Human-Computer Interaction Group

Leibniz Universität Hannover

Assignment 1

Programmieren 1 – WiSe 25/26

Prof. Dr. Michael Rohs, Falk Stock, M.Sc

Alle Assignments (bis auf das erste) müssen in Zweiergruppen bearbeitet werden. Ein Gruppenmitglied kann dabei die Lösung der Zweiergruppe gebündelt abgeben. Einzige Ausnahme ist dabei das erste Assignment, wo jedes Gruppenmitglied einzeln abgeben muss. Namen beider Gruppenmitglieder müssen sowohl in der PDF-Abgabe, als auch als Kommentar in jeglichen Quelltextabgaben genannt werden. Plagiate führen zum Ausschluss von der Veranstaltung.

Abgabe bis Donnerstag, den 23.10. um 23:59 Uhr über https://assignments.hci.uni-hannover.de/WiSe2025/Prog1. Die Abgabe muss aus einer einzelnen Zip-Datei bestehen, die den Quellcode, eine PDF für Freitextaufgaben und alle weiteren nötigen Dateien (z.B. Eingabedaten oder Makefiles) enthält. Lösen Sie Umlaute in Dateinamen auf.

Zum Bestehen der Studienleistung müssen Sie mindestens zwei von vier Punkten pro Assignment erzielen. Möchten Sie zusätzlich den Klausurbonus erreichen, müssen Sie über alle Assignments hinweg 75% der Punkte erreichen.

Aufgabe 1: Skript (1 Punkt)

Unter https://postfix.hci.uni-hannover.de/files/prog1script-postfix/script.html findet sich ein Skript, das u.a. die in der Vorlesung vorgestellte Vorgehensweise bei der Lösung von Programmieraufgaben beschreibt.

a) Lesen Sie Kapitel 1 (Introduction) und beantworten Sie folgende Frage in eigenen Worten: Warum werden in dieser Vorgehensweise Beispiele für Eingaben und erwartete Ausgaben erstellt, bevor die Implementierung erfolgt?

b) Lesen Sie Kapitel 2 (Recipe for Atomic Data) und beantworten Sie folgende Frage in eigenen Worten: Warum sollte man Konstanten definieren (z.B. WEEKLY_HOURS), statt die entsprechenden Werte direkt an den benötigten Stellen ins Programm zu schreiben?

c) Was war Ihnen beim Lesen der Kapitel 1 und 2 unklar? Wenn nichts unklar war, welcher Aspekt war für Sie am interessantesten?

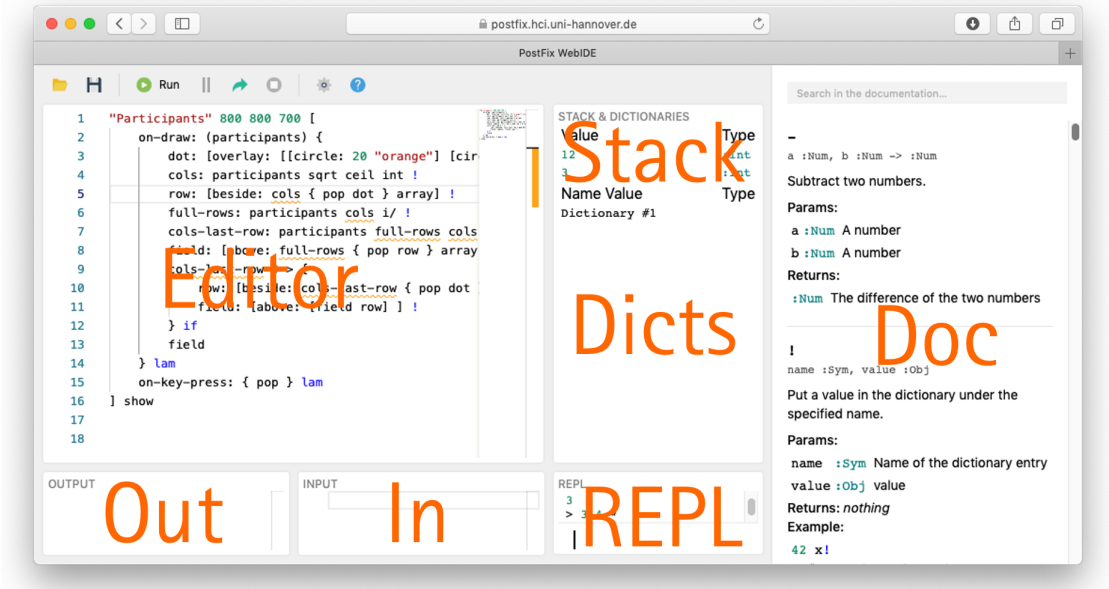

Abbildung 1: Die PostFix-Entwicklungsumgebung

Aufgabe 2: PostFix Entwicklungsumgebung (1 Punkt)

