Januar 2024

Praktikum zur Lehrveranstaltung Mikrorechentechnik I
Versuch 3, C-Programmierung

Organisation des Praktikumsversuchs: M. Herhold

Inhaltsverzeichnis

1 Ziele dieses Praktikumsversuchs

- Festigung des Vorlesungsstoffes zur C-Programmierung
  - durch Benutzung von Arrays und Speicherbereichen in C
  - durch den Austausch von Daten innerhalb eines C-Programms mittels Zeiger und Strukturen
  - durch Aufteilen von Funktionalität eines C-Programms auf mehrere C-Module
- Eigenständiges Entwerfen und Implementieren eines Programms in der Programmiersprache C.
- Schulung der Teamfähigkeit.

2 Wichtiger Hinweis

Der Zeitaufwand der Aufgabenstellung ist für Studenten, welche keine Programmiererfahrung in C haben umfangreich, da viele Programmierkonzepte und Herangehensweisen erst erlernt werden müssen. Fangen Sie daher rechtzeitig an die Aufgabe zu bearbeiten und arbeiten Sie als Gruppe.

3 Versuchsaufgabe

Programmieren Sie unter Zuhilfenahme der Programmiersprache C ein Computerprogramm, welches eine grafische Animation, nach vorgegebenen Regeln generiert und mit Hilfe der Bibliothek `SDL2` auf dem Bildschirm ausgibt. Benutzen Sie dazu die Funktionen des vorgegebenen Quelltextes und ergänzen Sie diesen Quelltext um die notwendige Funktionalität. Arbeiten Sie dazu im Team und verteilen Sie selbstständig Teilaufgaben unter den einzelnen Gruppenmitgliedern.

3.1 Animationsbeschreibung

Bei der Animationsfläche handelt es sich um ein Gitter mit `x` Pixeln horizontal und `y` Pixeln vertikal. Die Pixel-Koordinate `(x=0, y=0)` befindet sich links oben. Jedes Pixel stellt eine Einheit dar und kann die Werte `frei` oder `belegt` annehmen.

Der Folgezustand eines Pixels (zum Zeitpunkt t_{n+1}) ergibt sich aus den derzeitigen Zuständen (Zeitpunkt t_n) seiner 8 Nachbarpixel.

? ? ?
? ! ?
? ? ?

Zur Berechnung eines Animationsschrittes sind eine Reihe von Regeln – beschrieben in Abschnitt 4.3 – für jeden einzelnen Pixel anzuwenden.

4 Teilaufgaben

Folgende Teilaufgaben sind zu bearbeiten um die Praktikumsaufgabe zu lösen.

4.1 Datenstruktur- und Funktionsprototypen definieren

Dies ist eine Teilaufgabe, welche zuallererst in Angriff genommen werden und an der alle Teammitglieder beteiligt werden sollten.

Diese Teilaufgabe soll als Ergebnis eine geeignete Datenstruktur zum Austausch wichtiger Laufzeitdaten zwischen den einzelnen C-Modulen, sowie erste Schnittstellen-Prototypen der wichtigsten Funktionen der einzelnen C-Module, haben.

Folgende Daten sollen in der Datenstruktur `Laufzeit-Daten` auf jeden Fall enthalten sein.:

- die Anzahl der Pixel der Animationsfläche in Richtung der X-Achse (`Spalten`)
- die Anzahl der Pixel der Animationsfläche in Richtung der Y-Achse (`Zeilen`)
- ein Zähler, welcher den derzeitigen Animationsschritt speichert (`Schritt`)
- die maximale Anzahl an Animationsschritten die berechnet und angezeigt werden sollen (`Epochen`)
- ein Wert, welcher die Pause zwischen zwei Animationsschritten in Millisekunden angibt (`Verzögerung`)
- ein ein- oder zwei-dimensionales C-Array, im Folgenden als `Animations-Puffer` bezeichnet, welches den letzten berechneten Animationsschritt speichert. Dieses Array benötigt Platz für $Zeilen * Spalten$ Elemente.

  Wie groß der Speicher des `Animations-Puffers` sein muss, wissen Sie erst, wenn Sie einen Teil der Konfigurationsdatei eingelesen haben. Deshalb ist es hier sinnvoll mit einem Pointer auf den (anfangs unbekannten) Speicherbereich zu arbeiten.

