Institut für Informatik

Lehrstuhl für Technische Informatik
Prof. Dr. Björn Scheuermann

Digitale Systeme 2020
C-Programmierpraktikum

1 Aufgabenstellung

In dieser Aufgabe sollen Sie den Kern einer Computerspiel-Engine schreiben. Die Spiel-Mechanik kann man sich als eine Mischung aus „Vier gewinnt“ und „Candy Crush“ vorstellen: Auf ein diskretes Spielfeld $\mathbb{Z} \times \mathbb{N}_0$ fallen von oben (aus positiver y-Richtung) nacheinander einzelne Spielsteine (nachfolgend „Steine“) unterschiedlicher Farben herunter und bilden Stapel. Sobald vier oder mehr Steine der selben Farbe vertikal, horizontal oder diagonal in einer geraden Linie zusammenhängen, lösen diese sich auf; die darüber liegenden Steine rutschen dann nach unten nach. Auch wenn ein Stein Teil zweier Viererlinien ist, lösen sich beide auf (z. B. wenn sieben Steine T-förmig angeordnet sind oder sich zwei Linien in anderer Weise überschneiden). Die Schritte des Auflösens und Nachrutschens werden wiederholt, bis keine Viererlinie gleicher Steine mehr existiert. Erst dann kann der nächste Stein von oben in das Spielfeld hineinfallen. Wenn durch das Nachrutschen mehrere Viererlinien gleichzeitig entstehen, so lösen sich diese auch gleichzeitig auf.

Ihr Programm soll für eine gegebene Sequenz hereinfallender Steine die Spielmechanik simulieren und den resultierenden Endzustand auszugeben.

2 Ein- und Ausgabedaten

Ihr Programm soll ohne jegliche Interaktion und ohne Benutzeroberfläche mit einem einzelnen Aufruf `./loesung` von der Kommandozeile aus nutzbar sein. Kommandozeilenargumente werden nicht verwendet. Das Spielfeld sei zu Beginn leer. Als „Farben“ werden ganze Zahlen in $[0, 254]$ verwendet. Eine endliche Sequenz von (Farbe, x-Koordinate)-Paaren wird als ASCII-codierter Zeichenstrom von der Standardeingabe (`stdin`) gelesen: eine Zeile beschreibt genau einen Stein und besteht aus zwei dezimal notierten Ganzzahlen getrennt durch ein oder mehrere Leerzeichen.[1] Die erste Zahl steht für die Farbe des Steins, die zweite Zahl beschreibt in welcher Spalte der Stein herunterfällt. Jeder Zeilenumbruch wird gemäß UNIX-Konvention mit einem Linefeed (`0x0a`) codiert. Die letzte Zeile kann, muss aber nicht mit einem Zeilenumbruch terminiert werden.

Die Ausgabe ist ein ASCII-codierter Zeichenstrom, der den finalen Zustand des Spielfeldes beschreibt: Jede Zeile soll einen Stein beschreiben und als Tripel (Farbe, x-Koordinate, y-Koordinate), getrennt durch jeweils genau ein Leerzeichen, formatiert sein. $y = 0$ entspricht dabei der Grundlinie, auf der die ersten Steine zum liegen kommen.

[1] Als *Extended Regular Expression* wäre das zu akzeptierende Format also `^[0-9]+ +-?[0-9]+$`

Es folgt ein Beispiel für gültige Ein- und Ausgaben:

stdin

```
1 0
1 1
1 2
0 3
0 -1
0 -2
0 -3
0 -3
3 -2
1 -1
2 0
2 1
2 2
3 3
3 2
3 1
3 0
2 -1
0 -2
0 -1
0 -4
```

stdout

```
3 0 0
3 3 1
0 3 0
3 2 0
3 -2 0
3 1 0
```

Beachten Sie: In einem ersten Auflösungsschritt lösen sich sieben Nullen auf. In einem zweiten Auflösungsschritt lösen sich *gleichzeitig* vier Einsen und vier Zweien auf, sodass zu keinem Zeitpunkt eine horizontale Linie von vier Dreien entsteht.

