C++ Einführung

Dirk Frettlöh

verwendet Material von Gundolf Haase, Uni Graz

Technische Fakultät
Universität Bielefeld

April 8, 2024

Was ist C++?

- Vorgestellt 1985 als Erweiterung von C (1972)
- "C with classes": C mit Objektorientierung
- Sehr häufig in der Praxis verwendet ("My compiler compiles your compiler")

  Beispiele (ganz oder teilweise in C++)
  - Unity Game Engine (Call of Duty, Pokemon Go.....)
  - MacOS X, Windows 10...
  - Firefox, Chrome, Safari...
  - MS Office, LibreOffice...
  - Facebook, youtube... Frontend: python, Backend: C++
  - ...

Warum so häufig? Python und Java sind auch häufig (siehe: TIOBE).
Kurz: Objektorientiert **und** schnell.

Diese Einführung wird kurz, denn

- Ich selbst bin kein C++-Experte
- Sie können schon programmieren, auch objektorientiert

Also brauchen wir nur noch die Syntax.

Das "Hello World"-Programm in C++:

```
1  #include <iostream>                    // Ein-Ausgabe nutzen
2  using namespace std;                   // setze Namensraum auf std
3                                         //  (sonst: std::cout statt cout usw)
4  int main()                             // Beginn Hauptprogramm
5  {                                      // Block durch { ... } einfassen
6    cout << "Hello World" << endl;
7    return 0;                             // Beende Programm
8  }
```

Dasselbe “Hello World”-Programm in C++:

```
1  #include <iostream>                    // Ein-Ausgabe nutzen
2  using namespace std;                   // setze Namensraum auf std
3                                         //  (sonst: std::cout statt cout usw)
4  int main()                             // Beginn Hauptprogramm
5  {                                      // Block durch { ... } einfassen
6    cout << "Hello World" << endl;
7    return 0;                            // Beende Programm
8  }
```

Wir sehen daran schon:

- Kommentare bis zum Zeilenende werden mit // eingeleitet.
- Der C-Kommentar // kann auch in C++ verwendet werden.
- Befehle enden mit Semikolon ;
- Jedes Programm benötigt eine Funktion main(), genannt Hauptprogramm.
  - int main() deklariert das Hauptprogramm in Zeile 4.
  - Die geschweiften Klammern { } in Zeilen 5 und 8 begrenzen den Funktionskörper der Funktion main.
  - In Zeile 7 wird der Ausdruck return 0 durch das Semikolon ; zu einer Anweisung im Programm. Diese spezielle Anweisung beendet das Hauptprogramm mit dem Rückgabewert 0.

Fast dasselbe “Hello World”-Programm in C++:

```
1  #include <iostream>                    // Ein-Ausgabe nutzen
2  // using namespace std;                // setze Namensraum auf std
3                                         //  (sonst: std::cout statt cout usw)
4  int main()                             // Beginn Hauptprogramm
5  {                                      // Block durch { ... } einfassen
6    std::cout << "Hello World" << std::endl;
7    return 0                             // Beende Programm
8  }
```

- Die Ausgabe in Zeile 6 nutzt I/O-Bibliotheken von C++.
  - `cout` ist ein Bezeichner für die Ausgabe im Terminal.
  - << leitet den nachfolgenden String auf den Ausgabestrom (cout) um. Dies kann wiederholt in einer Anweisung geschehen.
  - endl ist der Bezeichner für eine neue Zeile im Ausgabestrom.
  - Die Preprocessor-Anweisung (beginnt mit #) in Zeile 1 fügt das Headerfile iostream in den Quelltext ein. Erst dadurch können Bezeichner wie `cout` und `endl` der I/O-Bibliothek benutzt werden.
  - Ohne Zeile 2 muss der Ausgabestrom etc. über die explizite Angabe des Namensraumes `std` angegeben werden, also als `std::cout`. Mit Zeile 2 wird automatisch der Namensraum std berücksichtigt.

Wie bringe ich's ans Laufen?

Wie immer: Quelltext eingeben und compilieren, Programm ausführen.

Viele Möglichkeiten. IDE: geany usw.

Oder "zu Fuß": hier für Linux.

