

Einführung in die Programmiersprachen C und C++

Markus Kirschmer, Fakultät für Mathematik

Übungsblatt 6

Die Sprache C++ ist im Wesentlichen eine Erweiterung der Sprache C.

Wie bei C besteht ein C++-Programm aus Ausdrücken und Anweisungen, die elementaren Hilfsmittel zur Programmsteuerung wie `if-else-`, `while-`, `for-` Konstrukte usw. sind mit denen in C identisch.

Für die Ein- und Ausgabe können die C-Funktionen wie `printf`, `scanf`, `putchar`, `getchar` verwendet werden; es gibt aber auch C++-spezifische Implementationen.

Etliche der im folgenden betrachteten Programme sind analog zu denen auf Blatt 1, jedoch mit den C++-spezifischen Ein- und Ausgaberroutinen.

In C++ gibt es das Konzept des Datenstroms. Z.B. bezeichnet `cout` den Strom der Standard-Ausgabe, auf den mit `cout << "Hallo\n"`; ein String geschickt werden kann. Das Programm

```
// welt.cc

using namespace std;    // Namensraum der C++ Standardbibliothek
#include <iostream>
int main() {
    cout << "Hallo, Welt!\n";
}
```

schickt dann diesen String auf den Bildschirm (die Standard-Ausgabe).

Dies Programm wird übersetzt mit dem Befehl `g++ welt.cc`. Hierbei ist `g++` der Name des C++-Compilers (ggf. ersetzen!). Man beachte die gegenüber C veränderte `#include`-Anweisung.

Dabei wird das Zeichen `<<` als binärer Operator aufgefasst. Die beiden Operanden sind der Ausgabestrom `cout` und der auszugebende String `"Hallo"`. Das Resultat ist wieder der Ausgabestrom `cout`. Das bedeutet, daß man den Operator `<<` auf das Resultat und auf einen weiteren String anwenden kann:

```
cout << "Hallo\n" << " Welt\n";
```

ist gleichbedeutend mit:

```
cout << "Hallo\n Welt\n";
```

Der Operator `<<` ist also, wie von C gewohnt, linksassoziativ. Die Bedeutung des Bitshift-Operators `<<` für den Fall, daß die Operanden `int` sind, bleibt bestehen. Man spricht vom **Überladen** des Operators, wenn er in verschiedenen Kontexten verschiedene Bedeutungen hat.

Auf der rechten Seite des Operators `<<` dürfen nicht nur Strings stehen, sondern irgendwelche Objekte, die auszugeben sind, z.B. Zahlentypen wie `int`, `float`, `double`.

In Analogie hierzu gibt es den Eingabestrom `cin`, der den von der Tastatur oder der Standard-Eingaben kommenden Datenstrom in das Programm einliest. Z.B. schreibt nach der Deklaration `float x; char ch;` die Anweisung

```
cin >> x >> ch;
```

eine zunächst eingegebene Fließkommazahl nach `x` und einen im Anschluss eingegebenen Buchstaben nach `ch`.

Erscheinen in einer Programmzeile eines C++-Programms die Zeichen `//`, so sind diese und der Rest der Zeile ein Kommentar.

Aufgabe 6.1. Welche Ausdrücke und Anweisungen enthalten die folgenden Programme?

Einige C++-Programme (vgl. Blatt 1)

```

// inch.cc
// Umwandlung von Inch in cm und umgekehrt
using namespace std;
#include <iostream> // Praeprozessor-Anweisung

int main() {
    const float faktor = 2.54; // Deklaration u. Initialisierung
    float x, in, cm; // Deklaration
    char ch; // Deklaration
    cout << "Bitte Laenge eingeben, gefolgt von i oder c: "; // Ausgabe
    cin >> x >> ch; // Eingabe nach x und ch
    if (ch == 'i') { // Falls ch gleich dem Buchst. i
        in = x; // kopiere x nach in
        cm = x*faktor; // mult. x mit faktor, Erg. nach cm
    }
    else { // andernfalls (ch ungleich i)
        in = x/faktor; // teile x durch faktor, Erg. n. in
        cm = x; // kopiere x nach cm
    }
    cout << in << " in = " << cm << " cm\n"; // Ausgabe ...
}

```

Aufgabe 6.2. Übertragen Sie nach dem Muster der folgenden Beispiele alle Programme der Übungsblätter 1,2 nach C++, so daß die Ein- und Ausgabekanäle `cin`, `cout` anstelle von `scanf`, `printf`, `getchar`, `putchar` verwendet werden.

