ulm university universität
uulm

Institut für Numerische Mathematik

Dr. Andreas F. Borchert
Moritz Carmesin

25. Oktober 2019
Blatt 2

Systemnahe Software I (WS 2019/2020)

Abgabe bis zum 8. November 2019, 14:00 Uhr

Lernziele:

- Grundlegende Kontrollstrukturen in C
- Einfache Ein- und Ausgabe mit *scanf* und *printf* (Aufgabe 3)
- Implementierung von Bit-Arrays auf Basis eines ganzzahligen Datentyps (Aufgabe 4)

Aufgabe 3: „Aliquot Game“

Das Ziel dieser Aufgabe ist es, ein kleines Spiel zu programmieren, in diesem Fall das sogenannte Aliquot Game. Ein aliquoter Teiler (echter Teiler) einer natürlichen Zahl ist ein Teiler, der nicht identisch mit der Zahl selbst ist. So hat zum Beispiel die Zahl 12 die aliquoten Teiler 1, 2, 3, 4 und 6. Primzahlen haben die 1 als einzigen aliquoten Teiler. Die 1 selbst hat keinen aliquoten Teiler. Das „Aliquot Game“ ist ein Spiel für zwei Spieler, die abwechselnd ziehen. Das Spiel beginnt mit einer beliebigen natürlichen Zahl. Der Spieler, der am Zug ist, wählt einen der aliquoten Teiler aus und zieht diesen von der Zahl ab. Der Spieler, der die 1 gezeigt bekommt (und aus diesem Grund nicht mehr ziehen kann), verliert das Spiel.

Ein Spielablauf könnte so aussehen:

```
theon$ ./aliquot
*** Aliquot Game ***
Number: 65
Your move: 2
2 is not a proper divisor of 65!
Your move: 5
Number: 60
My move: 15
Number: 45
Your move: 15
Number: 30
```

```
My move: 15
Number: 15
Your move: 5
Number: 10
My move: 5
Number: 5
Your move: 1
Number: 4
My move: 1
Number: 3
Your move: 1
Number: 2
My move: 1
You lose!
theon$
```

Das Spiel soll so programmiert werden, dass ein menschlicher Spieler gegen den Computer antritt. Das Programm soll dem Spieler die aktuelle Zahl präsentieren und ihn dann nach einem aliquoten Teiler fragen, den es anschließend vom Terminal einliest. Es sollen nur valide Antworten zugelassen werden. Der Gegenspieler soll ein Teil des Programmes selbst sein. Im einfachsten Fall kann der Gegenspieler immer die 1 wählen. Ihm ein wenig mehr Intelligenz zu spendieren, wäre aber auch nicht verkehrt.

Hinweise:

Der Return-Wert von *scanf* und die Benutzereingaben sind zu überprüfen. Wenn keine Zahlen eingegeben werden, soll eine entsprechende Fehlermeldung ausgegeben und die Ausführung des Programms mit *exit(1)* abgebrochen werden (hierzu wird **#include** <stdlib.h> benötigt). Wenn die eingegebene Zahl kein aliquoter Teiler ist, soll das ebenfalls abgefangen und eine entsprechende Meldung ausgegeben werden. Beim Ende der Eingabe liefert *scanf EOF* zurück. Eine Zufallszahl lässt sich mit Hilfe der Bibliotheksfunktion *rand()* ermitteln, wobei der Pseudozufallszahlengenerator zuvor mit *srand()* in geeigneter Form initialisiert werden sollte, etwa mit Hilfe des Systemaufrufs *time()*, wofür die Header-Datei *time.h* einzubinden ist.
Einreichen können Sie Ihre Lösung mit folgendem Kommando auf der Theon:

```
submit ss1 3 aliquot.c
```

Wenn Sie Ihre Lösung einreichen, wird diese einer Reihe automatisierter Tests unterzogen, über die sie sofort interaktiv und auch per E-Mail einen Bericht erhalten. Bei den Tests gibt es sowohl solche, die nur den Quelltext betrachten, als auch solche, die Ihr Programm mit Testdaten ausführen. Es ist dabei durchaus möglich, dass die statischen Tests etwas beanstanden, das in Ordnung ist. Die Laufzeittests können auch mit Meldungen abbrechen wie „Execution aborted by signal“ mit einer Signalnummer, die größer oder gleich 30 ist. Dann wurde es wegen Ressourcenverbrauch beendet. Das kann passieren, wenn Ihr Programm

z.B. in eine Dauerschleife gerät. Bei diesem Übungsblatt ist das leicht möglich, wenn Sie das Resultat von *scanf* nicht korrekt überprüfen. Wenn Sie Fragen dazu haben, können Sie uns gerne kontaktieren.

Im übrigen werden die Lösungen verwendet, um ein generelles Feedback in der nächsten Übungsstunde zu geben.

