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

WiSe 2012/13
Lösungsblatt 1
22. Oktober 2012

Algorithmen und Datenstrukturen

Aufgabe 1 Hello, World! (Beispiellösung)

Die Lösung steht auch als Quellcode zur Verfügung, den Sie über die Website beziehen können.

a) Der Hauptteil des Programms ist die `main`-Funktion, der sogenannte Einsprungpunkt. Wird ein Programm ausgeführt, wird die Kontrolle vom Betriebssystem anfänglich an diese Funktion abgegeben, und das Programm endet (im Allgemeinen) mit dem Verlassen der Funktion. Dementsprechend muss jedes Programm eine derartige Funktion aufweisen um ausführbar zu sein. Die Struktur ist stets wie folgt:

```
int main(int argc, char** argv)
{
    // Hier steht der eigentliche Programm-Code...
    return 0;
}
```

Über die beiden Parameter `argc` und `argv` können Informationen vom Aufrufer an das Programm übermittelt werden. Umgekehrt kann das Programm über den Rückgabe-Wert 0 Erfolg und über alle anderen Werte Misserfolg zurück signalisieren, wobei es sich hierbei lediglich um eine Konvention handelt. Wir ignorieren diese Möglichkeiten bis auf Weiteres aus Gründen der Vereinfachung und verwenden die `main` einfach „so".

In C/C++ werden Befehle normalerweise mit einem Strichpunkt bzw. Semikolon (`;`) abgeschlossen. Der kürzeste Befehl ist dementsprechend der Semikolon selbst, der für einen leeren Befehl steht und dementsprechend nichts macht.

Befehle können in einen Programmblock gruppiert werden, in dem mehrere Befehle in geschweiften Klammern (`{`, `}`) eingeschlossen werden. Die ganze Gruppe zählt dann als einzelner Befehl.

Neben Befehlen kann der Programmcode auch Kommentare enthalten, die hauptsächlich zur Dokumentation gedacht sind. Derartiger Text wird vom Compiler ignoriert, ist also nur im Kontext des Quellcodes vorhanden. In C/C++ gibt es dazu zwei Mechanismen: Der doppelte Schrägstrich (`//`) markiert den Beginn eines Kommentars, der sich bis zum Ende der aktuellen Zeile erstreckt. Der Kommentarblock beginnt auch mit einer Anfangsmarkierung (`/*`), endet aber erst an der entsprechenden Schlussmarkierung (`*/`). Bitte kommentieren Sie Ihren Programmcode stets, derartige Notizen können sich schon nach kurzer Zeit als essentiell herausstellen, sollen die Befehle nachvollzogen werden. Mit Kommentaren kann man aber natürlich auch einen Befehl temporär ausschalten.

Der letzte Teil des Programmes ist die Zeichenausgabe. Diese Funktionalität muss erst „beschafft" werden, wozu das `#include <iostream>` zu Beginn dient. Ähnlich können andere „Bibliotheken" von externen Prozeduren verfügbar gemacht werden. Die Ausgabe selbst ist nun modelliert wie eine Rohrleitung (bzw. als *Stream* oder Strom), wobei wir die Zeichenketten, Werte und Befehlsresultate (wie hier die Summe) einfach mittels `<<` „eingießen". Der Stream `std::cout` führt dabei auf den Bildschirm, aber man kann leicht andere derartige Streams erzeugen, die dann etwa in Dateien oder ins Netzwerk führen.

b) Ein wichtiger Teil von Programmen sind Variablen, also kleine Speicherbereiche, in denen das Programm seine Daten hält und Ergebnisse berechnet. Eine Variable in C/C++ benötigt einen Typen (bislang normalerweise `int` von *Integer*, also Ganzzahl), einen eindeutigen Namen (hier etwa `ersteZahl` und `zweiteZahl`) und einen Wert zur Initialisierung, hier 3 und 5. Sie werden weitere Datentypen in der Vorlesung und in der Übung kennenlernen.

Variablen können dann verarbeitet werden, wobei etwa für Zahlentypen mathematische Operationen wie Addition (`+`), Subtraktion (`-`), Multiplikation (`*`) und Division (`/`) bereitstehen, wobei natürlich auch geklammert werden kann (`(`, `)`). Als Operanden können dabei andere Variablen oder feste Werte verwendet werden.

