Objektorientierte Programmierung
(Winter 2010/2011)

Kapitel 1: Einführung

- Computer-Grundbegriffe, Programm-Ausführung
- Geschichte der Programmierung und von C++
- Erstes Beispiel ("Hello, World")
- Bedienung von Werkzeugen zur Programmierung (Editor, Compiler, Linker, Entwicklungsumgebung)

Inhalt

Computer, Programme (1)

- Der Kern eines Computers, der die Programme ausführt, heißt CPU oder Prozessor.

  Z.B. Pentium 4, UltraSparc IV. CPU = Central Processing Unit.

- Die auszuführenden Befehle entnimmt der Prozessor dem Hauptspeicher (RAM).

  RAM = Random Access Memory. Der Hauptspeicher enthält sowohl Programme (Maschinenbefehle) als auch Daten (die von den Programmen verarbeitet werden).

- Der Hauptspeicher besteht aus vielen Speicherzellen, die über Adressen (z.B. von 0 bis 268 435 455) (das wären 256 MByte) angesprochen werden.

Computer, Programme (2)

- Jede Speicherzelle enthält ein Byte (bestehend aus 8 Bits, die jeweils 0 oder 1 sein können, dies ergibt 256 verschiedene Werte).

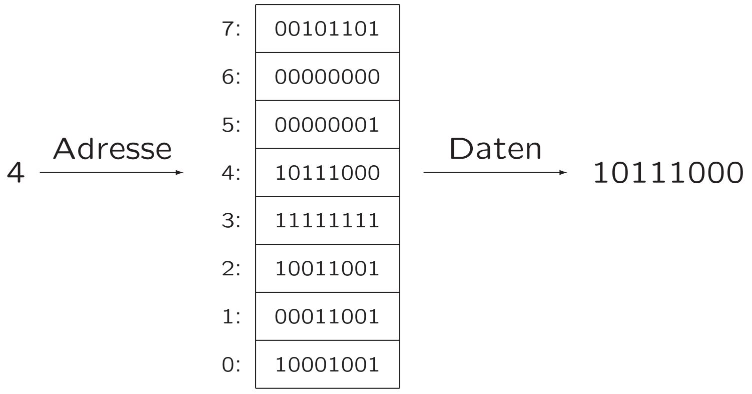
  Man kann Bytes zu größeren Einheiten (Worte) zusammenfassen, manche CPUs können auch einzelne Bits ansprechen. Deswegen kann man nicht genau sagen, was eine einzelne Speicherzelle ist.

- Man kann sich den Hauptspeicher also wie einen Schrank mit vielen Schubladen vorstellen. In jeder Schublade steckt eine Zahl zwischen 0 und 255.

  Informatiker beginnen häufig mit 0 zu zählen, da es ja eigentlich Folgen von Nullen und Einsen sind, und 00000000 einfach 0 entspricht. Das ist eine Interpretationsfrage. Z.B. auch möglich: -128 bis +127.

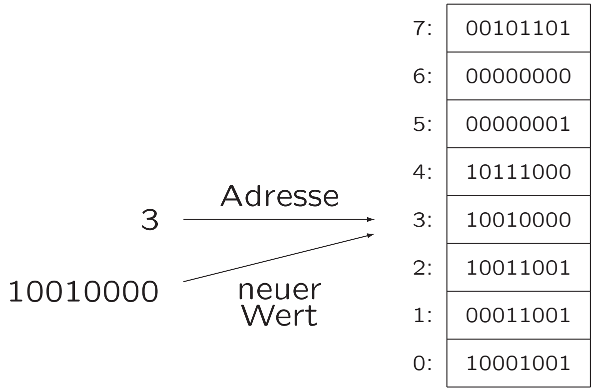
Computer, Programme (3)

Lesezugriff auf ein Byte des Hauptspeichers:



Computer, Programme (4)

Schreibzugriff auf ein Byte des Hauptspeichers:



Computer, Programme (5)

- Die CPU enthält einen “Instruction Pointer” (oder “Program Counter”), der die Adresse des nächsten auszuführenden Befehls enthält.
- Sie holt sich also den Wert aus dieser Speicherzelle.

  Eventuell auch die Werte aus einigen folgenden Speicherzellen: Viele Befehle sind länger als 1 Byte (z.B. 4–6 Byte). Typischerweise erkennt sie am ersten Byte des Befehls, wieviele Bytes noch nötig sind.
- Sie führt diesen Befehl aus, erhöht den Instruction Pointer, und holt sich den nächsten Befehl.

  Einige Befehle sind Sprungbefehle: Dann würde der Instruction Pointer auf einen neuen Wert gesetzt und nicht einfach der folgende Befehl geholt. Dies kann auch abhängig von Bedingungen geschehen.

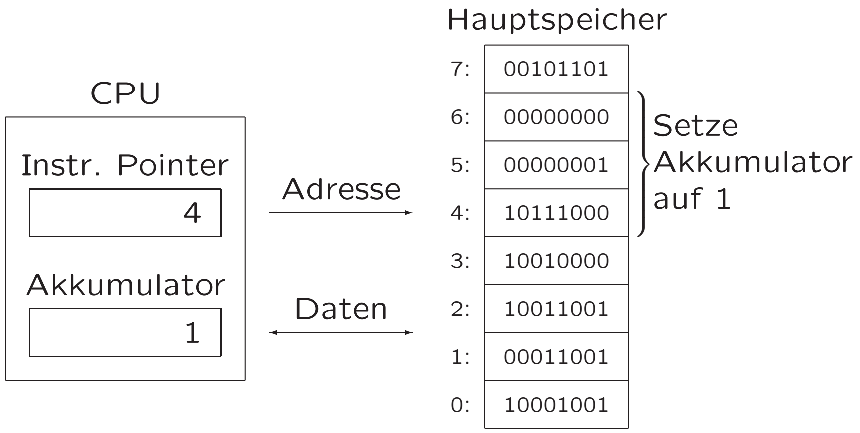
Computer, Programme (6)

- Die einzelnen Befehle sind sehr einfach, z.B.
  - ◇ Lade den Inhalt von Speicherzelle X in eine spezielle Speicherzelle in der CPU (“Akkumulator”).

    Viele CPUs haben mehr als eine solche Speicherzelle. Sie werden Register genannt. Es gibt aber normalerweise nur wenige Register. Die Register sind häufig größer als ein Byte, z.B. 4 Byte (32 Bit) oder 8 Byte (64 Bit).
  - ◇ Addiere den Inhalt von Speicherzelle X zum aktuellen Inhalt des Akkumulators hinzu.
  - ◇ Springe zu Speicherzelle X.

    Dies setzt also den Wert im Instruction Pointer, der auch ein spezielles Register der CPU ist. Der nächste Befehl wird dann von Speicherzelle X geholt.

Computer, Programme (7)



Computer, Programme (8)

- Programme bestehen also letztendlich aus Folgen von Nullen und Einsen im Hauptspeicher.
- Anfangs mußte man tatsächlich in dieser Form programmieren (Maschinensprache).
- Dann wurden Assemblersprachen erfunden. Sie sind ein 1:1 Abbild der Maschinensprache, aber mit für den Menschen lesbaren Befehlen:

```
mov AX, 1
```

Speichere den Wert 1 in das Register AX, den Akkumulator.
`mov` steht kurz für "move": Bewege den Wert 1 nach AX.

Computer, Programme (9)

- Der Assembler ist ein Programm, das solche Programme (Texte) in die internen Bitmuster für die Befehle übersetzt.
- Die Texte können auch im Hauptspeicher des Rechners repräsentiert werden.
- Dazu interpretiert man die Bytes (Bitmuster) einfach als Buchstaben/Zeichen. Z.B. wäre ein "a" nach dem ASCII-Code das gleiche Bitmuster wie die Zahl 97.

ASCII = American Standard Code for Information Interchange.

Computer, Programme (10)

	0	1	2	3	4	5	6	7	8	9
0	NULL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT
10	LF	VT	FF	CR	SO	SI	DLE	DC1	DC2	DC3
20	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS
30	RS	US	␣	!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~	DEL		

Computer, Programme (11)

- Texte (z.B. ein Assembler-Programm) können mit einem weiteren Programm, dem Editor, eingegeben und geändert werden (über die Tastatur).
- Der Inhalt des Hauptspeichers geht verloren, wenn der Computer ausgeschaltet wird. Daher wird man den Text bzw. das Programm auf die Festplatte (oder einfach Platte, engl. Disk) abspeichern.

  Zu Anfang hatte man Lochkarten oder Lochstreifen, die mit einer Art Schreibmaschine gestanzt wurden. Der Computer hat die Daten von so einem Kartenstapel (oder Lochstreifen) eingelesen und dann ausgeführt.

Computer, Programme (12)

- Assembler-Sprachen hatten drei Nachteile:
  - ◇ Die Programme liefen nur mit dem CPU-Typ, für den sie geschrieben wurden (nicht portabel).
  - ◇ Die Befehle der CPUs sind sehr einfach, kompliziertere Programme also entsprechend lang.

    Es gibt Untersuchungen, nach denen C-Programme dreimal kürzer sind als äquivalente Assembler-Programme, und auch entsprechend schneller entwickelt werden (d.h. Programmierer brauchen in diesen Sprachen die gleiche Zeit pro "Line of Code").
  - ◇ Die Programme sind schlecht strukturiert und unübersichtlich, schwierig zu warten.

    Es geschehen auch leicht Fehler, da der Assembler alles zulässt.

Computer, Programme (13)

- Daher wurden höhere Programmiersprachen erfunden, die erste erfolgreiche war Fortran (1954–57).

  Es hat auch etwas frühere Versuche gegeben.
  Fortran = FORmula TRANslator.
- Insbesondere konnte man jetzt die übliche mathematische Notation für Formeln verwenden, z.B.

  `X = 3 * Y + Z`

  Dies entspricht einer Reihe von Maschinenbefehlen: Zuerst muß man den Wert von `Y` in den Akkumulator laden. Dann muß man den Inhalt des Akkumulators mit 3 multiplizieren, dann `Z` aufaddieren, und zum Schluß den aktuellen Inhalt des Akkumulators in die für `X` reservierte Speicherzelle schreiben.

Computer, Programme (14)

- Solche Texte (Programme) wurden mit Hilfe eines Programms, des Compilers, in Maschinensprache übersetzt.

  Der erste Compiler mußte natürlich in Assembler geschrieben werden. "compile": zusammentragen, zusammenstellen (Maschinencode aus Mustern, Bibliotheken).
- Erst dadurch werden die Programme ausführbar: Die CPU selbst versteht ja nur Maschinenbefehle.
- Im Laufe der Zeit wurden viele Programmiersprachen vorgeschlagen (und Compiler für diese Sprachen entwickelt), u.a. auch die Sprache C++.

Computer, Programme (15)

- Ein Algorithmus ist ein Verfahren, mit dem eine Aufgabe gelöst werden soll.

  Ein Algorithmus ist unabhängig von einer speziellen Programmiersprache. Z.B. wurden viele Algorithmen zum Sortieren vorgeschlagen.

- Einen Algorithmus kann man in einer Programmiersprache formal aufschreiben ("codieren").
- Quellcode (engl. "source code") ist die Eingabe für den Compiler (z.B. ein Programm in C++).
- Ziel der Übersetzung ist ein ausführbares Programm (Maschinenbefehle für die Ziel-Hardware).

Computer, Programme (16)

Aufgabe:

- Angenommen, Sie haben ein Programm in C++ geschrieben und es in ein ausführbares Programm übersetzt (mit einem Compiler).
- Sie möchten Ihr Programm mit verschiedenen Eingaben testen. Müssen Sie den Compiler jedesmal neu aufrufen?
- Ihr Programm ist so nützlich, dass Sie im Internet veröffentlichen wollen. Welche Vor- und Nachteile hat es, wenn Sie Quellcode oder das ausführbare Programm auf Ihre Webseite stellen?

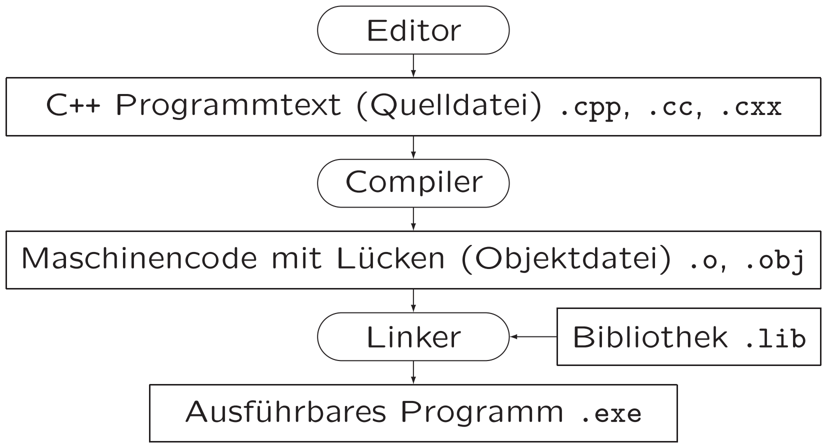
Computer, Programme (17)

- Größere Programme bestehen meist aus mehreren Teilen (Module), und verwenden Funktionen aus Bibliotheken (allgemein nützlicher Programmcode).

  Z.B. wäre es doch unsinnig, wenn jeder Programmierer sich selbst ein Programmstück für die Sinus-Berechnung schreiben müßte. Auch die Ein-/Ausgabe geschieht über Bibliotheksfunktionen.

- Die Module können mit dem Compiler getrennt übersetzt werden, und die Bibliothek wird gleich übersetzt ausgeliefert.
- Man benötigt dann einen Linker, der die verschiedenen Stücke von Programmcode zusammensetzt.

Computer, Programme (18)



Computer, Programme (19)

- Es gibt noch weitere Programme zur Unterstützung der Programmentwicklung, z.B.
  - ◇ Ein Debugger erlaubt es, Programme Schritt für Schritt auszuführen, um Fehler zu finden.

    Fehler in Programmen werden Bugs genannt (Wanze, Käfer). Das Wort wurde aber schon für Edison für kleine Schwierigkeiten/Probleme verwendet (vor Erfindung des Computers).
  - ◇ Mit einem Programmcode-Browser kann man die Definition eines verwendeten Symbols finden.

    Und umgekehrt von der Definition zu den Verwendungen. ("cross reference").

Computer, Programme (20)

- Bei Projekten mit mehreren Modulen benötigt man ein Programm zur Verwaltung der Abhängigkeiten zwischen diesen Teilen.

  Was muß neu übersetzt werden, wenn eine bestimmte Quelldatei geändert wurde?
- Software, die Editor, Compiler, Linker und weitere Funktionen zur Unterstützung der Programmentwicklung enthält, heißt Entwicklungsumgebung (IDE, "Integrated Development Environment").

Betriebssystem, Dateien (1)

- Wenn Daten länger als ein Programmlauf benötigt werden (und auch das Ausschalten des Rechners überstehen sollen), speichert man sie heute meistens auf einer Platte ("disk").
- Platten sind logisch gesehen ähnlich zum Hauptspeicher: Man kann Daten unter Adressen ablegen, und später unter dieser Adresse wiederfinden.
- Die Zugriffe sind aber viel langsamer und die adressierbaren Einheiten größer (512–8192 Byte).

  Die Adressen sind nicht nur eine Zahl, sondern beschreiben den Speicherort auf den Magnetscheiben: Zylinder, Spur (Kopf), Sektor.

Betriebssystem, Dateien (2)

- Auf der Platte können mehrere Texte, Programme, oder andere Datensammlungen (Dateien) stehen.
- Eine Datei ist einfach eine Folge von Bytes (die wiederum aus Bits bestehen).
- Anfangs mußte man genau sagen, wo auf der Platte die zu lesenden Daten standen: unpraktisch.
- Daher hat man Datenstrukturen entwickelt, mit denen Namen (Dateinamen) auf Speicherorte auf der Platte abgebildet werden können.

Betriebssystem, Dateien (3)

- Die Verwaltung von Dateien auf der Platte ist eine der Funktionen des Betriebsystems (z.B. Windows, Linux, Solaris).
- Programme können durch Aufruf einer Funktion des Betriebssystems
  - ◇ Daten von einer Datei auf der Platte in den Hauptspeicher laden (lesen),
  - ◇ bzw. umgekehrt vom Hauptspeicher in eine Datei schreiben.

