Objektorientierte Programmierung
(Winter 2006/2007)

Kapitel 1: Einführung

- Computer-Grundbegriffe, Editor, Compiler, Linker
- Geschichte von C++
- Erstes Beispiel
- Bedienung einer Entwicklungsumgebung

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) 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).

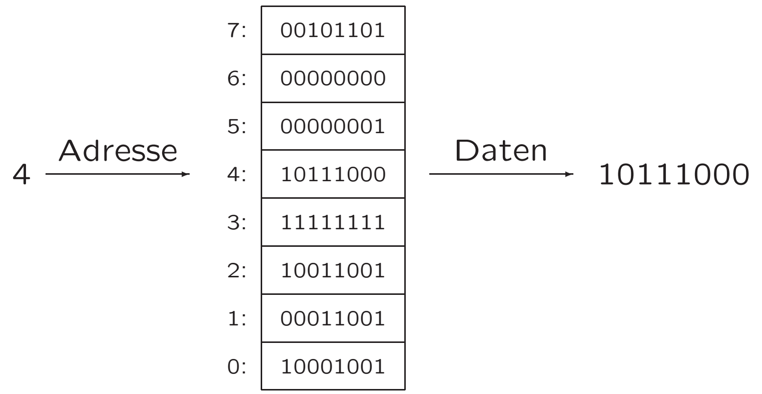
  Man kann Bytes zu größeren Einheiten (Worte) zusammenfassen, manche CPUs können auch einzelne Bits ansprechen. Deswegen kann man nicht genau sagen, was eine einzelne Speicherzelle ist.

- 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.

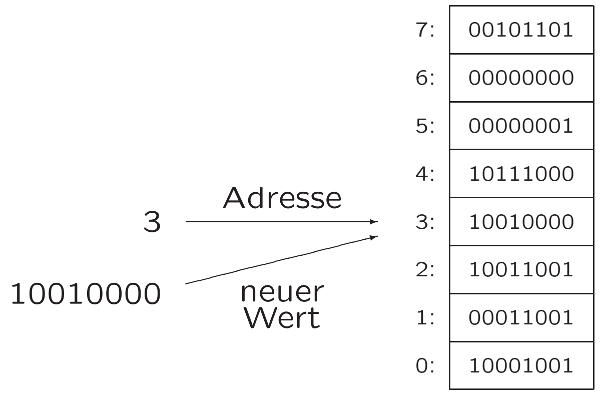
Computer, Programme (3)

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.

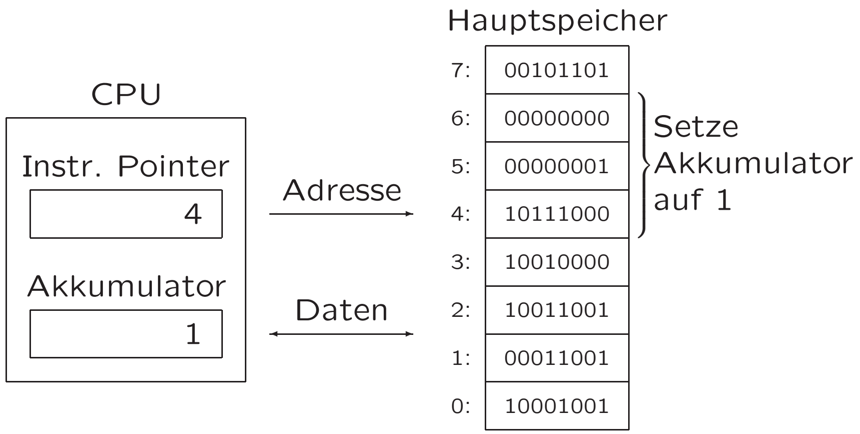
Computer, Programme (6)

- 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)

- 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 (11)

- 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 (12)

- 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 (13)

- 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++.

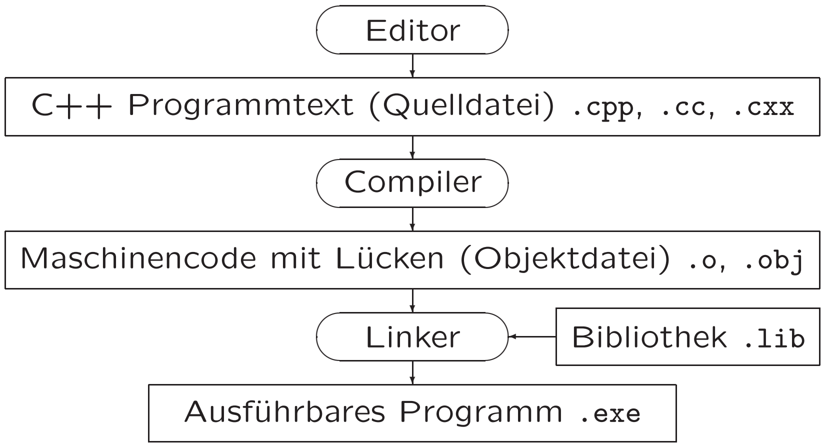
Computer, Programme (14)

