technische universität dortmund

fakultät für informatik

Andre Droschinsky
Roman Kalkreuth
Denis Kurz

Dortmund, den 15. November 2018

Praktikum zur Vorlesung Einführung in die Programmierung WS 18/19

Blatt 4

Es können 16 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.

Grundlage: Strukturierte Fehlersuche (Debugging)

Ein Debugger ist ein Werkzeug zum Auffinden von *Bugs* (Fehlern) in Programmen. Dabei kann das Programm schrittweise ausgeführt und nach jedem Schritt der Zustand untersucht werden. Ein Schritt bezeichnet dabei immer eine Programmzeile und der Zustand eines laufenden Programms lässt sich (stark vereinfacht) durch die Variablenbelegung beschreiben.
Beispiel:

```
1       :
2   int x = 5;
3   cout << x << endl; // Zustand: x hat den Wert 5, ...
4   x = x - 1;
5   cout << x << endl; // Zustand: x hat den Wert 4, ...
6       :
```

Der folgende Abschnitt beschreibt, wie das Debuggen mit Eclipse funktioniert: Da zum Debugging zusätzliche Debug-Symbole in dem Programm gespeichert werden müssen und weniger Optimierungen beim Übersetzen verwendet werden können, erzeugt Eclipse (und alle anderen gängigen Entwicklungsumgebungen) für ein neues Projekt zwei Konfigurationen für den Compiler (Übersetzer). Mit der Konfiguration `Release` wird das Programm so übersetzt, dass es möglichst schnell ausgeführt wird. Mit der Konfiguration `Debug` übersetzt der Compiler das Programm so, dass mit Hilfe des Debuggers nach Fehlern gesucht werden kann. Zwischen den zwei Konfigurationen kann über den Menüpunkt `Project` → `Build Configurations` → `Set Active` umgeschaltet werden, wie in Abbildung 1 zu sehen. Achten Sie also darauf, das Debugging zum Testen immer einzuschalten und das Programm mit der Konfiguration `Debug` zu übersetzen.

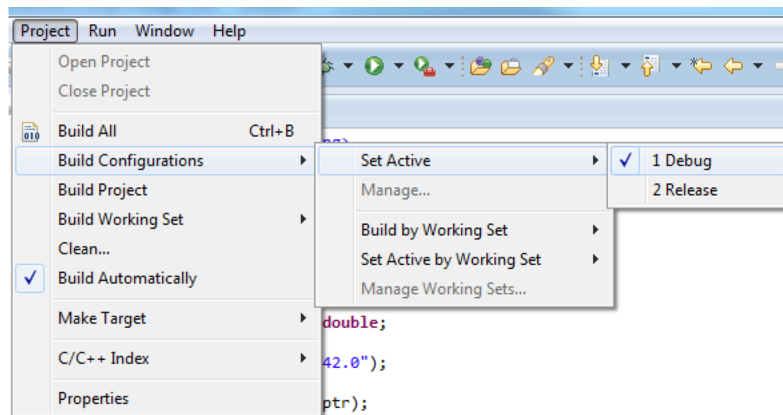

Abbildung 1: Eclipse: Umschalten zwischen Debug- und Release-Modus.