Wenn Ihr Programm eine Eingabe erhält, die nicht der obigen Spezifikation genügt, soll es gar nichts auf die Standardausgabe schreiben. Auf die Standardfehlerausgabe (`stderr`) soll dann eine Fehlermeldung geschrieben werden; als Returncode soll in diesem Fall ein Wert ungleich null zurückgegeben werden. Falls Ihr Programm regulär durchläuft, soll als Returncode null zurückgegeben werden.

Insbesondere sollten folgende Fälle als falsche Eingabe gewertet werden:

- Ein anderes Zeichen als eine Dezimalziffer, ein Minus, ein Leerzeichen oder ein Linefeed tritt auf.
- Eine Zeile enthält nicht genau zwei ganze Zahlen.
- Die Farbe liegt nicht im gültigen Zahlenbereich.

Der Einfachheit halber *dürfen* Sie neben einem einzelnen Linefeed (LF) auch ein Carriage Return (CR) oder ein „CR LF“ als Zeilenumbruch akzeptieren. Wir verzichten darauf, diesbezüglich zu testen.

3 Beispieldaten

Wir stellen Ihnen in Moodle einige Beispiele für Ein- und Ausgabepaare (`example01.stdin`, `example01.stdout`) zur Verfügung, anhand derer Sie selbst überprüfen können, ob Ihr Programm gewisse Eingaben korrekt bearbeitet. Da keine bestimmte Sortierung der Ausgabedaten

verlangt wird, müssen Sie Ihre Ausgabedaten ggf. mittels `sort` (GNU coreutils) sortieren, bevor Sie eine bytegenaue Übereinstimmung mit unseren Ausgabedaten mittels `diff` überprüfen können. Die Beispiel-Eingaben decken viele – aber nicht alle – Grenzfälle ab, mit denen Ihr Programm zurechtkommen muss.

Für die Überprüfung Ihrer Abgabe werden andere Datensätze zum Einsatz kommen.

Bezüglich falsch formatierter Eingaben oder fehlerhafter Aufrufe müssen Sie selbst kreativ werden. Versuchen Sie, Ihr Programm mit Hilfe bewusst bösartiger Eingaben zum Absturz zu bringen und vergewissern Sie sich, dass für jeden möglichen Programmdurchlauf *niemals* Schutzverletzungen auftreten, nur gültige Lese- und Schreiboperationen auf dem Speicher ausgeführt werden und immer aller vom Heap allozierter Speicher vor Programmende freigegeben wird. *Dies gilt auch für beliebige fehlerhafte Eingaben.*

4 Hinweise zum Algorithmus

Die Aufgabenstellung lässt sich auf folgende Weise lösen.

1. Zu jedem Zeitpunkt sei ein hinreichend großer Ausschnitt des Spielfeldes als zweidimensionales Array dargestellt.

2. Da die x-Koordinate negativ sein kann, benötigen Sie gegebenenfalls eine Offset-Variable, sodass der Zustand des Ortes (x, y) durch `array[offset+x][y]` beschrieben wird.

3. Kommt ein Stein außerhalb des aktuellen Ausschnitts hinzu, so wird ein größeres Array alloziert, gegebenenfalls der Offset angepasst, der alte Zustand in das neue Array kopiert und das alte Array freigegeben.

4. Kommt ein neuer Stein hinzu, so wird das Spielfeld nach Viererlinien abgesucht; diese werden aufgelöst.

5. Nach dem Auflösen ist das Nach-unten-Rutschen durchzuführen.

6. Anschließend muss nach weiteren Viererlinien gesucht werden; Auflösen und Nachrutschen ist so lange zu wiederholen, bis keine Viererlinien mehr existieren.

Es gibt deutlich einfachere, effizientere und elegantere Lösungsstrategien. Jede von Ihnen eingereichte Lösung ist akzeptabel, solange sie nicht noch ineffizienter als der von uns vorgeschlagene Lösungsweg ist und Sie im Abtestat begründen können, warum Ihre Lösung korrekte Resultate liefert.