Je nach Aufbau ihres Programms müssen Sie weitere Daten, z.B. einen Pointer zu einem zweiten `Animations-Puffer` für den Folgezustand, in der Struktur ablegen.

Hinweis: Die Schnittstellen der einzelnen Funktionen, sollten mit einem Zeiger auf eine Instanz dieser Datenstruktur arbeiten um unnötige Speicherkopieroperationen zu vermeiden. (Stichwort: call by reference)

4.2 Einlesen einer Konfigurationsdatei

Diese Teilaufgabe benötigt mittleren programmiertechnischen Aufwand, ist recht zeitaufwändig (benötigt viele Funktionstests), fordert Wissen zu Speicherallokation und C-Strings und sollte von von einem oder zwei Teammitgliedern bearbeitet werden.

Die Funktionalität der Teilaufgabe soll im C-Modul `config.c` implementiert werden.

Zweck der Teilaufgabe ist das Befüllen der zuvor ausgearbeiteten Datenstruktur `Laufzeit-Daten` mit, aus einer Datei eingelesenen, Werten.

Eine Beispielkonfigurationsdatei – `settings-1.txt` – befindet sich im Projektverzeichnis. Implementieren Sie die Funktionalität anhand dieser Datei. Während des Praktikumsversuchs, soll Ihr Programm in der Lage sein, diese und andere Dateien einzulesen. Die Datei – `settings-2.txt` – ist eine andere mögliche Beispielkonfigurationsdatei.

Zeigen Sie Fehler beim Einlesen der Datei, über den Standardfehlerausgabe-Stream (`stderr`), dem Nutzer an. Nutzen Sie perror und fprintf, wenn die Datei nicht geöffnet werden konnte, oder ein Parameter nicht im geforderten Datentyp vorliegt.

Im Folgenden wird der Aufbau einer Konfigurationsdatei erläutert:

```
Spalten:
 <Integer-Zahl>
Zeilen:
 <Integer-Zahl>
Epochen:
 <Integer-Zahl>
Verzögerung:
 <Integer-Zahl>
Initialisierung:
 <Lauflängenkodierung, siehe Beschreibung>
```

Schlüsselwörter in der Konfigurationsdatei erkennen Sie am Buchstaben `:` am Ende der Zeile. Schlüsselwörter stehen so wie oben aufgeführt in der Konfigurationsdatei. Schlüsselwörter markieren Eigenschaften im Speicher ihres Programms und nehmen je nach Konfigurationsdatei unterschiedliche Werte an. Auf jedes Schlüsselwort, folgt ein Zeilenumbruch. Auf der neuen Zeile findet sich der Wert der dieser Eigenschaft zugeordnet ist. Der Platzhalter `<Integer-Zahl>` steht hier für eine Ganzzahl, welche an dieser Stelle aus der Konfigurationsdatei gelesen werden soll.

Nachdem Sie diese Werte kennen, können Sie den Speicher für den `Animations-Puffer` vom Betriebssystem anfordern. Die Funktionen `malloc()` und `calloc()` der C-Standardbibliothek helfen ihnen dabei.

Das Füllen des initialen Zustands des Animations-Buffers erfolgt mit den Daten welche Sie aus der Konfigurationsdatei nach dem Schlüsselwort `Initialisierung:` herauslesen.
Bei der Art und Weise, wie die Daten gespeichert sind, handelt es sich um eine einfache Lauflängenkodierung.
Jede Zeile unter dem Schlüsselwort `Initialisierung:` in der Datei definiert eine Zeile im `Animations-Puffer`. Aufeinanderfolgende Zeilen in der Datei definieren aufeinanderfolgende Zeilen im `Animations-Puffer`.

Begonnen wird eine Zeile in der Datei mit dem Buchstaben `e` oder `f`. Wobei `e` für `empty` steht, und somit ein `freies` Pixel bedeutet. `f` steht für `full` und bedeutet, dass das Pixel `belegt` ist. Anschließend folgt immer ein Trennzeichen (`-`).

