```
================================================================================
Environment: vim
================================================================================

Vim startet man so, hier beispielsweise zum Editieren der datei "hello.cc":

# vim hello.cc

Vim hat mehrere Modi, für den Anfang genügt es, folgendes zu wissen:

Kommandomodus:

:q                        vim beenden
:q!                       vim beenden ohne Nachfrage,
                          wenn ungespeicherte Daten vorliegen
:w                        speichern
:w hello.h                Datei als "hello.h" speichern
h, j, k, l                Cursor nach links, unten, oben, rechts bewegen
dd                        Zeile löschen
i                         wechsel in den Eingabemodus
I                         an den Anfang der Zeile gehen, Wechsel in den
                          Eingabemodus
:42                       in Zeile 42 der Datei gehen
:syntax on                Syntaxhighlighting anstellen
:help                     startet die Onlinehilfe

Der Eingabemodus erlaubt normales Schreiben, und kann mit ESC verlassen werden.
```



```
================================================================================
Environment: g++
================================================================================

Üblicher übersetzungsvorgang:

Compilieren:

prog1.cc      -> prog1.o
prog2.cc      -> prog2.o
prog3.cc      -> prog3.o

Linken:

prog1.o prog2.o prog3.o -> prog

Damit ist prog das erwartete Ausführbare Programm, und prog1.cc prog2.cc
prog3.cc sind die vom Programmierer geschriebenen Dateien.

Optionen:

-c                        Compilieren
-o                        Ausgabedatei setzen
-O0, -O1, -O2             Optimierung einstellen (O0=keine, O1=wenig, O2=viel)
-Wall                     all Warnungen anzeigen
-g                        Debuggingsymbole eincompilieren
-p                        Profiling (zur Performancemessung) aktivieren

Beispiel:

# g++ -c -Wall -O2 -o prog1.o prog1.cc
# g++ -c -Wall -O2 -o prog2.o prog2.cc
# g++ -c -Wall -O2 -o prog3.o prog3.cc
# g++ -o prog prog1.o prog2.o prog3.o

Besteht ein Programm nur aus einer Datei, dann schreibt man auch kurz:

# g++ -Wall -O2 -o hello hello.cc
```

```
================================================================================
Environment: man
================================================================================

Die Manualpages enthalten Dokumentation zu Kommandos und Funktionen, die man
bei der Entwicklung benötigt. Beispiel für Kommandos:

# man vim
# man gcc

Die Manualpages sind in Unterabschnitte unterteilt:

Index für Unix-Handbücher
Abschnitt 1      Nutzer-Befehle
Abschnitt 2      System-Aufrufe
Abschnitt 3      Unter-Routinen
Abschnitt 4      Geräte
Abschnitt 5      Dateiformate
Abschnitt 6      Spiele
Abschnitt 7      Verschiedenes
Abschnitt 8      Systemverwaltung
Abschnitt 9      Kernel

C Funktionen finden sich normalerweise in den Abschnitten 2 und 3, hier ein
Beispiel.

# man 3 printf

Eine Schlüsselwortsuche liefert man -k:

# man -k printf

in KDE:
  Alt+F2 -> man:printf
  Alt+F2 -> man:

================================================================================
Programm: hello.cc
================================================================================

/* Das folgende Programm gibt "Hello World!" aus.
 */
#include <stdio.h>

int main()
{
    printf ("Hello World!\n");
    return 0;
}
```



```
================================================================================
C Strings im Speicher
================================================================================

Der C String "Welt" sieht im Speicher so aus:

| 'W' | 'e' | 'l' | 't' | 0 |
                          ^
                          Nullterminierung

In den ersten vier Bytes sind also die Buchstaben gespeichert, im letzten
Zeichen ist ein Nullbyte, welches anzeigt, dass der String hier endet. Die
Deklaration

const char *x = "Welt";

besagt:

const        Inhalt kann nicht verändert werden
char         Es handelt sich um einzelne Zeichen
*x           Es handelt sich um einen Zeiger, d.h. die Variable speichert
             die Adresse im Speicher, ab der der String beginnt.
= "Welt"     Der Zeiger Zeigt auf den C String Welt.

================================================================================
Programm: welt.cc
================================================================================

/* Das folgende Programm zeigt, wie man ein einzelnes Zeichen des Strings
 * "Welt" ausgibt.
 */

#include <stdio.h>

int main()
{
    const char *welt = "Welt";

    printf ("Das dritte Zeichen von '%s' ist '%c'.", welt, welt[2]);
    return 0;
}
```

```
===============================================================================
Datentypen: Boolean und ganze Zahlen
===============================================================================

Typ             Wertebereich auf einem x86er
-----------------------------------------
bool                true, false
char                -2^7..2^7-1
short               -2^15..2^15-1
int                 -2^31..2^31-1
long long           -2^63..2^63-1

Analog dazu gibt es jeweils einen vorzeichenlosen Typen (die nur positive Werte
annehmen können), also

unsigned char      0..2^8-1
unsigned short     0..2^16-1
unsigned int       0..2^32-1
unsigned long long 0..2^64-1

Für Funktionen gibt es den speziellen Typen

void                kein Wert

um Anzuzeigen, dass eine Funktion keinen Rückgabewert hat.

