Kurzanleitung C++

Christian Palus, 28. Mai 2019
Abteilung für Angewandte Mathematik
Albert-Ludwigs-Universität Freiburg

UNIX-Befehle

Achtung: Löschbefehle können nicht ohne Weiteres rückgängig gemacht werden.

`$ cd dir`	ins Verzeichnis `dir` wechseln
`$ cd ..`	ins übergeordnete Verzeichnis wechseln
`$ ls`	zeigt Inhalt des aktuellen Verzeichnisses an
`$ mv datei dir/`	verschiebt `datei1` in das Verzeichnis `dir`
`$ mv datei dir/dat`	verschiebt und ändert Namen in `dat`
`$ cp datei dir/dat`	kopiert `datei1` nach `dir/dat`
`$ mkdir dir`	erzeugt ein neues Verzeichnis namens `dir`
`$ rmdir dir`	löscht das (leere) Verzeichnis `dir`
`$ rm -r dir`	löscht rekursiv das Verzeichnis `dir`
`$ touch datei`	erzeugt eine leere Datein namens `datei`
`$ ps`	zeigt laufende Prozesse an
`$ kill id`	beendet den Prozess mit der PID `id`
`$ time programm`	misst die Laufzeit des Programms `programm`
`$ man befehl`	zeigt Hilfeseite zum Befehl `befehl` an
`~`	Abkürzung für das Home-Verzeichnis
`.`	Abkürzung für das aktuelle Verzeichnis
`..`	Abkürzung für übergeordnetes Verzeichnis

Kompilieren

Übersetzen der Datei `programm.cc` in eine ausführbare Datei namens `programm` im selben Verzeichnis:

```
$ g++ -o programm programm.cc
```

Ausführen des Programms:

```
$ ./programm
```

Laufende Programme können mit der Tastenkombination `<Ctrl>+<C>` bzw. `<Strg>+<C>` abgebrochen werden.

Kommentare

Kommentare werden beim Übersetzen des Programms durch den Compiler ignoriert.

```
// Ich bin ein einzeiliger Kommentar
/* Ich bin ein Kommentar
ueber mehrere Zeilen */
```

Variablentypen

Typ	Art, Minimaler garantierter Wertebereich
`bool`	kann nur die Werte `0`,`1` annehmen
`char`	Einzelnes Zeichen, etwa `'a'`
`int`	16 bit ganze Zahl, Bereich -32768 bis 32767
`unsigned int`	16 bit nicht-neg. ganze Zahl, 0 bis 65535
`long int`	32 bit ganze Zahl, -2,147,483,648 bis 2,147,483,647
`float`	32 bit Gleitkommazahl, Genauigkeit ~7 Ziffern
`double`	64 bit Gleitkommazahl, Genauigkeit ~15 Ziffern

Eine Variable wird deklariert durch Angabe ihres Typs und eines Namens:

```
int j; double x; unsigned int pos;
```

Wertzuweisungen erfolgen mit dem Operator `=`, z. B.:

```
j = 3; x = 3.141592; pos = 3;
```

Aufbau eines Programms, Ein- und Ausgabe

C++ Programme bestehen aus Anweisungen. Eine Folge von Anweisungen, die durch geschweifte Klammern begrenzt sind, heißt Block. Die Ein- und Ausgabe von Werten kann mit Hilfe der Streams `cin` und `cout` realisiert werden.

```
// Ein- und Ausgabebibliothek einbinden
#include <iostream>
// Die main()-Funktion ist der Hauptteil eines Programms
int main()
{
  // Deklaration einer Variablen vom Typ int
  int k;
  // Textausgabe
  std::cout << "Gib dein Alter ein: ";
  // Wert einlesen und in k speichern
  std::cin >> k;
  // Ausgabe eines gespeicherten Wertes
  std::cout << "Du bist " << k << " Jahre alt!" << std::endl;
  // Rueckgabewert 0 steht fuer "normal exit"
  return 0;
}
```

Arithmetische Operationen

Es gibt die üblichen arithmetischen Operationen:

`+`	Addition
`-`	Subtraktion
`*`	Multiplikation
`/`	Division
`%`	Rest bei Ganzzahldivision

Die Division zweier ganzer Zahlen wird dabei automatisch als Ganzzahldivision ohne Rest ausgeführt.

Zuweisungs-, Inkrement-, und Dekrementoperatoren

`a=b`	einfache Zuweisung	`++i, i++`	i um 1 erhöhen
`a+=b`	wie `a=a+b`	`--i, i--`	i um 1 verringern
`a-=b`	wie `a=a-b`		
`a*=b`	wie `a=a*b`		
`a/=b`	wie `a=a/b`		

Der Unterschied zwischen `++i` und `i++` liegt im Rückgabewert des Operators: `++i` gibt den um 1 erhöhten Wert zurück, während `i++` noch den alten Wert von `i` vor der Erhöhung zurückgibt (`i--` und `--i` analog).

