Ein- und Ausgabe mit Dateien

Edouard Lamboray
Informatik I für D-ITET (2004)

- Ein- und Ausgabekonzepte in C und C++
- `fstream` header
- Streamobjekt
- Files schreiben und lesen
- Argumente von `main`

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Copyright: M. Gross, E. Lamboray, ETHZ, 2004

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Vorbemerkungen

- Methoden zur Ein- und Ausgabe sind in C und C++ nicht Teil der Programmiersprache
- Historisch: Benutzer sollte Freiheit haben, eigene I/O Routinen für seine HW zu schreiben
- I/O Paket wurde zunächst für UNIX entwickelt und dann von ANSI standardisiert
  - `<stdio.h> // Original C-Header`
  - `<cstdio>  // C++-Anpassung`
- Library mit Hilfsfunktionen für Ein- und Ausgabe
- Enthält Funktionen wie z.Bsp.
  - `printf(); // C-Version von cout`
  - `scanf();  // C-Version von cin`
  - `fprintf();`
  - `fscanf();`

Copyright: M. Gross, E. Lamboray, ETHZ, 2004

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Vorbemerkungen

- In C++ wurde ein neues, objektorientiertes Konzept für Ein- und Ausgabe entwickelt
  - `<iostream>`
- Enthält Klassen zur Instantiierung von Objekten zur Ein- und Ausgabe
  - `cout  // Objekt`
- Konzeptionell wird Ein- und Ausgabe als Strom *(stream)* von Bytes angesehen
- Somit Ein- und Ausgabe über gleiches Interface auf
  - Tastatur
  - Bildschirm
  - Festplatte/File
  - andere Perpherie

Copyright: M. Gross, E. Lamboray, ETHZ, 2004

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Files in UNIX

- In UNIX ist ein File eine unstrukturierte Menge von Bytes
- Jede Struktur muss dem File von aussen (Anwendungsprogramm) aufgeprägt werden
- Alle internen und externen Devices werden als Files interpretiert
  - und mit gleichen Methoden bearbeitet
- Ein *Stream* verbindet eine Datenquelle (z. B. File) und eine Datensenke (z.B. Euer Programm)
- Quelle und Senke werden mittels **`ifstream`** und **`ofstream`** Objekten dargestellt.
- Wenn das File geöffnet ist, kann darauf zugegriffen werden
- Am Schluss muss das File geschlossen werden

Copyright: M. Gross, E. Lamboray, ETHZ, 2004

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

fstream

- Wichtige Funktionen zur Ein- und Ausgabe
  - `open()  // Oeffnen eines Files`
  - `close()  // Schliessen eines Files`
  - `eof()  // End of File`
  - `>>  // Formatiertes Lesen von Files`
  - `<<  // Formatiertes Schreiben von Files`
  - `read()  // schnelles, unformatiertes Lesen von Binärdaten`
  - `write()  // schnelles, unformatiertes Schreiben von Binärdaten`

Copyright: M. Gross, E. Lamboray, ETHZ, 2004

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Formatierte Ein- und Ausgabe

- Unterschied zwischen Textfiles und Binärfiles
- Zugriff auf Datei von einem Anwendungsprogramm aus, umfasst 5 Komponenten
  - Einbindung des Headers `#include <fstream>`
  - Definition eines File-Pointers: `ofstream fout;`
  - Öffnen der Datei: `fout.open("myDat");`
  - Datei lesen/schreiben: `fout << "Hello World!";`
  - Schliessen der Datei: `fout.close();`
- Einfache Beispiele für Lesen und Schreiben

Copyright: M. Gross, E. Lamboray, ETHZ, 2004

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Beispiel_1: Einfaches Schreiben

```
// simplewrite.cpp -- file write

#include <iostream>
#include <fstream>
using namespace std;

int main()
{
     // file object
     ofstream fout;

     // file opening
     fout.open("mydat");

     if (!fout.is_open())
     {
          cout << "File open error \n";
          return 0;
     }

     int a = 12345;

     // write file
     fout << a;

     // close file
     fout.close();

     return 0;
}
```

- Datei öffnen
- Schreiben
- Schliessen

Copyright: M. Gross, E. Lamboray, ETHZ, 2004

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Beispiel_2: Einfaches Lesen

```
// simpleread.cpp -- file read

#include <iostream>
#include <fstream>
using namespace std;

int main()
{

    // file object
     ifstream fin;

     // file open
     fin.open("mydat");

     if(!fin.is_open())
     {
          cout << "File open error \n";
          return 0;
     }

     int a;

     // read from file
     fin >> a;
     cout << "Inhalt der Datei:    " << a << "\n";

     // close file
     fin.close();

     return 0;
}
```

- Datei öffnen
- Lesen
- Schliessen

Copyright: M. Gross, E. Lamboray, ETHZ, 2004

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Zeichenweise Lesen und Schreiben

- Zeichenweises Lesen und Schreiben kann durch Hilfsfunktionen vereinfacht werden
- Folgende Funktionen als Beispiel
  - `int put(int c);`
  - `int get();`
- Zeichen c wird nach `unsigned char` umgewandelt und geschrieben oder gelesen
- Das Dateiende wird durch `eof()` angezeigt
- Nützliche Funktion, um Dateien zeichenweise zu scannen
- `fstream` bietet eine Vielzahl anderer Funktionen an