Das dritte Zeichen ist immer Beginn einer Ganzzahl und beschreibt, wie oft der zuvor definierte Zustand (`frei` oder `belegt`) auftritt, bevor eine Änderung des Pixelwerts (`(frei -> belegt)` oder `(belegt -> frei)`) erfolgt. Es folgen anschließend, bis zum Zeilenumbruch, nur noch Ganzzahl-Angaben. Jede Ganzzahlangabe ist durch das Trennzeichen – von der nächsten Angabe getrennt. Jede Ganzzahl spezifiziert wie viele aufeinanderfolgende Pixel den gleichen Zustand besitzen, bevor eine Pixelzustandsänderung eintritt.

Beispiele:

- Die Zeile `f-2-1-2` bewirkt, dass eine Zeile in Ihrem Animationspuffer die Werte `xx.xx` annimmt.
- Die Zeile `e-2-11-3-2` bewirkt dieses Muster `..xxxxxxxxxx...xx` in Ihrem Animationspuffer.

Folgendes ist beim Auslesen der Datei außerdem zu beachten:

- Die Reihenfolge der Schlüsselwörter kann variieren. Sie müssen also eine Zeile identifizieren, bevor Sie wissen, welche Einstellung gerade eingelesen wird.
- Wenn ein Schlüsselwort mehrfach, aber mit unterschiedlichen Parameterwert vorkommt, dann ist nur das letzte Vorkommen der Zeile in der Datei relevant (also Werte weiter unten in der Datei überschreiben vorherige Werte)
- Wenn Daten der `Initialisierung` den `Animations-Puffer` nicht ausfüllen, dann sind die ausgelesenen Daten mittig im `Animations-Puffer` zu platzieren. Nicht in der Datei beschriebene Pixel sind mit dem Wert `belegt`[1] zu initialisieren.

[1] Dies dient zur Behebung eines Fehlers, welcher in der vorherigen Version der Anleitung auftrat, ohne das komplette Regelwerk in der Anleitung ändern zu müssen.

- Die Datei kann leere Zeilen enthalten, diese sind beim Einlesen zu ignorieren.
- Die Daten der `Initialisierung`, welche den anfänglichen Inhalt des `Animations-Puffer` definieren, kommen immer am Ende der Datei.
- Die Größe der Daten der `Initialisierung` kann maximal Zeilen x Spalten Zeichen betragen.

Folgende Funktionen aus der C-Standard-Bibliothek[2] sind für die Lösung der Teilaufgabe hilfreich: `fopen()`, `fclose()`, `getline()`, `fscanf()`, `sscanf()`, `strncmp()`, `atoi()`, `atof()`, `malloc()`, `calloc()`, `memset()` und `perror()`.

4.3 Berechnen des nächsten Animationsschrittes

Diese Teilaufgabe hat mittleren Schwierigkeitsgrad und kann von einem einzelnen Teammitglied bearbeitet werden.

Die Funktionalität dieser Aufgabe soll im C-Modul `engine.c` implementiert werden.

Ziel der Teilaufgabe ist die pixelweise Berechnung und Speicherung des nachfolgenden Animationsschrittes (t_{n+1}).

Zur korrekten Berechnung des jeweils nächsten Animationsschrittes (t_{n+1}) muss der aktuelle Zustand (t_n) betrachtet und folgende Regeln eingehalten werden:

1. Ein `freier` Pixel mit weniger als 5 `belegten` Nachbarpixeln wird zu einem `belegten` Pixel.

   Beispiele: `. . .` `. O .` `x x x` $t_n \Rightarrow t_{n+1}$ `. . .` `. X .` `x x x` oder `. . x` `. O .` `x . .` $t_n \Rightarrow t_{n+1}$ `. . x` `. X .` `x . .`

2. Ein `belegter` Pixel mit 0 bis 4 `belegten` Nachbarpixeln behält seinen Zustand.

   Beispiele: `. x x` `. X .` `x x .` $t_n \Rightarrow t_{n+1}$ `. x x` `. X .` `x x .` oder `. . .` `. X .` `. . .` $t_n \Rightarrow t_{n+1}$ `. . .` `. X .` `. . .`

3. ein Pixel mit exakt 5 `belegten` Nachbarpixeln wird zum `freien` Pixel.