Nachdem das Debugging angeschaltet wurde, kann das Programm mit `F11` oder `Run` → `Debug` gestartet werden. Eclipse wechselt dann (ggf. nach einer Rückfrage) in die Debugging-Perspektive. Diese zeigt in einem Bereich rechts oben die Zustände der Variablen an. Soll wieder zurück zur Programmier-Perspektive (mit der Konsolenausgabe und der Fehlerliste) gewechselt werden, muss das Programm erst gestoppt werden (`Strg+F2`, `Run -> Terminate` oder der Stopp-Schaltfläche). Danach muss dann über einen separaten Klick auf das C++-Layout zurück gewechselt werden.
Wurde das Programm zum Debuggen gestartet, hält es am Beginn der `main` Funktion direkt an. Im angehaltenen Zustand kann dann in Ruhe der Zustand des Programms untersucht werden. Mit dem Befehl `Run` → `Step Over` (`F6`) kann das Programm schrittweise weiter ausgeführt werden.
Da ein Programm mitunter viele Schritte ausführt, wäre es mühselig, immer alle Schritte durchzugehen, bis die relevante Stelle im Code erreicht wird. Daher gibt es sogenannte *Haltepunkte* (engl. Breakpoints). Mit diesen markiert man einzelne Zeilen, an denen das Programm angehalten werden soll. In Eclipse kann ein Breakpoint in der aktuellen Zeile (in der sich der Cursor befindet) erstellt werden, indem `Umschalt+Strg+B` gedrückt, im Menü `Run` → `Toggle Breakpoint` ausgewählt oder ein Doppelklick auf die blaue Fläche links neben dem Zeilenanfang getätigt wird. Der Breakpoint wird anschließend über einen kleinen blauen Punkt angezeigt.

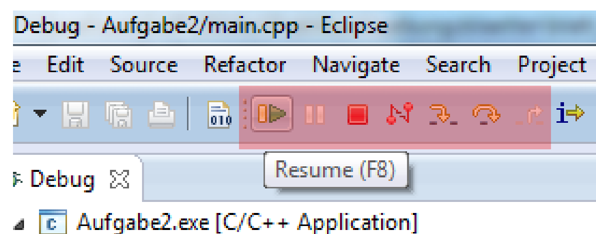

Abbildung 2: Eclipse: Debugging-Schaltflächen

Wurde das Programm schließlich an einem Haltepunkt angehalten, kann es entweder bis zum nächsten Haltepunkt ausgeführt (`F8-Taste`, `Run` → `Resume` oder der `Resume` Schaltfläche) oder wie oben beschrieben Schritt für Schritt (zeilenweise) durchgegangen werden (Eclipse: `F6` oder `Run` → `Step Over`).
In Tabelle 1 befindet sich noch einmal eine Zusammenfassung der verfügbaren Shortcuts.

Tabelle 1: Zusammenfassung der Shortcuts

Befehl	Tastenkombination
Haltepunkt setzen/entfernen	`Umschalt+Strg+B`
Debugging starten	`F11`
Debugging stoppen	`Strg+F2`
Bis zum nächsten Haltepunkt	`F8`
Eine Zeile ausführen	`F6`

HINWEIS: Wenn das Debugging eingeschaltet ist (Konfiguration `Debug` und Start mit `F11`) und ein Laufzeitfehler auftritt, hält das Programm automatisch an der fehlerhaften Stelle an und wechselt in die Debugging-Perspektive. Der Programmzustand zum Zeitpunkt des Fehlers kann direkt untersucht werden.

TIPP: Verwenden Sie den Debugger immer, wenn Sie sich das Verhalten eines Programms nicht erklären können und wenn Fehler auftreten. Es lohnt sich, mit dem Debugger vertraut zu sein. Die effiziente und effektive Fehlersuche in komplexen Programmen ist ohne Debugger kaum möglich!

TIPP: Das Zusammenspiel zwischen Eclipse und dem Debugger verläuft nicht immer reibungslos. Sollten Sie merkwürdiges Verhalten beobachten oder Probleme beim Debuggen haben, melden Sie sich bitte bei den Betreuern. Bekannte Probleme sind z. B.:

- Wenn das Programm noch im Debugger ausgeführt wird, kann der Quelltext nicht neu übersetzt werden, da das Programm nicht überschrieben werden kann. Achten Sie also unbedingt darauf, noch laufende Programminstanzen immer zu beenden (Befehl `Terminate`).
- Der verwendete Debugger ignoriert Haltepunkte in leeren Zeilen und bei einfachen Variablendeklarationen ohne gleichzeitige Initialisierung. Setzen Sie Haltepunkte deshalb immer in Zeilen, die Anweisungen enthalten.