Sie können der Einfachheit halber davon ausgehen, dass die eingegebenen x-Koordinaten betragsmäßig nicht größer als 2^{20} sind. Größere Koordinaten können Sie akzeptieren, dürfen stattdessen aber auch mit einer Fehlermeldung abbrechen. Falls Ihr Programm größere Werte akzeptiert, darf es jedoch nicht *ohne Fehlermeldung* falsche Resultate generieren.

Beachten Sie bei der Auswahl der Datenstrukturen und Ihrer Implementierung einen sorgsamen Umgang mit Speicherplatz. Insbesondere sollten Sie darauf achten, dass Ihre Datenstrukturen dynamisch mit der Größe des Spielzustands wachsen.

Gehen Sie zudem davon aus, dass Sie die Eingabedaten nur einmal sequenziell lesen können, dass also `fseek` und verwandte Operationen auf `stdin` nicht möglich sind. Vielleicht ist die Eingabe ja gar keine Datei sondern wird in Ihr Programm „hineingepipet“.

5 Zeitplan

Sie können sofort mit der Bearbeitung der Aufgabe beginnen. Bis zur finalen Abgabefrist haben Sie beliebig viele Abgabeversuche, eine erste Abgabe ist ab sofort möglich. Wir bemühen uns, Ihnen nach jedem Abgabeversuch kurzfristig Rückmeldung zu geben, ob Ihre Lösung korrekt ist oder, falls nicht, inwiefern sich Ihr Programm falsch verhält. **Nach der finalen Abgabefrist werden keinerlei weitere Abgaben mehr berücksichtigt.**

Wir ermutigen Sie ausdrücklich, so früh wie möglich mit der Bearbeitung der Aufgabe zu beginnen und einen ersten sinnvollen Lösungsversuch bis Anfang Juli abzugeben. Je nach Aufkommen von Abgaben können wir wenige Wochen vor der finalen Deadline nicht mehr garantieren, kurzfristig individuelles Feedback zu geben, das Ihnen auch wirklich weiter hilft. Da eine vollständig korrekte Lösung für das Bestehen erforderlich ist, sind fast immer mehrere Iterationen von Abgabe und Feedback notwendig. [2]

Wenn Sie eine korrekte Lösung eingereicht haben, müssen Sie uns für die Erbringung der Modulteilleistung zusätzlich noch im Rahmen eines kurzen persönlichen Abtestats Ihre Lösung erläutern und Fragen dazu beantworten. Der für die Durchführung der Abtestate vorgesehene Zeitraum liegt im September 2020. Nach individueller Absprache sind frühere Termine gegebenenfalls möglich.[3]

Datum	Beschreibung
09.07.2020	Empfohlene Deadline für eine erste Abgabe; bei Abgabe bis zu diesem Termin können wir im Falle von Problemen mit der Abgabe gewährleisten, dass nach einer ersten Rückmeldung noch Zeit zur Nachbesserung bis zur endgültigen Deadline bleibt.
27.08.2020 (23:59 Uhr MESZ)	Spätestmögliche Deadline für die Abgabe der Endversion, die alle Kriterien vollständig erfüllen muss.
07.09.2020 – 11.09.2020	In diesem Zeitraum finden die mündlichen Testate in Einzelsitzungen statt. Eine frühere Abnahme ist nach individueller Absprache möglich.

6 Wichtige Hinweise und weitere Anforderungen

Beachten Sie bei der Erstellung Ihres Programms **unbedingt** folgende Punkte:

- Halten Sie sich strikt an das oben definierte Ausgabeformat und die Aufrufkonventionen. Wir überprüfen Ihr Programm automatisiert auf die Korrektheit der ausgegebenen Lösungen. *Eine falsch formatierte Ausgabe wird nicht als korrekt anerkannt!* Im Zweifelsfall fragen Sie *rechtzeitig vor Abgabe* nach!
- Verwenden Sie keine Bibliotheken außer der C-Standardbibliothek wie sie in der 2011er Ausgabe des ISO C Standard beschrieben ist. Insbesondere Nutzer von Softwareprodukten

[2] Die von uns gesetzten Deadlines sind ernst gemeint. Von einer Abgabe ein paar Stunden, Minuten oder Sekunden vor der finalen Abgabefrist raten wir dringend ab. Technische Probleme bei der Abgabe sind kein Grund für eine Ausnahme von der Frist.

