technische universität dortmund

Fakultät für Informatik
Lehrstühle 2, 6, 8, 14

Prof. Dr. Christian Sohler, Alexander Munteanu, Benjamin Schowe
Martin Apel, Martin Sugioarto, Tristan Skudlik
http://www-ai.cs.tu-dortmund.de/DAP2P-SoSe2011/index.html

Sommersemester 2011

DAP2 Praktikum – Blatt 1

Ausgabe: 8. April — Abgabe: 11.–15. April

Studienleistung (Scheinkriterien)

- Zum bestehen des Praktikums muss jeder Teilnehmer die folgenden Leistungen erbringen
  - Es müssen mindestens 50 Prozent der Punkte in den Kurzaufgaben erreicht werden.
  - Es müssen mindestens 50 Prozent der Punkte in den Langaufgaben erreicht werden.
  - Man muss an mindestens 12 Terminen Anwesend sein, und aktiv mitarbeiten (Nach Absprache mit den Tutoren ist es auch möglich, Fehltermine in anderen Gruppen nachzuholen, sofern dort Platz ist).
- An Feiertagen suchen Sie sich bitte einen Ersatztermin, ansonsten wird dies als "nicht erschienen"gewertet.

Kurzaufgabe 1.1 (1 Punkt)

Lernziel: Verwendung von Kommandozeilenparametern

Schreiben Sie ein Programm, das mit zwei positiven, ganzzahligen Eingabeparametern a und b aufgerufen wird und den größten gemeinsamen Teiler von a und b ausgibt. Wird das Programm nicht korrekt aufgerufen, so soll es eine Fehlermeldung mit einer Beschreibung des korrekten Aufrufs ausgeben.

Benutzen Sie dazu Euklids Algorithmus, der sehr einfach rekursiv wie folgt formuliert werden kann:

```
function EUCLID(a, b)
    if b = 0 then
        return a
    else
        return EUCLID(b, a mod b)
    end if
end function
```

soll der ggT von 264 und 846 berechnet werden, so würde der Aufruf beispielsweise so aussehen:

```
java Euclid 264 846
```

Kurzaufgabe 1.2 **(1 Punkt)**

Lernziel: Verwendung von Arrays

Schreiben Sie ein Programm, das für einen Eingabeparameter n mit Hilfe des *Siebs des Eratosthenes* alle Primzahlen bis n berechnet. Dieser Algorithmus verwendet ein boolesches Array `isPrime`, d.h. er legt ein Array vom Typ `boolean` an, das mit den Zahlen von $2, \ldots, n$ indiziert werden kann und mit **true**, initialisiert ist.

Das bedeutet, zunächst sind alle Zahlen potentielle Primzahlen. Danach werden die Zahlen von $i = 2, \ldots, n$ durchgegangen. Falls `isPrime`$[i]$ **true** ist, dann ist i tatsächlich eine Primzahl und für alle Vielfachen $j > i$ von i wird `isPrime`$[j]$ auf **false** gesetzt.

Ihr Programm soll als Eingabe die Obergrenze n sowie einen optionalen Parameter `-o` (für *output*) bekommen, den Algorithmus ausführen und ausgeben, wieviele Primzahlen in dem Intervall gefunden wurden. Falls `-o` angegeben wurde, werden zusätzlich auch die Primzahlen ausgeben.

Ein Aufruf für die Primzahlen bis 100 mit Ausgabe würde beispielsweise so aussehen:

```
java Eratosthenes 100 -o
```

Beachten Sie die Hinweise und Tipps auf den folgenden Seiten.

Hinweise und Tipps

1.1 Arbeitsumgebung im Pool einrichten

Es empfiehlt sich, für jede Aufgabe oder jedes Aufgabenblatt (je nach Umfang) ein eigenes Arbeitsverzeichnis anzulegen, z. B. mit:

```
mkdir -p dap2praktikum/aufgaben/blatt01
```

Mit ...

```
cd dap2praktikum/aufgaben/blatt01
```

können Sie in dieses Verzeichnis wechseln.

