technische universität dortmund

fakultät für informatik

Andre Droschinsky
Roman Kalkreuth
Denis Kurz

Dortmund, den 23. November 2018

Praktikum zur Vorlesung Einführung in die Programmierung WS 18/19

Blatt 5

Es können 18 (+2 Bonus) Punkte erreicht werden.

Allgemeine Hinweise

1. Bitte lesen Sie vor der Bearbeitung **alle** Aufgaben sorgfältig durch! Dies erspart Ihnen unnötige Arbeit und somit auch Zeit!
2. Die einzigen Header, die Sie zur Bearbeitung der Aufgaben verwenden dürfen, sind `iostream` und solche, die laut Aufgabenstellung explizit erlaubt werden.
3. Lassen Sie sich fertiggestellte Aufgaben bitte möglichst **frühzeitig** testieren. In der letzten halben Stunde vor Schluss wird nur noch **eine** Aufgabe testiert!
4. Wir akzeptieren ein Testat nur, wenn die Lösung eigenständig auf Anhieb erklärt werden kann. Andernfalls müssen wir die entsprechende Teilaufgabe mit 0 Punkten bewerten.

HINWEIS: **Am 29.11.2018, findet von 14:15 - 15:45h im HG II, HS 3 eine Probeklausur statt!**

Grundlage: Debugging von Funktionen

Auf dem Praktikumsblatt 4 wurden einige Grundlagen des Debugging erläutert. Der folgende Teil erklärt, wie das Debugging beim Verwenden von Funktionen funktioniert.
Befindet sich in der aktuellen Zeile des mit dem Debugger geprüften Programms ein Funktionsaufruf, hat man zwei Möglichkeiten:

1. **Die komplette Funktion in einem Schritt ausführen:** Dieses Überspringen des Programmcodes der Funktion funktioniert wie gewohnt über den Menüpunkt `Run` → `Step Over` oder dem Tastenkürzel `F6`. Das Programm hält anschließend in der Zeile nach dem Funktionsaufruf.
2. **In die Funktion herabsteigen:** Möchte man den Rumpf der Funktion betreten, so wählt man den Menüpunkt `Run` → `Step Into` oder das Tastenkürzel `F5`. Das Programm hält anschließend in der ersten Zeile der Funktion und kann wie gewohnt fortgeführt werden, z. B. mit `F6`. Wird das Ende der Funktion erreicht, wird automatisch zur aufrufenden Stelle zurückgekehrt. Möchte man die Funktion vorzeitig verlassen und direkt zur aufrufenden Stelle zurückkehren, wählt man im Menü den Punkt `Run` → `Step Return` oder das Tastenkürzel `F7`.

Grundlage: Tipps zur Fehlersuche

Bisher wurden einige verschiedene Konzepte zum Aufspüren von Fehlern vorgestellt. Hier werden die bisher (nicht notwendigerweise von allen) im Praktikum verwendeten Methoden wiederholt und zusammengefasst. Die folgende Liste sollte also als Referenz zur Fehlersuche im Praktikum verwendet werden.

Der erste Schritt muss immer das Beseitigen von Syntaxfehlern sein, wenn solche existieren:

Umgebung aufräumen: Schließen Sie noch laufende Instanzen Ihrer Programme, bevor Sie das geänderte Programm übersetzen oder starten. Falls nicht alle Instanzen beendet sind, schlägt die Kompilierung fehl! Instanzen können unter anderem dann nicht automatisch beendet werden, wenn diese noch auf Eingaben warten oder durch den Debugger angehalten wurden. Eine Übersicht über die laufenden Instanzen gibt die `Debug-View` in der `Debug-Perspektive`. Über einen Rechtsklick auf eine solche Instanz erscheint das Kontextmenü. Über den Menüpunkt `Terminate and Remove` kann die aktuell gewählte Instanz beendet und aus der Übersicht entfernt werden. Möchte man alle Instanzen auf einmal beenden, so kann man den Menüpunkt `Terminate/Disconnect All` wählen.