Das folgende Programm benutzt – wie auf Blatt 1 – ein `while`-Konstrukt, um eine Tabelle von Geschwindigkeiten (m/sec – km/std) zu erstellen.

```

// ms-kmh.cc
// rechnet Geschwindigkeiten um
using namespace std;
#include <iostream>
#include <iomanip> // erforderlich fuer setw(i), setzt die naechste
// Ausgabe rechtsbuendig in ein Feld der Breite i

int main() {
    float msec, kmh;
    msec = 0;
    while ( msec <= 120 ) {
        kmh = msec*3.6;
        cout << setw(10) << setprecision(6) << msec
        << " m/sec = " << setw(10) << setprecision(6) << kmh << " km/h\n";
        msec = msec + 22.345;
    }
}

```

Hier modifiziert die Funktion `setw(3)` den Ausgabestrom so, daß er für die nächste Ausgabe ein Feld der Breite 3 vorsieht. Mit `setprecision(int)` kann ähnlich die Zahl der ausgegebenen Dezimalen festgesetzt werden; erproben Sie dies.

Hinweis: Man kann die C++-Ein/Ausgabe mit der in C üblichen mischen. (Man muss dann sowohl `iostream` wie auch `stdio.h` einbinden.) Bei formatierten Ausgaben ist oft die `printf`-Formatanweisung bequemer!

Aufgabe 6.3. Formulieren Sie das vorangegangene Programm so um, daß eine `for`-Schleife verwendet wird.

Die C++-Version der `getchar-putchar`-Schleife:

```
// io.cc
// liest die Eingabe zeichenweise,
// und gibt sie auf den Bildschirm aus.
using namespace std;
#include <iostream>

int main() {
    char c;
    while ( cin.get(c) )
        cout.put(c);
}
```

`cin.get(c)` versucht, ein Zeichen von der Tastatur ("Standard-Eingabe ") zu lesen. Gelingt dies, wird dies Zeichen nach `c` kopiert, und der Wert des Ausdrucks `cin.get(c)` ist $\neq 0$. Wird kein Zeichen mehr gelesen, ist der Wert des Ausdrucks $= 0$. `cout.put(c)` schreibt das Zeichen auf den Bildschirm ("Standard-Ausgabe ") Die Funktionen `cin.get()`, `cout.put()` entsprechen also den C-Funktionen `getchar()`, `putchar`. Unterschiede: `getchar()` nimmt kein Argument auf, `cin.get()` jedoch sehr wohl, *es modifiziert sein Argument c sogar*, so daß es nach dem Funktionsaufruf einen anderen Wert als vorher hat!

Dies ist ein wichtiger Unterschied gegenüber Funktionen in C. Während in C die Variablen in Funktionen immer lokal sind, ihre Änderungen also keine Auswirkungen auf die aufrufende Funktion hat, ist dies in C++ nicht unbedingt der Fall. Es können sogenannte *Referenzen* übergeben werden, s.u..

