OSTBAYERISCHE TECHNISCHE HOCHSCHULE REGENSBURG
INFORMATIK UND MATHEMATIK

Übungsaufgaben Blatt 12 - Verschiedenes

Programmieren 1 - Einführung in die Programmierung mit C - Prof. Dr. Ruben Jubeh

---

Aufgabe 1 : *Rock-Paper-Scissors-Lizard-Spock*

Das Spiel *Rock-Paper-Scissors-Lizard-Spock* ist eine Erweiterung des klassischen Spiels *Schere, Stein, Papier*, bei dem zwei Spieler gegeneinander spielen.

Beide Spieler entscheiden sich zeitgleich für ein Symbol, das sie mit der Hand formen. Jedes der Symbole kann gegen ein anderes Symbol gewinnen oder verlieren. Daher ist erst klar wer gewinnt, nachdem beide Spieler Ihren Spielzug durch Formen des Symbols mit der Hand offengelegt haben. Die folgenden beiden Abbildungen zeigen die Regeln des Spiels, sowie die dazugehörigen Handgesten.

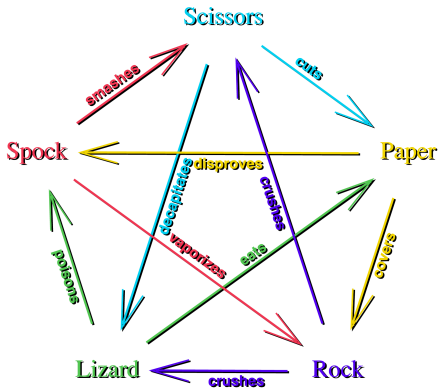


Abbildung 1: Wer gewinnt gegen wen?

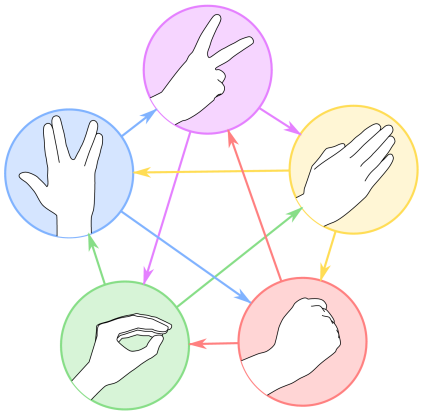


Abbildung 2: Handsymbole für das Spiel (nicht für die Implementierung benötigt)

Schreiben Sie ein Programm, um gegen den Computer "Rock-Paper-Scissors-Lizard-Spock" zu spielen. Die Ausgabe Ihres Programms sollte wie folgt aussehen:

```
1. ./rpslsNaive.o (rpslsNaive.o)
➜  PG1_MA_Uebungsblatt_05_Kontrollstrukturen git:(master) ✗ ./rpslsNaive.o
   Rock Paper Scissors Lizard Spock!
**************************************

Schere schneidet Papier
Papier bedeckt Stein
Stein zerstoert Echse
Echse vergiftet Spock
Spock zertruemmert Schere
Schere koepft Echse
Echse isst Papier
Papier widerlegt Spock
Spock verdampft Stein
Stein zerstoert Schere


0 - Beenden
1 - Stein
2 - Papier
3 - Schere
4 - Echse
5 - Spock
Ihre Wahl: 5
Spieler waehlt Spock
Computer waehlt Schere
Spock zertruemmert Schere - Gewonnen!

0 - Beenden
1 - Stein
2 - Papier
3 - Schere
4 - Echse
5 - Spock
Ihre Wahl:
```

Hinweise:

- Verwenden Sie den `enum`-Datentyp, um die Auswahl der Spieler festzuhalten.
- Erstellen Sie eigene Funktionen mit Rückgabewerten, um Ihren Code zu strukturieren und das Problem in Teilprobleme zu zerlegen.
- Für dieses Spiel gibt es eine naive Lösung, bei der Sie ausgehend von der Wahl des Spielers anhand der Wahl des Computers feststellen können, wer gewonnen hat. Verwenden Sie dazu die Kontrollstruktur *switch case*. **Lösen Sie das Problem erst auf diese Art, bevor Sie sich den zweiten Lösungsansatz ansehen.**
- Neben dieser Lösung gibt es auch eine elegantere Lösung, bei der Sie weniger Code schreiben müssen. Diese Lösung ermittelt durch Subtraktion der Auswahl der Spieler und Modulodivision einen deutlich vereinfachten Programmcode. Sie finden eine genauere Beschreibung dieses Lösungsansatzes auf Wikipedia beschrieben: http://en.wikipedia.org/wiki/Rock-paper-scissors-lizard-Spock

Aufgabe 2 : ***Rock-Paper-Scissors-Lizard-Spock* für kleine Kinder**
Schreiben Sie eine neue Version von "Rock-Paper-Scissors-Lizard-Spock" für kleine Kinder.

Kinder spielen gerne, wollen aber möglichst immer gewinnen. Daher spielt der Computer nicht fair: Er gewinnt zwar ab zu eine Runde, aber im Schnitt gewinnt das spielende Kind öfter als der Computer.

Aufgabe 3 : **Zahlen sprechen (optional)**
Schreiben Sie ein Programm, das Integer Werte in Worte übersetzt. Zum Beispiel wird für die Eingabe 895 der Text "acht neun fünf" ausgegeben.

Verwenden Sie eine `switch`-Anweisung und legen Sie Wert auf eine saubere Struktur.

Aufgabe 4 : **Selection Sort**
Schreiben Sie ein Sortierprogramm, das nach dem Selection-Sort Algorithmus arbeitet:

1. Suche das kleinste Element des Arrays.
2. Vertausche dieses mit dem ersten Element des Arrays.
3. Gehe wieder zu Schritt 1, jetzt aber mit dem verkürzten Array.