Bei einem neuen Poolaccount ist als Shell `csh` eingestellt. Komfortabler sind neuere Shells wie `tcsh` oder `bash` (Standard auf Linux-Systemen). In der laufenden Terminalsitzung können Sie vorübergehend durch Eingabe von `tcsh` bzw. `bash` die Shell wechseln. Dauerhaft können Sie die voreingestellte Shell auf einem Solaris-Rechner mit diesem Befehl wechseln:

```
passwd -r nis -e
```

Auf einem Linux-Rechner lautet der entsprechende Befehl:

```
chsh
```

Als neue Shell geben Sie dann `/bin/tcsh` oder `/bin/bash` an. (Ob auf einem Rechner Linux oder Solaris läuft, können Sie z.B. mit dem Befehl `uname -a` ermitteln.)

Falls sich das System bei Ihnen anders verhält als erwartet, könnte es daran liegen, dass Ihre Initialisierungsdateien nicht mehr im Originalzustand sind. In diesem Fall (und nur in diesem!) stellen Sie auf einem Solaris-Rechner die Standard-Initialisierung mit dieser Befehlsfolge wieder her:

```
cd $HOME
/bin/cp /opt/local/etc/skel/.??* .
```

Auf einem Linux-Rechner lautet die entsprechende Befehlsfolge:

```
cd $HOME
/bin/cp /etc/skel/.??* .
```

1.2 Kommentare im Quelltext

Auf diesem und auf den folgenden Aufgabenblättern wird (pro Aufgabe) ein halber Punkt abgezogen, wenn der Quelltext nicht sinnvoll kommentiert ist. Kommentare sollen das Programm erläutern und nicht den Quelltext beschreiben.

```
// Beispiel eines NICHT SINNVOLLEN Kommentars:
// Hier wird geprueft, ob a gleich 0 ist
if (a!=0) {
...
```

Besser ist es, zu erklären, warum `a` mit Null verglichen werden muss:

```
// Beispiel eines SINNVOLLEN Kommentars:
// a darf nicht 0 sein, da sonst spaeter durch 0 dividiert wuerde
if (a!=0){
...
```

Richten Sie sich nach folgender **Faustregel**: Stellen Sie sich vor, Sie möchten eine(r/m) anderen Programmierer(in) (mit ähnlichen Fähigkeiten) Ihr Programm erklären. Insbesondere soll sie/er das Programm möglichst schnell verstehen und davon überzeugt werden, dass es **korrekt** ist. Schreiben alle dazu nötigen Hinweise als Kommentare in Ihren Quelltext. Insbesondere **müssen** Schleifeninvarianten als Kommentar angegeben werden.

Wenn Ihr Tutor nicht in der Lage ist, Ihren Quelltext ohne große Mühe zu verstehen, ist diese Regel verletzt.

(Selbstverständlich, dennoch sei hier darauf hingewiesen: Für die Verständlichkeit ist richtiges Einrücken eine elementare Voraussetzung!)

1.3 Java in der Kommandozeile

Befindet man sich im selben Verzeichnis wie die zu kompilierende/auszuführende Klasse, so kann man mit dem Befehl:

```
javac Klasse.java
```

die Datei `Klasse.java` in Java-Bytecode (`Klasse.class` übersetzen.

Mit dem Befehl:

```
java Klasse
```

wird der Bytecode vom Interpreter ausgeführt. Wichtig ist es hierbei, dass die Endung `.class` beim Aufruf weggelassen wird.

1.4 Kommandozeilenparameter mit Java (im Terminal) Man kann Javaprogramme auch mit Parametern aufrufen, diese werden mit Leerzeichen getrennt und hinter den Aufruf des Interpreters (selbe Zeile) geschrieben. Beispiel:

```
java Klasse param1 param2 param3
```

Diese Parameter werden der `main`-Methode übergeben und befinden sich als Strings im Array `args`. Dies funktioniert aber auch nur, wenn die `main`-Mehtode die folgende Signatur hat:

```
public static void main(String[] args) { ... }
```