Der Rückgabe-Typ ist im Gegensatz zu `getchar` ein Objekt vom Typ `istream` (Eingabestrom), also wie `cin`. Wenn der Rückgabewert Null ist, ist das Ende des Eingabestroms erreicht.

```
// odh.cc, wandelt Zahldarstellungen um: oktal, dezimal, hex
using namespace std;
#include <iostream>
#include <iomanip>

int main() {
    int a; char c;
    while (1) {
        cout << "Bitte Zahl eingeben mit vorangestelltem o,d oder h: ";
        cin >> c;
        switch (c) {
            case 'o': cin >> oct >> a; break;
            case 'h': cin >> hex >> a; break;
            case 'd': cin >> dec >> a; break;
            default: continue;
        }
        cout << " Oktal: " << oct << a;
        cout << " Dezimal: " << dec << a;
        cout << " Hex: " << hex << a << "\n";
        if (a == 0)
            break; // Mit einer break-Anweisung kann man eine Schleife
    } // verlassen.
}
```

Hier finden wir wieder Modifikatoren der Ein-/Ausgabeströme: `oct`, `hex`, `dec` veranlassen oktale, hexadezimale oder dezimale Interpretation des jeweils nächsten Ein-/Ausgabe-Zeichens.

Aufgabe 6.4. Ändern Sie das Programm `odh.cc` so ab, daß es als Präfix nicht nur `o,d,h`, sondern auch `O,D,H` akzeptiert und sinngemäß arbeitet.

Funktionen in C++:

Die Deklaration einer Funktion geschieht wie in C, beispielsweise deklariert

```
int ggt(int, int)
```

eine Funktion namens `ggt`, die zwei Argumente vom Typ `int` aufnimmt und einen Wert vom Typ `int` zurückgibt. Im folgenden Beispiel berechnet eine solche Funktion den größten gemeinsamen Teiler zweier ganzer Zahlen.

```
// ggt.cc
// Groesster gemeinsamer Teiler ggt zweier Zahlen
using namespace std;
#include <iostream>

int ggt(int x, int y) {           // gibt ggt(x,y) zurueck, falls
    int c;                       // x oder y ungleich 0
    if ( x < 0) x = -x;          // und gibt 0 zurueck fuer x=y=0.
    if ( y < 0) y = -y;
    while ( y != 0 ) {
        c = x % y; x = y; y = c;
    }
    return x;                    // Dies ist der ggt.
}

int main() {
    int a,b;
    while (1) {
        cout << "Gib zwei ganze Zahlen ein: ";
        if ( !(cin >> a >> b) ) {
            cout << "Eingabefehler\n";
            break;
        }
        cout << "ggt(" << a << ", " << b << ") = " << ggt(a,b) << "\n";
    }
}
```

Erläuterungen:

1. Funktionsaufrufe werden behandelt wie in C: Die Funktionsdefinition von `ggt` enthält zwei formale Variablen `int x`, `int y`, deren Wert erst bei Aufruf der Funktion (in `main` durch den Ausdruck `ggt(a,b)`) feststeht. Die aktuellen Werte von `a`, `b` werden an die Funktion `ggt` übergeben, die Funktion berechnet dann den Rückgabewert, indem sie den Funktionscode so ausführt, als hätten `x,y` den Wert von `a,b`. Der Wert des Ausdrucks `ggt(a,b)` im Funktionsaufruf ist der Wert, den die Funktion mit `return` zurückgibt.
2. In `main` wird geprüft, ob der Ausdruck `cin >> a >> b` den Wert 0 hat. Dies ist der Fall, wenn das Einlesen der beiden Werte für `a,b` fehlerhaft war, z.B. weil versehentlich ein Buchstabe statt einer Zahl eingegeben wurde. Die `break`-Anweisung sieht für diesen Fall den Abbruch des Programms vor.

Aufgabe 6.5. Schreiben Sie eine rekursive Variante der Funktion `int ggt(int,int)`.

Überladen von Funktionen:

Hat man eine Funktion `funct` durch `typ1 funct(typ2, typ3)` deklariert, so darf man in C++ Funktionen gleichen Namens zu anderen Typen definieren, etwa `typ4 funct(typ5, typ6)`, wobei die Typen `typ4`, `typ5`, `typ6` von den Typen `typ1`, `typ2`, `typ3` verschieden sind, so daß also der Name `funct` in verschiedenen Zusammenhängen unterschiedliche Bedeutung hat.

