Übung 4

Abgabe: 15. Mai 2023, 18:00 Uhr

Hinweis. Wir haben die Abgabefrist auf Montag Abend verlängert :)

Aufgabe 4.1: Zeigerfehler [10 Punkte, Link zur Abgabe] In der Datei `main.c` im Repository haben sich neun fehlerhafte Zeilen eingeschlichen. Beheben Sie alle Fehler und sorgen Sie dafür, dass die Datei wieder kompiliert.

Anforderungen:

- Die Ausgabe des Programms darf nicht verändert werden.
- Sie dürfen keine Zeilen löschen.
- Markieren Sie alle fehlerhaften Zeilen mit Kommentaren.
- Benutzen Sie nicht `malloc` für die Lösung.

Aufgabe 4.2: Trim Hashes [10 Punkte, Link zur Abgabe] In dieser Aufgabe geht es darum, `#`-Zeichen vom Anfang und Ende eines Strings zu entfernen.

Im Repository befindet sich eine Datei `main.c` mit Code, der zunächst eine Zahl n und dann einen String der Länge n von `stdin` einliest. Dieser String wird einer Funktion `trim` aus `trim.c` übergeben und das Ergebnis des Funktionsaufrufes ausgegeben.

Ihre Aufgabe ist es, die Funktion `trim` in `trim.c` zu implementieren. Die Funktion bekommt einen String `str`. Sie soll einen Zeiger auf das linkeste Zeichen in `str` zurückliefern, das kein `#`-Zeichen ist. Außerdem soll sie dafür sorgen, dass `str` direkt nach dem rechtestens Zeichen endet, das kein `#`-Zeichen ist. Insbesondere soll die Funktion *in-place* arbeiten, d.h., sie soll keinen Speicher für einen neuen String anlegen.

Die Ausgabe des Programmes soll in einfache Anführungszeichen eingeschlossen sein.

Ein passendes Makefile ist bereits vorhanden, so dass Sie nur die Datei `trim.c` ändern sollen.

Verwenden Sie nicht die Funktionen aus `string.h` der C-Standardbibliothek.

Listing 1: Beispieldurchlauf

```
1 > cat input.in
2 21
3 ###Marie##Curie######
4 > make
5 > ./trim < input.in
6 'Marie##Curie'
7 >
```

Aufgabe 4.3: Hashtable [10 Punkte, Link zur Abgabe] Ein Team von Forscher:innen hat Vögel mit GPS Trackern ausgestattet und notiert, wo sich die markierten Vögel zu einem festen Zeitpunkt aufgehalten haben. Dabei wird jeder Vogel über eine nicht-negative, ganzzahlige ID identifiziert und seine Position ist durch eine ganzzahlige x- und eine ganzzahlige y-Koordinate gegeben.

Wir möchten jetzt ein Programm entwickeln, dass die Positionen der Vögel im Speicher ablegt und Anfragen der Art „Wo wurde der Vogel mit ID `id` beobachtet?“ beantworten kann. Da der mögliche Wertebereich für die IDs sehr groß ist, bietet es sich an, die Daten in einer Hashtabelle abzulegen.

Ihre Aufgabe ist es, die Hashtabelle zu implementieren. Die Hashtabelle soll die Operationen unterstützen, die in der Datei `hashtable.h` im Repository spezifiert sind. Implementieren Sie die Funktionen in `hashtable.c`.

Damit das Programm funktioniert müssen die eingelesenen Daten in der Hashtabelle abgelegt werden. Ergänzen Sie die Datei `main.c` so, dass die Funktionen der Hashtabelle geeignet aufgerufen werden. Die Hashtabelle soll eine feste Größe `TABLE_SIZE=23` haben und als Schlüssel die `id`s der Vögel benutzen. Dabei soll der Schlüssel `key` mit der Hashfunktion `h(key) := (a*key + b) % p` gehasht werden; a, b und p sind in der Eingabe gegeben. Im Allgemeinen ist `p > TABLE_SIZE`, so dass noch geeignet auf die Tabellenpositionen umgerechnet werden muss. Kollisionen sollen mit linearem Sondieren aufgelöst werden. Leere Einträge in der Hashtabelle können mit der Konstanten `EMPTY` markiert werden.

Als Eingabe erhält das Programm auf `stdin` n Beobachtungen und m Anfragen. Die erste Zeile der Eingabe enthält n und m. Es folgt eine weitere Zeile, die a, b und p enthält. Die folgenden n Zeilen enthalten jeweils eine ID sowie eine x- und eine y-Koordinate. Es folgen die m Zeilen mit Anfragen; jede Anfrage besteht aus einer ID.

Das Programm soll nach jedem eingefügten Schlüssel einmal die Hashtabelle mit der Funktion `print_table` aus `hashtable.h` ausgeben.

Anschließend soll für jede Anfrage ID der folgende Text ausgegeben werden:

- `Vogel ID bei (x, y) beobachtet.`, falls der Vogel ID tatsächlich beobachtet wurde.
- `Vogel ID nicht beobachtet.`, falls für den Vogel ID in der Eingabe keine Beobachtung existiert.

Als letztes soll das Programm den Text `k Kollisionen beim Einfügen.` ausgeben; dabei soll für k die Anzahl der beim Einfügen aufgetretenen Kollisionen eingesetzt werden. Kollisionen beim Suchen der Schlüssel werden ignoriert.

Kann ein Schlüssel nicht eingefügt werden, weil keine freie Position gefunden werden kann soll das Programm `Fehler: Hashtabelle voll` ausgeben.

Sie dürfen davon ausgehen, dass in den Beobachtungen keine `id` doppelt vorkommt.

Listing 2: Beispielausgabe mit reduzierter Tabellengröße `TABLE_SIZE=10`

```
> cat input1.in
5 3
1 0 1000
2 2 0
3 0 3
4 4 0
5 0 5
25 0 25
2
25
47
> make
> ./birds < input1.in
#  000#  001#  002#  003#  004#  005#  006#  007#  008#  009#
#     #     #    2#     #     #     #     #     #     #     #

#  000#  001#  002#  003#  004#  005#  006#  007#  008#  009#
#     #     #    2#    3#     #     #     #     #     #     #

#  000#  001#  002#  003#  004#  005#  006#  007#  008#  009#
#     #     #    2#    3#    4#     #     #     #     #     #

#  000#  001#  002#  003#  004#  005#  006#  007#  008#  009#
#     #     #    2#    3#    4#    5#     #     #     #     #

#  000#  001#  002#  003#  004#  005#  006#  007#  008#  009#
#     #     #    2#    3#    4#    5#   25#     #     #     #

Vogel 2 bei (2, 0) beobachtet.
Vogel 25 bei (0, 25) beobachtet.
Vogel 47 nicht beobachtet.
1 Kollisionen beim Einfügen.
```

Im Repository ist ein Teil der `main`-Methode in `main.c` schon vorimplementiert. Sie muss allerdings noch um geeignete Aufrufe der Hashtabellenfunktionen ergänzt werden.

Ein geeignetes Makefile ist vorhanden; ändern Sie nur die Dateien `hashtable.c` und `main.c`.