   Beispiele: `x . x` `x O .` `. x x` $t_n \Rightarrow t_{n+1}$ `x . x` `x O .` `. x x` oder `x . x` `. X x` `x . x` $t_n \Rightarrow t_{n+1}$ `x . x` `. O x` `x . x`

4. Ein Pixel mit exakt 6 `belegten` Nachbarpixeln behält seinen Zustand.

   Beispiele: `x . x` `x O x` `x . x` $t_n \Rightarrow t_{n+1}$ `x . x` `x O x` `x . x` oder `x x x` `. X x` `x x .` $t_n \Rightarrow t_{n+1}$ `x x x` `. X x` `x x .`

5. ein `freier` oder `belegter` Pixel mit mehr als 6 `belegten` Nachbarpixeln wird ein `belegter` Pixel

   Beispiele: `x x x` `x O x` `x . x` $t_n \Rightarrow t_{n+1}$ `x x x` `x X x` `x . x` oder `x x x` `x X x` `x x x` $t_n \Rightarrow t_{n+1}$ `x x x` `x X x` `x x x`

6. Randpixel, welche an den beiden Rändern oben und unten, außerhalb der dargestellten Animationsfläche liegen, werden als `belegte` Pixel behandelt.

   Nachbarpixel des rechten Bildrands sind die Pixel des linken Rands. Nachbarpixel des linken Bildrands sind die Pixel des rechten Rands. Stellen Sie sich die Animationsfläche als einen aufrecht stehenden Zylinder vor.

In den obigen Beispielen:

- . und O bedeutet `freier` Pixel.
- x und X bedeutet `belegter` Pixel.
- Es wird nur der Zustand des rot markierten Pixel in Bezug auf die Berechnungsvorschrift betrachtet.

Tips zur eigenständigen Kontrolle auf Korrektheit Ihres implementierten Algorithmus:

- Die Berechnung ist korrekt, wenn bei der vorgegebenen Konfiguration, nach 32 [3]Animationsschritten, das Muster von Animationsschritt 0, wieder im `Animations-Puffer` vorliegt.

[2] Dokumentationen sind durch Befehle wie z.B.: `man getline` in der Kommandozeile aufrufbar. Alle `Manpages` haben die gleich Grundstruktur.
[3] um ein Pixel nach rechts verschoben

- Testen Sie auch das Verhalten mit größeren x-y-Dimensionen und einer, mit Zufallswerten initialisierten Animationsfläche (wiederkehrende Muster ergeben sich nur selten).

4.4 Grafische Darstellung eines Animationsbildes

Diese Teilaufgabe ist einfach und kann von einem Teammitglied bearbeitet werden.

Die Funktionalität soll im C-Modul `gfx.c` implementiert werden.

Bei diesem Teil soll der aktuelle `Animations-Puffer` mit Hilfe der zur Verfügung gestellten Grafik-Funktionen aus dem C-Modul `graphic.c` in das Grafikausgabefenster gezeichnet werden.

Außerdem muss die Leinwand vor dem ersten Zeichnen in der korrekten Größe initialisiert werden.

Die aus dem Modul `graphic.c` anzuwendenden Funktionen heißen:
`grafik_create_paint_area()`, `grafik_paint_point()`, `grafik_lock_for_painting()` und `grafik_unlock_and_show()`.

4.5 Behandlung von Nutzereingaben

Diese Teilaufgabe ist einfach und kann von einem Teammitglied bearbeitet werden.

Die Funktionalität dieser Teilaufgabe soll im C-Modul `ui.c` implementiert werden.

Auf Tastatureingaben vom Nutzer soll das Programm folgendermaßen reagieren:

- Taste `q` : Programm beenden
- `Leertaste` : Animation pausieren, oder falls pausiert, weiterführen
- Taste `.` (Punkt) : den nächsten Animationsschritt anzeigen und dann die Animation pausieren
- Weitere Tastatureingaben dürfen ignoriert werden.

Bei dieser Teilaufgabe bietet es sich an, die benötigte Pause zwischen den einzelnen Animationszyklen einzubauen. Die Funktion `grafik_user_input` nimmt eine Wartezeit in Millisekunden als Parameter entgegen.

Hinweis: falls dieser Programmteil über einen längeren Zeitraum verhindert, dass die Grafikausgabe-Funktionen aufgerufen werden, könnte die Grafik im Ausgabefenster zerstört werden. Dies ist sehr gut während der `farb_demonstration` zu erkennen, wenn Sie das zusätzliche Informations-Dialogfenster mit der Maus verschieben.

4.6 Programm-Hauptschleife

Diese Teilaufgabe ist einfach und kann von einem Teammitglied bearbeitet werden.

