Wählen Sie aus den folgenden Aufgaben zur Bearbeitung drei Aufgaben aus.

3 Funktionen und Typen

Aufgabe 3.1. (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 3.2. (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. Benutzen Sie zur Implementierung beider Funktionen keine Funktionen der Standardbibliothek wie **sprintf**(3) oder **sscanf**(3).

Aufgabe 3.3. (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 3.4. (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, ohne die Mathebibliothek selber zu benutzen.

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

Aufgabe 3.5. (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 3.6. (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 3.7. (Stringsuche – mittel) Schreiben Sie Ihre eigene Variante der Funktion `strstr`(3). Schreiben Sie ein Programm, um diese zu testen. Hierzu bietet es sich an, dass Ihre eigene Implementierung mit der von System bereitgestellten Implementierung zu vergleich.

Analysieren Sie die Laufzeit Ihrer Implementierung und recherchieren Sie bessere Algorithmen zur Suche von Strings in Strings.

Aufgabe 3.8. (`env`(1) – mittel) Schreiben Sie eine Implementierung des Programmes `env`(1). Dieses hat zwei Aufgaben:

Ohne Argumente aufgerufen, gibt `env`(1) den Inhalt der zur Zeit gesetzten Umgebungsvariablen aus. Diese können Sie durch Inspektion der globalen Variable `environ`(7) auslesen.

Werden Argumente übergeben, so wird jedes Argument der Form

`Schluessel=Wert`

mit `putenv`(3) als Variable in die Umgebung aufgenommen. Alle Argumente ab dem ersten Argument, das nicht diese Form hat, werden als Name und Argumente eines auszuführenden Programmes verstanden, welches nach Setzen der Umgebungsvariablen mit `execvp`(2) ausgeführt wird.

Weitere Funktionalität von `env`(1) ist nicht Teil der Aufgabe.

Aufgabe 3.9. (Stack als Array – mittel) Ein Stack ist eine Datenstruktur auf der man auf das oberste Element zugreifen kann: Man kann es löschen (POP), angucken (PEEK) oder ein anderes Element „darüber legen“ (PUSH).

Implementiert diese Funktionalität mithilfe einer Struktur in welcher die Daten in einem Array gespeichert werden. Als Idee: Die Struktur benötigt mindestens Einträge um 1. die aktuelle Position des Stacks zu speichern und 2. zu wissen, ob der Stack leer oder voll ist. Diese wird dann als Pointer an die Funktionen übergeben, samt möglicherweise weiteren benötigten Argumenten.

Aufgabe 3.10. (Termauswerter – schwer) Schreiben Sie ein Programm, in das Sie einen C-Ausdruck mit Variablen eingeben können. Das Programm soll den Nutzer nach Belegungen für die Variablen fragen und den Wert des Terms ermitteln. Verwenden Sie für alle Variablen den Typ `long long int`. Sie können zur Lösung der Aufgabe den *Shunting-Yard-Algorithmus* verwenden; dafür bietet es sich an, zunächst einen Stack als Datenstruktur zu implementieren (siehe andere Aufgabe).

Es reicht, wenn Sie zunächst nur Variablen, deren Namen nur ein Buchstabe ist unterstützen. Sie können sich den Inhalt der Variablen in einem Array merken.

Aufgabe 3.11. (base64 – schwer) Schreiben Sie zwei Programme, die Dateien im Base64-Format kodieren bzw. aus dem Base64-Format dekodieren. Recherchieren Sie dazu wie das Format Base64 funktioniert. Sie können die Korrektheit Ihrer Implementierung gegen das Linux-Programm `base64` testen.

Aufgabe 3.12. (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.