[3] Wir bitten um Verständnis, dass wir derzeit selbst nicht abschätzen können, inwiefern dies zu welchem Zeitpunkt mit den geltenden Infektionsschutzmaßnahmen des Landes und den Regelungen der HU zum Prüfungsbetrieb kompatibel ist.

aus Redmond weisen wir darauf hin, dass Funktionen, die einen DromedaryCaseNamen tragen (wie `StrToInt`), sehr wahrscheinlich nicht Teil der C-Standardbibliothek sind.

- Ihr Programm soll als eine einzige C-Quelldatei mit dem Namen `loesung.c` in gültigem ISO C11 geschrieben sein. Ihr Programm muss sich mit folgendem einzelnen gcc-7.5.0-Compileraufruf[4] fehlerfrei übersetzen und linken lassen:
  `gcc -o loesung -O3 -std=c11 -Wall -Werror -DNDEBUG loesung.c`
  Um Fehler bei der Implementierung zu vermeiden, empfiehlt es sich, zusätzlich die Compiler-Flags `-Wextra` und `-Wpedantic` zu verwenden. Verlangt wird dies aber nicht.

- Ihr Programm muss auf der Kommandozeile ohne grafische Oberfläche lauffähig sein und ohne jegliche Kommandozeilenargumente laufen, d. h. es soll mit folgendem Aufruf (in `sh`, `bash` oder `zsh`) ausführbar sein:
  `cat example.stdin | ./loesung | tee myresult.stdout` Wenn Sie die Ausgabe Ihres Programms mit unseren Beispielausgaben vergleichen wollen, können Sie zum Beispiel folgenden Aufruf verwenden:
  `cat example01.stdin | ./loesung | sort | diff <(sort example01.stdout) -`

- Legen Sie Ihr Programm so aus, dass es mit beliebig großen Eingabedaten umgehen kann. Hierfür ist es unbedingt notwendig, dass Sie dynamische Speicherverwaltung einsetzen. Wir werden Ihr Programm mit *großen* Datensätzen testen!

- Ihr abgegebenes Programm wird automatisch auf Speicherlecks überprüft. Stellen Sie sicher, dass es keine Speicherlecks aufweist. Dies können Sie beispielsweise mit dem Werkzeug `valgrind`[5] bewerkstelligen. Geben Sie jeglichen dynamisch zugewiesenen Speicher vor Programmende korrekt wieder frei.

- Stellen Sie sicher, dass Ihr Programm *niemals* eine Schutzverletzung (segmentation fault, segfault) auslöst, egal welche gültige oder ungültige Eingabe es erhält. Achten Sie darauf, dass Ihr Programm auch auf Systemen mit sehr wenig Speicher keine Schutzverletzung auslöst. Verlassen Sie sich *niemals* darauf, dass ein `malloc` tatsächlich neuen Speicher alloziert. Falls `malloc` fehlschlägt, darf Ihr Programm natürlich abbrechen.

- Wir werden alle abgegebenen Lösungen benchmarken. Die ressourcensparsamsten Lösungen werden am Ende des Semesters mit einem kleinen Preis ausgezeichnet. Wir werden hierfür die Lösungen hinsichtlich ihrer Ausführungszeit und ihres Speicherverbrauchs untersuchen.

- Kommentieren Sie Ihren Quelltext in angemessenem Umfang. Im Testat werden Sie Ihren Quelltext detailliert erklären müssen. Dabei können Kommentare helfen. Erforderlich sind Kommentare natürlich nur an Stellen, an denen Ihr Code nicht in trivialer Weise für sich selbst spricht.

- Wir werden Feedback zu Ihren Lösungsversuchen ausschließlich als persönliche Moodle-Nachrichten versenden. Sorgen Sie dafür, dass Sie diese Nachrichten rechtzeitig lesen.

[4] Den gcc 7.5 finden Sie z.B. auf den Rechnern des Berlin-Pools ([alex|britz|buch|buckow|...|gruenau1|gruenau2|...].informatik.hu-berlin.de) installiert.

