr Operatoren mit unterschiedlichen Prioritäten
  - Meistens ist die Benutzung aber intuitiv und logisch
- Empfehlenswert: Klammern benutzen
  - dient auch der Lesbarkeit einer Anweisung
  - Auswertung wie in der Mathematik: Von innen nach außen

Allgemein: Anweisungen

- **Ziel:** Ausführen eines Befehls
- **Syntax:** `Anweisung;`
- **Beschreibung:**
  - Eine Anweisung ist das grundlegende Element jeder prozeduralen Programmiersprache
  - In C/C++ werden Anweisungen mit einem Semikolon abgeschlossen
  - Eine Anweisung entspricht einem oder mehreren zusammengesetzten Befehlen an den Computer

Beispiele für Anweisungen...

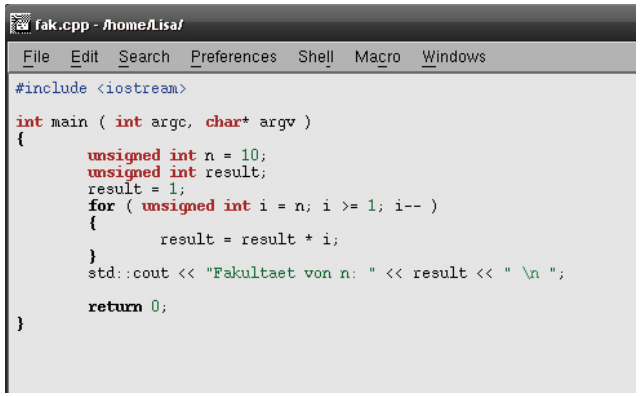
- Beispiele

```
int ergebnis = 1-2*3+4; //Delaration mit Zuweisung und
                        //Berechnungsanweisungen
return x;               //Rückgabe eines Wertes
result = result * i;    //Zuweisung mit Berechnung
k++;                    //Berechnung
```

- Anweisungen werden mit Semikolon abgeschlossen
- Kein Semikolon bei Schleifen und Funktionsdefinitionen (kommt beides später ausführlicher)

Quelltext compilieren und ausführen

- Quelltext steht in einer Datei mit der Endung .cpp:
- Benutzt werden kann jeder einfache Texteditor, z.B. gedit unter Linux



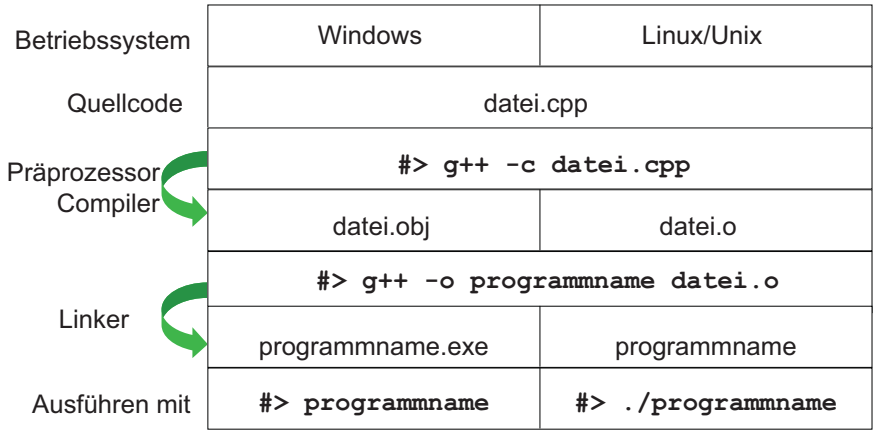
Quelltext compilieren und ausführen

Unser Ziel: Ein ausführbares Programm!

- Dafür brauchen wir einen **Compiler**
  - Hier benutzen wir den g++ (GNU C++ Compiler)
  - „Compiler“: Umgangssprachlich für Präprozessor, Compiler und Linker
  - **Präprozessor**: Textersetzung von **#**-Befehlen (z.B. Einfügen von zusätzlichem Quelltext mit **`#include`**, definieren von Konstanten mit **`#define`**...)
    - Merken: Der Präprozessor bereitet den Quelltext für den Compilevorgang vor
  - **Compiler**: Übersetzung des Quellcodes in ein maschinenlesbares Format (sog. „Objektdateien“)
  - **Linker**: Binden der Objektdateien zu einer ausführbaren Datei

Quelltext compilieren und ausführen

Kompiliervorgang (ausführlich):



Quelltext compilieren und ausführen

Kompiliervorgang (Kurzform):

- Compilieren durch einfachen Shell-Befehl:
  Entweder
  `#> g++ datei.cpp` (erzeugt standardmäßig die ausführbare Datei **`a.out`**)
  oder
  `#> g++ -o startdatei.o datei.cpp` (erzeugt eine ausführbare Datei mit dem selbstgewählten Namen **`startdatei.out`**)
- Nach erfolgreichem Compilieren befindet sich im gleichen Verzeichnis das ausführbare Programm **`a.out`** bzw. **`startdatei.out`**
- Das Programm wird ausgeführt mit:
  `#> ./a.out` bzw. **`./startdatei.out`**

Zusammenfassung

- Programmierung allgemein
  - Vielfältige Anwendungen, Sprachen, ...
- Ein erstes Programm (Fakultätsberechnung)
  - Algorithmus
  - Variablen
  - Datentypen
  - Operatoren (arithmetisch, Vergleichsoperatoren)
  - Prioritäten und Klammern
  - Anweisungen
- Kompiliervorgang und Ausführung
  - **`g++ -o startdatei.o datei.cpp`**