    Es muß dabei auch nicht immer die komplette Datei eingelesen oder geschrieben werden.

Betriebssystem, Dateien (4)

- Da es auf der Platte sehr viele Dateien geben kann, werden sie in Ordnern (Dateiverzeichnissen, Directories) strukturiert.
- Ordner können selbst wieder Ordner enthalten, so daß eine hierarchische Struktur entsteht.
- Beim Betriebssystem Windows stehen auf oberster Ebene die Laufwerke, wie z.B. `C:`.

  Man kann eine Platte auch in mehrere Abschnitte (Partitionen) unterteilen, die getrennt als Laufwerk angezeigt werde (z.B. Platten-Laufwerke `C:` und `D:` in einem Rechner mit nur einer Platte). Auch andere Speichermedien, wie CD-ROM/DVD oder USB-Stick werden als Laufwerke behandelt.

Betriebssystem, Dateien (5)

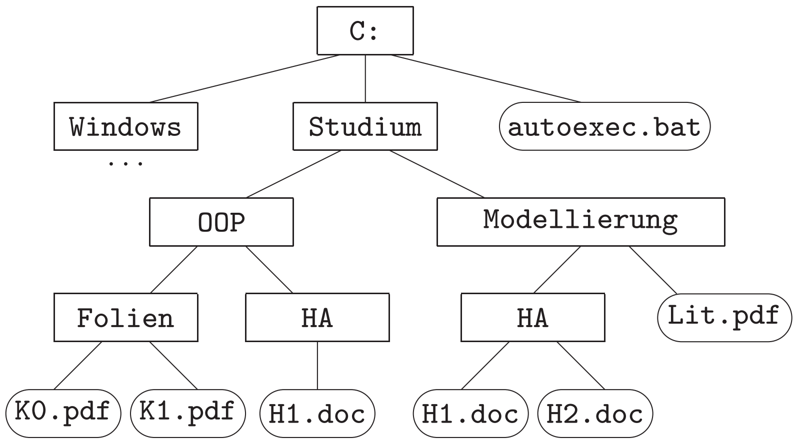
- Beim Betriebssystem UNIX (Linux, Solaris, ...) gibt es nur ein Hauptverzeichnis.

  Laufwerke können an beliebiger Stelle als Unterverzeichnisse in die Hierarchie integriert werden.
- Man kann Dateien über den vollständigen Namen (mit allen übergeordneten Ordnern) identifizieren:

  `C:\Studium\OOP\HA\H1.doc` (Windows)
  `/home/brass/OOP/HA/h1.tex` (Unix)

  Solche Dateibezeichnungen heißen absolute Pfadnamen. Es gibt ein aktuelles Verzeichnis, von dem aus relative Pfadnamen möglich sind: `H1.doc`, falls gerade `C:\Studium\OOP\HA` das aktuelle Verzeichnis ist, `HA\H1.doc`, falls `C:\studium\OOP` das aktuelle Verzeichnis ist, `..\OOP\HA\H1.doc`, falls `C:\Studium\Modellierung` akt. Verzeichnis.

Betriebssystem, Dateien (6)



Betriebssystem, Dateien (7)

- Obwohl es zwei Ordner `HA` und zwei Dateien `H1.doc` gibt, sind diese jeweils unterschiedlich, weil sie an verschiedenen Positionen in der Ordnerhierarchie ("Verzeichnisbaum") stehen.

  Nur innerhalb eines Ordners müssen die Dateinamen eindeutig sein.

- Es ist üblich, daß Dateinamen eine durch Punkt abgetrennte Endung haben ("Extension"), die die Art der Daten in dem Dokument anzeigt, z.B.:
  - ◇ `.pdf`: Textdokument (für Acrobat Reader).
  - ◇ `.cpp`: C++ Programm (verschiedene Compiler).

Betriebssystem, Dateien (8)

- Während das Betriebssystem anfangs eher eine Bibliothek von häufig verwendeten Funktionen war, ist es inzwischen auch eine Kontrollinstanz.
- Man will dem einzelnen Benutzerprogramm nicht mehr die volle Kontrolle über die Hardware geben.
- Ein Rechner wird eventuell von mehreren Benutzern verwendet, man darf dann nicht auf Dateien anderer Benutzer zugreifen.

  Es sei denn, diese Benutzer haben sich durch Vergabe entsprechender Zugriffsrechte damit einverstanden erklärt.

Betriebssystem, Dateien (9)

- Auf einem Rechner laufen heute mehrere Programme gleichzeitig, Störungen dazwischen müssen vermieden werden (Kontrolle des Betriebssystems).

  Die CPU führt tatsächlich nur eins dieser Programme gleichzeitig aus, aber sie (bzw. das Betriebssystem) schaltet so schnell zwischen den Programmen hin- und her, daß es für den Benutzer so aussieht, als würden die Programme gleichzeitig laufen. Oft warten ohnehin alle bis auf ein Programm auf Eingaben oder andere Ereignisse. Ein in Ausführung befindliches Programm heißt Prozess.

- Das Betriebssystem verwaltet Resourcen wie z.B. einen Drucker: Es können ja nicht mehrere Programme gleichzeitig Daten zum Drucker schicken.

Inhalt

Historische Bemerkungen (1)

- C++ ist eine Erweiterung der Sprache C.

  (Fast) jedes C-Programm ist auch ein C++-Programm.

- C wurde 1969–1973 von Dennis Ritchie in den Bell Labs entwickelt (kleinere Änderungen 1977-1979), das Lehrbuch von Kernighan/Ritchie erschien 1978.

  Siehe: [http://cm.bell-labs.com/cm/cs/who/dmr/chist.html]
  Vorläufer waren die Sprachen BCPL und B. Gehört zur Algol-Familie.

- C wurde parallel mit dem Betriebssystem UNIX entwickelt (Implementierungssprache von UNIX).

  Die Bell Labs hatten vorher bei dem Multics Projekt mitgearbeitet, aber das wurde ihnen zu groß/verspätet (Ausstieg 1969). Die für UNIX zuerst benutzte PDP-7 hatte 8K 18-bit Worte RAM.

Historische Bemerkungen (2)

- C erreichte große Verbreitung.

  Es war eine kompakte/kleine Sprache, deren Befehle sehr direkt in Befehle der CPU übersetzt werden konnten (hardware-nah, effizient).

- Der ANSI-Standard für C erschien 1989 (ISO C90).

  Er führte zu weiteren Änderungen in der Sprache (insbesondere die Deklaration von Parametern in Funktionsprototypen).

- Für große Programme hat sich eine objektorientierte Struktur als meistens sehr übersichtlich herausgestellt. Dies wird in C nicht direkt unterstützt.

  Man kann in (fast) jeder Sprache, auch in C, objektorientiert programmieren, aber das erfordert besondere Disziplin.

Historische Bemerkungen (3)

- Als erste objektorientierte Programmiersprache gilt heute Simula-67.

  Wie der Name schon sagt, war Simula für Simulationen gedacht, und wurde 1967 auf einer Konferenz vorgestellt (entwickelt von Ole-Johan Dahl and Kristen Nygaard in Oslo). Der Erfinder von C++, Bjarne Stroustrup, ist von Simula-67 wesentlich beeinflusst worden.

- Smalltalk-80 leiste einen wesentlichen Beitrag zur Verbreitung der Idee der Objektorientierung.

  Entwickelt in den 70er Jahren am XEROX PARC Forschungszentrum.

- Besonders graphische Benutzeroberflächen können objektorientiert gut entwickelt werden.

Historische Bemerkungen (4)

- Bjarne Stroustrup begann 1979, in den Bell Labs an einer objektorientierten Erweiterung von C zu arbeiten.

  Die Sprache hieß zuerst "new C", dann "C with Classes", und ab 1983 "C++". Eine erste kommerzielle Implementierung erschien 1985. 1989 erschien Version 2.0, 1990 das "Annotated C++ Reference Manual".

- Der ANSI-ISO Standard für C++ erschien 1998, eine neue Auflage 2003.

  Die Sprache C++ hat sich bis zum Erscheinen des ersten Standards wesentlich geändert, ältere Literatur ist heute kaum noch brauchbar. Eine Neuerung war auch die Hinzufügung der "Standard Template Library" STL entwickelt von Alexander Stepanov und anderen (HP, Veröffentlichung 1994).

Historische Bemerkungen (5)

- 2005 erschien ein Technischer Bericht mit einem Vorschlag für eine zukünftige, erweiterte Standard-Bibliothek (TR1).

  Viele neuere Compiler unterstützen diese Vorschläge bereits. Sie basieren auf der Boost C++-Bibliothek [http://www.boost.org/].

- Am 21.08.2010 erschien ein Entwurf für die nächste Version des Standards (erwartet für Ende 2011).

  Insbesondere ist eine bessere Unterstützung von Multithreading nötig.
  [http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2010/n3126.pdf]
  [http://en.wikipedia.org/wiki/C%2B%2B0x]

Historische Bemerkungen (6)

- Es gibt (natürlich) auch Konkurrenz zu C++:
  - ◇ Objective-C

    Eine andere objektorientierte Erweiterung von C, von Brad Cox ca. 1980-86 entwickelt, wurde durch Verwendung auf den NeXT-Rechnern bekannt, wird heute vor allem auf Mac-Computern und in den GnuStep-Bibliotheken eingesetzt.
  - ◇ Java

    Entwickelt von James Gosling und anderen bei Sun seit 1991, Java 1.0 erschien 1995. Wurde durch die Verwendung in Web-Browsern bekannt.
  - ◇ C#

    Entwickelt von Anders Hejlsberg und anderen bei Microsoft als Teil der .NET-Initiative. Erschien 2001.

Inhalt

Erstes Beispiel (1)

```
// Dies ist das klassische erste Programm.
// Es druckt den Text  Hello, world!

#include <iostream>
using namespace std;

int main()
{
        cout << "Hello, world!\n";
        return 0;
}
```

Erstes Beispiel (2)

- Die ersten zwei Zeilen sind ein Kommentar: Sie sind nur für den menschlichen Leser des Programms gedacht.
- Wenn der Compiler “//” sieht, ignoriert er alles bis zum Ende der jeweiligen Zeile.

  Man könnte da auch vollkommen unzutreffende Bemerkungen hinschreiben. Das würde uns zwar verwirren, aber den Compiler würde es nicht stören: Er erzeugt exakt den gleichen Maschinencode, das Programm würde sich also bei der Ausführung identisch verhalten.
- Es gilt als guter Stil, eine gewisse Menge Kommentar zum besseren Verständnis des Programms zu schreiben.

Erstes Beispiel (3)

- Das Programm benutzt Bibliotheksfunktionen zur Ausgabe des Textes.
- Die Bezeichner/Symbole “`cout`” und “<<” sind in die Sprache C++ nicht eingebaut.

  << eigentlich schon, aber nur für Zahlen (“Shift”), nicht zur Ausgabe.
- Der Compiler weiß also nicht automatisch, was sie bedeuten.
- Die Information, was diese Dinge überhaupt sind (ein Objekt, eine Funktion, etc.) steht in der Datei “`iostream`”.

Erstes Beispiel (4)

- Die Datei “`iostream`” wird mit dem Compiler geliefert, ist aber selbst nicht Teil des Compilers.

  Falls Sie sich die Datei mal anschauen wollen: In einer Entwicklungsumgebung wie MS Visual Studio sind unter “Options” die Verzeichnisse festgelegt, in denen der Compiler nach solchen Dateien sucht. Wenn Sie einen Compiler wie den `gcc` benutzen, rufen Sie ihn mit der Option `-v` (“verbose”) auf: “`g++ -v hello.cpp`”. Er gibt dann die verwendeten Dateien mit absolutem Pfadnamen aus. Allerdings ist der C++-Code in `iostream` nur etwas für Fortgeschrittene.
- Sie könnten sich auch eigene Funktionen zur Ein-/Ausgabe schreiben, und `iostream` nicht verwenden.

  Oder die klassische C-Bibliothek zur Ein-/Ausgabe verwenden.

Erstes Beispiel (5)

- Die Anweisung

  ```
  #include <iostream>
  ```

  teilt dem Compiler mit, daß er die Datei “`iostream`” an dieser Stelle lesen soll.

  Wenn er damit fertig ist, liest er Ihren Programmtext weiter. Formal hat das den gleichen Effekt, als wenn man einfach den Inhalt der Datei “`iostream`” an dieser Stelle in Ihr Programm einfügen würde. Englisch “include”: einschließen, umfassen, enthalten.
- Nachdem er die Datei gelesen hat, weiß er, was die Symbole “`cout`” und << bedeuten.

  Ganz stimmt das nicht: siehe nächste Folie.

Erstes Beispiel (6)

- Genauer sind in der Datei "`iostream`" nur Symbole wie "`std::cout`" definiert.
- `std` ist ein Namensraum (engl. "namespace").
- Auf diese Art werden Symbole aus der Bibliothek von Symbolen aus Ihrem Programm getrennt.

  Die Bibliothek definiert sehr viele Symbole. Sie können schlecht alle kennen. Es könnte sonst aber zu Namenskonflikten kommen, wenn Sie zufällig eine Funktion gleichen Namens definieren.
- Namensräume sind auch wichtig, wenn man mehrere Bibliotheken benutzt, die zufällig gleiche Symbole definieren.

Erstes Beispiel (7)

- Mit der Anweisung

  ```
  using namespace std;
  ```

  teilen Sie dem Compiler mit, daß Sie auf die Symbole aus dem Namensraum "`std`" auch ohne den Präfix "`std::`" zugreifen wollen.

  Genauer gilt dies nur, wenn in Ihrem Programm nicht zufällig ein Symbol gleichen Namens definiert ist. Sie brauchen so also nicht unbedingt alle Symbole aus dem Namensraum "`std`" zu kennen.
- Ohne `using` müßten Sie später schreiben:

  ```
  std::cout << "Hello, world!\n";
  ```

  Funktionen (<<) werden auch im Namespace ihrer Argumente gesucht.

Erstes Beispiel (8)

- Mit den Informationen aus `iostream` ist der Compiler in der Lage, die Benutzung der Symbole `cout` und `<<` in Maschinencode zu übersetzen.
- Der erzeugte Maschinencode ist aber unvollständig.
- Die Festlegungen in `iostream` sind nur ein Teil des Programmcodes, der für die Standard-Bibliothek geschrieben wurde ("Deklaration der Symbole").
- Der Hauptteil, die eigentliche Implementierung der Funktionen, liegt in bereits compilierter Form vor (Maschinecode, Bibliothek).

Erstes Beispiel (9)

- Der Linker fügt den Maschinencode, den der Compiler für Ihr Programm erzeugt, mit dem Maschinencode aus der Bibliothek zusammen.
- So entsteht das vollständige ausführbare Programm.

  Bei dynamischem Linken (heute üblich) geschieht das eigentliche Zusammenfügen erst beim Ausführen des Programms. Der Linker prüft dann nur, daß die aufgerufenen Funktionen auch in der Bibliothek definiert sind.
- Da die Entwicklungsumgebung normalerweise automatisch den Linker mit der Standard-Bibliothek aufruft, ist dieser Schritt anfangs eher implizit.

Erstes Beispiel (10)

- Das Programmstück

```
int main()
{
    ...
}
```

ist Ihre erste Prozedur/Funktion.

- Prozeduren sind benannte Stücke Programmcode: Man ruft sie unter ihren Namen auf, um den Programmcode (das Stück in `{...}`) auszuführen.
- Der Name dieser Prozedur ist “`main`”.

Erstes Beispiel (11)

- Die Prozedur `main` ist speziell:
  - ◇ Alle anderen Prozeduren müssen Sie in Ihrem Programm explizit aufrufen.

    Mehr oder weniger: In C++ gibt es auch nicht offensichtliche Aufrufe, in der ursprünglichen Sprache C dagegen nicht.
  - ◇ `main` wird dagegen automatisch vom Betriebssystem aufgerufen, wenn es Ihr Programm ausführt.

    Genauer behandelt der Linker das Symbol `main` speziell: Er muß in der ausführbaren Datei vermerken, wo die Programmausführung beginnen soll und nimmt dazu die Adresse, an der der Programmcode von `main` beginnt. Noch genauer gibt es ein wirkliches Hauptprogramm, das der Linker immer dazu tut, und das dann seinerseits die Prozedur `main` aufruft.

