

PR3 - Übungsblatt A.1

(Stand 2025-10-09 12:23)

Prof. Dr. Holger Peine
Hochschule Hannover
Fakultät IV – Abteilung Informatik
Raum 1H.2.60, Tel. 0511-9296-1830
Holger.Peine@hs-hannover.de

Thema

Übersetzungs vorgang, Programmstruktur

Termin

Ihre Arbeitsergebnisse zu diesem Übungsblatt führen Sie bitte in den Übungen bis zum 09.10.2025 vor.

Hinweise vor dem Start

Aufgaben mit Punkten sollten Sie bevorzugt bearbeiten, da sie sich mit zentralen Themenstellungen auseinandersetzen. Weitere Übungsaufgaben (0 Punkte) dienen der Vertiefung spezieller Teilbereiche. Es gilt: die Inhalte aller Pflicht-Übungsaufgaben und aller Vorlesungen sind Gegenstand der Prüfung.

Die Funktionen `printf` und `scanf` zur Konsolen-Ein/Ausgabe werden wir in einem späteren Kapitel der Vorlesung genauer unter die Lupe nehmen, weil dazu Wissen über Zeiger und Strings erforderlich ist, das erst später eingeführt wird. Leider werden Sie trotzdem nicht umhin gekommen, diese Funktionen bereits jetzt einzusetzen. Wenn Sie Hilfe beim Einsatz der Funktionen `printf` und `scanf` benötigen, finden Sie erstens das wichtigste mit einigen Beispielen (aber noch ohne Strings, die erst in Kapitel 4 eingeführt werden), im Foliensatz [PR3_printf+scanf.pdf bei den Vorlesungsfolien in Moodle](#). Zweitens finden Sie hier eine gute Referenz der C-Standardbibliothek: <http://www.cplusplus.com/reference/clibrary/>. `printf` und `scanf` sind dort unter dem Menüpunkt `cstdio` (`stdio.h`) erläutert. Der Einfachheit halber hier die beiden direkten Links:

- <https://cplusplus.com/reference/cstdio/printf/>
- <https://cplusplus.com/reference/cstdio/scanf/>

Unabhängig von den o. g. Links können Sie jedes beliebige Einsteigerbuch über C befragen. Hier gibt es z. B. eines gratis online: http://openbook.galileocomputing.de/c_von_a_bis_z/. Weitere Literaturhinweise finden Sie in den Vorlesungsfolien zu „Kapitel 0“ (am Ende des Foliensatzes).

1 Information zu den zu erbringenden Leistungen

Lesen Sie das Dokument [PR3_Leistungen.pdf](#) im Moodle-Kurs aufmerksam durch. Wenn Sie die Übungen als Pflichtaufgaben bearbeiten wollen, müssen Sie ein Formular ausfüllen (im genannten Dokument enthalten) und unterschrieben abgeben.

2 Ein Foto hochladen

Sie können den Ablauf der Übungen ein wenig erleichtern, indem Sie ein Foto von sich selbst in Ihr Moodle-Benutzerprofil hochladen: Dann wissen wir immer gleich, wen wir ansprechen müssen, wenn Ihr Übungs-Slot "dran" kommt 😊

3 Phasen der Übersetzung (1 Punkt)

basiert auf Vorlesung bis einschl. Abschnitt 2.b

Erstellen Sie mit einem Editor Ihrer Wahl ein Programm in C, das den Text „Hello, World!“ ausgibt. Übersetzen Sie das Programm mit dem gnu-Compiler `gcc`. Führen Sie es aus.

Ich empfehle Ihnen für "Programmieren 3" unter Linux zu arbeiten, entweder erstens auf den Pool-PCs, oder zweitens auf einem eigenen Linux-Rechner oder drittens in einer Linux-VM, die auf auf einem

eigenen Windows- oder MacOS-Rechner läuft. Eine vierte Alternative ist das Windows Subsystem für Linux (WSL), mit dem Sie auf einem eigenen Windows-11-Rechner ohne separate VM ein in Windows 11 enthaltenes Linux starten können. WSL empfehle ich für alle, die noch nicht viel Erfahrung mit virtuellen Maschinen haben. WSL wird wie folgt installiert:

Windows Power Shell als Administrator starten

z.B. durch Eintippen von "Power Shell" in der Windows-Suche (Windows-Taste) und "Starten als Administrator" und Eingeben des Administrator-Passworts

Im Power Shell Fenster WSL einschalten:

```
Enable-WindowsOptionalFeature -Online -FeatureName VirtualMachinePlatform
Enable-WindowsOptionalFeature -Online -FeatureName Microsoft-Windows-Subsystem-Linux
```

Dann den Rechner neu starten.