Separat kompilieren: Zuerst sollten die aktuellen Änderungen des Projekts gespeichert werden. Anschließend wird Eclipse über das Menü (`Project` → `Build Project`) angewiesen, das aktuelle Projekt zu kompilieren. Es ist ratsam, das Programm vor der Ausführung separat zu kompilieren, um die Ausgabe des Compilers in Bezug auf Fehlermeldungen und Warnungen untersuchen zu können (nach der Ausführung werden diese von den Programmausgaben überschrieben). Die Ausgabe des Compilers ist in der `Console-View` zu sehen. Zeilen, in denen ein Fehler ausgegeben wird, werden rot und Zeilen, in denen eine Warnung ausgegeben wird, gelb dargestellt. Nur wenn das Programm fehlerfrei übersetzt wurde, sollte es gestartet werden. Im anderen Fall wird die letzte fehlerfrei übersetzte Version (falls vorhanden) gestartet, welche nicht dem aktuellen Programmtext entspricht! Auch Warnungen sollten nicht einfach missachtet werden. Sie deuten mindestens auf einen schlechten Programmierstil und häufig auf Fehler hin, welche anschließend zur Laufzeit auftreten.

Die Fehlerliste: Schlägt die Kompilierung fehl, enthält das Programm höchstwahrscheinlich mindestens einen Syntaxfehler. Statt sich durch die komplexe Ausgabe des Compilers zu arbeiten, können Sie sich eine von Eclipse aufbereitete Liste von Fehlern in der `Problems-View` anzeigen lassen. Durch Doppelklick auf die einzelnen Einträge der Liste, die sich am unteren Bildschirmrand öffnet, gelangen Sie direkt in die Zeile, in der ein Fehler festgestellt wurde. Die Fehlerliste sollte dann Punkt für Punkt abgearbeitet werden, bis kein Fehler mehr beim Kompilieren auftritt. Starten Sie immer mit dem ersten Fehler und kompilieren Sie das Programm nach dessen Behebung neu. Häufig sind Fehler nur Folgefehler, welche bei der Behebung des Hauptfehlers mit beseitigt werden. Achten Sie unbedingt darauf, dass Sie nur das aktuell zu bearbeitende Projekt geöffnet haben. Ansonsten werden in der `Problems-View` auch die Fehler anderer Projekte angezeigt und die Anzeige wird unübersichtlich. Ein Projekt kann geschlossen (geöffnet) werden, indem mit einem Rechtsklick das Kontextmenü des Projekts aufgerufen wird und anschließend der Menüpunkt `Close Project` (`Open Project`) gewählt wird.

Wenn das Programm anstandslos übersetzt wird, kann es trotzdem passieren, dass es sich nicht so verhält, wie es gedacht war. In solchen Fällen sollten folgende Schritte zur Fehlersuche unternommen werden:

Programmlogik überdenken Zuerst sollten Sie sich kurz fragen, ob das Programm so, wie Sie es aufgeschrieben haben, überhaupt sinnvoll ist. Ist beim Überprüfen einiger Bedingungen vielleicht eine bestimmte Reihenfolge einzuhalten? Wird der Wert einer Schleifenabbruch-Variablen am Ende einer Schleife überschrieben, bevor die Abbruchbedingung überprüft werden konnte? Die Reihenfolge von Anweisungen kann eine ganz entscheidende Rolle spielen und sollte überprüft werden.

Code formatieren: Fehler lassen sich meist viel schneller finden, wenn man die Struktur des Programms an dessen Formatierung ablesen kann. Eclipse bietet daher die Möglichkeit, den kompletten Code durch das Drücken der Tastenkombination `Strg+Shift+F` automatisch zu formatieren. Anschließend können Sie beispielsweise `else`-Direktiven sehr schnell erkennen, die einem falschen `if` zugeordnet wurden, weil eventuell geschweifte Klammern fehlen (o.Ä.).

Debugging: Beim Debugging handelt es sich seit jeher um eines der nützlichsten Mittel zum Aufspüren von Fehlern. Verhält sich das Programm also seltsam, hilft es oft, vor fragwürdigen Zeilen einen Breakpoint zu

setzen und genau zu beobachten (Variablenbelegung in der `Variables-View`), was bei einer schrittweisen Ausführung passiert.

Tutoren fragen: Sollte keiner der oben genannten Schritte helfen, können Sie sich natürlich immer noch an einen Betreuer wenden. **Führen Sie aber in jedem Fall vorher die oben genannten Schritte aus!** Das eigenständige Finden von Fehlern hilft enorm, um das Verständnis über die Programmiersprache zu erhöhen. Auch wenn einige Schritte zuerst mühselig erscheinen (z.B. Debuggen, separates Kompilieren), so lassen sich mit diesen Werkzeugen und Vorgehensweisen Fehler in der Regel sehr viel schneller aufspüren, wenn man mit der Handhabung der Werkzeuge erst einmal vertraut ist.