Erstes Beispiel (12)

- Also:
  - ◇ Jedes Programm hat eine Prozedur “`main`”.

    Besonders einfache Programme, wie das erste Beispiel, bestehen überhaupt nur aus dieser einen Prozedur.
  - ◇ Die Programmausführung beginnt hier.

    Programmiersprachen wie Pascal hatten ein explizites Sprachkonstrukt für das Hauptprogramm. C versuchte minimalisitisch zu sein, und darauf zu verzichten. Das ist gelungen durch die Verwendung einer speziell benannten Prozedur. Wenn Sie “`Hauptprogramm`” statt “`main`” schreiben würden, würde es nicht funktionieren. Diese Lösung bot sich auch an, da C im Gegensatz zu Pascal keine Schachtelung von Prozeduren zulässt.

Erstes Beispiel (13)

- Das Wort “`int`” steht für “integer”, Englisch für “ganze Zahl”.
- Vor dem Namen der Prozedur schreibt man den Typ des Rückgabewertes der Prozedur (Ergebniswert).
- Der Aufrufer der Prozedur bekommt einen Wert von diesem Typ geliefert, nachdem die Prozedur ausgeführt wurde.
- Im Beispielprogramm geschieht die Rückgabe durch die Zeile

```
return 0;
```

Erstes Beispiel (14)

- Der Aufrufer erhält von diesem Programm also immer die Zahl 0 zurückgeliefert.
- Der Aufrufer ist in diesem Fall das Betriebssystem.

  Bzw. ein Programm, das andere Programme aufruft.
- Es interpretiert 0 als "fehlerfrei ausgeführt".
- Kleine positive Zahlen würden dagegen für einen Fehler stehen.

  Z.B. würde ein Programm, das mehrere Programme nacheinander aufruft, vermutlich abbrechen, wenn ein Programm einen von 0 verschiedenen Wert zurückliefert.
  Negative Werte sollten vermieden werden.

Erstes Beispiel (15)

- Der eigentliche Kern des Programms ist die Zeile:

  ```
  cout << "Hello, world!\n";
  ```
- `cout` ist ein Objekt, das den Standard-Ausgabestrom repräsentiert (Bildschirm bzw. Terminalfenster).
- `"Hello, world!\n"` ist eine Zeichenketten- (String-) Konstante, so wie `0` eine Zahlkonstante ist.
- Der Compiler ersetzt `\n` durch ein spezielles Steuerzeichen, den Zeilenvorschub (Linefeed, LF).

  Man kann so auch nicht direkt sichtbare Zeichen eingeben. Es gibt noch mehr Kombinationen aus Rückwärtsschrägstrich \ und weiteren Zeichen. Will man \, so muß man \\ schreiben.

Erstes Beispiel (16)

- Eine Bemerkung zur Portabilität:
  - C (und C++) stammen aus der UNIX-Welt.
  - In UNIX werden Zeilen mit dem Steuerzeichen Linefeed beendet (ASCII-Code 10).
  - Unter Windows wird dagegen die Kombination von Carriage Return (CR, 13) und LF verwendet.
  - Damit die existierenden Programme ohne Änderung auch unter Windows funktionierten, wird unter Windows tief in den Ausgabebibliotheken jedes LF in CR+LF umgewandelt.

    Falls nicht erwünscht: bei Dateieröffnung Binärmodus wählen.

Erstes Beispiel (17)

- `<<` ist ein Operator, dem eine Bibliotheksprozedur mit Namen "`operator<<`" hinterlegt ist.
- Die Zeile

  ```
  cout << "Hello, world!\n";
  ```

  ist äquivalent zu

  ```
  operator<<(cout, "Hello, world!\n");
  ```

  Genauer könnte es alternativ auch äquivalent sein zu `cout.operator<<("Hello, world!\n");`
  Der Entwickler von `iostream` entscheidet, welche Variante funktioniert (meistens beide).
- Natürlich sieht die Originalsyntax schöner aus.

Erstes Beispiel (18)

- Es wird also die Bibliotheksprozedur `operator<<` mit folgenden Eingabewerten aufgerufen:
  - `cout` (Ziel der Ausgabe: Bildschirm)
  - `"Hello, world!\n"` (Auszugebener Wert)
- Diese benutzt dann eine Funktion des Betriebssystems, um die einzelnen Zeichen am Ende wirklich auszugeben.

  Möglicherweise leitet das die Daten zunächst weiter an ein Programm, das ein Konsolenfenster darstellt. Das ruft dann wieder das Betriebssystem auf, um die Daten auf den Bildschirm zu zeichnen.

Erstes Beispiel (19)

Keine Panik!

- Es hört sich ziemlich kompliziert an, aber diese Dinge sind bei allen Programmen gleich, und mit der Zeit gewöhnt man sich daran, und denkt nicht mehr an die ganzen Details.

  Tatsächlich wäre C einfacher: Dann hätten wir nicht über Namenspaces und Operatoren sprechen müssen. C++ ist eine relativ komplexe Sprache, die aber bei großen Projekten Stärken hat. Außerdem sollen in dieser Vorlesung viele Konzepte vorgestellt werden, damit man andere Sprachen später leichter lernen kann.
- Alles wird später noch einmal diskutiert, und wird dann noch klarer werden.

Inhalt

Beispiel ausprobieren (1)

- Wenn man das Beispielprogramm nun an einem Rechner ausprobieren will, hat man zunächst die Wahl zwischen
  - einzelnen Kommandozeilen-Werkzeugen (Programmen wie Editor und Compiler), oder
  - einer Entwicklungsumgebung (“Integrated Development Environment”, IDE).

    Eine Entwicklungsumgebung enthält normalerweise den Editor, aber Compiler, Linker und ggf. andere Werkzeuge sind oft getrennte Programme, die dann die IDE aufruft. Z.B. würde Eclipse/CDT unter Linux auch den g++-Compiler verwenden.

Beispiel ausprobieren (2)

- Vorteile der Benutzung einzelner Werkzeuge:
  - ◇ Man sieht besser, was genau vorgeht.
  - ◇ Braucht keine graphische Oberfläche.

    Wenn man per `ssh`/`PuTTY` remote auf einem Linux-Rechner arbeitet (z.B. `anubis.informatik.uni-halle.de`), und selbst z.B. einen Windows Rechner (ohne X11) benutzt, hat man keine graphische Benutzeroberfläche. Die ist für die meisten IDEs aber notwendig.
  - ◇ Editor frei wählbar.
  - ◇ Für ungewöhnliche Situationen (z.B. Precompiler für Embedded SQL) ist die IDE eventuell nicht anwendbar oder schwerer anzupassen.
  - ◇ IDEs haben oft lange Startup-Zeiten.

Beispiel ausprobieren (3)

- Vorteile der Benutzung einer IDE:
  - ◇ Steuerung über graphische Oberfläche.

    Man braucht nicht im Handbuch nach Optionen zu suchen, dafür muß man eventuell Menüs und Dialogboxen durchforsten.
  - ◇ Per Mausklick auf eine Fehlermeldung kommt man im Editor direkt zu der Zeile, in der der Compiler den Fehler bemerkt hat.
  - ◇ Die Deklaration eines Symbols (Typ einer Variable, Argumente einer Funktion) wird als Tooltip angezeigt, wenn man das Symbol benutzt.

    Natürlich abhängigig von IDE. Ggf. auch Verlinkung mit Doku.

Beispiel ausprobieren (4)

- Vorteile der Benutzung einer IDE, Forts.:
  - ◇ Bei größeren Projekten, die aus vielen Quelldateien bestehen, werden Abhängigkeiten automatisch verwaltet und nur die von einer Änderung betroffenen Dateien neu übersetzt.

    Hierzu gibt es aber auch Kommandozeilen-Werkzeuge: `make` u.a.
  - ◇ Automatische Code-Vervollständigung, Anlegen eines Coderahmens für Klassen.

    Dies ist aber am Anfang auch ein Nachteil: In der Klausur stehen solche Funktionen nicht zur Verfügung. Wenn man es oft genug selbst aufschreibt, prägt sich die Syntax vermutlich besser ein, als wenn mit einem Klick das Gerippe/eine Schablone fertig ist.

Editor (1)

- Zum Eingeben und Ändern eines Programmtextes benötigt man einen Texteditor.

  Man kann nicht `Word` verwenden, das speichert die Texte mit Formatierungsangaben, die der Compiler nicht versteht. Es müssen einfach nur die reinen Zeichen des Programmtextes in der C++-Datei enthalten sein. `notepad` würde das leisten, ist aber etwas primitiv.
- Hier gibt es eine große Auswahl, z.B.
  - ◇ `gedit`

    Ist bei Linux-Systemen mit GNOME-Desktop (z.B. Ubuntu) immer dabei. Einfach mit Maus und Menüs zu bedienen, geht aber nicht für Remote Login (weil graphische Oberfläche nötig).

Editor (2)

- Beispiele für Editoren (Forts.):
  - ◇ `vi/gvim`

    `vi` ist wegen der Trennung von Kommandomodus und Einfügemodus etwas gewöhnungsbedürftig, aber durchaus mächtig und bei UNIX/Linux-Systemen immer mit dabei. Man kann ihn auch bei Remote Login verwenden (eine Text-Schnittstelle reicht). Es gibt auch eine Variante `gvim` mit graphischer Benutzeroberfläche.
  - ◇ `emacs`

    Bekannter, mächtiger Editor mit Tastatur- und graphischer Bedienung. In einem Lisp-Dialekt programmierbar. Er kann mit vielen Werkzeugen kombiniert werden, und wird so zu einer Art IDE. Wenn X11 (Graphiksystem u.a. von Linux/UNIX) zur Verfügung steht, öffnet der Aufruf `emacs` *Datei* automatisch ein Fenster, und man kann den Editor dann mit der Maus bedienen. Wenn Sie zu Hause unter Linux arbeiten, geht das auch remote mit "`ssh -X anubis.informatik.uni-halle.de`".

Editor (3)

- Beispiele für Editoren (Forts.):
  - ◇ `kate`

    KDE Advanced Text Editor, Grundlage der KDevelop IDE.
  - ◇ `pico/nano`

    Wenn man mit Remote Login ohne X11 arbeiten muss, wären dies einfache Editoren mit Tastatur-Schnittstelle, die unten immer die verfügbaren Kommandos anzeigen. Es sind allerdings nicht besonders viele. `^X` bedeutet gleichzeitiges Drücken von ⟨`Crtl`⟩ (bzw. ⟨`Strg`⟩) und `X`.
  - ◇ `nedit` (NEdit)

    Editor mit graphischer Benutzeroberfläche, er soll für Windows-Nutzer vertrauter wirken.

Editor vi (1)

- Es gibt u.a. Kommandomodus und Einfügemodus.

  Im Kommandomodus werden Buchstaben als Kommandos interpretiert, man kann also Text nicht einfach eintippen (erst nach Wechsel in den Einfügemodus). Dies ist gewöhnungsbedürftig, hat aber den Vorteil, dass viele Kommandos zur Verfügung stehen (und oft nur ein Tastendruck). Beim Start ist der Editor im Kommandomodus.

  Es hört sich am Anfang alles etwas kompliziert an, aber mit der Zeit braucht man nicht mehr über die Kommandos nachzudenken, und kann sich auf den Inhalt konzentrieren. Sie müssen den `vi` nicht unbedingt jetzt lernen, er ist nicht prüfungsrelevant. Sie müssen nur einen im Pool vorhandenen Editor bedienen können, dass kann z.B. auch der `gedit` sein. Wenn Sie aber zu Hause an einem Windows-Rechner arbeiten, und per Remote Login die geforderte Portabilität auf Linux testen wollen, könnte es nützlich sein, einen Editor bedienen zu können, der ohne graphische Oberfläche auskommt (z.B. auch `emacs`). Für UNIX/Linux-Experten wären `vi`-Basiskenntnisse allerdings Pflicht.

Editor vi (2)

- Es gibt immer eine aktuelle Position im Text (Cursor, blinkender Kasten um ein Zeichen).

  Bei der graphischen Variante `gvim` ändert sich die Form des Cursors im Einfügemodus: Dann ist es ein Strich hinter der aktuellen Position, also vor der Stelle, an der eingefügt wird.
- Die unterste Zeile ist die Statuszeile.

  Im rechten Bereich werden meist Zeilennummer und Spaltennummer angegeben, sowie ganz rechts die Nummer der mittleren Zeile im Vergleich zur Gesamtzeilenzahl des Textes in Prozent (bzw. `Top`, falls die erste Zeile angezeigt wird, `Bot` bei Sichtbarkeit der letzten Zeile und `All`, wenn die ganze Datei auf den Bildschirm passt). Links unten steht "`-- INSERT--`" im Einfügemodus. Die Statuszeile wird auch für spezielle Kommandos verwendet (s.u.).
- Aufruf: `vi` ⟨*Dateiname*⟩

Editor vi (3)

Basis-Kommandos:

- ⟨Esc⟩: zurück in den Kommandomodus.

  ⟨Esc⟩ bedeutet die Taste “Esc” / “Escape”. Durch Drücken von ⟨Esc⟩ können Sie auch ein eventuell begonnenes Kommando abbrechen. Wenn Sie nicht genau wissen, ob/welche Taste Sie schon gedrückt haben, können Sie mit ⟨Esc⟩ in einen definierten Zustand gelangen.

- a: Einfügen von Text hinter der aktuellen Cursor-Position (“append” / “anhängen”).

  Mit “a” wechselt man in den Einfügemodus. Dann kann man Text eingeben, auch mehrere Zeilen durch ⟨Enter⟩ / ⟨Return⟩ / ↵ getrennt. Muss man etwas korrigieren, kann man mit ⟨Backspace⟩ im Einfügemodus das jeweils letzte Zeichen löschen. Steuerzeichen kann man durch Voranstellen von ⟨Crtl⟩+V eingeben. Mit ⟨Esc⟩ beendet man die Eingabe (und ist dann wieder im Kommandomodus).

Editor vi (4)

Wechsel in Einfügemodus, Forts.:

- i: Einfügen von Text vor der aktuellen Cursor-Position (“insert”, “einfügen”).
- A: Einfügen am Ende der aktuellen Zeile.
- I: Einfügen am Anfang der aktuellen Zeile.

  Wobei Anfang hier nach der Einrückung durch Leer- und Tabulatorzeichen bedeutet.

- o: Einfügen von Text in einer (oder mehreren) neuen Zeilen nach der aktuellen Zeile (“open”).
- O: Einfügen vor der aktuellen Zeile.

Editor vi (5)

Cursor-Positionierung:

- ←, h: ein Zeichen nach links.

  Je nach Terminalprogramm werden für die Pfeil-Tasten eventuell Codes (Escape-Sequenzen) übermittelt, die der Editor nicht versteht. Deswegen ist es praktisch, daß man den Cursor (die aktuelle Position im Text) auch mit Buchstaben steuern kann. Die funktionieren immer (und liegen günstig für Zehn-Finger-Tippen). Wenn die Cursortasten gut unterstützt werden, kann man damit auch im Einfügemodus die Position ändern. Normalerweise muß man den Einfügemodus erst mit ⟨Esc⟩ beenden, bevor man sich frei im Text bewegen kann.

- ↓, j: eine Zeile nach unten.
- ↑, k: eine Zeile nach oben.
- →, l: ein Zeichen nach rechts.

Editor vi (6)

Cursor-Positionierung, Forts.:

- ⟨Ctrl⟩+F: Eine Seite vorwärts (“forward”).

  Man muß die Tasten “Crtl” (Control) und “F” gleichzeitig drücken. Auf deutschen Tastaturen heißt es “Strg” (Steuerung) statt “Ctrl”.

- ⟨Ctrl⟩+B: Eine Seite zurück (“backward”).
- ⟨Enter⟩: zum Anfang der nächsten Zeile.

  Hinter eine Einrückung aus Leer-/Tabulatorzeichen (auf echten Text).

- w: zum Anfang des nächsten Wortes.

  Wort ist hier eine Folge von Buchstaben und Ziffern oder eine Folge von Sonderzeichen. Bei der Variante W ist Wort etwas umfassender als “alles zwischen Leerzeichen” definiert. Zurück: b, B.