Vergleichsoperatoren und boolesche Verknüpfungen

`<, >`	kleiner, größer	`&&`	UND
`<=, >=`	kleiner-gleich, größer-gleich	`||`	ODER
`==, !=`	gleich, ungleich	`!`	NICHT

Verzweigungen

Die Schlüsselwörter **`if`** und **`else`** ermöglichen Fallunterscheidungen:

```
if ( j == 3 && x < 5 ) {
  /* Anweisungen in diesem Block werden ausgefuehrt falls
  der Wert von j gleich 3 und der von x kleiner als 5 ist */
}
else {
  /* Anweisungen in diesem Block werden ausgefuehrt falls
  der if-Block NICHT ausgefuehrt wird */
}
```

Schleifen

Die Anweisungen in einer „while“-Schleife werden solange wiederholt, bis ihre Bedingung verletzt ist. Ist die Bedingung schon vorher verletzt, so werden alle Anweisungen innerhalb des Schleifenkörpers übersprungen.

```
double zahl = 100.0;
while ( zahl > 0.5 )
{
  zahl = zahl / 10.0;
}
// Gibt 0.1 aus:
std::cout << zahl;
```

Die Anweisungen in einer „for“-Schleife werden solange wiederholt, bis eine Zählvariable eine Abbruchbedingung erfüllt. Nach jedem Durchlauf wird die Zählvariable gemäß einer Iterationsvorschrift verändert.

```
for ( int i = 1; i <= 10; i++ )
{
  // Wird 10 Mal ausgegeben:
  std::cout << "Ich bin eine for-Schleife\n";
}
```

Unterfunktionen

Eigene Funktionen müssen oberhalb der `main()`-Funktion deklariert werden. Dies kann auch in einer separaten Headerdatei geschehen, wenn diese dann mit einer **`#include`**-Anweisung eingebunden wird. Funktionen müssen einen Rückgabetyp (evtl. den leeren Typ **`void`**) haben und können von einem oder mehreren Parametern abhängen.

```
/* Erhaelt als Argument zwei Zahlen und
gibt deren Produkt zurueck */
double produkt( double x , double y )
{
  return x*y;
}
```

```
/* Gibt eine Fehlermeldung aus */
void fehlermeldung( int x )
{
  if ( x==1 )
  {
    std::cout << "Fehler 1 ist aufgetreten\n";
  }
  else if ( x==2 )
  {
    std::cout << "Fehler 2 ist aufgetreten\n";
  }
  else
  {
    std::cout << "Unbekannter Fehler!\n";
  }
}
```

Rekursive Programmierung

Funktionen können auch rekursiv definiert werden:

```
/*Berechnet rekursiv die Fakultaet einer natuerlichen Zahl*/
unsigned int fakultaet( unsigned int k )
{
  // Rekursionsverankerung
  if ( k==1 )
  {
    return 1;
  }
  else
  {
    // Rekursiver Funktionsaufruf
    return k * fakultaet( k-1 );
  }
}
```

Arrays

Arrays werden ähnlich wie Variablen deklariert, die Angabe ihrer Länge steht dabei in eckigen Klammern:

```
// Deklaration:
int lottozahlen[6];
```

Auch Arrays können bei der Deklaration initialisiert werden:

```
// Deklaration und Initialisierung:
int lotozahlen[6] = {3, 7, 13, 31, 37, 43};
```

Die Indizierung von Arrays beginnt in C bei 0. Der Zugriff auf einen Wert in einem Array geschieht mit eckigen Klammern:

```
std::cout << "Die Lottozahlen lauten: ";
for ( int i = 0; i < 6; ++ i)
{
  // Gibt die (i+1)-te Lottozahl aus
  std::cout << lottozahlen[i] << " ";
}
```

Außerdem können Arrays auch mehr als eine Dimension haben:

```
double matrix[3][3] = {{1,2,3},{4,5,6},{7,8,9}};
```

Mit dieser Belegung ist z.B. `matrix[0][0]` $= 1$, `matrix[0][1]` $= 2$, `matrix[0][2]` $= 3$, `matrix[1][0]` $= 4$, und so weiter.

Strings

Zur Arbeit mit Zeichenketten bindet man die Headerdatei `string` mittels `#include <string>` ein. Strings werden durch doppelte Anführungszeichen gekennzeichnet, im Gegensatz zu konstanten Ausdrücken vom Typ **char**, welche durch einzelne Anführungszeichen gekennzeichnet werden. Auf die einzelnen Zeichen eines Strings kann zugegriffen werden wie auf die Einträge eines Arrays. Vergleichsoperatoren wie > und < beziehen sich auf die Lexikographische Reihenfolge. Mit dem Operator + können zwei Strings aneinandergefügt werden.

```
std::string string1, string2, string3;
string1 = "Hallo"; string2 = ", Welt!";
string3 = string1 + string2       // "Hallo, Welt!"
string3[7] = 'Z';                 // "Hallo, Zelt!"
```

