C vs. C++

Sebastian Meyer

Universität Hamburg

Proseminar C - „Grundlagen und Konzepte“, 2013

Gliederung

Was ist C++?

C++ ist eine objektorientierte Weiterentwicklung von C

- Beinhaltet die Sprache C
- Ergänzungen und Aufweichungen
- Leider auch Verschärfungen und Änderungen
- C-Code vom C++-Compiler nicht immer kompilierbar
- C-Code von C++-Code benutzbar
- Neue Befehle und Libraries (C++ Libs benutzen)

Historische Einordnung

- Systemprogrammierung, Treiber etc. in C und Assembler
- Simula: geeignet für größere Projekte; aber nicht hocheffizient
- ab 1979: Bjarne Stroustrup entwickelt "C with Classes"
- 1983: Umbenennung in C++ und Erweiterung
- Systemprogrammierung, Anwendungssoftware auch mit C++
- Java, C# etc. eignen sich ebenfalls für Anwendungssoftware
- C++ rückt aufgrund der Effizienz wieder in den Vordergrund (Anwendungssoftware)

Vergleich der Konzepte von C / C++

C
imperativ
strukturiert
prozedural

C++
imperativ
strukturiert
prozedural
objektorientiert
generisch
(funktional)

- Objektorientierung auch in C mithilfe von Konventionen möglich

Vergleich der Spracheigenschaften

C (.c - nicht objektorientiert)	**C++ (.cpp - objektorientiert)**
Structs, Zeiger	(Structs), Zeiger, Klassen, Objekte, Mehrfachvererbung, Überladung, Exceptions
Standardlib: C	Standardlibs: (C) / C++
kleiner Overhead	größerer Overhead möglich
möglicherweise schneller	schnell
maschinennah +	maschinennah
min. Laufzeitu., kein Garbage Co.	min. Laufzeitu., kein Garbage Co.
malloc & Zeigerarithmetik	zuverlässigere Mechanismen: Zugriffsmodifizierer, new, delete()
gute Optimierung möglich	Optimierung möglich
Fachlichkeit schwierig modellierbar	Fachlichkeit gut modellierbar (Entwurfskonzepte & Methoden)
Übersichtlichkeit gefährdet	Übersichtlichkeit gefährdet
(schneller zu entwickeln)	(besser zu warten)

Neue Sprachelemente in C++: Ein- und Ausgabe

- Headerdatei: iostream.h / iostream
- Überladung der Bitshift-Operatoren statt printf()
- Eingabe: >>
- Ausgabe: <<
- Ein-/ Ausgabeort: cin, cout, cerr, clog
- Zeilenumbruch: endl = \n
- cout zusammen mit << entspricht printf()
- Selbständige Formatwahl

```
1  #include <iostream>
2  using namespace std;
3
4  int main(int argc, char* argv[])
5  {
6      int jahre_alt;
7      cout << "Wie alt?" << endl;
8      cin >> jahre_alt;
9      cout << "Du bist " << jahre_alt << " Jahre alt." << endl;
10     return 0;
11 }
```

Neue Sprachelemente in C++: Umgang mit Variablen

- Variablendeklarationen müssen nicht mehr am Anfang eines Blocks stehen (auch in C99)
- Neuer Operator für den Zugriff auf globale Variablen mit demselben Bezeichner

```
1  #include <iostream>
2  using namespace std;
3
4  int a = 5;
5
6  int main(int argc, char* argv[])
7  {
8      int a = 7;
9      cout << a << endl; //Ausgabe a lokal: 7
10     int b = 6; //In C nicht moeglich
11
12     cout << a << endl; //Ausgabe a lokal: 7
13     cout << ::a << endl; //Ausgabe a global: 5
14 }
```

Neue Sprachelemente in C++: Defaultparameter

- Defaultparameter im Funktionskopf möglich
- Sie müssen am Ende stehen

```
1  #include <iostream>
2  #include <cmath>
3  using namespace std;
4
5  int potenzen(int a, int potenz=2)
6  {
7      return pow((float)a, potenz);
8  }
9
10
11 int main(int argc, char* argv[])
12 {
13     cout << potenzen(3) << endl; //Ergebnis: 9
14     cout << potenzen(3, 3) << endl; //Ergebnis: 27
15     return 0;
16 }
```