Editor vi (7)

Cursor-Positionierung, Forts.:

- `nG`: Springe zu Zeile n ("go to").

  Z.B. würde man mit "58G" auf Zeile 58 wechseln, wenn die Datei mindestens 58 Zeilen hat, sonst auf die letzte Zeile. Bei den meisten anderen Kommandos wäre eine Zahl vor dem Kommando ein Wiederholungsfaktor. Wenn man versehentlich eine Ziffer gedrückt hat, gebe man ⟨Esc⟩ ein, um den Widerholungsfaktor zu löschen.

- `G`: Springe an das Ende der Datei (letzte Zeile).
- `n|`: Zu Spalte n.
- `|`: Zur ersten Spalte (Anfang der Zeile).
- `␣` (Leertaste): Nächstes Zeichen.

  Im Gegensatz zu "l" auch über Zeilengrenzen hinweg.

Editor vi (8)

Marken:

- Um später schnell zu einer bestimmten Position im Text zurückkehren zu können, kann man Zeilen mit den Marken a bis z (und A bis Z) markieren.

  Der Text wird dadurch nicht verändert, es sind nur Speicherstellen, in denen sich der Editor Positionen im Text (bestimmte Zeilen) merkt. Wenn man den Editor verläßt, oder eine andere Datei in den Editor läd, werden die Marken vergessen. Bei Einfügungen oder Löschungen vor der markierten Zeile wandert die Marke mit (so dass die gleiche Zeile markiert bleibt, auch wenn sich ihre Zeilennummer ändert).

- `mx`: Aktuelle Zeile mit Zeichen "x" benennen.

  Z.B. "ma" setzt Marke "a" auf die aktuelle Zeile.

- `'x`: Springe zu Marke "x".

Editor vi (9)

Puffer/Zwischenablagen:

- Es gibt Puffer/Zwischenablagen für Textstücke (zum Kopieren oder Verschieben von Text).

  Die Puffer-Inhalte werden vergessen, wenn man den Editor beendet, aber sie bleiben erhalten, wenn man eine andere Datei in den Editor läd (beim klassischen vi geht der Inhalt des Default-Puffers verloren).

- Sie sind mit a bis z und A bis Z benannt, außerdem gibt es den Default-Puffer und Puffer 1 bis 9.

  Die Puffer 1 bis 9 sind speziell: Puffer 1 enthält das zuletzt gelöschte Textstück, 2 das davor gelöschte u.s.w. (gilt nicht für Löschung einzelner Zeichen, nur für ganze Zeilen).

- Z.B. spricht man den Puffer a mit `"a` an.

Editor vi (10)

Löschen/Ausschneiden (Cut):

- `x`: Aktuelles Zeichen löschen.

  Dies geht nur innerhalb einer Zeile (Zeilenende wird nicht gelöscht). Das letzte gelöschte Zeichen steht im Default-Puffer.

- `X`: Löschen nach links.
- `D`: Rest der Zeile löschen.
- `J`: Aktuelle Zeile mit nächster verschmelzen.

  "Join" (zusammenfügen). Dies löscht also effektiv das Zeilenende.

- `dd`: Aktuelle Zeile löschen.

  Natürlich auch mit Widerholungsfaktor, z.B. "5dd" löscht 5 Zeilen. Der zuletzt gelöschte Text steht im Default-Puffer, außerdem in "1.

Editor vi (11)

Löschen/Ausschneiden (Cut), Forts.:

- `dw`: Aktuelles Wort löschen.

  Man kann nach dem "d" ein beliebiges Positionskommando anwenden, z.B. würde "dk" die aktuelle Zeile und die darüber löschen.

- `d'`x: Von aktueller Zeile bis zu Marke x löschen.

  Genauer: Bis zur Zeile mit Marke x einschließlich (Marken markieren immer nur Zeilen, nicht eine genaue Position in der Zeile).

- `"`a`d'`x: Ebenso, aber in Puffer a speichern.

  Das ist praktisch, wenn man den Text länger zwischenspeichern will. Gibt man den Puffer nicht explizit an, kommt es in den Default-Puffer. Dort steht immer das zuletzt Gelöschte. Eine weitere Löschung überschreibt es aus dann. Man kann aber die letzten neun gelöschten Textstücke noch mit "1 bis "9 ansprechen.

Editor vi (12)

Gelöschtes wieder einfügen ("Paste"):

- `p`: Inhalt des Default-Puffers einfügen.

  Hinter der aktuellen Position. Wenn der Puffer einzelne Zeichen oder Worte enthält, wird direkt hinter dem Cursor eingefügt. Wenn der Puffer ganze Zeilen enthält, wird hinter der aktuellen Zeile eingefügt. Der Puffer behält seinen Inhalt natürlich, man könnte ihn auch mehrfach einfügen.

- `"`a`p`: Inhalt von Puffer a einfügen.
- `P`: Entsprechend vor aktueller Position einfügen.

  Natürlich geht das auch mit einem benannten Puffer.

Editor vi (13)

In den Puffer kopieren ("Copy"):

- `Y`: Aktuelle Zeile in den Puffer kopieren ("yank").
- `y'`x: Aktuelle Zeile bis Marke x in den Puffer.
- `"`a`y'`x: Entsprechend in Puffer a kopieren.
- Bei `gvim` (mit graphischer Oberfläche) kann man einen Bereich mit der Maus markieren und ihn dann mit `y` in den Puffer kopieren.

  Man braucht ihn dort aber nicht in den Puffer zu kopieren, weil man durch Drücken der mittleren Maustaste den zuletzt markierten Text einfügen kann. Übrigens würde `d` den markierten Bereich löschen.

Editor vi (14)

Änderungen rückgängig machen ("Undo"):

- `u`: Letzte Änderung zurücknehmen.

  Beim klassischen `vi` nimmt ein zweites Drücken von "u" das Undo wieder zurück. Bei der heute meist eingesetzten moderneren Variante werden durch weiteres Drücken von "u" immer mehr Änderungen zurückgenommen (natürlich maximal bis zum Start des Editors). Man muß dann `⟨Crtl⟩+R` drücken, um das Undo rückgängig zu machen ("Redo").

- `U`: Alle Änderungen in aktueller Zeile zurücknehmen (nur solange Zeile noch nicht verlassen).
- `:q!⟨Enter⟩`: Beenden ohne abspeichern (s.u.).

  Die Datei bleibt dann auf dem letzten gespeicherten Stand.

Editor vi (15)

Speichern und Verlassen:

- `ZZ`: Abspeichern und Editor verlassen.
- `:q⟨Enter⟩`: Editor verlassen ("quit").

  Bei Drücken von ":" kommt man in die Statuszeile unten, die jetzt als Kommandozeile für kompliziertere Kommandos dient. Wurde die Datei geändert, kann man den Editor zur Sicherheit nicht einfach mit ":q" verlassen, dann wären die Änderungen ja verloren. Will man bewusst nicht abspeichern, muß man ":q!" eingeben.

- `:w⟨Enter⟩`: Abspeichern.

  Der aktuelle Änderungsstand im Editor wird dann auf die Platte geschrieben, und ist (recht) sicher bei Systemabstürzen/Stromausfällen. Der vi speichert Änderungen allerdings auf eine temporäre Datei automatisch. Nach einem Absturz versuche man vi -r ⟨Datei⟩.

Editor vi (16)

Mit mehreren Dateien arbeiten:

- `:e` D`⟨Enter⟩`: Datei "D" in den Editor laden.

  Weil damit die bisher aktuelle Datei aus dem Speicher des Editors entfernt wird, muß man vorher abspeichern. Will man die Änderungen nicht speichern, verwende man ":e! D⟨Enter⟩". Die Inhalte der (benannten) Pufer bleiben erhalten. Man kann also z.B. mit "y" einen Text in einen Puffer speichern, dann die Datei mit ":e" wechseln, und das gespeicherte Textstück mit "p" in die neue Datei einfügen.

- `:r` D`⟨Enter⟩`: Inhalt von Datei D nach der aktuellen Zeile einfügen ("read").
- `:n⟨Enter⟩`: Zur nächsten Datei.

  Falls man vi mit mehreren Dateien aufgerufen hat, z.B. "vi a.h b.h".

Editor vi (17)

Suchen:

- `/`T`⟨Enter⟩`: Text T suchen (nach unten).

  Es wird das von der aktuellen Cursor-Position aus nächste Vorkommen nach unten gefunden. Beim Dateiende wird automatisch zum Anfang gesprungen, und man erhält die Meldung "search hit Bottom, continuing at Top". Die Groß-/Kleinschreibung wird beachtet ("case-sensitive"). Der Text ist eigentlich ein regulärer Ausdruck, der viele Möglichkeiten für Suchmuster bietet. Daher müssen Zeichen wie ., *, [, die in regulären Ausdrücken eine besondere Bedeutung haben, durch Voranstellen von \ "entschärft" werden (z.B. "\." oder "\\").

- `n`: Nächstes Vorkommen des Suchstrings.
- `N`: Voriges Vorkommen (umgekehrte Richtung).
- `?`T`⟨Enter⟩`: Text T suchen (nach oben).

Editor vi (18)

Suchen und Ersetzen:

- `:1,$s/`T`/`S`/gc⟨Enter⟩`: Text T durch S ersetzen.

  Das wesentliche Kommando ist s ("substitute"). Davor wird der Zeilenbereich angegeben, auf den das Kommando angewendet werden soll, hier 1,$ (von der ersten bis zur letzten Zeile, d.h. ganze Datei). Z.B. würde 'a,'b die Ersetzung auf den Bereich von Marke a bis Marke b beschränken. Der Schrägstrich / trennt Suchtext S und Ersetzungstext T ab. Enthalten diese Texte Schrägstriche, kann man auch andere Zeichen verwenden, z.B. @ (das Zeichen nach s ist immer das Trennzeichen). Der Suchtext ist ein regulärer Ausdruck, man kann Teile darin mit \(...\) markieren, und im Ersetzungstext mit \1 u.s.w. einfügen — das kann u.U. viel manuelle Arbeit sparen. Dahinter stehen dann noch die Optionen g (global: alle Vorkommen in einer Zeile werden ersetzt, ohne g nur das jeweils erste) und c (conditional, es wird für jede Ersetzung nachgefragt — das kann man natürlich weglassen, wenn man sich sicher ist).

Editor vi (19)

Änderungskommandos:

- `.`: Wiederholung des letzten Kommandos.

  Außer Positions-Änderungen. Man kann damit die gleiche Änderung an verschiedenen Stellen im Text vornehmen.

- `cw`: Wort ersetzen ("change").

  Dieses Kommando löscht das aktuelle Wort und wechselt in den Einfügemodus. Weil "." nur das letzte Kommando wiederholt, ist es günstig, die ganze Änderung mit einem Kommando durchzuführen (bis ⟨Esc⟩). Nach `c` kann man wieder ein beliebiges Positionierungskommando verwenden, z.B. `c5l` für die Ersetzung von 5 Zeichen, oder `cf;` für die Ersetzung bis einschließlich zum Semikolon ("find"), oder `ct;` für die Ersetzung bis direkt vor dem Semikolon ("to"). Statt dem Semikolon sind natürlich auch beliebige andere Zeichen möglich, die Suche erstreckt sich jeweils nur bis zum Ende der Zeile.

Editor vi (20)

Änderungskommandos, Forts.:

- `C`: Rest der Zeile ersetzen ("change").
- `S`: Ersetze aktuelle Zeile ("substitute").
- `s`: Ersetze aktuelles Zeichen.
- `r`x: Ersetze aktuelles Zeichen durch x ("replace").

  Im Unterschied zu "`s`" wird hier nicht in den Einfügemodus gewechselt, so dass man am Ende auch nicht "⟨Esc⟩" drücken muss. Der Cursor bleibt auf dem ersetzten Zeichen (nützlich für Änderungen von Spalten).

- `~`: Groß/Kleinschreibung des aktuellen Zeichens ändern.

Editor vi (21)

Weitere nützliche Kommandos:

- ⟨`Crtl`⟩`+L`: Bildschirm neu zeichnen.
- ⟨`Crtl`⟩`+G`: Position im Text angeben, und Information, ob Änderungen seit letzten Speichern.

  Bei `gvim` steht in der Fensterüberschrift ein "`+`", wenn es nicht gespeicherte Änderungen gibt.

- `%`: Von einer Klammer (auf/zu) zur zugehörigen Klammer (zu/auf) springen.
- `>>`: Aktuelle Zeile eine Stufe weiter einrücken.
- `<<`: Aktuelle Zeile eine Stufe weniger einrücken.

Editor vi (22)

Anzeige von ungewöhnlichen Zeichen:

- Nicht druckbare Zeichen werden in einer der Formen `^`X, `\`abc oder <XY> dargestellt.

  `^`X steht für "Control-X", das man durch gleichzeitiges Drücken von ⟨`Crtl`⟩ ⟨`Strg`⟩ und X erhält. Es entspricht dem ASCII-Code von X minus 64. Bei der Form `\`abc sind die Ziffern a, b, c oktal zu interpretieren. Bei der Form <XY> sind X, Y Hezadezimalziffern.

- Steht ein `^M` an jedem Zeilenende, wurde die Datei unter Windows gespeichert.

  Unter Unix markiert nur LF das Zeilenende, unter Windows CR, LF. Die `^M` sind gerade die jetzt überflüssigen CR (ASCII Code 13). Man kann sie im `vi` entfernen mit `:1,$s/`⟨`Crtl`⟩`+V`⟨`Crtl`⟩`+M//`⟨`Enter`⟩. Alternativ kann man das Programm `tounix` verwenden, s.u.

Editor vi (23)

Anzeige von langen Zeilen:

- Ist eine Zeile im Dokument länger als eine Bildschirmzeile, wird sie einfach in der nächsten Bildschirmzeile fortgesetzt.

  Wenn man den Cursor mit `j` in die nächste Zeile bewegt, springt er dann entsprechend über mehrere Bildschirmzeilen, so dass er wirklich in der nächsten Dokument-Zeile landet. Wenn eine Zeile am unteren Bildschirmrand nicht vollständig angezeigt werden kann, wird sie im `vi` als "`@`" dargestellt. Da das alles verwirrend ist, empfiehlt es sich, in eigenen Dokumenten Zeilen länger als 79 oder 80 Zeichen zu vermeiden (80 Zeichen sind eine übliche Bildschirmbreite).

- In der Programmierung muß man explizit einen Zeilenumbruch durch Drücken von ⟨`Enter`⟩ verlangen.

Editor vi (24)

Siehe auch:

- `:help`⟨`Enter`⟩

  Es wird ein neuer Puffer im gleichen Fenster aufgemacht, den man mit "`:q`" wieder schließen kann. Wenn man auf einem Hyperlink steht, kann man ihm mit ⟨`Ctrl`⟩`+]` folgen (oder doppelter Masuklick). Zurück geht es mit ⟨`Ctrl`⟩`+T`.

- [http://vimdoc.sourceforge.net/]
- [http://unixhelp.ed.ac.uk/vi/ref.html]
- Programme `vimtutor` oder `gvimtutor`

Editor emacs (1)

- ⟨`Crtl`⟩`-x` ⟨`Crtl`⟩`-s`: Abspeichern.

  Weil viele Kommandos gleichzeitiges Drücken von ⟨`Crtl`⟩ verlangen, wird es in den meisten Anleitungen mit `C-` abgekürzt, dieses Kommando wäre also `C-x C-s`. Die andere häufige Abkürzung ist `M-` ("`M`" steht für Meta). Üblicherweise kann man `M-x` durch gleichzeitiges Drücken von ⟨`Alt`⟩ und `x` erreichen (je nach Betriebssystem gibt es eventuell auch eine andere "Meta"-Taste). Manchmal klappt das aber nicht, weil z.B. das Terminalprogramm bestimmte Tastenkombinationen mit ⟨`Alt`⟩ selbst interpretiert, und nicht an den Editor weitergibt. Dann kann man auch erst ⟨`Esc`⟩ drücken, und dann `x`. Manche `emacs`-Nutzer haben sich auch daran gewöhnt, `M-` Befehle immer mit ⟨`Esc`⟩ einzugeben. Neben dem `C/M-` Kürzel haben Kommandos auch einen langen Namen, dieses wäre `save-buffer`. Nach `M-x` (also ⟨`Alt`⟩`+x`) kann man den langen Namen eingeben. Die Zuordnung von Kürzeln zu den offiziellen langen Namen ist änderbar. Normale druckbare Zeichen sind an `self-insert-command` gebunden, werden also einfach eingefügt.