Danach in einem als normaler Benutzer laufenden Windows-Terminal-Fenster folgendes (der Rechner muss dabei online sein):

```
wsl --update
Aktualisiert den in Windows enthaltenen Linux-Kern
```

```
wsl --list --online
```

Zeigt die online für WSL verfügbaren „Linux-Distributionen“: Zusätzlich zum schon in Windows enthaltenen Linux-Kern müssen Sie noch eine von diesen Linux-Distributionen (die enthält u.a. die typischen Anwendungs-Programme) installieren. Installieren Sie mit dem folgendem Kommando z.B. die neueste Ubuntu LTS Distribution (als Beispiel dient hier Ubuntu-24.04):

```
wsl --install --distribution Ubuntu-24.04
```

Danach müssen Sie einen beliebigen Namen und Passwort für den ersten Linux-Benutzer eingeben, und dann befinden Sie sich auch schon in einer Linux-Shell. Ab jetzt können Sie das Linux in jedem Windows-Terminal-Fenster mit dem Kommando `wsl` starten und mit `exit` wieder beenden.

Sobald das Linux läuft, müssen Sie darin für PR3 noch `gcc`, `make` und `g++` installieren (`sudo` verlangt die Eingabe des zuvor von Ihnen vergebenen Passworts):

```
sudo apt install build-essential
```

Wenn alles funktioniert, sollte folgendes Kommando die Versionsnummer des `gcc` ausgeben:

```
gcc --version
```

Neben typischen Linux-Kommandos wie `cd`, `ls`, `gcc`, `grep` etc. kann man auch Windows-Programme aus diesem Linux heraus aufrufen, dann allerdings immer mit der Endung `.exe`, also z.B. `calc.exe`

Ihr Windows-Dateibaum ab `C:\` ist unter „`/mnt/c`“ unter Linux sichtbar, also z.B. Ihre persönlichen Windows-Dokumente unter `/mnt/c/Users/IhrWindowsBenutzerName/Documents`

Nun führen Sie die einzelnen Phasen der Übersetzung: Präprozessor, Compiler, Linker, nacheinander aus. Achten Sie darauf, jeden Schritt einzeln auszuführen. Die Vorlesungsfolien geben Ihnen Hinweise.

Heben Sie die bei den einzelnen Zwischenschritten entstehenden Dateien auf. Wenn Ihre Quelltextdatei z. B. `world.c` heißt, sollen Sie folgende Dateien erstellen:

- `world.E` mit dem Ergebnis des Präprozessors (Schritt 1)
 - Der Präprozessor druckt ohne weitere Angaben seine Ausgabe einfach aus. Um die Ausgabe in einer Datei zu erhalten, geben Sie die Option `-o datei` an.
- `world.o` mit dem Objectcode (Schritt 2)

- Dass `world.E` eine Präprozessor-Ausgabe enthält, erkennt der `gcc` leider nicht an der Dateiendung; dafür müssen Sie vor `world.E` noch folgende Option angeben: `-x cpp-output` (`cpp` steht für "C-Präprozessor").
- `world` mit dem gelinkten, ausführbaren Programm (Schritt 3)

Kopieren Sie die so erstellen Dateien an einen sicheren Ort.

Nun führen Sie den Präprozessor- und den Compilier-Vorgang (Schritt 1 und 2) in einem Zug durch Eingabe von `gcc -c world.c` durch. Die entstandene Objektcode-Datei müsste nun den gleichen Inhalt haben, wie die zuvor im Schritt 2 erstellte Datei. Überzeugen Sie sich durch Einsatz des Unix-Werkzeugs `diff` davon:

```
$ diff -s sicherer/ort/world.o world.o
Files sicherer/ort/world.o and world.o are identical
```

Entsprechend erstellen Sie nun direkt aus der Quelltextdatei `world.c` das ausführbare Programm `world`: `gcc -o world world.c`. Überzeugen Sie sich wieder durch Einsatz von `diff`, dass die entsprechenden Ergebnisse identisch sind.

(Dieser letzte Schritt klappt unter Windows nicht zuverlässig, vermutlich, weil der Linker unter Windows in der Programmdatei sekundengenau vermerkt, wann er diese erzeugt hat – und dieser Zeitpunkt unterscheidet sich natürlich in den meisten Fällen. Das ist eines von vielen Beispielen, die das Arbeiten unter Windows statt Linux in PR3 etwas mühsamer machen.)

4 Übersetzung mehrerer Quellcode-Module (1 Punkt)

basiert auf Vorlesung bis einschl. Abschnitt 2.b

Implementieren Sie zwei Funktionen in zwei Quellcode-Modulen:

- Das Modul `quadrat.c` soll die folgende Funktion `quadrat` definieren:


```
int quadrat(int x) {
    return x*x;
}
```
- Das Modul `main.c` soll die `main`-Funktion definieren. Aus `main` heraus soll `quadrat` aufgerufen und das Ergebnis mit `printf` ausgegeben werden.
- Erstellen Sie dazu eine Header-Datei `quadrat.h` mit dem Prototypen für die Funktion `quadrat`.

