Eine Übersicht von Unix-Kommandos: `http://www.maths.manchester.ac.uk/~pjohnson/resources/unixShort/examples-commands.pdf`

Aufgabe C++

Dieses Übungsblatt dient dazu Sie mit den grundlegenden Schritten für die Aufgaben fit zu machen und Sie in die Umgebung der Rechner auf E203 einzuführen.

Die Beschreibung dieser Aufgaben setzt eine Unix-ähnliche Umgebung voraus. Auf den Rechner in E203 ist MinGW (Minimalist GNU for Windows) installiert. Als Compiler ist g++ der GNU Compiler Collection (GCC) zu verwenden.

Raytracer übersetzen und ausführen

Laden Sie sich die Quelltexte für die Aufgaben aus Ilias herunter (raytracer.zip). Merken Sie sich diesen Dateipfad. Starten Sie eine Kommandoshell. Erzeugen Sie in Ihrem Home-Verzeichnis ein Ordner src und entpacken Sie die Dateien dorthin.

```
mkdir src
```

Navigieren Sie in dieses Verzeichnis:

```
cd src
```

Falls das aktuelle Verzeichnis nicht Ihre Home-Verzeichnis sein sollte, können Sie mit

```
cd ~
```

in Ihr Home-Verzeichnis navigieren.

Entpacken Sie die Dateien dorthin:

```
unzip pfad_zu_zip_file/raytacer.zip
```

`pfad_zu_zip_file` ist der absolute Pfad zum Ordner der von Ihnen am Anfang heruntergeladenen Datei. Man kann diesen Pfad auch mit Copy/Paste oder den kompletten Dateipfad per Drag and Drop den in die Kommandoshell einfügen. Falls das Programm unzip nicht vorhanden ist, können Sie die ZIP-Datei auch vom Explorer aus entpacken.

Übersetzen Sie das Programm mit

```
g++ -Wall -pedantic -O3 raytracer.cc statistics.cc
```

Führen Sie das Programm aus (unter Windows `./a.exe`):

```
./a.out --no_ppm
```

Das Programm ließt die Objektbeschreibung aus der Datei `examples/teapot.obj` ein (Wavefront OBJ Format) und schreibt das Bild in einem Textformat (Portable Pixmap, PPM) auf die Standard-Ausgabe. Es können nur Dreiecke im OBJ-Format eingelesen werden, keine Materialien.

Unter vielen Linux-Distributionen kann mit `display` eine Bilddatei angezeigt werden. Mit einer Pipe `|`— kann die Standardausgabe des Raytracer zur Standardeingabe für ein nachfolgendes Programm umgeleitet werden.

```
./a.out | display
```

Die Standard-Ausgabe eines Programms kann mit `>` in eine Datei geschrieben werden. Führen Sie das Programm erneut aus:

```
./a.out > teapot.ppm
```

Da unter Windows standardmäßig keine Anzeige für PPM-Dateien existiert, erzeugt das Programm eine Datei `output.bmp` im Windows BMP-Format. Dieses kann mit dem Standard-Anzeige-Programm unter Windows angeschaut werden, indem Sie im Explorer die Datei öffnen. Mit der Option `--no_ppm` wird kein PPM erzeugt und ausgegeben. Mit `--bmp BMP` kann der Name der BMP-Ausgabedatei auf BMP gesetzt werden.

Das Programm kennt einige Kommandozeilen-Argumente.

```
./a.out --help
```

Erzeugen Sie ein Bild mit größerer Auflösung:

```
./a.out --width 512 --height 512
```

Da keine Beschleunigungsdatenstrukturen verwendet werden, müsste die Ausführungszeit sich etwa vervierfachen.

G++ Compiler

Diese Aufgabe soll sie mit den wesentlichen Compiler-Optionen von g++ vertraut machen.

Erstellen Sie eine Datei `example.cc` mit folgendem Quelltext:

```
#include <cmath>

int main(void) {
  float root;

  root = sqrt(2.0);

  return 1;
}
```

Übersetzen Sie die Datei mit

```
g++ example.cc
```

Es sollten keine Fehler oder Warnungen angezeigt werden. Da der Quelltext die main-Methode enthält, ist eine ausführbare Datei erzeugt worden. Sie heißt `a.out`. Sie kann direkt aus der Kommandoshell ausgeführt werden. Da im Suchpfad für die ausführbare Dateien das aktuelle Verzeichnis `.` oft nicht standardmäßig gesetzt ist, sollte es mit dem relativen Pfad `./a.out` ausgeführt werden. Das Programm sollte nicht abstürzen. Es gibt nichts auf die Konsole aus.