Die PostFix-Entwicklungsumgebung (integrated development environment, IDE) ist verfügbar unter: https://postfix.hci.uni-hannover.de. Öffnen Sie die PostFix-IDE in einem Webbrowser Ihrer Wahl und führen Sie als Test folgendes Programm in der REPL und im Editor aus: `"Hello \{Ihr Name\}" println`

Lesen Sie das Tutorial zu PostFix (https://postfix.hci.uni-hannover.de/postfix-lang.html) bis einschließlich zum Abschnitt „Conditionals". Probieren Sie die Beispiele aus und machen Sie sich mit der Entwicklungsumgebung vertraut. Sie werden sie in den nächsten Übungen benötigen.

a) Wie in Abbildung 1 zu sehen ist, hat die PostFix-IDE sechs Bereiche: Editor, REPL, In, Out, Stack Dicts und Doc. Geben Sie in eigenen Worten wieder, wofür die einzelnen Bereiche in der IDE nützlich sind. Geben Sie (wenn möglich) Beispielcodefragmente (nicht aus der Vorlesung) an, um die Funktionen zu erklären. Speichern Sie Ihre Codefragmente ab und fügen Sie sie der Abgabe bei.

b) Im oberen Bereich der IDE finden sich vier Schaltflächen zum Steuern der Programmausführung (siehe Abbildung 2). Beschreiben Sie kurz, was diese Schaltflächen bewirken.

c) Ein Rechtsklick mit der Maus im Editor-Bereich öffnet ein Kontextmenü. Beschreiben Sie kurz, was die Menüpunkte „Add Conditional Breakpoint", „Toggle Breakpoint" und „Command Palette" bewirken.

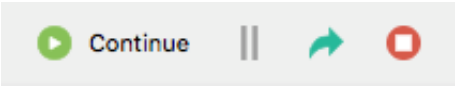

Abbildung 2: Schaltflächen zum Steuern der Programmausführung

Aufgabe 3: PostFix (1 Punkt)

Sollten Sie die Kommandos in der PostFix IDE testen, stellen Sie sicher, dass der Stack vor Ausführung jeder Teilaufgabe geleert ist. Rufen Sie dazu die Funktion clear in der REPL auf, um den Stack zu leeren.

a) Was passiert bei der Ausführung des folgenden Programms: **4 8 ***

b) Was steht auf dem Stack, nachdem das Programm **8 3 4 2 - *** ausgeführt wurde und warum?

c) Wofür sind die Operatoren **read-int**, **read-flt**, **print**, **println** zuständig (in eigenen Worten)?

d) Beschreiben Sie die Operatoren **or** und **and**. Geben Sie jeweils ein Beispiel.

e) Welche der folgenden Ausdrücke sind äquivalent (hinsichtlich des Ergebnisses):

(1) 1 2 3 4 + + +
(2) 1 2 + 2 5 + +
(3) 1 2 + 3 + 4 +
(4) 3 4 + 1 2 + +
(5) 4 3 2 1 + + +

Gilt dies auch für den Operator - (Subtraktion)? Begründen Sie kurz.

f) Führen Sie das folgende Code-Fragment von Hand aus und stellen Sie den Zustand des Stacks nach jedem gelesenen und verarbeiteten Token dar: **8 2 + -1 =**
Nutzen Sie als Hilfe das PostFix Execution Model aus der Vorlesung.

g) Was passiert, wenn Sie (nach **clear**) lediglich **1 /** in die REPL eingeben?

Aufgabe 4: Zahlentest (1 Punkt)

a) Implementieren Sie ein Programm in PostFix, das bei Eingabe einer ganzen Zahl prüft, ob sie den Wert 100 hat. Ist das der Fall soll „Match" ausgegeben werden. Ist der Wert kleiner als 100 soll „Too Low" ausgeben werden, ist der Wert größer als 100 „Too High". Hier soll nicht das Dictionary, sondern nur der Stack verwendet werden. Folgende Befehle können hierbei hilfreich sein:

- **read-int** : Liest eine ganze Zahl ein
- **println** : Ausgabe
- **dup** : Dupliziert das oberste Element auf dem Stack und legt es auf den Stack

- **<**, **>**: Relationale Operatoren
- **Wahrheitswert {dies} {das} if** : Wenn Wahrheitswert, dann mache dies, ansonsten das

b) Ändern Sie das obige Programm so um, dass die Eingabe aus einer Variablen mit dem Namen „test" gelesen wird. Weiterhin soll das Ergebnis in eine Variable „result" geschrieben werden. Dabei soll „result" auf 0 gesetzt werden, wenn „test = 100" erfüllt ist, -1 wenn „test < 100" oder 1 wenn „test > 100". Weisen Sie der Variablen „test" zu Beginn des Programms einen Wert zu. Prüfen Sie das Verhalten des Programms mit verschiedenen Werten für „test".