Editor emacs (2)

- ⟨`Crtl`⟩`-x` ⟨`Crtl`⟩`-c`: Beenden.
- ⟨`Crtl`⟩`-v`: Bildschirm weiter (nach unten).
- ⟨`Esc`⟩ `v`: Bildschirm zurück (nach oben).

  Dies ist eigentlich `M-v`, aber ⟨`Alt`⟩`-v` funktioniert nicht (unter Ubuntu Linux), weil das Terminalprogramm dann ein Menü aufklappt.

- ⟨`Alt`⟩`+<`: Zum Anfang der Datei.
- ⟨`Alt`⟩`+>`: Zum Ende der Datei.
- ⟨`Alt`⟩`+g g`: Zu bestimmter Zeilennummer.

Editor emacs (3)

- ⟨Crtl⟩+f: Zeichen nach rechts ("forward").

  Normalerweise sollten aber die Cursor-Tasten funktionieren.
- ⟨Crtl⟩+b: Zeichen nach links ("backward").
- ⟨Crtl⟩+p: Zeile nach oben ("previous").
- ⟨Crtl⟩+n: Zeile nach unten ("next").
- ⟨Crtl⟩+a: Zum Anfang der Zeile.
- ⟨Crtl⟩+e: Zum Ende der Zeile.

Editor emacs (4)

- ⟨Crtl⟩+d: Einzelnes Zeichen löschen ("delete").
- ⟨Crtl⟩+k: Rest der Zeile löschen ("kill").

  Ist man am Ende der Zeile, so wird das Zeilenende gelöscht. Ist man nicht am Ende der Zeile, so wird das Zeilenende nicht mitgelöscht. Wenn man also am Anfang der Zeile ist, kann man sie durch zweimaliges Drücken von ⟨Crtl⟩+k vollständig löschen. Der durch eine Folge von ⟨Crtl⟩+k-Kommandos gelöschte Text kommt in eine Zwischenablage (ein Element im "kill ring"). Im Gegensatz dazu werden mit ⟨Crtl⟩+d gelöschte Zeichen nicht zwischengespeichert.
- ⟨Crtl⟩+y: Zuletzt gelöschten Text einfügen ("yank").

  Falls dies nicht der richtige Text war, sondern ein vorher gelöschter Text gebraucht wird, kann man den zuletzt mit ⟨Crtl⟩+y eingefügten Text mittels ⟨Alt⟩+y durch den davor gelöschten Text ersetzen (u.s.w.).

Editor emacs (5)

- ⟨Crtl⟩+u n ⟨Crtl⟩+k: n Zeilen löschen.

  Mit ⟨Crtl⟩+u ("universal argument") kann man Kommandos einen Zahlwert mitgeben, der oft als Wiederholungsfaktor interpretiert wird.
- ⟨Crtl⟩+g: Kommando abbrechen.

  Wenn man nicht mehr genau weiss, welche Taste man gedrückt hat, oder der `emacs` auf eine Eingabe wartet, ist das praktisch.
- ⟨Crtl⟩+_, ⟨Crtl⟩+/: Undo.

  Die beiden Kommandos sind identisch. Mehrfaches Drücken dieser Kommandos macht weitere Änderungen rückgängig. Wenn man zwischendurch ein normales Kommando verwendet (z.B. eine Cursor-Bewegung), werden die Änderungen durch Undo als normale Änderungen betrachtet, und können dann durch eine neue Folge von Undo-Kommandos rückgängig gemacht werden.

Editor emacs (6)

- ⟨Crtl⟩+s: Inkrementell vorwärts suchen.

  Inkrementell bedeutet, dass während man den Suchstring eingibt, passende Vorkommen angezeigt werden. Durch nochmaliges Drücken von ⟨Crtl⟩+s während der Suche kommt man zum nächsten Vorkommen. Hat man die Suche schon beendet, muss man ⟨Crtl⟩+s zwei Mal drücken, um das nächste Vorkommen zu finden.
- ⟨Crtl⟩+r: Inkrementell rückwärts suchen.
- ⟨Alt⟩+%: Suchen und ersetzen.

  Es wird dann nacheinander jede Ersetzungsstelle angezeigt. Man hat die Wahl zwischen `y` (ersetzen), `n` (nicht ersetzen), `q` (abbrechen), `!` (alle folgenden ohne Nachfrage ersetzen), `.` (dieses Vorkommen ersetzen und Schluss).

Editor emacs (7)

- ⟨Crtl⟩+⟨Leertaste⟩: aktuelle Position markieren.

  Viele Kommandos, die zu einer längeren Bewegung im Text führen (z.B. die Suche), setzen die Marke automatisch auf die Position vor dem Kommando, so dass man dorthin leicht zurückkehren kann.

- ⟨Crtl⟩+x ⟨Crtl⟩+x: Zur Marke zurückkehren.

  Die genaue Erklärung ist: "swap mark and point", d.h. Marke und Cursor-Position werden ausgetauscht. Wenn man das gleiche Kommando noch einmal eingibt, gelangt man also zur Ursprungsposition.

- Der Bereich zwischen Marke und aktueller Cusor-Position heißt die "Region".

  Wenn man eine Marke setzt und dann den Cursor bewegt, wird die Region hervorgehoben. Will man das nicht, kann man ⟨Crtl⟩+g drücken.

Editor emacs (8)

- ⟨Crtl⟩+w: Markierte Region ausschneiden ("cut").

  Man kann das zum Löschen verwenden oder den Text danach mit ⟨Crtl⟩+y ("yank") an anderer Stelle wieder einfügen.

- ⟨Alt⟩+w: Region in Zwischenablage kopieren.
- ⟨Crtl⟩+x ⟨Crtl⟩+f: Datei laden / Puffer wechseln.

  Das Kommando `find-file` prüft zuerst, ob es einen Puffer gibt, in dem die Datei schon geladen ist. Wenn nicht, legt sie einen neuen Puffer an, und läd die Datei. Anschliessend wird in den Puffer gewechselt, d.h. er füllt nun den Bildschirm aus. Der "kill ring" bleibt erhalten, d.h. man kann im neuen Puffer mit ⟨Crtl⟩+y einen Text einfügen, den man in einem anderen Puffer gelöscht/kopiert hat. Mit ⟨Crtl⟩+x b kann man zwischen Puffern wechseln (oder einen neuen anlegen).

Editor emacs (9)

- ⟨Crtl⟩+h a: In Hilfe suchen ("apropos").

  Mit ⟨Crtl⟩+h k bekommt man die Dokumentation zu einem Tastatur-Kürzel, mit ⟨Crtl⟩+h f die zu einer Funktion (langer Kommando-Name). Mit ⟨Crtl⟩+h t starten Sie ein Tutorial. Weitere Tutorials finden Sie im Web, z.B. [http://www.gnu.org/software/emacs/tour/] [http://www2.lib.uchicago.edu/keith/tcl-course/emacs-tutorial.html]

- Dadurch wird der Bildschirmbereich in zwei Teilfenster unterteilt.
- ⟨Crtl⟩+x o: Wechsel zwischen Fenstern.
- ⟨Crtl⟩+x 0: Fenster löschen.

Syntax-Unterstützung (1)

- Es ist eine übliche Funktion heutiger Editoren, dass verschiedene Sprachkonstrukte unterschiedlich eingefärbt werden ("Syntax Highlighting").
- `vi/gvim` benutzen folgende Farben für C++-Dateien:
  - ◇ Kommentare: blau
  - ◇ Konstanten (Zahlen, Zeichenketten): rosa
  - ◇ Escape-Sequenzen in Strings (z.B. `\n`): grau
  - ◇ Schlüsselworte für Kontrollstrukturen: braun
  - ◇ Schlüsselworte für Typen: grün
  - ◇ Präprozessor-Anweisungen (z.B. `#include`): lila

Syntax-Unterstützung (2)

- Wenn der Cursor auf einer Klammer steht, wenn die zusammengehörigen Klammern hellblau unterlegt.
- Diese Färbung von bestimmten Textteilen macht der Editor für die Darstellung auf dem Bildschirm, es werden aber keineswegs Formatierungsanweisungen in den Text eingefügt.

  Die C++-Datei enthält nur die reinen Buchstaben u.s.w., spezielle Anweisungen für Farben würde der Compiler nicht verstehen. Wenn man einen anderen Editor benutzt, bekommt man für die gleiche Datei eventuell ein anderes Farbschema (oder auch gar keine Farben). Das Farbschema ist meist konfigurierbar, außerdem hängt es von der Programmiersprache ab, die der Editor an der Dateiendung erkennt.

Syntax-Unterstützung (3)

- Falls man eine Zeichenkette nicht schließt, oder ein reserviertes Wort als Variablenname verwendet, kann man das an der Farbe erkennen.

  Natürlich braucht man die Farben nicht auswendig zu lernen. Wenn man länger mit einem Editor arbeitet, merkt man an den Farben intuitiv, dass “etwas nicht stimmt”. Spätestens, wenn der Compiler einen Fehler gemeldet hat, kann es sich lohnen, auf ungewöhnliche Farben zu achten. Manchmal kommt aber auch der Editor mit den Farben durcheinander.
- Viele Editoren rücken den Text nach einer “{” auch eine Tabulatorbreite weiter ein.

Rechnernutzung am Institut (1)

- Die Übungen werden in zwei Pools abgehalten:
  - ◇ Raum 324 (PC-Pool) enthält PCs, die wahlweise unter Linux oder Windows XP laufen.

    Nach dem Einschalten und Selbsttest erscheint der Bootloader “GRUB”. Man kann mit den Cursortasten ein Betriebssystem auswählen und drückt dann ⟨`Enter`⟩ (wenn man sich zu viel Zeit läßt, startet ein System automatisch). Läuft der Rechner bereits, und will man ein anderes Betriebssystem, muß man das aktuell laufende System erst geordnet herunterfahren (“Ausschalten” oder “Neustart” auf dem Bildschirm anklicken). Bitte nicht einfach den Ausschalt-Knopf drücken!
  - ◇ Raum 322 (Thin-Client-Pool): s.u.

Rechnernutzung am Institut (2)

- Zuerst müssen Sie sich auf einem Rechner “einloggen” (beim Betriebssystem mit Benutzerkennung und Passwort anmelden).

  Benutzername und Passwort sollten Sie bei Ihrer Immatrikulation bekommen haben. Ihre Benutzerkennung muß aber für die Rechner unseres Instituts freigeschaltet werden. Bitte melden Sie sich dazu vor Ihrer ersten Übung bei der Poolaufsicht im Raum 333 (möglichst nach 14 Uhr). Nach der Betriebssystemauswahl dauert es noch etwas, bis die Aufforderung zum Einloggen erscheint (das Betriebssystem “fährt hoch”). Läuft der Rechner unter Windows, und wollen Sie auch unter Windows arbeiten, drücken Sie “⟨`Ctrl`⟩+⟨`Alt`⟩+⟨`Delete`⟩” (Steuerung+Alt+Löschen), um die Eingabeaufforderung zu erhalten. Nach Eingabe von Benutzername und Passwort dauert es wieder etwas, bis der Rechner bereit ist, Ihre Befehle entgegen zu nehmen.

Rechnernutzung am Institut (3)

- Zum Thin-Client Pool:
  - ◇ "Thin Clients" sind kleine Rechner, die nur Fensteroberfläche, Maus und Tastatur verwalten.

    Der Vorteil ist, dass die Thin Clients billiger sind als ein vollwertiger PC, meist geräuschlos, und keine Administration benötigen.
  - ◇ Die eigentliche Programmausführung geschieht über das Netzwerk auf einem größeren Server.

    Nach Einschalten und Hochfahren der Software auf dem "Thin Client" muß man sich mit Benutzername und Passwort als Nutzer des Thin Clients ausweisen. Dann bekommt man eine Auswahl zwischen verschiedenen Server-Rechnern unter unterschiedlichen Betriebssystemen. Für Linux wähle man z.B. den GNOME-Desktop. Hat man die Verbindung zum Server bekommen, muss man sich dort erneut mit Benutzername und Passwort anmelden.

Rechnernutzung am Institut (4)

- Man vergesse nicht, sich am Ende beim Betriebssystem abzumelden ("ausloggen").

  Jemand anders könnte sonst alle Dateien löschen, oder sogar richtig kriminelle Dinge unter Ihrer Benutzerkennung tun. Nach einiger Zeit der Inaktivität wird der Bildschirm automatisch gesperrt (Sie müssen dann Ihr Passwort erneut eingeben, um weiterarbeiten zu können).
- Natürlich sollte man das Passwort geheim halten.

  Das Passwort sollte auch nicht zu einfach sein: Es sollte Großbuchstaben, Kleinbuchstaben, Ziffern und Sonderzeichen enthalten, mindestens 8 Zeichen lang sein, und nicht im Duden stehen, auch nicht der Name der Freundin, der Liebings-Popgruppe, eine Telefonnummer oder ein Geburtsdatum sein. Niemand darf es von Ihnen erfragen. Reagieren Sie nicht auf Mails von Administratoren, die Sie bitten, es Ihnen mitzuteilen, oder bei einer obskuren Webseite einzugeben (Phishing).

Rechnernutzung am Institut (5)

- Haben Sie zu Hause einen Rechner unter Windows, und wollen Sie über das Internet auf einem Institutsrechner und Linux arbeiten, können Sie z.B. das Programm PuTTY verwenden.

  [http://www.chiark.greenend.org.uk/~sgtatham/putty/]
  Es enthält auch den Befehl `pscp` zum Kopieren von Dateien, z.B.
  `pscp hello.cpp brass@anubis.informatik.uni-halle.de:oop10/hello`
- Bei Schwierigkeiten wenden Sie sich an:
  - ◇ Den Tutor Ihrer Übungsgruppe.
  - ◇ Die Poolaufsicht in Raum 333.
  - ◇ Herrn Trull und Herrn Ohme in Raum 320.

Linux-Benutzung (1)

- Man braucht zunächst ein "Terminal"-Fenster.

  Man muß bei Ubuntu Linux auf "Applications" / "Anwendungen" links oben klicken, dann die Maus auf "Accessories" / "Zubehör" bewegen, und in dem sich dann öffnenden Submenü auf "Terminal" klicken. Wenn das Menü zu lang ist, erscheint unten bzw. oben ein Pfeil: Bewegt man die Maus darüber, wird der angezeigte Menü-Ausschnitt in der jeweiligen Richtung bewegt.
- Die Eingabeaufforderung ("Prompt") im Terminal-Fenster hat die Form

  "`Benutzer@Rechner:Verzeichnis$ `".

  Dabei wird das Home-Verzeichnis, in dem man sich zunächst befindet, mit "~" markiert. Der Prompt bedeutet, dass der Kommandointerpreter ("Shell") bereit ist, ein Kommando entgegenzunehmen.

Linux-Benutzung (2)

- Bei der Eingabe eines Kommandos im Terminal-Fenster kann man mit
  - ⟨`Backspace`⟩ das jeweils letzte Zeichen löschen,
  - ⟨`Crtl`⟩`+U` die ganze Eingabe,
  - ←, → den Cursor in der Zeile bewegen,

    Normale Zeichen werden dann eingefügt, ⟨`Backspace`⟩ löscht das Zeichen links vom Cursor, ⟨`Delete`⟩ das Zeichen unter dem Cursor.
  - ⟨`Crtl`⟩`+C` ein laufendes Programm abbrechen oder mindestens unterbrechen.
- Man beendet einen Befehl mit ⟨`Return`⟩/⟨`Enter`⟩/↵.

  Er wird dann ausgeführt.

Linux-Benutzung (3)

- Das erste Wort auf der Kommandozeile ist der Name des Programms, das ausgeführt werden soll, der Rest sind seine Argumente (Eingabewerte).

  Die Argumente werden also durch Leerzeichen getrennt.
- Ein "ungefährliches" Kommando zum Testen ist "`echo`". Es gibt nur seine Argumente aus.

  `echo Hallo` (gibt "Hallo" aus)
- Mit den Cursortasten ↑, ↓ kann man die letzten Kommandos in die Kommandozeile bekommen.