Die Programm-Hauptschleife (manchmal auch als `main event loop` bezeichnet) soll im C-Modul `main.c` implementiert werden.

Die Hauptschleife führt die oben beschriebenen Teilaufgaben zu einem funktionierenden Programm zusammen und ruft dabei die Funktionen der einzelnen C-Module in geeigneter Reihenfolge und Anzahl auf. Dies soll so lange geschehen, bis der Nutzer das Programm mit der Taste `q` beendet.

5 Hinweise zur Entwicklungsumgebung (Toolchain)

Die Lösung der Praktikumsaufgabe und der vorgegebene Quelltext wurden auf der `MRT-C-VM` (kurz: `C-VM`) entwickelt und getestet.

Die `C-VM` kann in die Virtualisierungssoftware Oracle VirtualBox importiert und anschließend darüber gestartet werden. Apple Nutzer mit ARM-Prozessor können die `C-VM` mittels der App `UTM` oder direkt mit dem Emulator `QUEMU` nutzen (siehe auch FAQ zur VM).

Das Kompilieren und die Präsentation Ihres Programms, während des Versuchstermins, erfolgt in der `C-VM` auf Praktikums-PCs.

Es ist möglich, die Aufgabe mit einer eingerichteten Toolchain unter Windows, Mac, oder Linux zu programmieren. Stellen Sie jedoch sicher, dass Ihr Programm für die Präsentation und Abgabe zum Praktikumstermin in der `C-VM` kompiliert und funktioniert! C-Quelltext, entwickelt unter Windows oder MacOS, lässt sich oft nur nach Quelltext-Anpassungen unter Linux kompilieren (und umgekehrt).

Hilfestellung kann aus Personal- und Hardwaregründen seitens des Praktikumsbetreuers nur für die `C-VM` und Debian-basierte Linux Systeme gegeben werden.

Es ist ihnen freigestellt, welche IDE oder Editoren[4] Sie verwenden möchten. Die IDE `Code::Blocks` ist in der `MRT-VM` bereits vorinstalliert. Der in der WS2023/24 Vorlesung genutzte Editor `VS Code` ist aufgrund Telemetrie-Daten-Sammlung und Microsofts EEE Strategie eine schlechte Wahl und ist nicht in der `C-VM` vorinstalliert. Eine Installationsanleitung für `VS Codium` in der `C-VM` finden Sie in der FAQ zur `C-VM`. `VS Code` / `VS Codium` wird auf den Praktikums-PCs nicht verfügbar sein.

Zum Kompilieren des Projekts, unabhängig von einer IDE oder Editor, können Sie das vorbereitete `Makefile` nutzen und verändern.

Wie Sie die `C-VM` auf Ihrem PC in Betrieb nehmen und Daten in die `C-VM` übertragen können, erfahren Sie aus der FAQ zur VM. Die Downloadlinks für `C-VM` und die `VM-FAQ` erhalten Sie auf der Webseite des C-Versuchs.

- Übertragen Sie den vorbereiteten Quellcode der Praktikumsaufgabe in die `C-VM`. Speichern Sie den Quellcode zum Bearbeiten in der `C-VM`. Arbeiten Sie bitte nicht im (optional) eingerichteten `Shared-Folder`.
- Der Quellcode kann direkt in der IDE `Code::Blocks` geladen, kompiliert und gestartet werden.
- Der vorgegebenen Quelltext kann auch ohne IDE kompiliert werden.
- Hinweise zur Arbeit mit dem Makefile (also ohne IDE):
  - Mittels folgendem Kommando von der Kommandozeile kann der Quellcode kompiliert werden:

    ```
    make
    ```

  - Das so entstandene Programm können Sie mittels der Eingabe

    ```
    ./animation
    ```

    von der Kommandozeile starten.
  - Zum Aufräumen des Verzeichnisses führen Sie folgendes Kommando aus:

    ```
    make clean
    ```

    Alle generierten Dateien `*.o`, `*.gch` und die ausführbare Datei werden dann gelöscht.