===============================================================================
Basiswissen: Berechnungen
===============================================================================

Zuweisung:

x = y;              // weise x den Wert von y zu

Summen:

x = a + b;          // weise x die Summe von a und b zu
x += b;             // weise x die Summe von x und b zu

analog dazu:
x = a - b           Differenz
x = a * b           Produkt
x = a / b           Division
x = a % b           a Modulo b

Postinkrement / Preinkrement:

x = a++;            // weise x a zu, und erhöhe dann a um 1
x = ++a;            // erhöhe a um 1, und weise das Ergebnis x zu
x = a--;            // weise x a zu, und erniedrige dann a um 1
x = --a;            // erniedrige a um 1, und weise das Ergebnis x zu
```



```
===============================================================================
Basiswissen: Vergleiche
===============================================================================

if (a == b)                         // Gleichheit
if (a != b)                         // Ungleichheit
if (a >= b)                         // a grösser gleich b
if (a > b)                          // a grösser b
if ((a == b) && (b == c))           // wenn a gleich b ist und b gleich c ist
if ((a == b) || (b == c))           // wenn a gleich b ist oder b gleich c ist
if (a)                              // wenn a wahr ist (bei boolean true,
                                    // sonst bedeuted wahr != 0)
if (!a)                             // wenn a falsch ist

Man beachte vor allem, dass (a == b) einen Vergleich durchführt, während
(a = b) den Wert der Variable a auf den Wert der Variable b setzt, also eine
Zuweisung durchführt. Dies kann gerade als Anfänger zu schwer am Sourcecode zu
findenden Fehlern führen.

===============================================================================
Programm: primes.cc
===============================================================================

/* Das folgende Programm berechnet die Primzahlen von 1 bis 100.
 */
#include <stdio.h>

bool isPrime (int p)
{
    for (int factor = 2; factor < p; factor++)
    {
        /* p can be divided by factor (p modulo factor is zero) */
        if (p % factor == 0)
            return false;
    }
    return true;
}

int main()
{
    for (int p = 2; p < 100; p++)
    {
        if (isPrime (p))      /* shortcut for isPrime (p) == true */
        {
            printf ("%d\n", p);
        }
    }
}
```

```
================================================================================
Aufgabe 1: vim
================================================================================
(a) Führe

# vimtutor

aus, und lerne damit vim kennen.

(b) Finde mit der Onlinehilfe heraus, wie Du zwei Dateien in verschiedenen Tabs
bearbeiten kannst.

(c) Schreibe ein Programm mit einem Syntaxfehler, und übersetze es mit der
Zeile

# g++ -o prog prog.cc -Wall -O2 2>&1 | tee /tmp/x

Der Fehler wird nun ausgegeben aber auch noch in der Datei /tmp/x gespeichert.
Probiere aus, wie vim mit

# vim -q /tmp/x

nun in die Zeile mit dem Fehler springt. Die Onlinehilfe erklärt das Feature
unter "Quickfix" näher.

================================================================================
Aufgabe 2: Strings
================================================================================
(a) Die Länge eines Strings kann man mit der Funktion strlen() berechnen. Lese
die Manualpage zu strlen(). Schreibe eine Funktion, welche auch die Länge eines
Strings berechnen kann.

(b) Versuche herauszufinden, ob Deine eigene Funktion zur Berechnung der Länge
eines Strings schneller oder langsamer ist als strlen().

Tip: Die Zeit, die ein Programm zur Ausführung benötigt, kann man mit

# time programm

herausfinden.

(c) Welche Rolle spielt in diesem Szenario die Einstellung des Optimizers?

(d) Schreibe ein Programm, welches folgende Ausgabe produziert:

H
He
Hel
Hell
Hello
Hello
Hello W
Hello Wo
Hello Wor
Hello Worl
Hello World
Hello World!
```



```
================================================================================
Aufgabe 3: einfache Funktionen
================================================================================
(a) Implementiere ein Funktion, die n! (n Fakultät) berechnet, schreibe ein
vollständiges Programm um die Funktion zu testen

Tip: Die Definition der Fakultät findet sich auf Wikipedia.

(b) Für welche n funktioniert Deine Funktion, für welche nicht? Warum?

(c) Implementiere ein Programm, welches die ersten paar Elemente der Fibonacci
Folge ausgibt.

Tip: Die Definition der Fibonacci-Folge findet sich auf Wikipedia.

================================================================================
Aufgabe 4: Operator Precedence
================================================================================
(a) Schau Dir 'http://www.cppreference.com/operator_precedence.html' an. Hier
wird beschrieben, daß bei

   x = a + b * c;

z.B. als

   x = a + (b * c);

berechnet wird, weil * eine höhere Operatorprezedenz hat als +
(Punkt-vor-Strich Regel).

Gibt es Dinge, die Dich an der Tabelle wundern?
```