Oft verwenden wir auch abkürzende Schreibweisen. So kann man – die Existenz einer `int`-Variable `wert` annehmend – statt `wert = wert + 5` auch `wert += 5` schreiben. Ähnliche Abkürzungen existieren auch für die anderen Operatoren. Die Spezialfälle `wert += 1` und `wert -= 1` lassen sich sogar noch weiter abkürzen zu `wert++` bzw. `++wert` und `wert--` bzw. `--wert`.

Ähnlich wie Variablen sind Funktionen definiert. Auch sie haben einen eindeutigen Namen und einen Rückgabewert. Im Unterschied zur Variable liegt dieser Wert allerdings nicht im Speicher, sondern wird aufgrund der Eingabewerte (und vielleicht des Programmzustandes) errechnet, wobei die entsprechenden Befehle im „Funktionskörper" aufgelistet sind. Innerhalb dessen sind die Eingabeparameter als Variablen verfügbar, und der Rückgabewert wird mittels `return` übermittelt und kann dann – muss aber nicht – in der aufrufenden Funktion weiterverarbeitet werden, also etwa in einer Berechnung verwendet, in einer Variable gespeichert oder auf einen Ausgabestrom geschickt werden.

Es ist auch möglich, dass Funktionen keine Rückgabe liefern, da sie etwa in eine Datei schreiben. Dafür ist der Spezialtyp `void` vorgesehen, sozusagen ein „Nicht-Typ". In diesem Fall kann mittels parameterlosem `return;` die Funktion verlassen werden.

Funktionen müssen in C/C++ im Quellcode vor einer Verwendung aufgeschrieben sein, daher wurde die Funktion vor die `main` gestellt. Wenn dies nicht möglich ist, genügt es aber, die Signatur (hier `int summe(int, int);`) vor der Verwendung aufzuschreiben, und die eigentliche Funktion dann später zu notieren.

c) Der Stream `std::cin` ist der Gegenstrom zu `std::cout` und führt vom Bildschirm ins Programm. Dementsprechend können mittels `>>` Daten eingelesen werden. Auch hier können alternative Eingabeströme erzeugt werden, über die dann aus Dateien oder dem Netzwerk gelesen werden kann.

Bitte bedenken Sie, dass wir stets von wohlwollenden Nutzern ausgehen und insofern normalerweise die Eingaben nicht prüfen. In einem echten Programm müsste dies **unbedingt** geschehen, wobei man etwa die Funktion `std::cin.fail()` verwenden könnte.

d) Die Grundstruktur eines `if`-Statements ist stets wie folgt:

```
if (bedingung)
    befehl;
```

Da natürlich auch hier ein Befehlsblock den einzelnen Befehl ersetzen kann, sieht man gewöhnlich eher diese Version:

```
if (bedingung)
{
    befehl1;
    befehl2;
}
```

An ein `if` kann optional mittels `else` ein alternativer Befehl oder Befehlsblock angefügt werden, der in dem Fall ausgeführt wird, dass die Bedingung nicht erfüllt ist, also:

```
if (bedingung)
    befehl1;
else
    befehl2;
```

Es können sogar ganze Ketten aus derartigen Strukturen zusammengebaut werden, um mehr als nur simple „entweder-oder" Probleme zu verarbeiten:

```
if (bedingung1)
    befehl1;
else if (bedingung2)
    befehl2;
else
    befehl3;
```

Wir empfehlen Ihnen, anfänglich auch einzelne Befehle **immer** mit geschweiften Klammern zu Blöcken zusammenzufassen, um mögliche Probleme (Stichwort „Dangling Else") zu umgehen.