1. Quelldatei editieren.
   $ [Lieblingstexteditor] `HelloWorld.cpp`

   Also z.B. `$ emacs Helloworld.cpp`
2. Quelldatei compilieren
   `$ g++ HelloWorld.cpp`
3. Programm ausführen.
   `$ ./a.out`

Wieso “cpp”? Wieso “g++”? Wieso a.out?

cpp Dateiendung im Prinzip beliebig.
Üblich: `.C`, `.cc`, `.cpp`, `.cxx`, `.c++` (bzw `.H`, `.hh`,...)

g++ Im Prinzip `gcc`. Gnu-Compiler: zuerst ein C-Compiler, dann auch C++, dann auch andere.

g++ ist `gcc -xc++ -lstdc++ -shared-libgcc`.

g++ erwartet ein C++-Programm (und nicht C oder so)

a.out `a.out` ist der Standard. Das kann dann umbenannt werden:

```
$ mv a.out hallowelt
```

Erweitertes “Hello World”-Programm in C++:

```
1  #include <iostream>                    // Ein-Ausgabe nutzen
2  #include <string>                      // Strings nutzen
3  using namespace std;
4
5  int main()
6  {
7    cout << "HelloWorld\n";
8    int i;                               // Variable i als int deklarieren
9    cout << "i=";
10   cin >> i;                            // Eingabe
11   cout << ``\n'' <<"i gleich " << i << ".\n"
12
13   float a,b;                           // Fliesskommazahl deklarieren
14   cout << "a, dann b eingeben:";
15   cin >> a >> b;
16   cout << "\n a=" << a << " und b=" << b <<"\n";
17                                        // Erstes Auftauchen von c,
18   float c=a+b;                         // also deklarieren
19   const string st("a+b ist ");         // Deklarieren der Konstante st
20   cout << st << c << "\n";
21   return 0 ;
22 }
```

Was sehen wir?

- Statt `endl` geht auch \n
- Eingabe mit `cin`
- `string, float, const`...

Variablen Wichtige grundlegende Datentypen:

Typ	Byte	Inhalt	Mögliche Werte
char	1	ASCII-Zeichen	H, e, \n
bool	1	Wahrheitswert	false, true
int, long	4	Ganze Zahlen	$[-2^{31}, 2^{31}-1]$
unsigned, unsigned long	4		$[0, 2^{32}-1]$
long long	8		$[-2^{63}, 2^{63}-1]$
unsigned long long	8		$[0, 2^{64}-1]$
float	4	Fließkommazahlen	1.1, -1.56e-32
double	8		1.1, -1.56e-132

Ein paar weitere wie signed char, short... Wie int mit weniger byte.

Beim ersten Auftauchen muss eine Variable mit ihrem Typ deklariert werden, z.B.:

- `int i;` oder
- `int c=5;` oder
- `string st;` oder
- `string st="Cpiderman"` oder wie oben:
- `string su("Cuperman");`

Variablen gelten innerhalb ihres Blockes: { ... int i=5; ... }

Variablen können ihren Wert ändern: c=6.

Es gibt auch Konstanten: `const int i=17;` usw.

Die haben immer denselben Wert.

for, while, if: Machen wir es kurz. Schleifen:

```
1  for (int i = 0; i < 10; i++)
2  {
3    // do something
4  }
```

oder

```
1  int i = 0;
2  while (i <= 10)
3  {
4    i++;
5    // do something
6  }
```

Bedingte Verzweigung (if .. else):

```
1    y=0.0;
2    if(x>=0.0)
3      y=1.0;        //nur eine Anweisung, keine { } noetig
4    cout << "Ergebnis:" << y <<endl;
```

oder

```
1    if(x>=0.0)
2      y=1.0;        //nur eine Anweisung, keine { } noetig
3    else
4      y=0.0;        //nur eine Anweisung, keine { } noetig
5    cout << "Ergebnis:" << y << endl;
```

oder

```
1    if(x>=0.0)
2    {
3      y=1.0;        // alles mit { }
4      z=1;
5    }
6    else
7    {
8      y=0.0;
9      z=2;
10   }
11   cout << "Ergebnis:" << y << endl;
```

...oder für viele Fallunterscheidungen
(vgl. pattern matching in `haskell`):

```
1  switch (foo)
2  {
3    case 1:
4      // do something
5      break;
6    case 2:
7      // do something else
8      break;
9    default:
10     // do something different
11     break;
12 }
```

Funktionen: z.B.

```
1  int add1(int a, int b)
2  {
3    a = a + b;
4    return a;
5  }
6
7  int main()
8  {
9    int a = 2;
10   int b = 4;
11   int s = 0;
12   s = add1(a,b);
13   cout << a << " + " << b << " = " << s << endl;
14
15   return 0
16 }
```