6 Hinweise zur Praktikumsdurchführung

- Durchführungsort ist im WS 2023/2024 der Raum BAR E57.
- Am Anfang des Praktikums wird ein kurzer (10-20 Minuten) schriftlicher Eingangstest durchgeführt. Abgefragt wird für den Versuch benötigtes Grundlagenwissen. Wenn Sie aktiv an der Lösung aller Teilaufgaben des Praktikum beteiligt waren, sollten Sie den Test bestehen können. Das Bestehen des Tests ist zum Bestehen des Versuchs Voraussetzung.
- Ihre Gruppe bringt zum Praktikumstermin ein funktionierendes[5] Programm mit, welches die Praktikumsaufgabe löst und stellt es dem Betreuer vor. Das Programm muss unter Linux in der `C-VM` laufen. Das Programm muss am Ende des Termins fehlerfrei funktionieren und auch andere Konfigurationsdateien verarbeiten können, damit Sie den Durchführungsteil bestehen können.
- Nach erfolgreicher Präsentation dürfen Sie den Praktikumstermin vorzeitig beenden.

[4] Anleitung & Quellcode erstellt mit GNU Emacs & Org-mode & Magit.
[5] Ein "Programm funktioniert, aber compiliert noch nicht"führt erfahrungsgemäß nur selten zum Bestehen des Versuchs.

- Es können zum Praktikumstermin Probleme zusammen mit dem Betreuer gelöst und Fragen an den Betreuer gestellt werden. Es ist jedoch aufgrund der knappen Zeit und der begrenzten IT–Ressourcen/Räumlichkeiten nicht realistisch die Aufgabe ohne ausreichend Vorarbeit seitens der Gruppe vollständig zu lösen.
- Stellen Sie Fragen bei Problemen vor Ihrem Versuchstermin per E-Mail an den Praktikumsversuchsbetreuer. Sinnvolle Fragen im Rahmen Ihrer Versuchsvorbereitung hat keinen negativen Einfluss auf Ihre Versuchsbewertung.

7 Hinweise zur Programmdokumentation

Zwei Wochen nach dem Praktikumstermin ist eine Programmdokumentation als PDF per E-Mail beim Betreuer abzugeben.

Das Dokument ist in Art und Weise einer wissenschaftlichen Arbeit zu schreiben. Sie werden dies in Ihrem Studium noch oft tun müssen. Dies ist eine Gelegenheit zum üben und mit verfügbaren Werkzeugen zu experimentieren.

7.1 Umfang

Vier bis Acht A4-Seiten und zusätzlich vorgegebenes Deckblatt.

7.2 fakultativer Inhalt des Dokuments

- Einleitung oder Einführung oder Aufgabenstellung
- Nennung/Beschreibung/Begründung verwendeter Konzepte/Besonderheiten zur Aufgabenlösung (z.B. Arten der: Datenspeicherung/Verwaltung/Datenübergabe an andere Programmteile, verwendete Algorithmen)
- Programmablauf in sinnvollem Detailgrad und geeigneter Präsentationsweise.
- Quellcode nur als kurze Ausschnitte, entsprechend hervorgehoben und mit Erläuterungen
- Dokumentation von Quellcode-Abschnitten: was und warum er so arbeitet, aber nicht wie er eine Aufgabe löst, dies dokumentiert bereits der Quelltext.
- Quellenangaben für fremden Quellcode oder Text (sofern notwendig)
- Analyse der aufgetreten Probleme beim Bearbeiten der Aufgabe
- Verbesserungsvorschläge (welche Teile des eigenen Programm-Quellcode ändern und warum)

7.3 Was nicht in die Dokumentation gehört:

- Ich-Form
- kompletter Quellcode des Programms
- Im Detail erklären wie einzelne Funktionen Ihres Programms arbeiten. Gehen Sie davon aus, dass der Quelltext Ihres Programms bereits digital vorliegt und vom Leser verstanden wird.

8 Hinweise zur Versuchsaufgabe:

Schauen Sie sich die Funktion `farb_demonstration` genau an, darin sind einige Lösungshinweise enthalten.

Die Versuchsaufgabe ist weniger komplex als es diese Aufgabenstellung vermuten lässt. Diese Aufgabe legt in Umfang und Struktur, besonderes Augenmerk auf das Aufteilen der notwendigen Arbeiten, damit Sie ihre Teamfähigkeit trainieren.

Das Erlernen der Programmiersprache C benötigt aktives üben und damit etwas Zeit.

In diesem praktischen Teil der Lehrveranstaltung geht es um Lernen durch Übung. Programmieren Sie daher aktiv einen Teil der Aufgabe selbst.