Zeiger

Zeiger sind Variablen, die als Wert die Speicheradresse einer anderen Variablen enthalten. Wichtig in diesem Zusammenhang sind die Operatoren * (Dereferenzierung) und & (Adressermittlung).

```
int i;             // Variable vom Typ integer
int *zeiger1;      // Zeiger auf eine Variable vom Typ int
int *zeiger2;      // Noch ein Zeiger auf eine int-Variable
i = 10;            // Wertzuweisung
zeiger1 = &i;      // zeiger1 zeigt nun auf die Variable i
zeiger2 = zeiger1; // zeiger2 zeigt nun auch auf i
int j;
j = *zeiger1;      // j hat den Wert, auf den zeiger1 zeigt
```

Call-by-Value und Call-by-Reference

Je nachdem, wie eine Funktion definiert ist, werden bei ihrem Aufruf entweder nur die Werte der Parameter (call-by-value) an die Funktion übergeben oder aber deren Speicheradressen (call-by-reference). Der Unterschied besteht darin, dass im ersten Fall etwaige Veränderungen der Funktionsargumente keinen Einfluss auf den Wert der entsprechenden Parameter im Hauptprogramm haben, im zweiten Fall jedoch schon, da direkt mit dem Wert des entsprechenden Parameters im Hauptspeicher gearbeitet wird. Standardmäßig arbeitet C mit call-by-value. Eine Parameterübergabe im Sinne von call-by-reference kann mit Hilfe von Zeigern oder mit dem Adressermittlungsoperator realisiert werden:

```
// Standardfall call-by-value
void by_value( int a ) {
  a = a + 10;
}
// Call-by-reference mit Zeigern
void by_reference1( int *a ) {
  *a = *a + 10;
}
// Call-by-reference mit Adressoperator
void by_reference2( int &a, int &b ) {
  a = a + 10;
}

int main() {
  int x = 10;
  // Standardfall
  by_value( x, y );         // immernoch x==10;
  // Bei der Zeigervariante werden die Speicheradressen
  // uebergeben
  by_reference1( &x, &y );  // jetzt ist x==20;
  // Bei dieser Variante werden die Speicheradressen in der
  // Funktion ermittelt
  by_reference2( x, y );    // jetzt ist x==30;
}
```

Im obigen Beispiel ist im Aufruf der Funktion `by_reference2` nicht ersichtlich, ob die Parameterübergabe als call-by-value oder call-by-reference realisiert wird. Daher ist es häufig sinnvoll, Variablen, deren Wert sich nicht verändern soll, direkt bei der Initialisierung als Konstanten zu kennzeichnen. Dies geschieht durch Voranstellen des Schlüsselwortes **const** bei der Deklaration. Im obigen Beispiel etwa gibt der Compiler eine Fehlermeldung aus, falls x mittels `const int x = 10;` initialisiert wird.

Die `vector`-Klasse

Ist die Größe eines Arrays bei der Übersetzung des Programms noch nicht bekannt, bzw. möchte man sie während der Laufzeit verändern, so bietet sich die `vector`-Klasse an. Ein Vektor in C++ ist ein dynamisches Array welches im Gegensatz zu den statischen Arrays keine feste Größe hat. Für die Benutzung muss die Bibliothek `<vector>` eingebunden werden. Ein leerer Vektor `v` mit Einträgen des Typs `T` wird dann deklariert durch `std::vector<T> v;`. Auf die Einträge eines nicht leeren Vektors kann genauso zugegriffen werden wie auf die Einträge eines Arrays.

```
#include <vector>
int main() {
  // Vektor mit ganzen Zahlen
  std::vector<int> quadratzahlen;
  // Vektor mit Gleitkommazahlen
  std::vector<double> w;
  /* Das Anhaengen von Werten an das Ende  eines Vektors
  geschieht mit der Methode push_back(): */
  w.push_back( 3.14159 );
  w.push_back( 2.71828 );
  for ( int i = 0; i < 100; ++i ) {
    quadratzahlen.push_back( i*i );
  }
  // Gibt 9 aus:
  std::cout << quadratzahlen[ 3 ] << std::endl;
  // Gibt 3.14159 aus:
  std::cout << w[ 0 ] << std::endl;
}
```

Weitere nützliche Methoden sind:

`v.size()`	gibt die Anzahl der Elemente in `v` zurück,
`v.back()`	gibt das letzte Element in `v` zurück,
`v.pop_back()`	entfernt das letzte Element aus `v`.

Ein- und Ausgabe von Dateien

Dateioperationen können mit der Klasse `fstream` durchgeführt werden:

```
#include <fstream>
#include <string>
int main() {
  // Deklariere eine Variable vom Typ string
  std::string stringvariable;
  // Deklariere zwei Objekte der Klasse fstream
  std::fstream datei_in, datei_out;
  // Oeffne Datei "datei1.txt" zum Lesen
  datei_in.open("datei1.txt", std::ios::in);
  // Oeffne Datei "datei2.txt" zum Schreiben
  datei_out.open("datei2.txt", std::ios::out);
  // Lese eine Zeichenkette aus der Quelldatei...
  datei_in >> stringvariable;
  // ...und schreibe sie in die Zieldatei
  datei_out << stringvariable;
  // Schliesse die Dateien
  datei_in.close(); datei_out.close();
}
```