  Und dann erneut ausführen oder ggf. vorher modifizieren.

Linux-Benutzung (4)

- Letztes Kommando wiederholen:

  `!!`
- Zuletzt ausgeführtes Kommando, das mit "`e`" beginnt, wiederholen:

  `!e`
- Letztes Kommando wiederholen, aber Zeichenfolge "Hallo" durch "Tschues" ersetzen:

  `^Hallo^Tschues`

  Es wird nur das erste Vorkommen von "`Hallo`" ersetzt.

Linux-Benutzung (5)

- Es empfiehlt sich für diese Vorlesung ein Unterverzeichnis (z.B. "`oop`") anzulegen, das geht mit:

  `mkdir oop` ("make directory")
- Man wechselt in dieses Unterverzeichnis mit:

  `cd oop` ("change directory")
- Groß-/Kleinschreibung ist wichtig unter Linux!
- Eventuell legt man hierin noch ein weiteres Unterverzeichnis für das spezielle Programm an (`hello`), und wechselt dann da hinein.

Linux-Benutzung (6)

- Nun muss man einen Editor aufrufen, um den Programmtext einzugeben, z.B. mit

  `gedit hello.cpp`

  Statt `gedit` sind natürlich auch `vi`, `gvim`, `emacs` möglich.

- Geben Sie das Beispielprogramm ein und speichern Sie ab ("Save").

  Wenn der Editor ein eigenes Fenster hat, brauchen Sie ihn nicht unbedingt zu beenden. Läuft der Editor im Terminal-Fenster, können Sie auch ein weiteres Terminal-Fenster aufmachen, um darin die folgenden Kommandos einzugeben (dort zuerst `cd oop/hello`). Sie aktivieren ein Fenster (es bekommt den "keyboard focus", d.h. Tastatureingaben landen dort) durch Mausklick in das Fenster. Fenster verschieben: Linke Maustaste über Kopfzeile drücken, Maus verschieben, loslassen.

Linux-Benutzung (7)

- Inhalt des aktuellen Verzeichnisses anzeigen:

  `ls` ("list directory")

  Es sollte nun `hello.cpp` geben, ggf. auch `hello.cpp~` (ältere Version).

- Mehr Informationen bekommt man mit

  `ls -l` ("ls, long form")

  Die Ausgabe ist z.B.

  ```
  insgesamt 1
  -rw-r--r-- 1 brass pib 155 2010-09-17 16:45 hello.cpp
  ```

  Die erste Zeile besagt, wie viele Blöcke (zu 1KB) die Dateien insgesamt belegen. Dann kommt eine Zeile pro Datei mit Dateityp, Zugriffsrechten, Datum und Uhrzeit der letzten Änderung und anderen Informationen. Ein "d" in der ersten Spalte markiert Unterverzeichnisse ("directory"). Der genaue Aufbau wird auf Folie 1-125 erläutert.

Linux-Benutzung (8)

- Alle Dateien anzeigen (auch "versteckte"):

  `ls -al` ("ls, all entries, long form")

  Dateien und Verzeichnisse, die mit einem Punkt "." beginnen, werden normalerweise nicht angezeigt. Im Homeverzeichnis gibt es davon viele (z.B. üblich zur Speicherung persönlicher Voreinstellungen u.s.w.). Das aktuelle Verzeichnis wird als "." ausgegeben, und das übergeordnete Verzeichnis als "..". Wenn man nicht die lange Form des Anzeige wünscht, kann man natürlich auch "`ls -a`" verwenden.

- Dateien im Verzeichnis `hello` listen:

  `ls -l hello`

  Ohne `-l` würden wieder nur die Dateinamen gelistet. Wünscht man Informationen über das Verzeichnis, aber nicht den Inhalt, kann man `-d` ("directory") verwenden.

Linux-Benutzung (9)

- Das Zeichen `*` passt auf eine beliebige Folge beliebiger Zeichen (auch "."), das Zeichen `?` auf ein einzelnes beliebiges Zeichen.

- Alle C++Dateien auflisten:

  `ls -l *.cpp`

- Die "Wildcards" `*` und `?` funktionieren mit beliebigen Befehlen, nicht nur `ls`.

  Der Kommandointerpreter (die "Shell") ersetzt sie vor Aufruf des eigentlichen Kommandos. Die meisten Kommandos erlauben mehr als ein Argument, und führen dann einfach die gleiche Aktion mit allen Argumenten durch (sonst würden die Wildcards nicht funktionieren).

Linux-Benutzung (10)

- Name des aktuellen Verzeichnisses anzeigen:

  `pwd` ("print working directory")

- In das übergeordnete Verzeichnis wechseln:

  `cd ..` ("change directory")

- In das Home-Verzeichnis wechseln:

  `cd` ("change directory")

- Platzbelegung (in 1 KByte Einheiten) anzeigen (dies listet auch alle Unterverzeichnisse auf):

  `du` ("disk usage")

Linux-Benutzung (11)

- Datei kopieren:

  `cp hello.cpp x.cpp` ("copy")

- Datei in anderes Verzeichnis kopieren:

  `cp hello.cpp hello` ("cp Dateien Verzeichnis")

  Hier wird "`hello.cpp`" in das Unterverzeichnis "`hello`" kopiert.

- Datei in das aktuelle Verzeichnis kopieren:

  `cp hello/hello.cpp .`

  Das aktuelle Verzeichnis kann immer mit "." angesprochen werden.

- Verzeichnis mit Inhalt kopieren:

  `cp -r hello test` ("copy recursive")

Linux-Benutzung (12)

- Datei (oder Verzeichnis) umbenennen:

  `mv x.cpp test.cpp` ("move")

  Gab es vorher schon eine Datei "`test.cpp`", wird sie (die alte Version) gelöscht (überschrieben). Das kann auch beim `cp`-Befehl passieren.

  Dateinamen mit Leerzeichen (oder Spezialzeichen wie "`$`") sind in `'...'` einzuschliessen: `mv 'Ein Versuch.cpp' v.cpp`. Dies gilt nicht nur für "`mv`", sondern für beliebige Kommandos (Leerzeichen trennen sonst die Argumente/Eingabewerte eines Kommandos).

- Datei/Verzeichnis verschieben (hier ins übergeordnete Verzeichnis):

  `mv hello.cpp ..` ("move")

Linux-Benutzung (13)

- Datei löschen:

  `rm hello.cpp` ("remove")

  Gelöschte/überschriebene Dateien sind weg! Es empfiehlt sich, Dateien regelmäßig zu sichern (z.B. auf USB-Stick).

- Leeres Verzeichnis löschen:

  `rmdir hello` ("remove directory")

- Verzeichnis mit Inhalt löschen:

  `rm -r hello` ("remove recursive")

- Löschen aller Objektdateien mit Nachfrage:

  `rm -i *.o` ("remove interactive")

Linux-Benutzung (14)

- Es gibt auch ein Werkzeug mit graphischer Benutzeroberfläche für die Dateiverwaltung ("Nautilus" File Browser).

  Er ist zu erreichen über das Menü "Places" (dann "Home Folder") oben links.

- Den "Firefox" Web Browser erhält man durch Klick auf die Weltkugel in der Menüleiste oben oder durch Eingabe von "`firefox`" im Terminalfenster.
- Die Möglichkeit zum Ausloggen oder Herunterfahren des Rechners erhält man durch Klick auf das Ein/Aus Symbol oben rechts.

Linux-Benutzung (15)

- Steckt man einen USB-Speicherstick in einen Rechner unter Ubuntu Linux, wird er automatisch unter `/media` in das Dateisystem eingefügt.

  Der Name des Verzeichnisses in `/media` ist der Name des USB-Sticks (oft kryptisch). Der USB-Stick wird auch links oben auf dem Bildschirm angezeigt, durch Doppelklick darauf öffnet man ihn in Nautilus. Durch Klick mit der rechten Maustaste öffnet sich ein Pop-up Menü, dort gibt es auch einen Punkt "Safely Remove Drive", den man anklicken sollte, bevor man den USB-Stick herauszieht. Man kann statt dessen auch "`umount /media/*`" eingeben. Auf den "Thin Clients" funktionieren USB-Sticks unter Linux bisher nicht (sie sind nicht vom Server aus zugreifbar). Unter Windows geht es.

  Wenn Sie die Zeilenenden einer Datei zwischen UNIX und Windows konvertieren wollen, können Sie das mit "`fromdos hello.cpp`" und "`todos hello.cpp`" machen.

Linux-Benutzung (16)

- UNIX/Linux ist grundsätzlich ein System, das auf Kooperation der Nutzer ausgelegt ist.
- Sie können sich z.B. das Verzeichnis mit dem "Hello World" Beispiel des Professors anzeigen lassen:

  ```
  ls ~brass/oop10/hello
  ```

- Allgemein kann man das Homeverzeichnis des Benutzers B mit ~B ansprechen.
- Sie können sich auch das Beispiel kopieren:

  ```
  cp ~brass/oop10/hello/hello.cpp .
  ```


Linux-Benutzung (17)

- Was genau geht, hängt an den Zugriffsrechten, die der jeweilige Benutzer vergeben hat.
- Z.B. erhält man hier eine Fehlermeldung:

  ```
  ls ~brass/oop10/exam
  /home/brass/oop10/exam: Permission denied
  ```

- Man kann normalerweise auch keine Dateien fremder Benutzer löschen oder verändern.

  Bei ungünstiger (zu liberaler) Setzung der Zugriffsrechte wäre das aber grundsätzlich möglich. Manchmal möchten ja auch einige Freunde/Kollegen zusammen an den gleichen Dateien arbeiten.

Linux-Benutzung (18)

- Die Zugriffsrechte werden mit `ls -l` angezeigt:

```
-rwxr-xr-x 1 brass pib 54936 2010-09-22 19:06 hello
-rw-r--r-- 1 brass pib   176 2010-09-22 19:06 hello.cpp
```

- Das erste "-" ist der Dateityp (normale Datei). Bei Verzeichnissen wird hier "`d`" angezeigt.
- Dann kommen Dreiergruppen "`rwx`" (read, write, execute) von Zugriffsrechten für den Dateibesitzer, seine Gruppe, und alle anderen Rechner-Nutzer.

  Read: Leserechte, Write: Schreibrechte, Execute: Ausführungsrechte. Es folgen die Anzahl der "Hard Links" (Einträge dieser Datei im Verzeichnisbaum), der Dateibesitzer, seine Gruppe, die Dateigröße in Byte, Datum und Uhrzeit der letzten Änderung, und der Dateiname.

Linux-Benutzung (19)

- Die Zugriffsrechte für die Datei `hello.cpp` sind `rw- r-- r--`:
  - ◇ Der Besitzer (`brass`) darf die Datei lesen und schreiben (ausführen geht nicht).
  - ◇ Die Mitglieder seiner Gruppe sowie alle anderen Nutzer des Rechners dürfen die Datei nur lesen.
- Die Rechte für das Programm `hello` sind `rwx r-x r-x`, d.h. alle dürfen das Programm ausführen.

  Das Programm läuft mit den Rechten des Aufrufers. Selbst wenn das Programm auf Dateien schreiben würde, könnten andere Benutzer damit nicht den Zugriffsschutz für Dateien von `brass` umgehen.

Linux-Benutzung (20)

- Bei Verzeichnissen bedeutet das `x`-Bit, dass man in Pfaden durch das Verzeichnis wechseln darf.

  Wenn man keine Leserechte, aber Ausführungsrechte für ein Verzeichnis hat, kann man es sich nicht anzeigen (auflisten) lassen, aber man kann auf Dateien und Unterverzeichnisse zugreifen, falls man für sie Zugriffsrechte hat, und weiss, wie sie heissen.
- Sie können Zugriffsrechte mit dem Befehl `chmod` ("change mode") setzen, z.B. der Gruppe (`g`) und aller Welt (`o`, "others") die Leserechte nehmen:

```
chmod go-r hello.cpp
```

  Sie selbst wären `u` ("user"), alle zusammen `a` ("all").
  Statt `+` geht auch `-` und `=`.

Linux-Benutzung (21)

- Wir würden es schätzen, wenn Sie dem Kopieren Ihrer Hausaufgaben entgegen wirken, indem Sie das Verzeichnis sperren, z.B.

```
chmod go= oop
```

  Das bedeutet: gar keine Rechte für die Gruppe und alle anderen.
- Wenn Sie das nicht machen, ist das Risiko, dass Sie 0 Punkte bekommen, weil jemand anders Ihre Hausaufgabe kopiert hat.

  Wir prüfen nicht, was Original und was Kopie ist, und halten zumindest aktive Weitergabe zwecks kopieren für nicht in Ordnung.
- Administratoren sind nicht an die Rechte gebunden.

Linux-Benutzung (22)

- Man kann Handbuchseiten zu einem Kommando (z.B. `mkdir`) abfragen mit

  `man mkdir` ("manual")

  Mit ⟨`Enter`⟩ kommt man eine Zeile im Text weiter, mit ⟨`Leertaste`⟩ eine Seite, mit "`b`" ("back") eine Seite zurück, und mit "`q`" ("quit") kann man die Anzeige der Handbuchseite verlassen. Das Handbuch besteht aus mehreren Bänden, z.B. stehen Systemaufrufe im zweiten Band: "`man 2 mkdir`" sucht `mkdir` in Band 2.

- Viele Kommandos geben eine Kurzanleitung aus, wenn sie mit der Option `--help` aufgerufen werden:

  `mkdir --help` (Hilfe zu `mkdir`)

Linux-Benutzung (23)

- Nach Stichworten im Handbuch sucht man mit

  `apropos mkdir`

  Alternativ auch mit "`man -k mkdir`".

- GNU-Software ist häufig durch `info`-Seiten besser beschrieben, als durch den `man`-Eintrag:

  `info mkdir`

  Info ist ein Hypertextsystem, das ohne Maus auskommt: `n` führt zu nächsten Seite, `p` zur vorherigen, `u` eine Stufe hoch, `/` zur letzten besuchten Seiten und wenn der Cursor über einem mit "`*`" markierten Hyperlink steht, folgt man ihm durch Drücken von ⟨`Enter`⟩.

- Siehe: [http://www.linux.org/lessons/beginner/]

Fenster in Ubuntu Linux (1)

- Genauer: Fenster beim GNOME-Desktop.
- Wenn mehrere Fenster offen sind, ist eins das aktive Fenster (in dem Tastatureingaben verarbeitet werden), zu erkennen an den hellen/farbigen Symbolen in der Kopfzeile.

  Durch Mausklick in ein Fenster (oder auf seine Kopfleiste) wird das das aktive Fenster.

- Man schließt ein Fenster durch Klick auf das Kreuz im orangen Kreis links oben.

  Das zugehörige Programm wird dann beendet. Gleiche Funktion wie das Kreuz rechts oben in der Kopfleiste von Windows-Fenstern.

Fenster in Ubuntu Linux (2)

- Klickt man auf den Pfeil nach unten (oben links in jedem Fenster), wird das Fenster "minimiert".

  Es wird dann nicht mehr im Hauptbereich des Bildschirms dargestellt, sondern nur noch in der Fensterliste unten. Klickt man dort unten, wird es wieder normal dargestellt. Klickt man auf den Eintrag des Fensters unten, während es oben zu sehen ist, wird es auch minimiert. All das funktioniert wie bei Windows.

- Klickt man auf den Pfeil nach oben (oben links in jedem Fenster), wird das Fenster maximiert, es füllt dann den ganzen Bildschirm aus.

  Das Symbol in der Kopfzeile ändert sich dann in ein Quadrat: Klickt man darauf, erhält das Fenster wieder seine alte Größe und Position.

Fenster in Ubuntu Linux (3)

- Man kann Fenster in in Windows verschieben.

  Die linke Maustaste über der Kopfzeile drücken, Maus verschieben, Maustaste loslassen.

- Fenster können in ihrer Größe verändert werden.

  Auch wie in Windows: Linke Maustaste am rechten oder unteren Rand (oder in der rechten unteren Ecke) drücken (wenn sich der Mauszeiger entsprechend geändert hat), ziehen und loslassen.

- Auch der Scrollbar funktioniert wie in Windows.