Die Bedingung im `if`-Statement wird als mit „Ja"/„Nein" (bzw. `true`/`false`) zu beantwortende Frage formuliert. Dies geht etwa mit Vergleichen auf Gleichheit (`==`, nicht zu verwechseln mit der Zuweisung von Werten mittels einfachem `=`), Ungleichheit (`!=`) und Sortierung (`>`, `<`, `>=`, `<=`). Eine Bedingung kann auch negiert (`!`) werden, oder sich aus anderen Bedingungen zusammensetzen, die dann etwa beide gelten müssen („und" bzw. `&&`), oder von denen mindestens eine gelten muss („oder" bzw. `||`). In der Vorlesung und nachfolgenden Übungen wird dies noch eingehender behandelt werden.

Der Vollständigkeit halber sei noch das `switch`-Statement erwähnt, das für Fallunterscheidungen ebenfalls Anwendung finden kann. Wir werden diese Struktur bis auf Weiteres nicht verwenden, da es im Umgang weniger elegant ist.

e) Schleifen führen den Befehl bzw. Befehlsblock des *Schleifenkörpers* solange aus, bis die *Schleifenbedingung* nicht mehr erfüllt ist. Die beiden grundliegenden Schleifenkonstrukte sehen damit aus wie folgt:

```
while (bedingung)
    befehl;

do
    befehl;
while (bedingung);
```

Der Unterschied ist also der, dass bei `do-while` der Schleifenkörper mindestens einmal durchlaufen wird, da die Schleifenbedingung erst danach getestet wird.

Eine etwas erweiterte Schleife, die primär zum Zählen bzw. Iterieren gedacht ist, ist die `for`-Schleife, wie sie in der Aufgabe Anwendung fand. Dazu muss zunächst eine Zähl-Variable initialisiert werden, wobei dies durch einen oder mehrere Befehle[1] erfolgen kann. Anschließend wird dieser Zähler dann in einer Bedingung getestet, normalerweise etwa gegen eine Schranke. Nach Durchlaufen des Schleifenkörpers wird diese Variable dann in einem aus einem oder mehreren Befehlen bestehenden Aktualisierungsschritt neu gesetzt, etwa hochgezählt. Die Grundstruktur ist dementsprechend:

```
for (initialisierung; bedingung; aktualisierung)
    befehl;
```

[1] **Nur** in der `for`-Schleife werden mehrere Befehle in Initialisierung oder Aktualisierung durch Kommata getrennt, da ansonsten der Compiler die Semikolons von den in der Syntax der Schleife vorgesehenen nicht mehr unterscheiden könnte.

Diese Schleifenform ist also lediglich ein „Schönschreib-Konstrukt" für bessere Lesbarkeit. Folglich kann der Code einer `for`-Schleife gleichbedeutend ersetzt werden durch folgende Konstrukte:

```
for (initialisierung; bedingung;)
{
    befehl;
    aktualisierung;
}

initialisierung;
for (; bedingung; aktualisierung)
    befehl;

initialisierung;
for (; bedingung;)
{
    befehl;
    aktualisierung;
}

initialisierung;
while (bedingung)
{
    befehl;
    aktualisierung;
}
```

Umgekehrt kann natürlich auch die `while`-Schleife via `for` geschrieben werden, und folgende Statements sind äquivalent:

```
while (bedingung)
    befehl;

for (; bedingung;)
    befehl;
```

Bitte bedenken Sie, dass diese Auflistung gleichbedeutender Schreibweisen zur Illustration der Funktionsweise gedacht ist. Bitte „missbrauchen" Sie die Konstrukte nicht, um Ihren Code (auch für Sie) les- und wartbar zu halten. In diesem Sinne sollten Sie auch hier kompliziertere Einzelbefehle als Befehlsblock schreiben, und sie also mit geschweiften Klammern umfassen.

Alle Schleifenkörper können mittels `break` abgebrochen werden, wodurch das Programm dann mit den auf die Schleife folgenden Befehlen fortfährt. Auf ähnliche Weise kann die Ausführung des Schleifenkörpers mittels `continue` für die aktuelle Iteration abgebrochen und nach Prüfung der Schleifenbedingung mit der nächsten Wiederholung fortgefahren werden.

Aufgabe 2 Euklidischer Algorithmus (Beispiellösung)

Die Lösung steht als Quellcode zur Verfügung, den Sie über die Website beziehen können. Beachten Sie dort bitte auch die Erklärungen auf den Folien.