Sämtliche Personenbezeichnungen in diesem Dokument wurden zum Zweck der Verständlichkeit im geschlechterneutralen Sinn (generisches Maskulin) verwendet.[6]

Bei Fragen – organisatorischer oder inhaltlicher Natur, und auch bei Problemen mit der Aufgabe oder Fehlern in der Aufgabenstellung – können Sie sich per E-Mail an den Betreuer des Praktikumsversuchs Mario Herhold wenden. Die E-Mailadresse finden sie auf den Webseiten der TU Dresden. Fragen Anmerkungen und konstruktive Kritik führen nicht zur Abwertungen bei der Bewertung des Versuchs. Wichtige oder Wiederkehrende Fragen werden in das entsprechende FAQ-Dokument aufgenommen.

[6] Die derzeit propagierte Handhabung ist ungünstig, denn sie macht die deutsche Sprache ungenauer und schwerer verständlich (Was versteht man unter "Flüchtender"? Ein vor der Polizei flüchtender Verbrecher, oder ein sich legal im fremden Land aufhaltender Flüchtling?), ist uneinheitlich (Wieso wird die Bezeichnung "Falschfahrer"weiterhin verwendet? Wie lautet die "korrekte"Bezeichnung für "Hexe"?) und diskriminiert umso deutlicher bestimmte Personengruppen.

9 Arbeits- und Brandschutzhinweise

9.1 Vorbeugende Maßnahmen:

- Die Praktikumsteilnehmer haben sich so zu verhalten, dass Gefahrensituationen und Unfälle vermieden werden.
- Die Befugnis zum Bedienen und Nutzen von Geräten ist auf den zugewiesenen Praktikumsplatz beschränkt.
- Eingriffe in die zum Praktikumsaufbau gehörenden Geräte sind nicht erlaubt.
- Der Anschluss und der Betrieb privater Geräte in den Praktikumsräumen ist verboten.
- Defekte an Geräten oder Gebäudeeinrichtungen sind unverzüglich dem Betreuer mitzuteilen. Betroffene Geräte sind außer Betrieb zu nehmen. Andere Personen sind vor Gefahren zu warnen.
- Den Anweisungen der Praktikumsbetreuer bzw. anderer aufsichtsführender Personen ist unbedingt Folge zu leisten.
- Betriebsfremde dürfen sich nur mit Erlaubnis des Praktikumsbetreuers in den Praktikumsräumen aufhalten.
- Rauchen und Umgang mit offenem Feuer ist nicht gestattet.
- Nach Ende des Praktikums ist der Arbeitsplatz in sauberem und aufgeräumtem Zustand zu hinterlassen.
- Außergewöhnliche Ereignisse bzw. besondere Vorkommnisse sind umgehend dem Betreuer oder dem diensthabenden Assistenten zu melden.

9.2 Verhalten im Falle eines Brandes:

- Beachten der richtigen Reihenfolge: **MELDEN - RETTEN - LÖSCHEN**

9.2.1 Feuer melden:

- Telefonische Brandmeldung:
  - Notruf 112 der Feuerwehr (von jedem Telefon aus möglich)
  - Notruf HA 34515 der Technischen Leitzentrale der TUD
- Deutliche, genaue und vollständige Angaben:
- Wo brennt es?
- Was brennt?
- Angaben zu verletzten oder gefährdeten Personen
- Wer meldet?

9.2.2 Personen retten:

- Erste Hilfe leisten
- Weitere Hilfe organisieren, medizinische Hilfe anfordern
- Gefahrenbereich räumen; Fluchtwege benutzen, keine Aufzüge
- Andere Personen warnen, Sammelplatz (Platz vor Turmeingang zum Barkhausenbau) aufsuchen
- Behinderten und älteren Personen helfen

9.2.3 Löschversuch unternehmen

- Feuerlöscher verwenden (Standorte: Gänge des Barkhausenbaues), dabei sich nicht selbst gefährden
- Fenster und Türen schließen, aber nicht abschließen
- Möglichst elektrische Verbraucher abschalten

9.3 Rufnummern für Notfälle:

Helfer	Telefonnummer
Rettungsdienst	112
Polizei	110
TUD-Notruf	34515
Betriebsärztlicher Dienst	36199
Klinikum Friedrichstadt Notaufnahme	0351-480 1938