  Mit dem Scrollbar am rechten Fensterrand kann man den dargestellten Ausschnitt des Textes verschieben. Man drückt die linke Maustaste über dem Schieber, verschiebt die Maus nach oben oder unten, und läßt los. Alternativen: oberhalb oder unterhalb des Schiebers klicken, oder auf die kleinen Pfeile am Ende, oder Drehrad in der Maus.

Fenster in Ubuntu Linux (4)

- Wenn man ein Textstück markiert, indem man die Maustaste am Anfang drückt und mit der Maus zum Ende fährt, und dann loslässt, kommt es in die Zwischenablage ("Copy").

  Man braucht also nicht wie bei Windows ⟨`Crtl`⟩-`C` zu drücken.

- Durch Klick mit der mittleren Maustaste kann man es wieder einfügen ("Paste").
- Durch Klick mit der rechten Maustaste erreicht man üblicherweise ein Pop-Up Menü.
- [https://help.ubuntu.com/7.04/user-guide/C/]

Aufruf des Compilers `g++` (1)

- `g++` ist der Compiler von GNU für die Sprache C++.

  Die GNU-Compiler sind sehr verbreitet und gelten als gut. Das Ziel der Organisation ist die Verbreitung freier Software (Open Source). Siehe [http://www.gnu.org/]. `g++` ist eine Variante von/intern eigentlich der `gcc` (der Name ist historisch, er compiliert verschiedene Sprachen).

- Der einfachste Aufruf des `g++`-Compilers ist:

  ```
  g++ hello.cpp
  ```

  Das C++-Programm wird übersetzt, mit den Bibliotheken gelinkt, und das ausführbare Programm in der Datei `a.out` abgelegt.

  Der Dateiname ist historisch zu verstehen: Ausgabe des Assemblers.

Aufruf des Compilers `g++` (2)

- Man führt das erzeugte Programm dann aus mittels

  ```
  ./a.out
  ```

  "." ist das aktuelle Verzeichnis. Man sagt also explizit, dass man das Programm "`a.out`" im aktuellen Verzeichnis ausführen möchte. Die meisten Programme, die man ausführt, stehen in Systemverzeichnissen. Z.B. würde Ihnen "`which g++`" sagen, wo das Programm `g++` gespeichert ist. Der Suchpfad bestimmt, in welchen Verzeichnissen automatisch nach Programmen gesucht wird. Sie können ihn sich mit `echo $PATH` anschauen (die Verzeichnisse sind durch ":" getrennt). Normalerweise ist aus Sicherheitsgründen das aktuelle Verzeichnis "." nicht enthalten (bösartige Nutzer hatten Programme angelegt, die wie normale Systembefehle hießen, und als der Administrator in ihrem Verzeichnis war, wurden diese Programme mit Admin-Rechten ausgeführt). Am Ende des Suchpfades wäre "." aber relativ sicher. Ist "." im Suchpfad enthalten, können Sie einfach "`a.out`" eingeben.

Aufruf des Compilers g++ (3)

- Man kann den Namen der erzeugten Programm-Datei festlegen mit der Option `-o`:

  `g++ -o hello hello.cpp`

  Nun wird das Programm in die Datei `hello` geschrieben, und kann entsprechend mit `./hello` aufgerufen werden.

  Es gibt keine spezielle Ende für Programmdateien. Programmdateien sind daran zu erkennen, dass bei `ls -l` das `x` ("execute") Bit angezeigt wird (das ist allerdings auch für Verzeichnisse gesetzt, und bedeutet dort, dass man in sie hinein wechseln darf). Mehr Informationen zum Dateityp bekommt z.B. mit `file hello`.

Aufruf des Compilers g++ (4)

- Übersetzung nur teilweise durchführen:
  - ◇ Compiler ausführen, aber nicht Linker:

    `g++ -c hello.cpp`

    Dies ist wichtig für Projekte aus mehreren Quelldateien (später).
  - ◇ Nur Präprozessor ausführen (`#include` etc.):

    `g++ -E hello.cpp >hello.ii`

    Ergebnis in `hello.ii`. Man kann so sehen, ob der Präprozessor den Text auf unerwartete Weise geändert hat (für Experten).
  - ◇ Compiler ausführen, aber nicht Assembler:

    `g++ -S hello.cpp`

    Das Assembler-Programm steht dann in `hello.s` (für Neugierige).

Aufruf des Compilers g++ (5)

- Es empfiehlt sich, alle Warnungen anzuschalten, die der Compiler bei verdächtigen Konstrukten (möglichen Fehlern) geben kann:

  `g++ -Wall -Wextra -o hello hello.cpp`

  Meist kann man den Fehler so leichter finden, als wenn er erst bei Ausführung des Programms auftritt, oder vielleicht nur in bestimmten Testfällen. Auch `-Wall -Wextra` sind noch nicht ganz alle Warnungen, siehe Handbuch oder "`g++ --help=warnings`".

- Man sollte Standard-konformes C++ schreiben (ohne spezielle g++-Erweiterungen):

  `g++ -ansi -pedantic -o hello hello.cpp`

Aufruf des Compilers g++ (6)

- Wenn man später einen Debugger verwenden will, ist es günstig, wenn der Compiler mehr Information im erzeugten Programm ablegt Das geht mit:

  `g++ -ggdb -o hello hello.cpp`

  Mit einem Debugger kann man ein Programm schrittweise ausführen, um damit Fehler zu finden. Die Zusatzinformation beschreibt die Beziehung zwischen Maschinenbefehlen und C++ Quellprogramm. Allgemeine Debugger-Information wird mit `-g` erzeugt, mit `-ggdb` erhält man zusätzliche Informationen speziell für den `gdb` Debugger. Wenn ein Programm fertig entwickelt ist, und ohne Fehler läuft, braucht man die Debugger-Information nicht mehr, und sollte diese Option weglassen. Besonders bei kommerziellen Programmen versucht man, die Symbol-Information zu entfernen (→ `man strip`), um die Rückgewinnung des Quellcodes ("Reverse Engineering") zu erschweren.

Aufruf des Compilers g++ (7)

- Wenn das erzeugte Programm möglichst schnell laufen soll, kann man es mit verschiedenen Stufen optimieren, z.B.

```
g++ -O2 -o hello hello.cpp
```

Das Compilieren dauert durch die Optimierung natürlich länger. Diese Option ist also nur interessant, wenn das Programm hinterher öfters ausgeführt wird, oder lange läuft (oder man einen Wettbewerb gewinnen will). Da bei der Optimierung Code umgestellt wird, wird das Debuggen schwieriger bis unmöglich. Diese Option ist also nur für fertig ausgeteste Programme gedacht. In sehr seltenen Fällen könnte ein Programm, das ohne Optimierung funktioniert hat (aber schon versteckte Fehler/Unsauberkeiten enthielt) mit Optimierung eventuell nicht mehr laufen. Sie auch `g++ --help=optimizers`.

Make (1)

- Wenn Ihnen die Eingabe der Optionen zum `g++` zu mühsam wird, wäre die professionelle Lösung, das Werkzeug `make` zu verwenden.

  `make` ist eigentlich für größere Projekte mit mehreren C++-Quelldateien gedacht, und sorgt dafür, dass nur die jeweils von Änderungen betroffenen Dateien neu compiliert werden.

- Sie schreiben die Abhängigkeiten und auszuführenden Kommandos in eine Datei `Makefile` (s.u.) und geben dann nur noch das Kommando `make` ein.

  Ein Nebeneffekt ist, dass, wenn das übersetze Programm `hello` bereits existiert, und `hello.cpp` nicht geändert wurde, `make` darüber informiert und weiter nichts tut.

Make (2)

- Legen Sie eine Datei `Makefile` (oder `makefile`) mit folgendem Inhalt an:

```
CXXFLAGS = -Wall -Wextra -ansi -pedantic -ggdb

hello: hello.o
	g++ -o hello hello.o

hello.o: hello.cpp
	g++ -c $(CXXFLAGS) hello.cpp
```

- Es ist wichtig, dass die eingerückten Zeilen mit einem Tabulator-Zeichen beginnen.

Make (3)

- Die erste Zeile definiert eine Abkürzung (“Makro”) `CXXFLAGS`. Anschließend wird dann `$(CXXFLAGS)` immer durch die Optionen `-Wall` u.s.w. ersetzt.

  Solange man nur eine Quelldatei hat, bringt die Abkürzung höchstens etwas mehr Übersichtlichkeit. Wird der Compiler dagegen mehrfach aufgerufen, sollte man die Optionen an zentraler Stelle ändern können.

- Dann kommen zwei Regeln, eine für den Aufruf des Compilers, die andere für den Aufruf des Linkers.

  Solange es nur eine C++-Quelldatei gibt, könnte man das natürlich auch in einem Schritt machen. Eventuell ist ein Vorteil, dass man klarer sieht, ob die Fehlermeldungen vom eigentlichen Compiler oder vom Linker kommen. Der Linker wird hier auch über `g++` aufgerufen, weil dann automatisch die C++-Bibliotheken dazugebunden werden.

Make (4)

- Die Regeln bestehen aus drei Teilen:
  - ◇ Links vom Doppelpunkt das zu erzeugende Ziel.
  - ◇ Rechts vom Doppelpunkt die Dateien, von denen es abhängig ist.
  - ◇ In den mit Tabulator eingerückten Zeilen darunter stehen die Kommandos, die auszuführen sind, um das Ziel zu erzeugen.

    Angenommen, `make` soll das Ziel einer Regel erstellen. Dazu wird zuerst rekursiv die Existenz und Aktualität der Dateien rechts vom ":" geprüft, und sie ggf. mit ihren Regeln erstellt. Dann werden die Kommandos dieser Regel ausgeführt, wenn das Ziel nicht existiert, oder eine der Dateien rechts vom ":" geändert wurde, nachdem das Ziel zuletzt geändert wurde.

Make (5)

- Wird `make` ohne Argumente aufgerufen, versucht es, das erste Ziel im `Makefile` zu erstellen.

  Insofern ist die Reihenfolge der Regeln im Beispiel wichtig.

- Oft wird ein Default-Ziel (Voreinstellung) explizit mit der ersten Regel definiert, danach ist die Reihenfolge der weiteren Regeln egal:

```
all: hello
```

  Es ist kein Problem, dass eine Datei `all` nie erzeugt wird, und diese Regel auch gar keine Kommandos enthält. Wenn `all` das erste Ziel im `Makefile` ist, wird `make` dann automatisch rekursiv das Ziel `hello` erstellen. Danach würden die Kommandos dieser Regel ausgeführt, die sind aber leer.

Make (6)

- Man kann weitere Befehle ins `Makefile` schreiben:

```
test: hello
        ./hello

clean:
        rm -f hello hello.o
```

- Gibt man nun `make test` ein, wird `hello` ausgeführt, nachdem es bei Bedarf vorher übersetzt wurde.

  An dem Verhalten bei Eingabe von `make` (ohne explizites Ziel) ändert sich durch die zusätzlichen Regeln (unten in der Datei) nichts.

- `make clean` löscht die erzeugten Dateien.

  `-f` ("force") vermeidet eine Fehlermeldung, wenn sie nicht existieren.

Make (7)

- Kommentare in Makefiles beginnen mit `#` und erstrecken sich bis zum Zeilenende.

  Kommentare sind Erklärungen für Menschen, die die jeweilige Datei verstehen sollen. Das Programm, das die Datei verarbeitet, ignoriert die Kommentare. Das entspricht also genau den oben besprochenen Kommentaren in C++, nur dass die Syntax hier anders ist (`#` statt `//`).

- Wenn Sie genauer sehen wollen, was `make` macht, probieren Sie:

```
make --debug
```

- Siehe auch: [http://www.gnu.org/software/make/manual/]

Fehlermeldungen (1)

- Wenn man das Programm richtig abgetippt hat, sollte es ohne Fehlermeldungen übersetzt werden.
- Es ist aber normal, dass man beim Übersetzen gelegentlich Fehlermeldungen des Compilers bekommt.
- Man muss dann verstehen, was die Ursache ist, und die C++-Datei mit dem Editor entsprechend korrigieren. Dann ruft man den Compiler erneut auf.

  Manchmal bekommt man auch viele Fehlermeldungen. Das ist kein Grund zur Panik. Oft ist es nur eine kleine Ursache (z.B. eine fehlende "{"), die für alle Fehlermeldungen verantwortlich ist. Gelegentlich führt auch die vom Compiler implizit durchgeführte Korrektur zu weiteren Fehlermeldungen ("Folgefehler").

Fehlermeldungen (2)

- Beispiel: Man hat das Semikolon nach der Ausgabeanweisung vergessen:

```
...
int main()
{
        cout << "Hello, world!\n" ←—
        return 0;
}
```

- Der Compiler g++ gibt folgende Fehlermeldung aus:

```
hello.cpp:10: error: expected ';' before 'return'
```

  In der Datei `hello.cpp`, Zeile 10, erwartet er ein Semikolon vor dem Schlüsselwort `return`. Zeile 10 ist die `return`-Zeile der eigentliche Fehler ist in der Zeile darüber.

Fehlermeldungen (3)

- Die Fehlermeldung bezieht sich recht häufig auf eine Stelle kurz nach dem eigentlichen Fehler:
  - ◇ Der Compiler liest das Programm von vorne nach hinten (Teile werden nachträglich weiterverarbeitet).
  - ◇ Er gibt eine Fehlermeldung aus, wenn keine gültige Fortsetzung mehr möglich ist.
  - ◇ Die Ausgabeanweisung hätte aber noch fortgesetzt werden können, z.B. mit `<< "Here I am.\n";`
  - ◇ Erst wenn der Complier das `return` gesehen hat, ist klar, dass ein Syntaxfehler vorliegt.

Fehlermeldungen (4)

- Beispiel: Man hat den Typ `int` groß geschrieben:

```
INT main()
```

- Der Compiler g++ gibt folgende Fehlermeldung aus:

```
hello.cpp:7: error: 'INT' does not name a type
```

- Die Groß/Kleinschreibung ist in C++ wichtig!
- An dieser Stelle erwartet der Compiler einen Typ und er beschwert sich korrekt, dass es keinen Typ mit Namen `INT` gibt.

  Einige Schlüsselworte wie z.B. `class` wären an dieser Stelle auch möglich, die Fehlermeldung zielt also nur auf den häufigsten Fall.

Fehlermeldungen (5)

- Beispiel: Man hat für die Zeichenkettenkonstante einfache Anführungszeichen ' (Apostroph) statt der korrekten doppelten " verwendet:

```
cout << 'Hello, world!\n';
```

- Man erhält folgende Fehlermeldung:

```
hello.cpp:9:10: warning:
      character constant too long for its type
```

- Man muß dazu wissen, dass ' in C++ für Zeichenkonstanten (einzelne Zeichen) verwendet wird.

Fehlermeldungen (6)

- Beispiel: Man nennt die Funktion nicht `main`, sondern `hauptprogramm`:

```
int hauptprogramm()
```

- Die Fehlermeldung (siehe nächste Folie) kommt jetzt nicht vom Compiler,

  Ein C++ Programm kann beliebige viele Funktionen enthalten, für die Sie die Namen relativ frei wählen dürfen, insbesondere dürfen Sie eine Funktion mit Namen `hauptprogramm` definieren.

- sondern vom Linker: Es muss immer eine Funktion mit Namen `main` geben.

Fehlermeldungen (7)

- Man erhält folgende Fehlermeldung:

```
/usr/lib/.../crt1.o:
In function ‘_start’:
(.text+0x20): undefined reference to ‘main’
collect2: ld returned 1 exit status
```

Wie oben erläutert, wird jedes C++Programm mit einem Rahmenprogramm verbunden, das sich um Intialisierungen kümmert, und dann `main` aufruft. Teil davon ist `crt1.o`. Dies ist bereits compiliert, deswegen können sich Fehlermeldungen nicht mehr auf Zeilen beziehen. Im Programmsegment, Adresse 20 (hexadezimal) steht der Sprungbefehl zu `main`. Diesen kann der Linker `ld` nicht auflösen. Wir haben `g++` aufgerufen, der seinerseits intern das Programm `collect2` benutzt, das wiederum den Linker `ld` aufruft. Man muss das alles aber nicht verstehen, "`undefined reference to function main`" ("undefinierter Verweis auf Funktion "`main`") sagt ja eigentlich alles.

Fehlermeldungen (8)

- Man kann viel aus Fehlern lernen, aber nur, wenn Sie vollständig aufgeklärt werden.