Aufgabe 1: Benutzen des Debuggers (6 Punkte)

Erstellen Sie ein neues Projekt mit dem Namen `Aufgabe_4_1` und fügen Sie die zur Verfügung gestellte C++-Quelldatei `debug.cpp` hinzu.

a) Starten Sie das Programm im Debugging-Modus. Normalerweise ist der Debugger so eingestellt, dass das Programm direkt anhält. Tut es das nicht, setzen Sie in der Programmzeile bei der Definition der Variablen `andereZahl` der Funktion `main()` einen Haltepunkt. Gehen Sie dann das Programm schrittweise durch. Angenommen, Zeile x wird gerade durch einen Pfeil und besondere Hintergrundfärbung hervorgehoben. Welche Zeilen des Programms wurden in diesem Moment schon ausgeführt? Wie verhält sich der Debugger bei Bedingungen? Welche Zeilen werden ausgewertet?

____________ (2)

b) Setzen Sie einen Haltepunkt in der Zeile mit der Anweisung `cout << "Zahlenvergleich" << endl;`. Welchen Wert haben die Variablen `eineZahl`, `andereZahl`, `text1` und `text2`, wenn das Programm an diesem Haltepunkt anhält? Welche Werte haben die Elemente des Arrays `text2`? Sind die Werte über mehrere Läufe stabil? Wie erklären Sie das Verhalten?

____________ (2)

c) Legen Sie in dem Programm zwei Variablen vom Datentyp `char` an und weisen Sie ihnen die Werte `'q'` und `'&'` zu. Welche Informationen lassen sich mit dem Debugger über die Repräsentationen von `'q'` und `'&'` gewinnen?

____

____

__________ (1)

d) Legen Sie einen Haltepunkt in der Zeile mit der Anweisung `if (andereZahl > eineZahl)` an und lassen Sie das Programm bis zu diesem Haltepunkt laufen. Ändern Sie nun im View `Variables` den Wert von Variable `andereZahl` in 15 und lassen Sie das Programm weiterlaufen. Was passiert?

____

____

__________ (1)

Aufgabe 2: Wurzelziehen (4 Punkte)

Erstellen Sie zunächst ein neues Projekt `Aufgabe_4_2` und darin eine C++-Quelltextdatei `wurzel.cpp`.

a) Schreiben Sie eine Funktion `wurzel`, die eine Zahl a vom typ `double` als Eingabe erhält. Diese Funktion soll das Heron-Verfahren implementieren, welches die Quadratwurzel $\sqrt{a}$ iterativ annähert.
Das Heron-Verfahren ist definiert durch einen Startwert

$$x_0 = \frac{a+1}{2}$$

und einen Näherungsschritt

$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{a}{x_n}\right).$$

Die Variable $x_i \approx \sqrt{a}$ ist dabei der Näherungswert für die Quadratwurzel in Iteration i. Setzen Sie innerhalb der Funktion ein Schleifenkonstrukt ein, das abbricht, sobald $x_n = x_{n+1}$, spätestens aber nach 100 Iterationen. Der approximierte `double`-Wert für $\sqrt{a}$ soll von der Funktion zurückgegeben werden.

__________ (2)

b) Schreiben Sie eine `main`-Funktion, welche die von `wurzel` benötigte Anzahl Schleifendurchläufe, den errechneten Wert von $\sqrt{a}$, das Ergebnis der Funktion `sqrt`, sowie die Differenz `sqrt(a) - wurzel(a)`, mit 17 Stellen auf dem Bildschirm ausgibt.

Eingabe	Anzahl Schleifen-durchläufe	Ergebnis `wurzel(a)`	Ergebnis `sqrt(a)`	Differenz
2				
100				
5698				

HINWEIS:

- Inkludieren Sie zusätzlich zu `iostream` die Bibliotheken `iomanip` sowie `cmath` (enthält `sqrt`).
- Die Anweisung `cout << setprecision(n);` (mit integer-Argument n) legt die Anzahl der auf dem Bildschirm ausgegebenen Stellen (insgesamt vor und nach dem Komma) fest.