Wenn Sie im Team arbeiten sollten, können Sie bei dem *submit*-Befehl auch noch eine *team*-Datei hinzufügen, bei der jede Zeile aus einem Loginnamen bei uns besteht. Alle Mitglieder müssen bei SLC für diese Vorlesung registriert sein. Den Testbericht mitsamt der eingereichten Lösung erhalten dann alle Teammitglieder per E-Mail.

Aufgabe 4: „Grundy's Game"

Zu implementieren ist das Spiel Grundy's Game. Bei diesem Spiel treten zwei Spieler gegeneinander an. Zu Beginn liegen n gleiche Gegenstände zusammen auf einem einzigen Haufen. Die Zahl n sollte zu Beginn innerhalb geeigneter Grenzen pseudo-zufällig gewählt werden. Die Spieler wählen abwechselnd einen der Haufen aus und teilen diesen in zwei unterschiedlich große Haufen. Derjenige Spieler, der keinen gültigen Zug mehr machen kann (das ist der Fall, wenn nur noch Haufen der Größe 1 oder 2 übrig sind), verliert das Spiel.

Hier ist ein beispielhafter Spielverlauf:

```
theon$ grundy
*** Grundy's Game ***
One large heap of 22 beans is given. In each move one of the heaps
may be split into two smaller heaps, provided the resulting heaps
are of different size. The player who is unable to split any remaining
heap loses.

Beans are represented by dots, possible splitting points with digits.
If you are asked for a move, specify one of the breaking points.
Do you want to start? Yes=1 No=2 1
.01.02.03.04.05.06.07.08.09.10..12.13.14.15.16.17.18.19.20.21.
Your move: 14
.01.02.03.04.05.06..08.09.10.11.12.13. .15.16.17..19.20.21.
Computer splits a heap at position 1.
. .02.03.04.05.06.07.08.09.10.11.12.13. .15.16.17..19.20.21.
Your move: 3
. .. .04.05.06.07.08.09.10.11.12.13. .15.16.17..19.20.21.
Computer splits a heap at position 4.
. .. . .05.06.07.08..10.11.12.13. .15.16.17..19.20.21.
Your move: 6
. .. . .. .07.08.09..11.12.13. .15.16.17..19.20.21.
Computer splits a heap at position 7.
. .. . .. . .08.09.10.11.12.13. .15.16.17..19.20.21.
Your move: 21
. .. . .. . .08.09.10.11.12.13. .15.16.17.18.19.20. .
Computer splits a heap at position 8.
. .. . .. . . .09.10..12.13. .15.16.17.18.19.20. .
Your move: 10
```

```
. .. . .. . . .. .11..13. .15.16.17.18.19.20. .
Computer splits a heap at position 11.
. .. . .. . . .. . .12.13. .15.16.17.18.19.20. .
Your move: 17
. .. . .. . . .. . .12.13. .15.16. .18..20. .
Computer splits a heap at position 12.
. .. . .. . . .. . . .. .15.16. .18..20. .
Your move: 15
. .. . .. . . .. . . .. . .. .18..20. .
Computer splits a heap at position 18.
. .. . .. . . .. . . .. . .. . .19.20. .
Your move: 19
. .. . .. . . .. . . .. . .. . . .. .
Congratulations, you've won!
theon$
```

Im Rahmen der Aufgabe ist der Spielstand mit Hilfe einer Variable des Typs *unsigned long int* zu realisieren, die uns mindestens 32 Bits zur Verfügung stellt. Die Verwendung von regulären Arrays für die Verwaltung des Spielstands ist ausdrücklich nicht zulässig.[1]

Zu implementieren ist dann ein Spielverlauf, bei dem dann abwechselnd ein Zug vom menschlichen Spieler und dann von Ihrem Programm bestimmt wird. Ihre Lösung muss keine optimalen Züge finden; es genügt vollkommen, wenn die Züge zulässig sind. Sie sind aber herzlich eingeladen, mit Hilfe der Literatur[2] Ihre Lösung zu verbessern.

Sicherzustellen ist aber, dass bei Einlesefehlern (wie etwa einem nicht erfolgreichen *scanf*) der Ablauf terminiert wird. Die eingegebenen Spielzüge sind zu überprüfen und bei unzulässigen Zügen ist eine passende Fehlermeldung auszugeben und die Gelegenheit für eine erneute Zugeingabe zu geben.

Die Lösung kann wiederum auf der Theon eingereicht werden:

```
submit ss1 4 grundy.c
```

Viel Erfolg!

[1] Es ist aber zulässig, ein Array für bereits berechnete Nim-Werte zu verwenden, um schneller zum Sieg führende Züge zu bestimmen. Diese sollten dann aber in der entsprechenden Funktion mit **static** deklariert werden.

[2] Elwyin R. Berlekam, John H. Conway, Richard K. Guy: *Winning Ways for Your Mathematical Plays*. Volume 1, Second Edition, A K Peters, Natick 2001, ISBN 1-56881-130-6, S. 96–97.