Beispiel: Es gibt eine Funktion `double pow(double, double)`, die in der C++-Bibliothek bereits vorhanden ist, mit der Eigenschaft, daß `pow(x, y)` die Potenz x^y berechnet. Diese Funktion wird im folgenden Beispiel durch eine selbstdefinierte Funktion `double pow(double, int)` überladen.

Im Gegensatz dazu gibt es für C mehrere Bibliotheksfunktionen, die zur Funktion `pow` Varianten `powf`, `powl` für weitere Float-Typen definieren, vgl. `man pow`.

Gründe für diese Möglichkeit: Es könnte sein, daß eine Funktion für gewisse Typen (int statt double) bei spezieller Implementation schneller ist als eine generelle Implementation.

Ein anderer Grund: In C++ kann man eigene Typen definieren, und man will vorhandene Funktionen auf diese Typen „fortsetzen“. Etwa kann man die obige Funktion `pow` erweitern auf komplexe Zahlen.

Achtung: Im vorliegenden Fall muss die Mathematik-Bibliothek mit eingebunden werden. Dies geschieht einerseits durch Verwendung der Datei `math.h`, andererseits durch Ergänzung des Compilationsaufrufes um die *Linker-Option* `-lm`. Dies ist für C++ ebenso wie für C.

Im Beispiel wird von der „selbstdefinierten“ Funktion `double pow(double, int)` eine Ausgabe "(Integer Exponent)" veranlasst, um zu zeigen, daß wirklich der eigene Code verwendet wird.

```
// ueberl.cc
// Ueberlaedt die Bib-Fkt double pow(double,double)
// durch double pow(double,int)
// Zu uebersetzen mit: g++ ueberl.cc -o ueberl
using namespace std;
#include <iostream>
#include <math.h>

double pow(double f, int i) { // zweites Argument jetzt int !!
    cout << " (Integer Exponent) : ";
    if (i == 0) return 1;
    if (i < 0) { i = -i; f = 1/f; }
    if (i == 1) return f;
    double res = f; // Deklarationen muessen in C++ nicht am Blockanfang stehen
    while (i-- > 1)
        res *= f;
    return res;
}

int main() {
    double f = 1.234, e = 2.718;
    for (int i = 0; i < 4; i++) { // int i hier: C++ -spezifisch !
        cout << f << " hoch " << i << " = " << pow(f,i) << '\n';
        cout << f << " hoch " << i << " = " << pow(f,(double)i) << '\n';
        cout << f << " hoch " << e+i << " = " << pow(f,e+i) << '\n';
    }
}
```

Aufgabe 6.6. Überladen Sie die Funktionen `pow` durch eine Version vom Prototyp `int pow(int, int)`. Dabei soll `pow(x, y)` für $y < 0$ eine Fehlerausgabe liefern.

Funktionen und Referenzen in C++

In C++ gibt es für Funktionen neben dem von C gewohnten Aufruf durch *Call by value* auch den *Call by reference*. Typischer Einsatz: Die Funktion soll eine Variable ändern, die unabhängig von ihr definiert ist.

Standardbeispiel: `int a; scanf("%d", &a);` In die Variable `a` soll durch die Funktion ein neuer Wert eingelesen werden. Um diesen Effekt zu erreichen, muss man in C explizit eine Adresse (hier `&a`) übergeben, also als Argument eine Zeigervariable verwenden.