____________ (2)

Aufgabe 3: Anagramme (6 Punkte)

Erstellen Sie zunächst ein neues Projekt `Aufgabe_4_3` und fügen Sie die zur Verfügung gestellte C++-Quelldatei `anagramme.cpp` hinzu.
Ein Wort ist ein *Anagramm* eines anderen Wortes, wenn beide Wörter dieselben Buchstaben mit jeweils derselben Anzahl enthalten. Beispielsweise ist „Lager" ein Anagramm von „Regal", weil beide jeweils aus einem A, einem E, einem G, einem L und einem R bestehen. Im Gegensatz dazu sind „Banane" und „Bannen" keine Anagramme; sie unterscheiden sich in der Anzahl der A und der N.
In dieser Aufgabe soll ein Programm geschrieben werden, das feststellt, ob zwei vorgegeben Wörter Anagramme voneinander sind. Dazu wird in einem `int`-Array der Länge 26 zunächst gezählt, wie häufig jeder Buchstabe in einem der Wörter vorkommt. Anschließend wird in demselben Array für jeden im zweiten Wort vorkommenden Buchstaben wieder heruntergezählt. Es handelt sich dann um Anagramme, wenn am Ende jeder Eintrag im Array 0 ist.
Im Rahmen dieser Aufgabe soll Groß-/Kleinschreibung ignoriert werden (wie im Beispiel Lager/Regal). Außerdem dürfen Sie davon ausgehen, dass die übergebenen Wörter nur Buchstaben enthalten (von a bis z und von A bis Z, also keine Ziffern, Sonderzeichen, Umlaute oder sonstige Buchstaben). Eine gesonderte Fehlerbehandlung für Wörter mit anderen Zeichen ist nicht gefordert.

a) Schreiben Sie eine Funktion `position` mit Rückgabetyp `int`, die für einen übergebenen Buchstaben (Typ `char`) zurückgibt, an welcher Position des Alphabets er steht. Für den Buchstaben A (also `'a'` oder `'A'`) soll 0 zurückgegeben werden, für B (`'b'` oder `'B'`) soll 1 zurückgegeben werden, und so weiter.

HINWEIS: Eine einzige `if`-Anweisung ist hierfür ausreichend.

____________ (1)

b) Schreiben Sie eine Funktion `zaehleHoch` mit einem Parameter `buchstabe` vom Typ `char` und einem Parameter `anzahlen` vom Typ `int[]`. Innerhalb der Funktion soll der Eintrag des `anzahlen`-Arrays, der der Position des übergebenen Buchstaben entspricht, um eins erhöht werden. Für `buchstabe == 'a'` soll also beispielsweise das Element mit Index 0 inkrementiert werden. Die Funktion soll nichts zurückgeben.
Schreiben Sie außerdem eine Funktion `zaehleRunter` mit derselben Signatur und demselben Rückgabetyp wie `zaehleHoch`. Die Funktion soll analog zu `zaehleHoch` den entsprechenden Eintrag im Array um eins verringern.

____________ (2)

c) Schreiben Sie eine Funktion `zaehleAlleHoch`, die die obige Funktion `zaehleHoch` für jeden Buchstaben eines Wortes aufrufen soll. Dabei soll immer dasselbe `anzahlen`-Array verwendet werden, das der Funktion `zaehleAlleHoch` geeignet übergeben wird. Auch das Wort soll der Funktion geeignet übergeben werden.
Schreiben Sie außerdem eine Funktion `zaehleAlleRunter`, die analog für jeden Buchstaben eines Wortes die Funktion `zaehleRunter` aufruft.

____________ (2)

d) Schreiben Sie eine Funktion `sindAlleNull` mit Rückgabetyp `bool`, einem Parameter `anzahlen` vom Typ `int[]` und einem Parameter `laenge` vom Typ `int`. Der Parameter `laenge` soll dabei für die Länge des übergebenen Arrays stehen. Die Funktion soll `true` zurückgeben, wenn alle Einträge des Arrays gleich 0 sind, und `false` sonst.

____________ (1)