Technische Universität München
Institut für Informatik I-16
Dr. Tobias Lasser
Jakob Vogel

WiSe 2012/13
Übungsblatt 1
17. Oktober 2012

Algorithmen und Datenstrukturen

In den Beispielen und Übungen verwenden wir C und C++ als Programmiersprachen. Sie benötigen daher mindestens einen entsprechenden Compiler, um die Programmieraufgaben bearbeiten zu können. Wir empfehlen jedoch die Nutzung einer IDE (Integrierte Entwicklungsumgebung), da dort über diesen Übersetzungsprozess hinausgehende Hilfsmittel vorhanden sind, die insbesondere bei der Fehlersuche von großem Nutzen sein können.

Je nach Ihrer Ausstattung können Sie kostenfrei folgende IDEs nutzen:

- Für *Windows* ist *Microsoft Visual Studio* verfügbar. Sie erhalten die Version *Express 2012 for Windows Desktop* direkt von Microsoft unter `http://microsoft.com/express` (erfordert Online-Registrierung bei Microsoft). Alternativ können Sie die Vollversion über die Universität beziehen unter `https://maniac.tum.de/` (erfordert Online-Registrierung bei der TUM). Bei MANIAC erhalten Sie auch kostenfreie Lizenzen anderer Microsoft-Produkte.
- Für *Mac OS X* erhalten Sie *Xcode*, das Sie im App Store unter `https://itunes.apple.com/de/app/xcode/id497799835` beziehen können.
- Für beide Plattformen sowie *GNU/Linux* und andere gibt es *Qt Creator* unter `http://qt.digia.com/product/developer-tools/` oder *Eclipse* unter `http://eclipse.org/`.

Die Übungsaufgaben dieser Veranstaltung sind so gestellt, dass sie mit jeder Standard-konformen C/C++ Umgebung gelöst werden können. Etwa bereits vorhandene alternative Lösungen sind also ebenso einsetzbar.

Aufgabe 1 (P) Hello, World!

Sie werden im Rahmen eines separaten Praktikums im Laufe des Semesters die Programmierung in C einüben. Diese Aufgabe dient dazu, Ihnen bereits jetzt einen Überblick zu verschaffen, und Ihnen die grundliegenden Kenntnisse zu vermitteln, die Sie zur Bearbeitung der nachfolgenden Aufgaben benötigen werden. Bitte nutzen Sie auch externe Quellen für weitere Informationen!

a) Erstellen Sie eine Textdatei `helloworld.cpp` und speichern Sie darin folgende Befehle:

```
#include <iostream>

int main(int argc, char** argv)
{
    // Nachfolgend kommt eine Ausgabe!
    std::cout << "Hello, World!" << std::endl;

    // Das ist eine Ausgabe mit Variablen
    std::cout << "Die Summe von " << 5 << " und " << 3
              << " ist " << 5+3 << "!" << std::endl;

    return 0;
}
```

Übersetzen Sie dieses Programm und führen Sie es aus! Können Sie den verschiedenen Befehlen eine Bedeutung zuordnen?

b) Komplexe Programme müssen strukturiert werden. Ein Mittel dazu sind Funktionen, die Funktionalität kapseln und aufgrund einer oder mehrerer Eingaben eine Ausgabe berechnen. Erweitern Sie nun den zuvor erstellten Code, indem Sie folgende Befehle **vor** der Funktion `main` einfügen:

```
int summe(int a, int b)
{
    return a+b;
}
```

Nutzen Sie diese neue Funktion nun, indem Sie die `main` abändern:

```
int ersteZahl=5,
    zweiteZahl=3;
std::cout << "Die Summe von " << ersteZahl << " und " << zweiteZahl
          << " ist " << summe(ersteZahl, zweiteZahl) << "!" << std::endl;
```

Was passiert jetzt bei der Ausführung? Was sind `ersteZahl` und `zweiteZahl`?

c) Natürlich sollen Programme nicht nur Daten verarbeiten können, die direkt eincodiert wurden, sondern müssen auch Eingaben zur Laufzeit verarbeiten können. Dementsprechend sollen die beiden Summanden nun vom Nutzer eingegeben werden. Ändern Sie also die `main` neuerlich ab:

```
int ersteZahl, zweiteZahl;
std::cout << "Bitte geben Sie zwei Summanden ein!" << std::endl;
std::cin >> ersteZahl >> zweiteZahl;
std::cout << "Die Summe von " << ersteZahl << " und " << zweiteZahl
          << " ist " << summe(ersteZahl, zweiteZahl) << "!" << std::endl;
```

Testen Sie dieses Programm! (Die Eingabe von Zahlen schließen Sie bitte mit `Enter` ab!)

d) Die letzte „Zutat" für mächtige Programme sind Befehle, mit denn der lineare Programmablauf durchbrochen werden kann, sogenannte *Control Statements*. Dazu zählen insbesondere `if`-Konstrukte, bei denen ein Befehlsblock nur dann ausgeführt wird, wenn eine logische Bedingung erfüllt ist.