In C++ kann man die Adressenübergabe durch eine *Referenz* vermeiden. Dazu ein Beispiel:

```
// refs.cc Beispiel fuer Funktion mit Referenzuebergabe
// swap_ soll die Werte zweier Variablen vertauschen
using namespace std;
#include <iostream>
// FALSCHER Variante:
void swap0(int x, int y) { // lokale Kopien der Werte
    int tmp = x;
    x = y; y = tmp;      // diese lokal (!) vertauscht
}                        // bleibt ausserhalb ohne Effekt
// Zeiger-Variante, in C ueblich, in C++ moeglich
void swap1(int* x, int* y) { // lokale Kopien von Zeigern auf Variable
    int tmp = *x;          // die Inhalte der Variablen, auf die
    *x = *y;  *y = tmp;    // die Zeiger weisen, werden vertauscht.
}
// Referenzen-Variante, gleiche Semantik wie swap1, in C++ moeglich
void swap2(int& x, int& y) { // "Referenzen" = Adressen werden uebergeben
    int tmp = x;          // Wert einer Referenz ist der Inhalt der
    x = y; y = tmp;      // Speicherstelle, auf die die Referenz zeigt.
} // code- und aufrufgleich mit swap0, abgesehen von der Deklaration von x,y
int main() {
    int a, b;
    cout << "Eingabe a b "; cin >> a >> b;
    cout << "a = " << a << ", b = " << b << '\n';
    swap0(a,b);           // Werte uebergeben
    cout << "a = " << a << ", b = " << b << '\n';
    swap1(&a,&b);         // Adressen uebergeben
    cout << "a = " << a << ", b = " << b << '\n';
    swap2(a,b);          // Referenzen uebergeben
    cout << "a = " << a << ", b = " << b << '\n';
}
/* Ein-/Ausgabe:
Eingabe a b 3 5
a = 3, b = 5
a = 3, b = 5          swap0 hat nicht vertauscht
a = 5, b = 3          swap1 hat vertauscht
a = 3, b = 5          swap2 hat vertauscht          */
```

Aufgabe 6.7. Schreiben Sie eine Funktion, die die Werte von drei Variablen gleichen Typs zyklisch vertauscht; verwenden Sie Referenzen.

Aufgabe 6.8. Schreiben Sie eine Funktion `twice()`, die als Argument ein `int` nimmt, und als Rückgabewert ebenfalls ein `int` liefert, so daß der Aufruf `c = twice(a)` folgendes leistet: Nach dem Aufruf ist der Wert von `a` verdoppelt und der Wert von `c` ist der Wert von `a` vor dem Aufruf.

Funktionstemplates

Manche Funktionen können für ganze Serien von Typen definiert werden. Beispiel: `Max(a,b)` kann so für alle Zahltypen und sogar für alle Typen definiert werden, für die die Operation `a > b` sinnvoll ist (z.B. für Strings). C++ sieht hierfür *Templates* vor:

```
// templ.cc
using namespace std;
#include <iostream>
#include <string>
```

```

template <class T>
T Max(T x, T y) {
    return ( x > y ) ? x : y;
}
int main() {
    int    x = 1, y = 2;
    float  u = 1.0, v = 0.4;
    double e = 2.7182818, pi = 3.14159265;
    string s = "qwerty";
    string t = "uiop";
    cout << "Max("<<x<<","<<y<<") = "<<Max(x,y)<<'\n';
    cout << "Max("<<u<<","<<v<<") = "<<Max(u,v)<<'\n';
    cout << "Max("<<e<<","<<pi<<") = "<<Max(e,pi)<<'\n';
    cout << "Max("<<s<<","<<t<<") = "<<Max(s,t)<<'\n';
} /* Typische Ausgabe:      Max(1,2) = 2
                           Max(1,0.4) = 1
                           Max(2.71828,3.14159) = 3.14159
                           Max(qwerty,uiop) = uiop          */

```

Aufgabe 6.9. Schreiben Sie Templates für die Funktionen `min`, `abs` (Berechnung des Minimums von 2 (oder mehr?) Zahlen bzw. des Absolutbetrages von Zahlen).

Ein weiteres Beispiel: Die Swap-Funktion (Vertauschen der Werte zweier Variablen) hätte man sicher gern für jeden Typ, auch für Typen, die man erst noch entwickeln will.

Bemerkung: Die Templates `max` und `swap` gibt es bereits im namespace `std`, zur Demonstration definieren wir sie hier noch einmal unter den Namen `Max`, `Swap`.