- 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 (15)



Computer, Programme (16)

- 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 (17)

- 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)

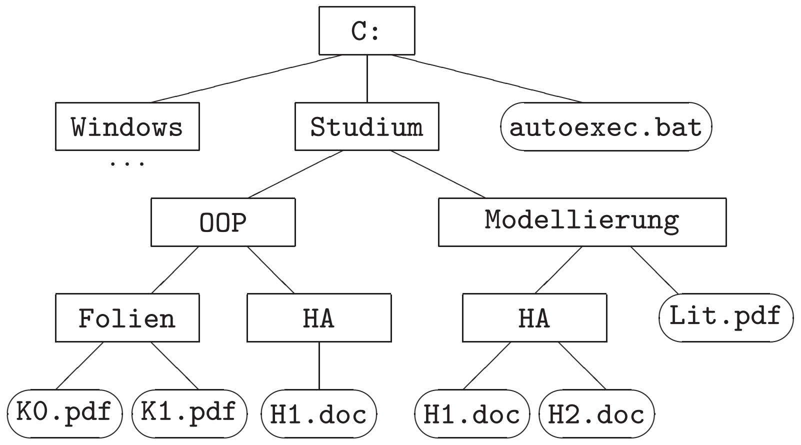
- Beim Betriebssystem UNIX (Linux, Solaris, ...) gibt es nur ein Hauptverzeichnis.

  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.

Historische Bemerkungen (1)

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

  Siehe: Ritchie: Development of the C Language.
  [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.

  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)

- Bjarne Stroustrup begann 1979, in den Bell Labs an einer objektorientierten Erweiterung von C zu arbeiten, die zuerst "new C", dann "C with Classes", und ab 1983 "C++" hieß.

  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.
- C ist im Prinzip eine Teilmenge von C++.
- C++ ist wesentlich größer/komplexer als C.

Historische Bemerkungen (4)

- 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.

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 der Klasse `ostream`. Es repräsentiert den Standard-Ausgabestrom (Bildschirm).
- `"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.

Benutzung einer IDE (1)

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

  Abgeben müssen Sie nur den C++ Programmcode.
- Als Beispiel verwenden wir Microsoft Visual C++ (Teil von Microsoft Visual Studio).

  Die Übungsleiterin hätte lieber Eclipse verwendet. Das ist eine gute Alternative. Ich verwende unter UNIX meist einfach den `gcc` als Compiler, sowie `make` zum automatischen Aufruf von Compiler und Linker, und `vi` als Editor. Das bin ich seit vielen Jahren so gewöhnt.
- Microsoft Visual Studio ist auf den Rechnern im Windows-Pool installiert.

Benutzung einer IDE (2)

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

  Dazu müssen Sie ggf. den Rechner und/oder den Bildschirm einschalten. Nach dem Einschalten des Rechners dauert es einige Minuten, bis die Aufforderung zum Einloggen erscheint (das Betriebssystem “fährt hoch”, d.h. führt Initialisierungsarbeiten durch und startet verschiedene Hintergrundprozesse). Falls der Rechner bereits eingeschaltet ist, 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.

Benutzung einer IDE (3)

- Dann wählen Sie Start → Programme → Entwicklungsumgebungen → Microsoft Visual Studio .NET 2003 → Microsoft Visual Studio .NET 2003.

  “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.

- Jetzt startet die Entwicklungsumgebung.

  Das dauert wieder etwas.

Benutzung einer IDE (4)

- Wählen Sie Datei → Neues Projekt.
- Es erscheint eine Dialogbox.
  - ◇ Hier können Sie den Projekttyp auswählen. Wählen Sie “C/C++ Konsolenanwendung”.
  - ◇ Wählen Sie einen Projektnamen, z.B. hello.
  - ◇ Sie müssen auch eine Projektmappe angeben (einen Ordner für die Dateien des Projekts).
  - ◇ Klicken Sie am Ende auf “Fertig stellen”.

Benutzung einer IDE (5)

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

  Machen Sie sich mit den Cursortasten (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.

Benutzung einer IDE (6)

- 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.

- Das können ganz einfache Dinge wie ein fehlendes Semikolon sein.

  Der Compiler ist ein Computer-Programm: Es führt exakt die von seinem Programmierer vorgegebenen Regeln aus, aber es “denkt nicht mit”. Formale Syntax besprechen wir später (im dritten Kapitel).

Benutzung einer IDE (7)

- 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.

Benutzung einer IDE (8)

- 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.

- Die Datei `hello.exe` (das ausführbare Programm) steht in `hello\Debug` in dem Projektmappen-Ordner.
- Sie können es auch außerhalb von Visual Studio ausführen.

  Da es nicht auf eine Eingabe wartet, ist das geöffnete Fenster sofort wieder verschwunden. Sie können es aber im MS-DOS Prompt (Kommandoschnittstelle) ausführen, um die Ausgabe zu sehen.

Benutzung einer IDE (9)

Noch etwas zum Schluß:

- 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.