Neue Sprachelemente in C++: Überladung

```
1  #include <iostream>
2  #include <cmath>
3  using namespace std;
4
5  int potenzen(int a, int potenz)
6  {
7      return pow((float)a, potenz);
8  }
9
10 int potenzen(int a)
11 {
12     return pow((float)a, 2);
13 }
14
15 int main(int argc, char* argv[])
16 {
17     cout << potenzen(3) << endl; //Ergebnis: 9
18     cout << potenzen(3, 3) << endl; //Ergebnis: 27
19     return 0;
20 }
```

- In C mit Namenskonventionen: z.B. potenzen_ii() und potenzen_i() - C++ kodiert diese selbst

Neue Sprachelemente in C++: Klassen

Structs:

- Attribute

Klassen:

- Attribute
- Funktionen

In C mit Namenskonventionen: z.B. structname_funktionsname()

Neue Sprachelemente in C++: Klassen (Negativbeispiel)

```
1  #include <iostream> //include einer Headerdatei ohne .h
2  using namespace std;
3
4  class Bankkonto //Mehrere Klassen in einer Datei moeglich
5  {
6
7  protected:
8      int _kontostand;
9
10 public: //fuer alle folgenden Deklarationen gueltig
11     Bankkonto()
12     {
13         _kontostand = 0;
14     }
15
16     int getKontostand()
17     {
18         return _kontostand;
19     }
20
21     int zahleEin(int wert)
22     {
23         if(wert >=0)
24         {
25             _kontostand = _kontostand + wert;
26             return 0;
27         }
28         return 1;
29     }
30 };
```

Neue Sprachelemente in C++: Klassen

Konto.h

```
1  class Bankkonto
2  {
3
4  protected:
5      int _kontostand;
6
7  public:
8      Bankkonto(); //Konstruktor
9      ~Bankkonto(); //Destruktor
10
11     int getKontostand();
12     int zahleEin(int wert);
13 };
```

Neue Sprachelemente in C++: Klassen

Konto.cpp

```
1  #include "Konto.h"
2
3  Bankkonto::Bankkonto() //Konstruktor
4  {
5      _kontostand = 0;
6  }
7
8  Bankkonto::~Bankkonto() //Destruktor
9  {
10 }
11
12 int Bankkonto::getKontostand()
13 {
14     return _kontostand;
15 }
16
17 int Bankkonto::zahleEin(int wert)
18 {
19     if(wert >=0)
20     {
21         _kontostand = _kontostand + wert;
22         return 0;
23     }
24     return 1;
25 }
```

Neue Sprachelemente in C++: Objekte und Dyn. Speicher

- Objekte auf dem Heap erzeugen: Pointerdeklarartion + new + Konstruktoraufruf
- Objekte auf dem Stack erzeugen: Objektdeklaration + Konstruktoraufruf
- Objekte auf dem Heap löschen: delete()
- malloc() und free() weiterhin einsetzbar

```
1  #include <iostream>
2  #include "Konto.h"
3  using namespace std;
4
5  int main(int argc, char* argv[])
6  {
7      Bankkonto* konto1 = new Bankkonto(); //Objekt auf dem Heap
8      konto1->zahleEin(2);
9      cout << konto1->getKontostand() << endl;
10     delete(konto1); //Destruktoraufruf und Speicherfreigabe
11     konto1 = NULL;
12
13     Bankkonto konto2 = Bankkonto(); //Objekt auf dem Stack
14     konto2.zahleEin(10);
15     cout << konto2.getKontostand() << endl;
16
17     return 0;
18 }
```

Neue Sprachelemente in C++: Vererbung

Vererbung:

```
1  class Girokonto :public Bankkonto
2  {
3      //...
4  };
```

Mehrfachvererbung:

```
1  class Girokonto :public Bankkonto, public Finanzbehaelter
2  {
3      //...
4  };
```

- Mehrfachvererbung kann unangenehme Folgen haben

„Es gibt nur 5 Personen auf der Welt, die Mehrfachvererbung verstehen.“(Axel Schmolitzky, 2013)

„Es ist richtig, dass man sehr genau wissen muss, was man tut, aber dann ist es ein mächtiges Werkzeug.“(Nathanael Hübbe, 2013)