Aufgabe 1: Primzahlen (6 Punkte) + (2 Bonuspunkte)

Eine Primzahl ist eine natürliche Zahl, die größer als 1 und ausschließlich durch sich selbst und durch 1 teilbar ist. Eine Primfaktorzerlegung ist die Darstellung einer natürlichen Zahl n als Produkt aus Primzahlen, die dann als Primfaktoren von n bezeichnet werden
Erstellen Sie ein neues Projekt `Aufgabe_5_1` und darin die Quelldatei `prim.cpp` mit der `main`-Funktion.

a) Legen Sie in der `main`-Funktion ein Array `ist_prim` der Länge 1000 vom Typ `bool` an. Initialisieren die das gesamte Array mit `true`, außer an den Positionen 0 und 1.

EINSCHRÄNKUNG: Legen Sie das Array auf dem Heap an.

_____ (1)

b) Implementieren Sie das Sieb des Eratosthenes zur Bestimmung von Primzahlen. Durchlaufen Sie dazu von der kleinsten Primzahl 2 aus die ganzen Zahlen in aufsteigender Reihenfolge bis zur Zahl 1000. Wenn das Array `ist_prim` für die Zahl i den Wert `true` enthält, geben Sie die Zahl auf der Konsole aus und setzen die Werte in `ist_prim` für alle Vielfachen von i auf `false`.

_____ (2)

c) Erweitern Sie den Code aus Teil b) um eine Primfaktorzerlegung für die Zahlen, die nicht prim sind. Die Primfaktorzerlegung einer Zahl n kann durch das Verfahren der sogenannten Probedivision durchgeführt werden. Dabei wird für alle Primzahlen $2 \leq p \leq n$ geprüft, ob $n \mod p = 0$ ist. Ist dies der Fall, wurde ein Primfaktor gefunden und das Verfahren wird mit $n := n/p$ wiederholt. Wir können abbrechen, wenn n selbst prim ist. Implementieren Sie dieses Verfahren iterativ in einer Funktion mit der Signatur `void gibPrimfaktorenAus(int zahl, bool ist_prim[])`. Die Funktion soll aus der Schleife in b) aus aufgerufen werden und alle gefundenen Primfaktoren sofort auf der Konsole ausgeben, so dass die Ausgabe des Gesamtprogramms wie folgt aussieht:

```
2: prim
3: prim
4: 2, 2
5: prim
6: 2, 3
7: prim
8: 2, 2, 2
9: 3, 3
10: 2, 5
⋮
```

_____ (3)

d) Bonusaufgabe:
Der Ansatz aus c) ist relativ ineffizient, da wir das Array `ist_prim` durchlaufen müssen, das nicht nur die Informationen zu den Primzahlen, sondern auch zu allen anderen Zahlen enthält. Erweitern Sie daher den Code in der `main`-Funktion durch ein Array `primzahlen` vom Typ `int` mit derselben Länge wie `ist_prim`. Befüllen Sie dieses Array mit den gefundenen Primzahlen und zählen Sie diese. Mit diesen Informationen können Sie nun eine weitere Funktion zur Primfaktorzerlegung schreiben, mit der Signatur `void gibPrimfaktorenAus(int zahl, int anz_primzahlen, int primzahlen[])`. Diese Funktion kann nun direkt die Primzahlen durchgehen und stoppen, sobald für eine Primzahl p gilt $p^2 > n$.

_____ (2)

Aufgabe 2: ISBN-Nummern prüfen (6 Punkte)

Es sollen 10-stellige ISBN-Nummern (**I**nternational **S**tandard **B**ook **N**umber) überprüft werden. Hierbei sind die ersten 9 Ziffern z_1 bis z_9 der Reihe nach von links mit den Faktoren 1,2,3,4,... zu multiplizieren und aufzusummieren. Das Ergebnis wird dann durch 11 geteilt. Der Divisionsrest muss der 10. Ziffer (z_{10}) entsprechen. Hat der Divisionsrest den Wert 10, so wird er durch das Zeichen 'x' dargestellt, sonst durch die Ziffern 0 bis 9. Für die Berechnung der Prüfziffer ergibt sich dann folgende Formel:

$$z_{10} = \left(\sum_{i=1}^{9} i \cdot z_i \right) \mod 11$$

Für die korrekte ISBN-Nummer 349913599x ergibt sich daraus folgende Berechnung:

$$1 \cdot 3 + 2 \cdot 4 + 3 \cdot 9 + 4 \cdot 9 + 5 \cdot 1 + 6 \cdot 3 + 7 \cdot 5 + 8 \cdot 9 + 9 \cdot 9 = 285$$

$$285 \mod 11 \quad = \quad 10 \quad (\widehat{=} \texttt{'x'})$$

Erstellen Sie ein neues Projekt `Aufgabe_5_2` und darin die Quelldatei `isbn.cpp` mit der `main`-Funktion.