  Rufen Sie ggf. den Tutor zu Hilfe. Wenn kein Tutor verfügbar ist, speichern Sie die C++-Datei noch einmal unter einem anderen Namen ab, so daß Sie ggf. später oder per EMail/Forum Hilfe bekommen können. Natürlich können Sie ein bisschen herumprobieren (das Programm ändern, und die Reaktion des Compilers anschauen). Es ist aber sehr wichtig, dass Sie am Ende verstanden haben, was der ursprüngliche Fehler war. Geben Sie sich nicht damit zufrieden, dass es zufällig funktioniert. Wenn Sie den Fehler nicht verstanden haben, werden Sie ihn wieder machen. Außerdem funktioniert das Programm vielleicht nur für diese eine Eingabe, aber nicht allgemein.

Inhalt

MS Visual C++ (1)

- Sie können die Entwicklungsumgebung verwenden, mit der Sie am besten zurechtkommen.

  Abgeben müssen Sie nur den C++ Programmcode. Der Programmcode muß allerdings mit dem Compiler g++ unter Linux lauffähig sein, Sie dürfen also z.B. keine speziellen Microsoft-Erweiterungen des Sprachumfangs oder spezielle Windows-Systemaufrufe verwenden. Es empfiehlt sich also, das abzugebene Programm am Ende nochmal unter Linux zu testen, wenn Sie es unter Windows entwickelt haben.

- Die Benutzung von Linux ist erwünscht, aber nicht erzwungen.

  Wir glauben, dass es gut für Sie wäre, wenn Sie bis zum Ende Ihres Studiums UNIX/Linux kennengelernt haben. Aber wenn Sie sich im Moment überfordert fühlen, wäre C++ das Wichtige, nicht Linux.

MS Visual C++ (2)

- Als Beispiel soll im Folgenden die Benutzung von Microsoft Visual Studio 2010 kurz erläutert werden.

  Microsoft Visual Studio ist auf den Rechnern im Windows-Pool installiert. Es ist ein umfangreiches Paket für die Entwicklung von Programmen in verschiedenen Programmiersprachen, wir interessieren uns hier nur für den C++-Teil.

- Microsoft Visual C++ Express ist kostenlos.

  Und teils noch einfacher zu benutzen, weil es weniger Funktionen gibt. Siehe:
  [http://www.microsoft.com/germany/express/products/windows.aspx]

MS Visual C++ (3)

- Start → Alle Programme → Entwicklungsumgebungen → Microsoft Visual Studio 2010 → Microsoft Visual Studio 2010.

  Starten Sie einen Rechner im PC-Pool unter Windows, oder öffnen von einem "Thin Client" eine Sitzung auf dem Windows Server. Loggen Sie sich ein. "Start" ist rechts unten im Bildschirm. Klicken Sie mit der Maus darauf (linke Taste). Es erscheint ein Menü. Klicken Sie auf den Menüpunkt "Programme". Es eröffnet sich ein Untermenü (schon wenn Sie den Mauszeiger auf "Programme" bewegen und darauf etwas verweilen). Und so weiter. Schneller geht es, wenn Sie die Maustaste drücken, wenn der Zeiger auf "Start" steht, und die Taste gedrückt halten, bis Sie den richtigen Untermenüpunkt gefunden haben, und dann loslassen.

MS Visual C++ (4)

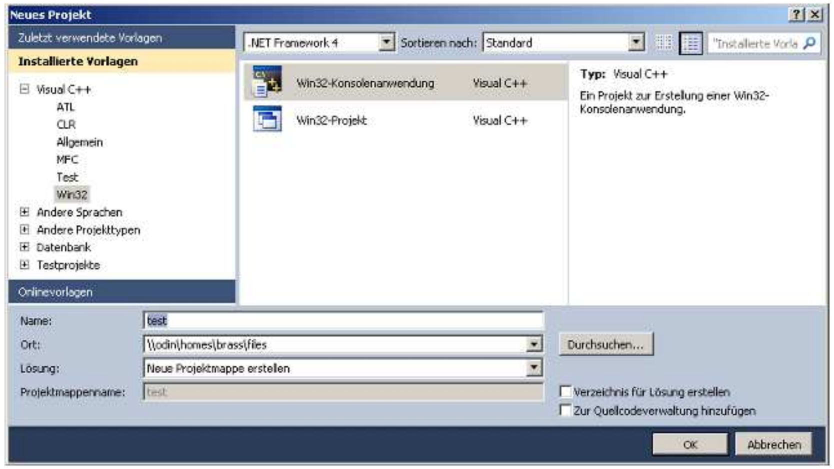
- Beim ersten Mal erscheint “Standardumgebungseinstellungen auswählen”. Wählen Sie “Visual C++ Entwicklungseinstellungen”, klicken Sie dann auf “Visual Studio starten”.
- Jetzt startet die Entwicklungsumgebung.
- Wählen Sie Datei → Neu → Projekt.
- Wählen Sie in der Dialogbox zuerst unter Installierte Vorlagen links “Win32”, anschließend unter Projektvorlagen rechts “Win32-Konsolenanwendung”.

MS Visual C++ (5)

- Geben Sie im unteren Teil der Dialogbox einen Namen für Ihr Projekt ein (z.B. `hello` oder `test`).
- Bei Visual Studio gibt es noch den Begriff “Projektmappe”: Sie kann mehrere Projekte enthalten.

  Wie ein Verzeichnis mehrere Unterverzeichnisse enthält. Sie könnten das neue Projekt einer existierenden Projektmappe zuordnen. Für den Anfang ist es aber das Einfachste, wenn jede Projektmappe genau ein Projekt enthält. Lassen Sie die Standardeinstellung in der Dialogbox “Neue Projektmappe erstellen”.
- Entfernen Sie das Häkchen bei “Verzeichnis für Lösung erstellen”.

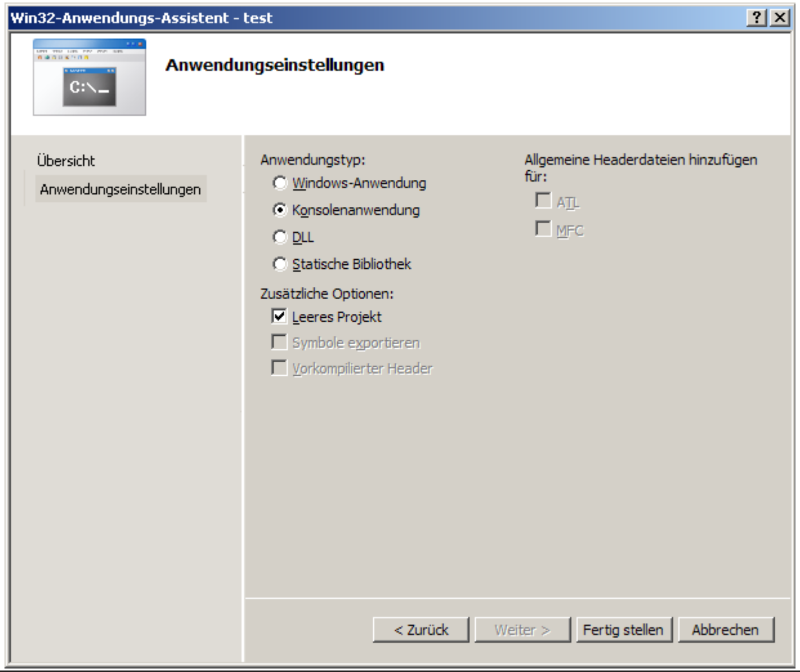
  Sonst erhalten Sie eine Verzeichnis-Schachtelungsebene mehr.



MS Visual C++ (7)

- Klicken Sie in “Neues Projekt erstellen” auf “OK”.
- Es erscheint der Win32-Anwendungs-Assistent.

  Er könnte Ihnen schon eine Datei mit etwas C++-Code erzeugen. Die wäre für uns aber unpassend, da sie nur unter Windows funktioniert. Sie müssen aber das oben gezeigte “Hello World” Beispiel als Grundlage verwenden, damit die Programme auch unter Linux funktionieren.
- Klicken Sie im linken Bereich auf “Anwendungseinstellungen” (Optionen zum neuen Projekt).
- Löschen Sie das Häkchen bei “Vorkompilierter Header”, setzen Sie ein Häkchen bei “Leeres Projekt”.
- Klicken Sie dann auf “Fertig stellen”.



MS Visual C++ (9)

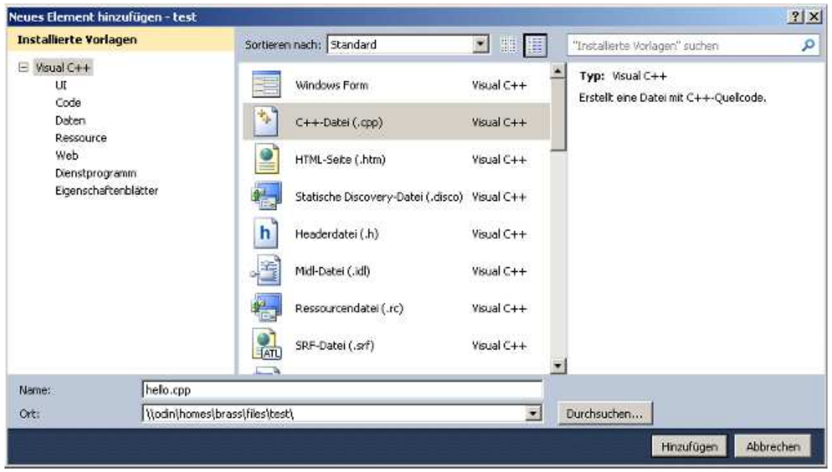
- Es erscheint eine Warnung, dass Suchdatenbank und IntelliSense-Datei nur auf einem lokalen Laufwerk gespeichert werden können.

  Ich verstehe das leider nicht, auch nicht die Konsequenzen, wenn es in einem temporären Verzeichnis gespeichert wird, wie Visual Studio dann vorschlägt. Wenn Sie dazu mehr wissen, sagen Sie es mir. Ansonsten klicken Sie einfach auf "OK".

- Das erstellte Projekt ist zunächst leer. Es muss eine Quellcodedatei "`hello.cpp`" hinzugefügt werden.
- Dafür gibt es zwei Möglichkeiten, siehe nächste Folien.

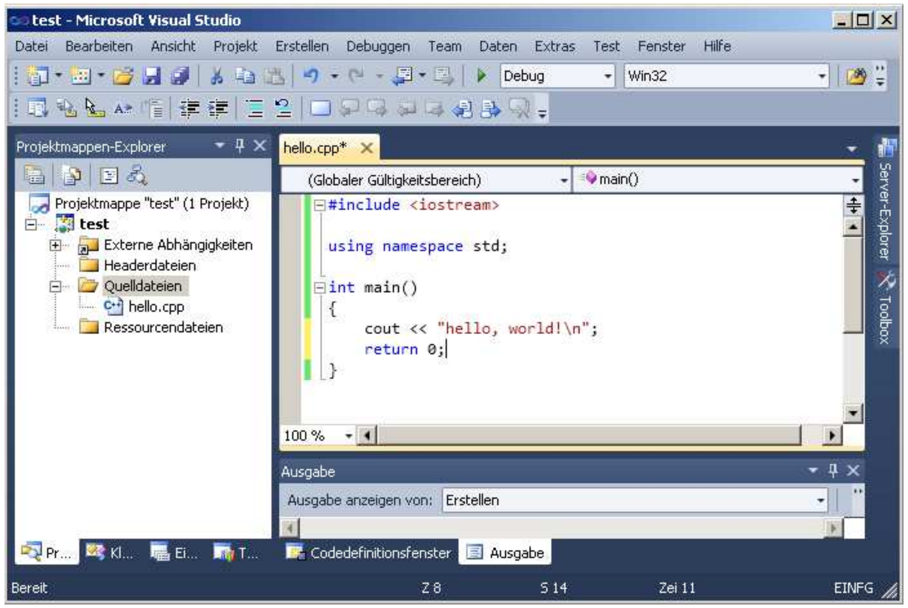
MS Visual C++ (10)

- Bewegen Sie die Maus über "Quelldateien" im linken Bereich ("Projektmappen-Explorer") drücken Sie die die rechte Maustaste.
- Es erscheint ein "Pop-Up Menu", wählen Sie "Neues Element hinzufügen".
- Wählen Sie im oberen Bereich "C++-Datei (.cpp)".
- Geben Sie unten einen Dateinamen ein (`hello.cpp`).
- Klicken Sie auf "Hinzufügen".



MS Visual C++ (12)

- Alternative: Wählen Sie Datei → Neu → Datei.
- Wählen Sie links die Kategorie "Visual C++" und prüfen Sie, daß rechts C++ Datei (.cpp) als Dateityp ausgewählt ist.
- Die Datei heißt zunächst "Quelle1.cpp".
- Wählen Sie nun Datei → Verschiebe Quelle1.cpp in → hello (das Projekt, das Sie angelegt haben).
- Es erscheint die "Datei speichern unter" Dialogbox, dort können Sie einen Dateinamen eingeben, z.B. "hello.cpp".



MS Visual C++ (14)

- Es erscheint ein Fenster, in das Sie das Beispielprogramm eingeben können.

  Machen Sie sich mit den Cursortasten (Pfeile) (Pfeiltasten) vertraut, sowie mit Backspace, Delete, der direkten Positionierung der Cursors mit der Maus, etc.

- Wählen Sie nun Erstellen → hello erstellen.
- Es erscheinen Meldungen, daß der Compiler und der Linker aufgerufen werden.

  Falls Sie beim Eintippen des Programms einen Fehler gemacht haben, erscheinen Fehlermeldungen. Durch Anklicken der Fehlermeldung kommen Sie an die Stelle des Programms, an der der Compiler den Fehler erkannt hat. Der tatsächliche Fehler liegt möglicherweise davor.

MS Visual C++ (15)

- Falls es fehlerfrei durch Compiler und Linker gelaufen ist, können Sie Ihr Programm mit "Debuggen → Starten ohne Debuggen" ausführen.

  Alternativ können Sie auch Crtl+F5 drücken. Für Konsole-Programme wird automatisch der "Command Prompt" `cmd.exe` gestartet. Es erscheinen verschiedene Meldung vor der Ausgabe "Hello, world!" Ihres Programms. Danach hat Visual C++ dafür gesorgt, dass `cmd.exe` noch auf einen Tastendruck wartet, bevor es sich beendet (sonst könnten Sie die Ausgabe nicht sehen, weil das Fenster gleich wieder geschlossen wird). Das ist aber keine Funktion Ihres Programms.

- Bei der Variante mit Debuggen können Sie einen "Breakpoint" setzen, und das Programm im Einzelschrittmodus abarbeiten.

MS Visual C++ (16)

- Die Datei **hello.exe** (das ausführbare Programm) steht im Ordner **Debug** im Projekt-Ordner.

  Visual Studio kann verschiedene Konfigurationen eines Projektes verwalten. Standardmässig werden bei der Projekterstellung "`Debug`" und "`Release`" angelegt. Die Konfigurationen unterscheiden sich insbesondere in den Optionen für Compiler und Linker. Am Anfang ist die "`Debug`"-Konfiguration ausgewählt, das ist zur Programmentwicklung sinnvoll. Wählt man "`Release`", laufen die erzeugten Programme schneller, Fehler werden aber nicht so leicht gefunden.

- Sie können **hello.exe** auch außerhalb von Visual Studio ausführen (im "Command Prompt").

  Wenn Sie auf eine Eingabe warten wollen, bevor sich das Programm beendet, können Sie `string s; getline(cin,s);` hinzufügen.

Schlussbemerkung

- Computer gehen durch Fehlbedienung (über Tastatur und Maus) praktisch nicht kaputt.

  Durch gewaltsames Hereinstecken einer Floppy Disk verkehrt herum dagegen schon.

- Schlimmstenfalls können Sie sich Ihre eigenen Dateien löschen oder überschreiben.

  Deswegen empfiehlt es sich, sie regelmäßig zu sichern, z.B. auf einen USB-Stick. Wenn Sie als Administrator arbeiten, können Sie den Rechner dagegen so unbrauchbar machen, daß Sie das Betriebssystem neu installieren müssen.

- Man kann also auch mal etwas ausprobieren.