[5] `http://valgrind.org/`

- Wir empfehlen Ihnen, Ihr Programm vor jedem Abgabeversuch selber mit der gegebenen Compiler-Version sowie -Flags zu übersetzen und mit allen von uns zur Verfügung gestellten Testdaten und unter Verwendung von Valgrind zu testen.
- Für das Bestehen ist es nicht erforderlich, ein besonders effizientes Programm zu schreiben. Dennoch sind wir aus naheliegenden praktischen Gründen gezwungen, bei der Überprüfung sowohl die CPU-Zeit eines Prozesses als auch die Menge an gleichzeitig verwendetem RAM[6] zu begrenzen. Obwohl wir die Grenzen hierfür so großzügig gewählt haben, dass die meisten Implementierungen wahrscheinlich nicht ansatzweise in deren Nähe kommen werden, geben wir sie der Transparenz halber an:
  - Ihr Programm sollte jedes der von uns gestellten Beispielein- und Ausgabepaare auf einem der Pool-Rechner in je 50 s CPU-Zeit bearbeiten können.
  - Ihr Programm sollte zu keinem Zeitpunkt mehr als $\max\{10^6, 200A_{\text{max}}^{\text{Hüllkörper}}\}$ Byte RAM benötigen, wobei $A_{\text{max}}^{\text{Hüllkörper}}$ das über alle im Verlauf der Bearbeitung einer Eingabe auftretenden geometrischen Spielzustände gebildete Maximum der Fläche des rechteckigen Hüllkörpers bezeichnet.
    (Zum Beispiel wäre $A^{\text{Hüllkörper}}\big(\{(0,23,0),(1,43,0),(0,43,1)\}\big) = 42$.)
- Bearbeiten Sie diese Programmieraufgabe *alleine*. Gerne dürfen Sie Lösungsstrategien, auftretende Probleme, etc., *verbal* mit Ihren Kommilitonen diskutieren. Aber *teilen Sie keinen Code!*

6.1 Valgrind

Valgrind ist ein sehr vielseitiges Werkzeug. Im einfachsten Fall testen Sie Ihr Programm mittels
`cat example01.stdin | valgrind ./loesung 1> /dev/null`

Wenn Ihr Programm keine Fehler in der Speicherverwaltung aufweist, sollten in der Valgrindausgabe folgende zwei Zeilen vorkommen:

```
==1234== All heap blocks were freed - no leaks are possible
==1234== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

(Statt „1234“ wird dort in der Regel eine andere Zahl stehen.)

6.2 SSH

Ob Ihr Code compiliert, hängt von der verwendeten gcc-Version ab. Machen Sie sich am besten mit `ssh` vertraut und testen Sie vor jeder Abgabe, ob sich Ihr Code auf einem `gruenau` oder einem Pool-Rechner übersetzen lässt. Verwenden Sie zum Beispiel folgenden Aufruf, um Ihre im aktuellen Verzeichnis liegende Lösung direkt auf `gruenau6` zu bauen:

```
cat loesung.c | ssh myldapusername@gruenau6.informatik.hu-berlin.de \
'cat > ~/loesung.c && gcc-7 -o loesung -O3 -std=c11 -Wall -Werror -DNDEBUG loesung.c'
```

6.3 Randfälle

- Die leere Eingabe ist eine gültige Eingabe.
- Ist der Endzustand des Spiels leer, so ist die leere Ausgabe die einzig gültige Ausgabe.

[6] „maximum resident set size“, MRSS

- Wenn die Eingabe formal richtig ist, Ihr Programm aber nicht genug Speicher allozieren kann, soll es mit einer entsprechenden Fehlermeldung abbrechen. Verwenden Sie Stack-Speicher nicht exzessiv, sondern besser Heap-Speicher.

Sollten Sie Fragen zur Aufgabenstellung haben, stellen Sie diese bitte im Moodle Forum (`https://moodle.hu-berlin.de/mod/forum/view.php?id=2158042`) und **nicht via E-Mail oder persönlicher Moodle Nachricht**.

Viel Erfolg!