Der C und C++-Standard kennt keine Error oder Warnings, sondern nur Diagnosics. Warning sind meist Hinweise auf Fehler. Deswegen sollten immer alle Warnung beim Übersetzen angeschaltet werden:

```
g++ -Wall -pedantic example.cc
```

Diesmal sollte ein Warnung über die Variable `a` erscheinen. Die ausführbare Datei wird trotzdem erzeugt.

Statt direkt ein ausführbares Programm zu erzeugen, können mit der Option `-c` zuerst Objekt-Dateien erzeugt, die dann anschließend mit dem Linker zu einem ausführbaren Programm zusammengebunden (to link) werden. Dies ist bei sehr großen modularisierten Programmen sinnvoll, um nur die Programmteile neu zu übersetzen, die sich geändert haben. Die Objektdateien sind plattformabhängig.

```
g++ -Wall -pedantic -c example.cc
```

Es wird eine Objektdatei `example.o` erzeugt. Werden bei Aufruf des Compilers nur Objektdateien übergeben, dann wird nur der Linker ausgeführt.

```
g++ example.o
```

Der Namen der ausführbaren Datei lässt sich mit der Option `-o datei` angeben.

```
g++ example.o -o example
```

Statt einer ausführbaren Datei kann mit der Option `-S` —auch nur der Assemblercode für die Zielplattform (hier x86-Architektur) erzeugt werden.

```
g++ -Wall -pedantic -S example.cc
```

Der Assemblercode wird in Dateien mit der Endung `.s` geschrieben. Bei der Verbesserung von Programmen für eine Zielplattform, wird es immer wieder notwendig sein diesen Assemblercode zu inspizieren. Öffnen Sie `example.s` und schauen sie hinein. An welcher Stelle wird die sqrt-Funktion berechnet?

Standardmäßig sind Debug-Informationen bei der GCC ausgeschaltet. Sie werden mit Option `-g` angeschaltet. Die Programme können dann mit einem Quelltextdebugger ausgeführt werden.

```
g++ -Wall -pedantic -S -g example.cc
```

Schauen Sie wieder in die Assemblerdatei hinein. Sie müsste größer geworden sein. Am Anfang sollte sie fast gleich sein, am Ende sind zusätzliche Informationen.

Mit dem GCC selbst kann kein mit dem Quelltet annotierter Assembler erzeugt werden. Mit dem Programm `objdump` können Objektdateien dissassembliert werden. Falls die Objektdateien Debug-Informationen enthalten, dann kann mit der Option `-S` (für Source) ein annotierter Assembler erstellt werden. Der Text wird auf die Standard-Ausgabe geschrieben und sollte in eine Datei umgeleitet werden:

```
g++ -Wall -pedantic -c -g example.cc
objdump -S example.o > example.s
```

Durch die Umleitung wird `example.s` überschrieben. Schauen Sie wieder hinein. Es sollte diesmal einfacher sein, herauszufinden wo die Quadratwurzel berechnet wird.

Bei der Verbesserung von Programmen wird man in jedem Fall alle Compiler-Optimierungen anstellen. Falls für die Ausführungszeit optimiert werden soll, ist `-O3` die höchste Optimierungsebene.

```
g++ -Wall -pedantic -c -g -O3 example.cc
objdump -S example.o > example.s
```

Schauen Sie sich wieder den Assemblercode an. Er scheint sehr kurz geworden zu sein. Was macht das Programm überhaupt noch? Was ist geschehen?

Ändern Sie im Quelltext `return 1` zu `return root` ab, übersetzen Sie das Programm erneut wie zuvor, erzeugen den Assemblercode und inspizieren Sie ihn. Die Warnung ist endlich weg. Was ist jetzt geschehen?

Mit der Option `-D symbol=wert` können Präprozessor-Symbole definiert (ohne `=wert`) oder mit einem bestimmten Werte belegt werden. Dies entspricht im Quelltext einem

```
#define symbol wert
```

Ändern Sie das Literal 2.0 im Quelltext zu ZAHL ab.

```
g++ -Wall -pedantic -c -g -O3 -D ZAHL=4.0 example.cc
objdump -S example.o > example.s
```

Im Assembler sollte der Rückgabewert jetzt 2 sein.

Ändern Sie das Programm wie folgt ab:

```
#include <cmath>
#include <iostream>

int main(void) {
  float root;
  float a;
  std::cin >> a;
  root = sqrt( a );

  return root;
}
```

Inspizieren Sie den für dieses Programm erzeugten Assemlbercode erneut.