Ausgabe: $2 + 4 = 6$

Funktionen: z.B.

```
1  int add1(int &a, int &b)
2  {
3    a = a + b;
4    return a;
5  }
6
7  int main()
8  {
9    int a = 2;
10   int b = 4;
11   int s = 0;
12   s = add1(a,b);
13   cout << a << " + " << b << " = " << s << endl;
14
15   return 0
16 }
```

Ausgabe: $6 + 4 = 6$

Funktionen: z.B.

```
1  int add1(int &x, int &y)
2  {
3    x = x + y;
4    return x;
5  }
6
7  int main()
8  {
9    int a = 2;
10   int b = 4;
11   int s = 0;
12   s = add1(a,b);
13   cout << a << " + " << b << " = " << s << endl;
14
15   return 0
16 }
```

Ausgabe: $6 + 4 = 6$

Vektoren:

```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4
5  int main()
6  {
7    vector <double> v(3);    // Deklariere Vektor in R^3
8    v[0]=1;  v[1]=3;  v[2]=0.5;
9
10   cout << "\n v = (" ;
11   for (int k=0, k<3, ++k){ cout << v.at(k) << ", ";}
12   cout << ") \n" ;
13   return 0;
14 }
```

Ausgabe: `v = (1, 3, 0.5,)`

`v[k]` wird nicht abgefangen für z.B. $k > 2$ (Programm stürzt ab, oder unsinniger Wert, oder...)

`v.at(k)` wird abgefangen für z.B. $k > 2$ (Programm endet mit Fehlermeldung)

Vektoren:

```
1  #include <cmath>          //  sqrt ()
2  #include <iostream>
3  #include <vector>         //  vector<>
4  using namespace std;
5
6  int main()
7  {
8    int n;
9    cout << "\n Anzahl der Vektoreintraege:";
10   cin >> n;                        // Format des Vektors
11   vector <double> v(n);            // Deklariere Vektor v
12   for (int k=0; k<v.size(); ++k)
13   {
14     v.at(k)=1.0/(k+1.0);           // Belege Vektoreintraege
15   }
16   double norm=0.0;
17   for (int k=0; k<v.size(); ++k)
18   {
19     norm+=v[k]*v[k];               // Berechne Vektornorm
20   }
21   norm=sqrt(norm);
22   cout << "\n Norm: " << norm << endl;
23   return 0;
24 }
```

$$\text{Für } n = 3: \quad \|v\|_2 = \left\| \begin{pmatrix} 1 \\ 1/2 \\ 1/3 \end{pmatrix} \right\|_2 = 1.16667$$

Matrizen sind Vektoren mit Vektoren als Einträgen.

```
1
2  #include <iostream>
3  #include <vector>
4  using namespace std;
5
6  int main()
7  {
8    int mrow = 4;                    // Zeilen
9    int ncol = 3;                    // Spalten
10   vector<vector<float>> A(mrow,vector<float>(ncol));
11
12   for (int i=0; i<mrow; ++i) {
13     for (int j=0; j<ncol; ++j) {
14       A[i][j]=i+j;
15     }
16   }
17   cout << A.at(1).at(2) << endl;    // Elementzugriff
18
19   return 0;
20 }
```

$$\text{Hier ist } A = \begin{pmatrix} 0 & 1 & 2 \\ 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \end{pmatrix}$$

Ausgabe: 3

Matrix mal Vektor:

```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4
5  int main()
6  {
7    double s;
8
9    vector<float> v(3);    v[0]=2; v[1]=1; v[2]=0.5;
10
11   [...] // A wie oben: 0 1 2
12         //              1 2 3
13         //              2 3 4
14         //              3 4 5
15
16   vector<float> b(A.size());
17   for (int i=0; i<A.size(); ++i) {
18     s=0;
19     for (int j=0; j<A[1].size(); ++j) {
20       s+=A.at(i).at(j)*v.at(j);
21     }
22     b.at(i)=s;
23   }
24
25   cout << "\n b = (";
26   for (int k=0; k<b.size(); ++k){ cout << b.at(k) << ","; }
27   cout << ")" << endl;
28   return 0;
29 }
```

Damit kommen wir schon ziemlich weit.

Viel mehr in dem sehr empfehlenswerten Skript von Gundolf Haase.

Ein paar gemischte Tipps:

- Listen, arrays, Vektoren ... starten mit 0
- Fehlende } oder ; erzeugen seltsame Fehlermeldungen
- Test auf Gleichheit mit ==, nicht =
- `float` oder `double` nie auf Gleichheit testen

Viel Spaß!