Wieso ist dieses Verfahren effizienter als Bubble-Sort?

Aufgabe 5 : **Hailstone-Rekursiv**
Konvertieren Sie ihr Programm aus Übung 4 Aufgabe 2, die Hailstone-Sequenz, in ein rekursives Programm. Das bedeutet die Funktion

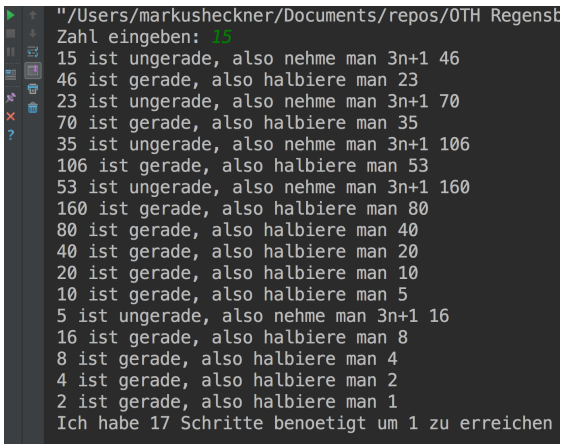
```
1 void hailstone(int num, int steps) {
2     // your *recursive* code goes here
3 }
```

soll ich mit dem nächsten Iterationsschritt als Parameter aufgerufen werden. Eine for-Schleife ist nicht zulässig. Der zweite Parameter `steps` zählt die Rekursionsschritte mit und gibt am Ende die benötigten Schritte aus. Wir nehmen an, dass die Hailstone-Sequenz immer gegen 1 konvergiert, d.h. der Basisfall zum Abbruch ist n bzw. `num==1`.

Zur Wiederholung, die Hailstone-Sequenz kann folgendermaßen beschrieben werden:

> Man nehme eine positive Ganzzahl und nenne sie n.
> Falls n gerade ist, halbiere man sie.
> Ist n ungerade, multipliziere man sie mit drei und zähle eins dazu.
> Dann setze man diesen Prozess solange fort, bis n den Wert
> eins angenommen hat.

Ihr Programm soll dieselbe Ausgabe erzeugen wie in der Übungsaufgabe Blatt 4 bzw. im folgenden Screenshot dargestellt:

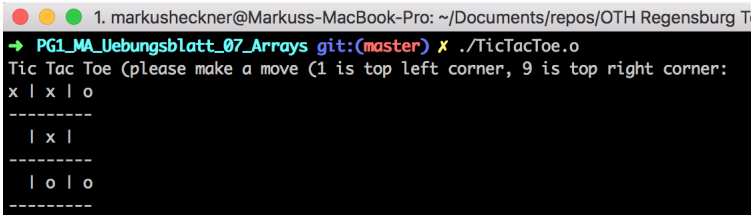


Wie könnten Sie den Parameter `steps` benutzen, um sicher zu stellen, dass die Rekursion keinen Stackoverflow für große Eingabe-Zahlen produziert? Wie reagiert ihr Programm auf negative Eingaben bzw. wie behandeln Sie solche sicher?

Aufgabe 6 : **Tic Tac Toe (optional)**

Bei dem Spiel Tic Tac Toe geht es darum, dass zwei Spieler abwechselnd gegeneinander spielen und versuchen, drei Symbole (Kreis oder Kreuz) in einer Reihe anzuordnen. Implementieren Sie Tic Tac Toe als Zweispielervariante. Fordern Sie die Spieler jeweils abwechselnd zu einem Zug auf und zeichnen Sie abwechselnd das Spielfeld. Die Spielfelder können Sie dabei von 1 - 9 von oben links nach unten rechts durchnummerieren.

Die Ausgabe des Spielfelds könnte z.B. wie folgt aussehen:



Erweitern Sie Ihr Spiel so, dass man gegen den Computer spielen kann. Der Computer sollte sinnvolle Züge machen.

Aufgabe 7 : Präzedenz und Assioziativität

Malen Sie die Operatorbäume so auf, wie ein C-Compiler die folgenden Ausdrücke parsen würde. Wir nehmen an, dass `f(int)` eine Funktion ist, die bereits deklariert ist.

```
a = b * c == 2;
```

```
a = f(x) && a > 100;
```

```
a == b && x != y;
```

Aufgabe 8 : Pointer

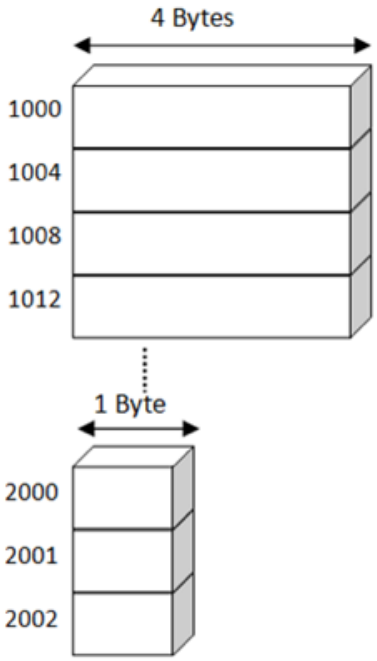
Was macht folgendes Programm?

```c
void set(int *n) {
  *n = 10;
}

int main(void) {
  int *x;

  set(x);
  printf("%d\n", *x);
  return 0;
}
```

Beschriften Sie das folgende Diagramm so, dass die Speicherbelegung veranschaulicht wird. Wir nehmen an, dass ein Integer bzw. ein Pointer 4 Bytes belegen.


4 Bytes
1000
1004
1008
1012
1 Byte
2000
2001
2002