Übung 3

Abgabe: 8. Mai 2023, 08:00 Uhr

Hinweise zur Abgabe

Aus Sicherheitsgründen löscht das Bewertungsskript grundsätzlich keine Dateien. Es kann daher passieren, dass das Skript bei mehrfacher Ausführung Ihren Code nicht neu kompiliert! Empfehlung: Vor jeder Ausführung des Skriptes einmal per Hand kompilieren :)

Aufgabe 3.1 [10 Punkte, Link zur Abgabe] Sie finden im Repository eine Implementierung von 3-dimensionalen Punkten und einigen passenden Rechenoperationen. Die Implementierung ist fertig, aber was noch fehlt ist das Makefile!

Ihre Aufgabe: Schreiben Sie ein Makefile für das Projekt im Repository. Sorgen Sie dafür, dass das Makefile nicht unnötig viel kompiliert, indem Sie passende Abhängigkeiten eintragen. Sie dürfen implizite Regeln verwenden. Das Ziel ist, dass das Projekt mit dem Aufruf `make` kompiliert und anschließend als `./linalg` ausgeführt werden kann.

Wichtige Informationen:

- Es soll mindestens mit den Compilerflags `-Wall -g` kompiliert werden. Fügen Sie nach Bedarf weitere Compilerflags hinzu.
- Die kompilierte ausführbare Datei muss `linalg` heißen – sonst funktionieren die Tests nicht.
- Es soll die Debugsausgabe aktiviert werden; dafür muss das Makro `CA_DEBUG` definiert werden.

Verändern Sie *nur* das Makefile und keine andere Datei im Repository.

Listing 1: Beispieldurchlauf *ohne* Debugausgabe

```
~/caldat> cat input.in
1 1 1
0 1 0
~/caldat> ./linalg < input.in
Quadrierte Länge von p: 3.000
Quadrierte Länge von q: 1.000
Quadrierter abstand von p und q: 2.000
Summe von p und q: [1.000, 2.000, 1.000]
~/caldat>
```

Listing 2: Beispieldurchlauf *mit* Debugausgabe

```
~/caldat> cat input.in
1 1 1
0 1 0
~/caldat> ./linalg < input.in
[Debug] Punkt [1.000, 1.000, 1.000] gelesen.
[Debug] Punkt [0.000, 1.000, 0.000] gelesen.
Quadrierte Länge von p: 3.000
Quadrierte Länge von q: 1.000
Quadrierter abstand von p und q: 2.000
Summe von p und q: [1.000, 2.000, 1.000]
~/caldat>
```

Aufgabe 3.2 [10 Punkte, Link zur Abgabe] In dieser Aufgabe geht es darum, Headerdateien zu schreiben.

Sie finden im Repository ein Programm, das mit rationalen Zahlen („Brüchen“) rechnet. Dazu verwendet es eine Struct `Fraction`, die einen Bruch repräsentiert sowie einige Bruchrechenoperationen aus der Datei `fraction.c`. Die Datei `io.c` enthält Hilfsfunktionen für die Ausgabe von Brüchen. In `genmath.c` finden Sie einige Hilfsfunktionen, die zur Implementierung der Bruchrechenoperationen benötigt werden. Die `main`-Funktion ist in `main.c`.

Alle Funktionen sind bereits fertig implementiert und es gibt auch schon ein fertiges Makefile. Leider sind alle Headerdateien im Projekt noch leer und es fehlen etliche `#include`-Anweisungen!

Ihre Aufgabe: Schreiben Sie die Headerdateien `fraction.h`, `genmath.h` und `io.h` so, dass alle Funktionen in `main.c` sichtbar sind und das gesamte Programm ohne Warnungen kompiliert. Fügen Sie dazu auch geeignete `#include`-Anweisungen hinzu.

Die Datei `main.c` darf nicht verändert werden!

Listing 3: Beispieldurchlauf

```
1  ~/caldat> cat input.in
2  1 2
3  3 4
4  ~/caldat> ./fracdemo
5  x = 1/2
6  y = 3/4
7  1/2 ist kleiner als 3/4
8  1/2 + 3/4 = 5/4
9  1/2 + 3/4 = 3/8
10 ~/caldat>
```

Aufgabe 3.3 [10 Punkte, Link zur Abgabe] In dieser Aufgabe sollen Sie das *Sieb des Eratosthenes* implementieren; das ist ein Algorithmus zur Berechnung von Primzahlen. Eine natürliche Zahl $n \geq 1$ ist eine Primzahl genau dann, wenn sie genau zwei verschiedene Teiler (1 und n) besitzt. Nach Definition sind 0 und 1 keine Primzahlen; 2 ist eine Primzahl.

Der Algorithmus basiert auf der Beobachtung, dass jede natürliche Zahl ≥ 2 entweder selbst eine Primzahl ist oder ein echtes Vielfache einer Primzahl ist. Ein echtes Vielfache einer Zahl ℓ ist ein Vielfaches $k \cdot \ell$, das nicht ℓ selbst ist.

Für eine gegebene natürliche Zahl $n \geq 1$ berechnet das *Sieb des Eratosthenes* alle Primzahlen zwischen 1 und n (inklusive) auf folgende Weise:

1. Markiere alle Zahlen $1, \ldots, n$ als potentielle Primzahlen.

2. Markierungsphase:

   a) Streiche die 1.

   b) Streiche alle echten Vielfachen von 2.

   c) Streiche alle echten Vielfachen von 3.

   d) Streiche alle echten Vielfachen von 5.
   (die Vielfachen der 4 müssen nicht gestrichen werden, weil sie Vielfache von 2 sind.)

   e) Streiche alle echten Vielfachen von 7.

   f) ...

   g) Streiche alle echten Vielfachen von n.

3. Gib alle Zahlen aus, die jetzt noch markiert sind.

Ihre Aufgabe: Implementieren Sie das Sieb des Eratosthenes. Sie erhalten als Eingabe eine natürliche Zahl $1 \leq n \leq 10000$ auf der Standardeingabe. Geben Sie alle Primzahlen zwischen 1 und n (inklusive) auf der Standardausgabe in aufsteigend sortierter Reihenfolge aus.

In der Datei `main.c` finden Sie Code, der eine natürliche Zahl einliest und dann eine Funktion `print_primes(int n)` aufruft. Implementieren Sie die Funktion `print_primes(int n)` in der Datei `primes.c` mit einer geeigneter Headerdatei `primes.h`.

Verändern Sie die Datei `main.c` nicht!

Die Testfälle werden versuchen, Ihr Programm mit `make` zu kompilieren. Schreiben Sie also auch ein passendes Makefile und erzeugen Sie damit eine ausführbare Datei `primes`.

Listing 4: Beispieldurchlauf

```
~/caldat> cat input.in
20
~/caldat> ./primes < input.in
2
3
5
7
11
13
17
19
~/caldat>
```