

1 C-Syntax I

Aufgabe 1.1. (Bitfunktionen – leicht) Schreiben Sie C-Funktionen zur Berechnung der folgenden Bit-Operationen. Jede Funktion soll ein Argument vom Typ `unsigned int` entgegennehmen 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 1.2. (Zahlenkonvertierung – leicht) Die Bibliotheksfunktion `atoi(3)` konvertiert eine Zeichenkette in eine Ganzzahl. Implementieren Sie Ihre eigene Version dieser Funktion. Implementieren Sie ein Gegenstück `itoa()`, welches eine Zahl in eine Zeichenkette konvertiert und prüfen Sie beide Funktionen gegeneinander.

Aufgabe 1.3. (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 1.4. (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 1.5. (Maschinen-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 1.6. (hd(1) – mittel) Schreiben Sie einen Klon des Programms **hd(1)**, welches den Inhalt von Dateien in Binär angibt. Experimentieren Sie dazu mit **hd(1)**, um das Ausgabeformat des Programms zu verstehen. Dieses besteht aus drei Teilen: ganz links die Adresse der aktuellen Zeile, dann 16 Bytes in hexadezimal und ganz rechts die selben Daten in ASCII, wobei nicht druckbare Zeichen (vgl. **isprint(3)**) durch einen Punkt wiedergegeben werden.

Aufgabe 1.7. (Datum aus Systemzeit – schwer) Die Funktion **time(2)** (je nach System ggf. **time(3)**) liefert die aktuelle Zeit in Sekunden seit dem 1. Januar 1970, 00:00 UTC zurück.

Schreiben Sie ein Programm, das aus der Rückgabe von **time(2)** Zeit, Datum, Wochentag, und Uhrzeit bestimmt. Alternativ soll es dem Nutzer möglich sein, eine zu konvertierende Sekundenzahl per Argument an das Programm zu übergeben.

Denken Sie sich selbst einen Algorithmus aus oder recherchieren Sie im Internet. Sie können die Ausgabe Ihres Codes anschließend mit dem von **ctime(3)** bestimmten String vergleichen. Beachten Sie dabei, dass der Ausgabe von **ctime(3)** abhängig von der aktuellen Zeitzone ist.

Aufgabe 1.8. (Kommentarentferner – sehr 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 nachzuvollziehen.