Beispielsweise soll unser Programm bei einer Nutzereingabe ausgeben, ob die Zahl gerade oder ungerade ist. Dazu nutzen wir den Rest nach Teilung durch 2. Erweitern Sie nun die `main` erneut:

```
int zahl;
std::cout << "Bitte geben Sie eine Zahl ein!" << std::endl;
std::cin >> zahl;
if ((zahl%2) == 0)
{
    std::cout << "Die Zahl " << zahl << " ist gerade!" << std::endl;
}
else
{
    std::cout << "Die Zahl " << zahl << " ist ungerade!" << std::endl;
}
```

e) Neben diesen bedingten Befehlsblocks sind Schleifen die anderen wichtigen Kontrollstrukturen. Es gibt `for`-, `while`- und `do-while`-Schleifen, die jeweils einen Programmblock wiederholt ausführen bis eine gewisse Abbruchbedingung erfüllt ist.

Um also die Zahlen 1 bis 10 aufzuaddieren, könnte man folgende Konstruktion verwenden:

```
int ergebnis=0;
for (int i=1; i<=10; i=i+1)
{
    ergebnis = ergebnis+i;
}
std::cout << "Die Summe der Zahlen 1 bis 10 ist "
          << ergebnis << "." << std::endl;
```

Alternativ könnte man exakt die gleiche Funktionalität mit einer `while`-Schleife erreichen:

```
int ergebnis=0, i=1;
while (i<=10)
{
    ergebnis = ergebnis+i;
    i=i+1;
}
std::cout << "Die Summe der Zahlen 1 bis 10 ist "
          << ergebnis << "." << std::endl;
```

Ein anderes Schleifenkonstrukt wäre angebracht, wenn der Nutzer etwa zur Eingabe einer positiven Zahl gezwungen werden soll:

```
int wert;
do
{
    std::cout << "Bitte geben Sie eine positive Zahl ein!" << std::endl;
    std::cin >> wert;
}
while (wert <= 0);
std::cout << "Die Zahl " << wert << " ist positiv!" << std::endl;
```

Testen Sie alle Varianten!

Aufgabe 2 (P) Euklidischer Algorithmus

In der Vorlesung haben Sie den *Euklidischen Algorithmus* besprochen, mit dem der größte gemeinsame Teiler (ggT) zweier Zahlen berechnet werden kann. Ihnen wurde dort die Originalformulierung Euklids vorgestellt, wobei *AB* und *CD* zwei Streckenlängen bezeichnen:

> Wenn *CD* aber *AB* nicht misst, und man nimmt bei *AB*, *CD* abwechselnd immer das kleinere vom größeren weg, dann muss (schließlich) eine Zahl übrig bleiben, die die vorangehende misst.[1]

a) Implementieren Sie diesen Algorithmus direkt als C++ Programm! Lassen Sie den Nutzer die beiden positiven Zahlen eingeben, berechnen Sie den ggT und geben Sie diesen dann wieder aus. Kapseln Sie dabei bitte den eigentlichen Algorithmus in einer eigenen Funktion `int euklid_original(int ab, int cd)`!

[1] Euklid, Clemens Thaer (Hrsg.): Die Elemente; übernommen von `http://de.wikipedia.org/wiki/Euklidischer_Algorithmus` (Stand vom 16. Oktober 2012)

b) Gehen wir nun davon aus, dass $AB \gg CD$, also erheblich größer als CD ist. Der Algorithmus wird also mehrfach hintereinander CD abziehen, bis vom ursprünglichen AB nur noch ein Rest übrig bleibt, der kleiner ist als CD. Dies ist aber genau der Rest, der auch bei ganzzahliger Teilung von AB durch CD entsteht. In C/C++ errechnen Sie diesen Rest über den *Modulo-Operator* als `rest = ab % cd`. Schreiben Sie nun eine neue Funktion `int euklid_modulo(int ab, int cd)` mit der neuen Berechnung, und vergleichen Sie die Resultate der beiden Methoden! Womit müssen Sie den Gleichheitstest aus dem Original ersetzen?

c) Weitere wichtige Beobachtungen sind die folgenden Beziehungen:

$$\begin{aligned} \mathrm{ggT}(ab,cd) &= \mathrm{ggT}(cd,ab) \\ \mathrm{ggT}(ab,cd) &= \mathrm{ggT}(ab, cd \bmod ab) \\ \mathrm{ggT}(ab,cd) &= \mathrm{ggT}(ab, cd + \lambda \cdot ab) \end{aligned}$$

Hierbei bezeichnet „mod" den Rest nach Teilung, und λ einen beliebigen ganzzahligen Faktor. Aufgrund dieser Eigenschaften kann der ggT statt durch iterative Berechnung mittels Schleifen auch durch eine rekursive Implementierung erhalten werden. Bei letzterer ruft die Funktion sich selbst mit veränderten Eingabewerten wieder auf *ohne* dabei Schleifen zu nutzen.

Implementieren Sie also zu den beiden bestehenden iterativen Methoden jeweils eine rekursive Variante, und vergleichen Sie auch deren Ergebnisse mit den bereits zuvor errechneten!