Um die Ausgabeanweisungen im nächsten Beispiel abzukürzen, wird gleich ein weiteres Template entwickelt. Der Typ von `cout` ist `ostream&`, also eine Referenz auf einen Typ namens `ostream`. Daher schreiben wir eine Ausgabefunktion mit diesem Rückgabetypp, die nach der Ausgabe das Objekt `cout` zurückgibt. Auf diese Weise können wir den `<<`-Operator verwenden, um weitere Ausgaben anzuhängen. Mit dem Schlüsselwort `inline` wird der Compiler veranlasst, den Code für `Swap` nicht als Funktionsaufruf zu generieren, sondern direkt in die aufrufende Funktion einzubetten. Dies liefert kürzere Laufzeiten.

```

// swap-template.cc
using namespace std;
#include <iostream>
#include <string>

template <class T>
inline void Swap(T& x, T& y) {
    T tmp = x;
    x = y; y = tmp;
}

template <class S>
ostream& out(string xx, S x, string yy, S y) {
    return cout<<xx<< " = "<<x<< " , "<<y<< " = "<<y;
}

int main() {
    int    x = 1, y = 2; float  u = 1.0, v = 0.4;
    double e = 2.7182818, pi = 3.14159265;
    out("x",x,"y",y)<<'\n';  Swap(x,y);  out("x",x,"y",y)<<'\n';
    out("u",u,"v",v)<<'\n';  Swap(u,v);  out("u",u,"v",v)<<'\n';
    out("e",e,"pi",pi)<<'\n'; Swap(e,pi); out("e",e,"pi",pi)<<'\n';
}

```

Quicksort ist ein schnelles Sortier-Verfahren, das folgendermaßen arbeitet:

Aus einem zu sortierenden Array wird ein Wert p herausgegriffen, danach werden alle anderen Array-Elemente sortiert in einen unteren Abschnitt, der alle Elemente $< p$ enthält, sowie einen oberen Abschnitt mit den Elementen, die $\geq p$ sind, jedoch ohne das Element p selbst. Sodann wird auf diese beiden Abschnitte (rekursiv) das gleiche Verfahren angewandt.

Ein Quicksort-Template als Anwendung von swap:

```
// quicksort.cc Template-Version
using namespace std;
#include <iostream>

int scount=0;          // Zaehler fuer die Aufrufe von Swap
template <class T>      // Swap template
inline void Swap(T& x, T& y) {
    T tmp = x;
    x = y; y = tmp;
    scount++;
}

template <class T>      // Ausgabe-Template
ostream& out(T v[], int size) {
    for (int i=0; i<size; i++)
        cout << v[i] << ' ';
    return cout;
}

template <class T>      // Quicksort-Template
void quicksort(T v[], int n) { // cf. K&R, p.87, dort fuer C und festen Typ
    if(n <= 1)
        return;
    int last = 0;
    for (int i = 1; i < n; i++)
        if (v[0] > v[i])
            Swap(v[++last], v[i]);
    Swap(v[0],v[last]);
    quicksort(v, last); quicksort(v+last+1, n-last-1);
}

#define size(A) (sizeof((A))/sizeof((A)[0]))

// Diese Arrays verschiedener Typen werden sortiert:
int    a[] = { 3, 1, 4, 1, 5, 9, 2, 6, 3, 1, 4, 1, 5, 9, 2, 6};
float  b[] = { 3.1, 4.3, 5.9, 2.7, 5.3, 5.9, 9.7, 9.9,
              3.2, 4.1, 5.7, 2.6, 5.4, 5.8, 9.3, 9.3};
string s[] = { "Sonntag", "Montag", "Dienstag", "Mittwoch", "Donnerstag",
              "Freitag", "Samstag", "Mittwoch", "Donnerstag", "Freitag", "Samstag" };
int main() {
    scount=0;
    quicksort(a,size(a)); out(a,size(a))<<'\n';
    cout<<scount<<'\n';scount=0;
    quicksort(b,size(b)); out(b,size(b))<<'\n';
    cout<<scount<<'\n';scount=0;
    quicksort(s,size(s)); out(s,size(s))<<'\n';
    cout<<scount<<'\n';scount=0;
}
```

Aufgabe 6.10. *Kann man für die Berechnung von `size(a)` und `size(b)` ein Template formulieren?*