a) Die ISBN-Nummern sollen durch ein Array vom Datentyp `char` dargestellt werden. Legen Sie in der `main`-Funktion zwei `char`-Arrays an und initialisieren Sie diese mit den ISBN-Nummern `349913599x` und `2871499367`.

_____ (1)

b) Da Sie für die Überprüfung der ISBN-Nummern mit Ganzzahlen rechnen, müssen die Ziffern vom Typ `char` konvertiert werden. Schreiben Sie eine Funktion `umwandlung`, welche einen Wert vom Datentyp `char` als Funktionsargument aufnimmt und diesen zum Datentyp `int` umwandelt. Der umgewandelte `int`-Wert soll dann von der Funktion zurückgegeben werden.

_____ (2)

c) Schreiben Sie eine Funktion `isbn10check`, die eine ISBN-Nummer übernimmt, aus den ersten 9 Ziffern die Prüfziffer berechnet und das Ergebnis mit der 10. Stelle vergleicht. Die Funktion soll den Datentyp `bool` zurückgeben. Wenn die ISBN-Nummer richtig ist, wird `true` zurückgegeben, sonst `false`. Beachten Sie, dass die ISBN-Nummer in einem `char`-Feld übergeben wird. Wenn eine Umwandlung von `char` nach `int` notwendig ist, soll die Funktion `umwandlung` aufgerufen werden.

EINSCHRÄNKUNG: Verwenden Sie zur Berechnung der Prüfziffer eine Schleife!

_____ (2)

d) Rufen Sie in der `main`-Funktion die Funktion `isbn10check` auf und reagieren Sie auf den Rückgabewert der Funktion. Ist die ISBN-Nummer gültig, so soll eine entsprechende Erfolgsmeldung ausgegeben werden, ansonsten eine Fehlermeldung. Überprüfen Sie ihr Programm mit den in Teil a) genannten ISBN-Nummern.

_____ (1)

Aufgabe 3: Summe zweier Quadratzahlen (6 Punkte)

a) Einige ganze Zahlen lassen sich als Summe aus zwei ebenfalls ganzzahligen Quadraten darstellen. Hierzu einige Beispiele:

$$10 = 1^2 + 3^2$$
$$13 = 2^2 + 3^2$$
$$17 = 1^2 + 4^2$$
$$18 = 3^2 + 3^2$$

Erstellen Sie ein neues Projekt `Aufgabe_5_3` und darin die Quelldatei `quadratzahlen.cpp` mit der `main`-Funktion.
Schreiben Sie eine Funktion `istSummeQuadratzahlen`, die untersucht, ob sich der im Parameter n übergebene Wert als Summe der Quadrate zweier ganzer Zahlen a und b darstellen lässt:

$$n = a^2 + b^2 \tag{1}$$

Wenn das zutrifft, gibt die Funktion den boolschen Wert `true` zurück, ansonsten `false`. Übergeben sie a und b als Zeiger und weisen sie nach der Berechnung beiden Variablen die entsprechende Werte zu.

HINWEIS: Für einen gegebenen Wert von n können der Reihe nach in 1er Schritten die Werte von $a = 1$ bis $a \leq \sqrt{n/2}$ untersucht werden. Damit lässt sich für ein so gegebenes a nach entsprechender Umstellung von Gleichung 1 der Wert von b berechnen: $b = \sqrt{n - a^2}$. Wenn b ganzzahlig ist, ist die Bedingung erfüllt.

_____ (4)

b) Schreiben Sie ein kurzes Hauptprogramm in welchem die Zahlen von 1 bis 50 mit Hilfe der Funktion `istSummeQuadratzahlen` auf diese Eigenschaft hin untersucht werden. Nur die Zahlen, die sich als Summe zweier Quadratzahlen darstellen lassen, soolen ausgegeben werdens.

_____ (2)