Neue Sprachelemente in C++: Vererbung

Konten.h

```
1  #include "Konto.h"
2
3  class Girokonto :public Bankkonto //Standardwert: private
4  {
5  protected:
6      int _dispolimit;
7
8  public:
9      Girokonto();
10     int zahleAus(int wert);
11 };
```

Neue Sprachelemente in C++: Vererbung

Konten.cpp

```
1  #include "Konten.h"
2
3  Girokonto::Girokonto()
4  {
5      this;
6      _dispolimit = 100;
7  }
8
9  int Girokonto::zahleAus(int wert)
10 {
11     if((_kontostand - wert) >= (0 - _dispolimit))
12     {
13         _kontostand = _kontostand - wert;
14         return 0;
15     }
16     return 1;
17 }
```

Neue Sprachelemente in C++: Virtuelle Funktionen

- Speziellere Implementationen haben Vorrang

Konto.h

```
1  using namespace std;
2
3  class Bankkonto
4  {
5  public:
6      virtual string getKontoTyp(); //Virtuelle Funktion
7  };
```

Konto.cpp

```
1  #include <string>
2  #include "Konto.h"
3  using namespace std;
4
5  string Bankkonto::getKontoTyp()
6  {
7      return "Bankkonto";
8  }
```

Neue Sprachelemente in C++: Virtuelle Funktionen

Konten.h

```
1  #include "Konto.h"
2  using namespace std;
3
4  class Girokonto :public Bankkonto
5  {
6  public:
7      string getKontoTyp();
8  };
```

Konten.cpp

```
1  #include <string>
2  #include "Konten.h"
3  using namespace std;
4
5  string Girokonto::getKontoTyp()
6  {
7      return "Girokonto";
8  }
```

Neue Sprachelemente in C++: Virtuelle Funktionen

Main.cpp

```
1  #include <string>
2  #include <iostream>
3  #include "Konten.h"
4  using namespace std;
5
6  int main(int argc, char* argv[])
7  {
8      Bankkonto* konto1 = new Bankkonto();
9      cout << konto1->getKontoTyp() << endl; //Ausgabe: Bankkonto
10
11     Bankkonto* konto2 = new Girokonto(); //Ausgabe: Girokonto (Bankkonto-Pointer)
12     cout << konto2->getKontoTyp() << endl;
13
14 return 0;
15 }
```

Neue Sprachelemente in C++: Abstrakte Klassen

- Durch "... = 0"wird die Klasse zur abstrakten Klasse
- Objekte können nicht erzeugt werden
- Abstrakte Klasse muss die Funktion nicht implementieren
- Stärker als Interfaces → als Ersatz einsetzbar

Konto.h

```
1  using namespace std;
2
3  class Bankkonto
4  {
5  public:
6      virtual string getKontoTyp() = 0;
7  };
```

Konto.cpp

```
1  #include <string>
2  #include "Konto.h"
3  using namespace std;
4
5  string Bankkonto::getKontoTyp()
6  {
7      return "Bankkonto";
8  }
```

Neue Sprachelemente in C++: Templates

- Definieren Vorlagen für Funktionen oder Klassen
- Wenig Code steht für viel Code
- Implementation für verschiedene Typen (generische Programmierung)
- Implementationsspezialisierung möglich
- Funktionieren mit Ersetzung
- Sozusagen: Weiterentwicklung von Makros
- Unterteilung möglich in:
  - Funktionstemplates
  - Klassentemplates
- Achtung: Können Code unübersichtlich machen und Fehler verursachen

Neue Sprachelemente in C++: Templates

```
1  #include <string>
2  #include <iostream>
3  using namespace std;
4
5  template <typename T> //Allgemeines Template
6  T addiere(T obj1, T obj2)
7  {
8      return obj1 + obj2;
9  }
10
11 template <> //Spezialisierung
12 string addiere(string str1, string str2)
13 {
14     return str1 + " " + str2;
15 }
16
17 int main(int argc, char* argv[])
18 {
19     string a = "Hallo";
20     string b = "Welt";
21     cout << addiere(2, 3) << endl; //Ergebnis: 5
22     cout << addiere(2.1, 2.1) << endl; //Ergebnis: 4.2
23     cout << addiere(a, b) << endl; //Ergebnis: Hallo Welt
24     //cout << addiere(2.3, 3) << endl; --> ist nicht moeglich
25     return 0;
26 }
```