Copyright: M. Gross, E. Lamboray, ETHZ, 2004

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Beispiel_3: Einfaches Copy

```
// Filecopy Program Version 1
#include <iostream>
#include <fstream>
using namespace std;

void filecopy (ifstream &fin, ofstream &fout); // prototype

int main() {
     ifstream fin;  // Define input and output file objects
     ofstream fout;

     char inname[20], outname[20];      // Read file names from command line
     cout << "Filename in?\n";          cin >> inname;
     cout << "Filename out?\n";         cin >> outname;

     fin.open(inname);                  // Open files
     if(!fin.is_open()) {
          cout << "can't open infile \n";
          return 0;
     }
     fout.open(outname);
     if(!fout.is_open()) {
          cout << "can't open outfile \n";
          fin.close();
          return 0;
     }

     filecopy (fin, fout);    // Call file copy function

     fin.close();             // Close files
     fout.close();
     return 0;
}
```

Copyright: M. Gross, E. Lamboray, ETHZ, 2004

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Beispiel_3: Einfaches Copy 2

```
void filecopy (ifstream &fin, ofstream &fout) {
    int c;

    // scan file char by char and copy
    while (true) {
        c = fin.get();
        if(fin.eof()) return;
        fout.put(c);
    }
}
```

Copyright: M. Gross, E. Lamboray, ETHZ, 2004

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Argumente von `main()`

- Argumente der **`main`**-Funktion ermöglichen den Programmaufruf mit Uebergabeparametern
- Beispiel: **`cp myfile yourfile`**
- Programmaufruf **`cp`** enthält zwei String-Parameter mit den Filenamen
- Werden beim Aufruf vom System an das Anwendungsprogramm übergeben
- Dazu sind zwei Argumente notwendig
  - **`void main(int argc, char *argv[])`**
- **`argc`**: Integer, enthält die Anzahl der Argumente
- **`argv`**: Array von **`char`**–Pointern mit den Anfangsadressen der Argumente (grundsätzlich Strings!)

Copyright: M. Gross, E. Lamboray, ETHZ, 2004

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Beispiel UNIX cp

- Eine etwas elaboriertere Variante von **`cp`** soll nun folgendermassen aufgerufen werden
  - **`cp <filename1> <filename2>`**
- Die Argumente von **`main`** beinhalten nun die folgenden Daten
  - **`argv[0]=„cp“`**
  - **`argv[1]=<filename1>`**
  - **`argv[2]=<filename2>`**

Copyright: M. Gross, E. Lamboray, ETHZ, 2004

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Beispiel_4: Verbessertes Copy

```
// Filecopy Program Version 2

#include <iostream>
#include <fstream>
using namespace std;

void filecopy (ifstream &fin, ofstream &fout); // prototype

int main(int argc, char **argv) {
     // check arguments
     if(argc < 3) {
          cout << "copy <infile> <outfile>\n";
          return 0;
     }
     // Define input and output file objects
     ifstream fin; ofstream fout;
     // Open files
     fin.open(argv[1]);
     if(!fin.is_open()) {
          cout << "can't open infile \n";
          return 0;
     }
     fout.open(argv[2]);
     if(!fout.is_open()) {
          cout << "can't open outfile \n";
          fin.close();
          return 0;
     }
     // Call file copy function
     filecopy (fin, fout);
     // Close files
     fin.close(); fout.close();
     return 0;
}
```

Copyright: M. Gross, E. Lamboray, ETHZ, 2004

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Ein- und Ausgabe von Binärdateien

- Komplexe Datentypen können wesentlich effizienter in Binärdateien geschrieben werden
- Dabei muss die Variable nach **`char*`** gecastet werden und die Grösse der Variable muss angegeben werden
- Funktionen dazu
  - **`read(char *data, size_t size);`**
  - **`write(char *data, size_t size);`**
- Beim Umgang mit komplexeren Datentypen ist Vorsicht geboten
- Der Stream muss beim Lesen jeweils geprüft werden

Copyright: M. Gross, E. Lamboray, ETHZ, 2004

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Beispiel_5: Binärdateien

```
#include <iostream>
#include <fstream>
using namespace std;

struct planet {
     char name[16];          // name of the planet
     double population;      // its population
     double g;               // its acceleration of gravity
};

int main() {
     planet pout = {"Earth", 6e9, 9.81};

     // Writing binary file
     ofstream fout;
     fout.open("planets");
     fout.write((char *)&pout, sizeof planet);
     fout.close();

     // Reading binary file
     ifstream fin;
     planet pin;
     fin.open("planets");
     fin.read((char *)&pin, sizeof planet);
     // Testing the stream on errors
     if(fin.eof() || fin.fail() || fin.bad()) cout << "Error while reading file\n";

     fin.close();

     cout << "PIN = {" << pin.name << ", " << pin.population << ", " << pin.g << "}\n";
     return 0;
}
```

Copyright: M. Gross, E. Lamboray, ETHZ, 2004

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Formatierte Ausgabe

- Breite eines Ausgabefeldes
  - `cout.width(12);`
  - `... << setw(12) << ...`
- Füllzeichen
  - `cout.fill('*');`
  - `... << setfill('*') << ...`
- Floatgenauigkeit
  - `cout.precision(3);`
  - `... << setprecision(3) << ...`
- Fixe Anzahl Stellen
  - `cout.setf(ios::fixed, ios::floatfield)`
- Benutzen von **`set***`** erfordert
  - `#include <iomanip>`
- Formatierung funktioniert auch für Files
- Mehr Informationen im Prata, Kapitel 17