Wie dieses Beispiel zeigt muss beim Optimieren immer darauf geachtet werden, dass der Compiler den zu verbessernden Abschnitt nicht komplett wegoptimiert.

Die x86-Prozessor-Architektur unterstützt noch zusätzliche SIMD-Instruktionen für ganzen Zahlen (MMX) und Gleitkommazahlen (SSE, float, SSE double) mit vier (MMX, SSE) bzw. acht (AVX, AVX2) parallel ausgeführter Operationen. Dieser Standard kann auch für einzelne Werte verwendet werden. Er unterstützt auch eine Quadratwurzelberechnung. Die Befehle nutzen zusätzliche MMX-Register (xmm). Zum Anschalten muss die Zielarchitektur SSE unterstützen: Mit `-march=native` versucht der Compiler die Zielarchitektur des Systems herauszufinden, auf dem er ausgeführt wird. Das funktioniert nicht immer! Mit `-msee` werden die SSE-Erweiterungen eingeschaltet. Sie sind allerding nur wirksam, wenn sie für die arithmetischen Berechnungen mit `-mfpmath=sse` eingeschaltet werden.

Übersetzen Sie das Programm erneut. Erzeugen sie den annotierten Assemblercode. Und suchen sie nach der Quadratwurzelberechnung.

```
g++ -Wall -pedantic -c -g -O3 -mavx2 -march=native
    -mfpmath=sse example.cc
objdump -S example.o > example.s
```

C++

Für die Aufgaben benötigen Sie grundlegende C++11-Kenntnisse. Die folgenden Fragen sollten Sie mehrheitlich ohne Nachschlagen beantworten und erklären können. Falls nicht, sollten Sie Ihre Kenntnisse mit einem Buch über C++11 auffrischen.

- Mit wie vielen Bits wird der Datentyp `int` codiert?
- Was ist der Datentyp `long double`?
- Was ist der Datentyp `size_t`? Wofür benötigt man ihn?
- Welchen Wertebereich hat der Datentyp `bool`?
- Wie sollte man einen Null-Pointer in C++11 angeben?
- Wie wird eine Referenz auf eine Variable deklariert? Wie wird diese verwendet?
- Was ist der Unterschied zwischen einem C-String und einen C++ std::string?
- Was ist der Unterschied zwischen einem struct und class in C++?
- Was versteht man unter dem Überladen von Operatoren? Können neue Operatoren definiert, die Klammerungsregeln oder Operatorenbindung geändert werden?
- Der Cast-Operator aus C sollten in C++ nicht verwendet werden. Welche Cast-Operatoren stellt C++ zur Verfügung?
- Was ist eine Template-Funktion? Wie behandelt der Compiler Templates?
- Die C-Funktionen zur Ein- und Ausgabe von Daten sollten nicht verwendet werden. Wie funktioniert die Ein- und Ausgabe in C++? Warum sollten diese verwendet werden?
- Was versteht man unter nicht definiertem Verhalten? Nennen Sie Beispiele.
- Was versteht man unter plattformabhängigem Verhalten? Nennen Sie Beispiele.
- Wie ist dieses plattformabhängige Verhalten in der GCC und unter 64-Bit-Windows definiert. Wo muss man nachschlagen, um das herauszufinden?
- Erklären Sie nicht definiertes und plattformunabhängiges Verhaltem am Beispiel des Rechtsschift-Opterators.
- Was ist der Call-stack?
- Was ist der Heap?
- Was ist der statische Speicherbereich?
- Die Funktionen zur dynamischen Speicherverwaltung aus C sollten in C++ nicht verwendet werden. Wie erzeugt man in C++ dynamisch Speicher für einzelne Werte wie ein int und gibt ihn wieder frei.

- Wie werden Felder dynamisch angelegt und wieder freigegeben. Wie Objekte?
- C++ kennt keine automatische Speicherverwaltung. Wie kann man in C++11 sicherstellen, dass dynamisch angeforderter Speicher, der nicht mehr benötigt wird, garantiert immer freigegeben wird?
- Was bedeutet Call-by-reference, call-by-value? Was sind die Unterschiede? In welchen Situationen wirken sich diese Unterschiede aus?
- Was bedeutet die Dreier-Regel?
- Was (ab C++14) bedeutet die Fünfer-Regel?

Kenntnisse über Einfach- und Mehrfachvererbung, abstrakte Klassen und virtuelle Methoden sind nicht nötig.