- Ähnliches ist auch mit Klassen möglich

Neue Sprachelemente in C++: Namespaces

Namespace festlegen:

```
1  namespace bank
2  {
3      int a;
4  }
```

Auf Namespace zugreifen:

```
1  #include "Bank.h"
2  using namespace bank;
3
4  int main(int argc, char* argv[])
5  {
6      a = 0;
7      return 0;
8  }
```

oder

```
1  #include "Bank.h"
2
3  int main(int argc, char* argv[])
4  {
5      bank::a = 0;
6      return 0;
7  }
```

Neue C++ Standardbibliothek

Einbindung:

```
1 #include <paket-headerdatei>
2 using namespace std;
```

Einige Header:

- <iostream>
- <string>
- <list>
- <set>
- <stack>
- <vector>
- <functional>

Weitere Header (von C übernommen):

- <cstdlib> (<stdlib.h>)
- <cstdio> (<stdio.h>)

Vom C++-Compiler nicht kompilierbarer C-Code

- Bei Void-Pointern in C++: explizite Typumwandlung nötig

```
1  #include <stdlib.h>
2
3  int main(int argc, char* argv[])
4  {
5      int* a1 = malloc(sizeof(int)); //malloc gibt void Pointer zurueck
6      int* a2 = (int*) malloc(sizeof(int)); //Gueltiger C++ Code
7      return 0;
8  }
```

- Allgemein: oft explizite Typumwandlung notwendig
- Rekursiver Aufruf von main() ist in C++ nicht möglich
- Reservierte Namen nicht mehr als Bezeichner verwendbar
- Datentyp von Zeichenliteralen: C → int, C++ → char

```
1  int main(int argc, char* argv[])
2  {
3      printf("%i", sizeof('a')); //Verschiedene Ergebnisse
4      return 0;
5  }
```

Vom C++-Compiler nicht kompilierbarer C-Code

- funktion() bedeutet in C++ "keine Parameter" und in C "unbestimmt"

```
1  int funktion1() //C++: keine Parameter, C: unbestimmt
2  {
3      return 0;
4  }
5
6  int main(int argc, char* argv[])
7  {
8      funktion1(1); //In C++ ungueltig
9      return 0;
10 }
```

- Es gibt noch weitere Einschränkungen

Fazit

Welchen Compiler benutzen?

- C-Code nicht vom C++-Compiler kompilieren lassen
- Ausnahme: C-Code in einem C++-Projekt verwenden
- Besser: C-Code als .c oder Bibliothek einbinden

Welche Sprache benutzen?

- Bevorzugter Programmierstil?
- Effizienz, Schnelligkeit, Mikrocontroller $\rightarrow$ C
- Größere Projekte $\rightarrow$ C++

Zusammenfassung

- C++ ist eine objektorientierte Weiterentwicklung von C
- Daraus resultieren viele neue Sprachelemente
- Viele andere Weiterentwicklungen
- Einige Weiterentwicklungen hat C bereits berücksichtigt
- Nicht jeder C-Code ist vom C++-Compiler kompilierbar
- Kriterien bei der Sprachwahl:
  - Maschinennähe, möglicher Overhead, Schnelligkeit, Optimierbarkeit
  - Wartbarkeit, fachliche Modellierbarkeit, Projektgröße, bevorzugter Programmierstil

Quellen

C++ - Programmieren mit einfachen Beispielen - Dirk Louis, Markt+Technik Verlag

C / C++ - Die praktische Referenz - Dirk Louis, Markt+Technik Verlag

http://de.wikibooks.org/wiki/C++-Programmierung

http://de.wikipedia.org/wiki/C%2B%2B

http://en.wikipedia.org/wiki/Compatibility_of_C_and_C%2B%2B

http://www.asdala.de/algorithmik/cpp/non-oop.html

http://www.textarchiv.alojado.de/text/vorteile_und_nachteile_der_objektorientierung-ap645.html

http://www.mikrocontroller.net/articles/C_vs_C%2B%2B

http://www.willemer.de/informatik/cpp/cincout.htm

http://www.willemer.de/informatik/cpp/pointer.htm

http://docwiki.embarcadero.com/RADStudio/XE4/de/Virtuelle_Funktionen

http://www.namespace-cpp.de/std/doku.php/kennen/header

http://de.over-blog.com/C_vs_Java_Vor_und_Nachteile-1228321776-art185420.html