2 C-Syntax I

Aufgabe 2.1. (Binomialkoeffizienten – leicht) Schreiben Sie eine Funktion

```
unsigned long choose(unsigned long n, unsigned long k)
```

die den Binomialkoeffizienten $\binom{n}{k}$ errechnet. Dieser ist definiert Quotient zweier Produkte:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{n(n-1)(n-2)\cdots(n-k+1)}{1 \cdot 2 \cdots k}$$

Schreiben Sie ein Programm, das Ihre Implementierung von `choose` aufruft und vergewissern Sie sich, dass Ihre Implementierung korrekt ist.

Untersuchen Sie, was passiert, wenn Sie größere Werte eingeben. Versuchen Sie, das Verhalten zu erklären. Können Sie Abhilfe schaffen?

Analysieren Sie die Laufzeit Ihrer Implementierung und denken Sie darüber nach, ob es einen besseren Algorithmus zur Berechnung des Binomialkoeffizienten gibt.

Aufgabe 2.2. (Rate die Zahl – leicht) Schreiben Sie ein Programm, dass den Benutzer zum Raten einer Zahl auffordert. Dem Benutzer soll nach jedem Versuch angezeigt werden, ob die geratene Zahl größer, kleiner, oder gleich der vom Programm erdachten Zahl ist. Hat der Benutzer richtig geraten, so soll ihm angezeigt werden, wie viele Versuche er gebraucht hat. Seien Sie kreativ in der Ausgestaltung des Spiels.

Verwenden Sie `rand()` aus `<stdlib.h>`, um eine Zufallszahl zu erzeugen, nachdem sie den Zufallszahlengenerator mit `srand(time(NULL))` aus `<time.h>` initialisiert haben. Verwenden Sie `scanf`, um eine Zahl vom Benutzer zu lesen. Lesen Sie die entsprechenden Seiten aus dem Online-Handbuch, wenn Sie sich in den Details unsicher sind.

Aufgabe 2.3. (Bitfunktionen – leicht) Schreiben Sie C-Funktionen zur Berechnung der folgenden Bit-Operationen. Jede Funktion soll ein Argument vom Typ `unsigned int` entgegen nehmen und das Ergebnis erneut als `unsigned int` ausgeben.

sadd(x) zählt, wie viele Bits in `x` gesetzt sind

ctz(x) zählt, wie oft `x` glatt durch 2 teilbar ist

rev(x) kehrt die Reihenfolge der Bits von `x` um

bswap(x) kehrt die Reihenfolge der Bytes von `x` um

Schreiben Sie ein Programm, das Ihre Implementierungen ausprobiert und vergewissern Sie sich ihrer Korrektheit.

Versuchen Sie, Ihre Implementierung hinsichtlich der Performanz zu optimieren.

Aufgabe 2.4. (Wurzelziehen – mittel) Die Mathematikbibliothek `-lm` enthält die Funktion `sqrt()`, die die Quadratwurzel einer Gleitkommazahl bestimmt.

Implementieren Sie Ihre eigene Quadratwurzelfunktion `my_sqrt()` und vergewissern Sie sich der Korrektheit ihrer Implementierung.

Untersuchen Sie die Genauigkeit Ihrer Implementierung indem Sie sie mit `sqrt()` vergleichen.

Aufgabe 2.5. (Machinen-Epsilon – mittel) Das Maschinen-Epsilon ist die kleinste positive Gleitkommazahl ε, sodass $1.0 \neq 1.0 + \varepsilon$. Idealerweise wäre $\varepsilon = +0$, aber leider haben Gleitkommazahlen nur endliche Genauigkeit.

Finden Sie eine Möglichkeit, das Maschinen-Epsilon zu bestimmen. Bestimmen Sie das Maschinen-Epsilon für die Typen `float`, `double`, und `long double`. Der Platzhalter `%e` ist hilfreich, um derartig kleine Zahlen mit `printf` auszugeben.

Versuchen Sie das Ergebnis zu interpretieren.

Aufgabe 2.6. (Kommentarentferner – schwer) Schreiben Sie ein Programm, welches aus einer C-Quelltextdatei alle Kommentare entfernt. Ihr Programm muss in der Lage sein, sowohl Zeilen-Kommentare der Form `// ...` als auch Bereichs-Kommentare der Form `/* ... */` zu erkennen, und zu entfernen. Dazu müssen Sie Fortsetzungszeilen erkennen und Kommentare innerhalb von Zeichenketten geschickt ignorieren. Lesen Sie ggf. den C-Standard, um die genauen Feinheiten der Syntax zu verstehen.