Übersetzen Sie jedes Modul einzeln mit dem `gcc`-Compiler (Präprozessor *nicht* einzeln aufrufen). Verwenden Sie dabei bitte unbedingt folgende Kommandozeilenoptionen:

| | |
|-------------------------------|---|
| <code>-c</code> | Compiler erstellt aus dem Quellcode-Modul eine Objektcode-Datei |
| <code>-Wall</code> | schaltet alle Warnungen ein |
| <code>-std=c99</code> | Compiler verwendet den C99-Standard |
| <code>-pedantic-errors</code> | Compiler weist nicht standardgemäße Programme zurück |

Linken Sie die beiden Objektcode-Dateien schließlich mit einem separaten Aufruf des `gcc`.

In `.o`-Dateien steht der Objektkode (auch Maschinencode oder Binärkode genannt) derjenigen C-Funktionen, die in der entsprechenden `.c`-Datei definiert wurden – aber nicht der Objektkode für Funktionen, die in der `.c`-Datei nur benutzt, aber dort nicht definiert wurden. Machen Sie sich klar, welchen Objektkode `quadrat.o` und `main.o` enthalten und welchen

nicht. Wo steht der Objektcode für die `printf`-Funktion, und wie kommt er letztlich in das ausführbare Programm? Was genau ist die Aufgabe des Linkers?

5 Deklarationen und Definitionen (0 Punkte)

basiert auf Vorlesung bis einschl. Abschnitt 2.a

- a) Rufen Sie sich nochmals in Gedächtnis, was in C der Unterschied zwischen einer Deklaration (also der Beschreibung einer Schnittstelle zu einem Ding) und einer Definition (also der Implementierung dieses "Dings") ist, und entscheiden Sie dann für die folgenden C-Konstrukte, ob es sich um eine Deklaration oder um eine Definition handelt (das können Deklarationen bzw. Definitionen für Variablen, Funktionen, oder Typen sein):

```
1. int i;
2. extern int summe(int a, int b);
3. int summe(int a, int b) { return a + b; }
4. extern int k;
5. typedef int Entfernung;
   // neuen Typ "Entfernung" einführen (wird später in Vorlesung genauer behandelt)
6. Entfernung hannoverNachHamburg;
```

- b) Was darf eine `#include`-Datei bzw. Header-Datei enthalten: Deklarationen/Schnittstellen, Definitionen/Implementierungen, oder beides?
- c) Falls ein Quelltextmodul `main.c` versehentlich eine unnötige Header-Datei `unnoetig.h` (Inhalt siehe unten) inkludiert, wächst dann die Größe der Objektcode-Datei `main.o`? Warum bzw. warum nicht?
 Inhalt der Datei `unnoetig.h`:

```
extern void unnoetige_funktion(void);
```

6 Inhalt von Objektcode-Dateien (0 Punkte)

basiert auf Vorlesung bis einschl. Abschnitt 2.b

Das Kommando `nm` listet auf, welche Funktionen und Variablen in einer Objektcode-Datei definiert sind und welche Funktionen und Variablen dort zwar nicht definiert sind, aber von definierten Funktionen benutzt werden (solche "externen Definitionen" müssen dann in einer anderen Objektcode-Datei desselben Programms definiert sein, damit insgesamt ein lauffähiges Programm aus diesen Objektcode-Dateien zusammengelinkt werden kann).

Beschreiben Sie die Ausgaben von `nm quadrat.o` und `nm main.o` aus Aufgabe 4. Funktionen, die in der Datei definiert sind, werden mit `T` ("text", damit ist Code gemeint) markiert, und Variablen mit `D` ("data"); Funktionen oder Variablen, die in dieser Datei zwar nicht definiert sind, aber von in dieser Datei definierten Funktionen benutzt werden, werden mit `U` ("undefined") markiert.

Fügen Sie in `main.c` die Zeile `#include "quadrat.h"` zweimal ein, erzeugen Sie `main.o` neu und vergleichen Sie mit dem ursprünglichen `main.o` – was stellen Sie fest? Wie erklären Sie Ihre Beobachtung?

Untersuchen Sie mit `nm` nur anhand der *.o-Dateien, wo in Ihrem Programm aus Aufgabe 4 die Funktion `printf` **benutzt** wird. Gibt es auch eine *.o-Datei, in der `printf` **definiert** wird?

Das folgende grep-Kommando sucht in seiner Standard-Eingabe nach `printf` mit einem Leerzeichen direkt davor und dem Zeilenende direkt dahinter – achten Sie also darauf, das Kommando exakt abzutippen (der angegebene Dateipfad passt für die Pool-PCs – auf Ihrem eigenen Rechner liegt `libc.a` möglicherweise an einem anderen Pfad):

```
nm /usr/lib/x86_64-linux-gnu/libc.a | grep ' printf$'
```

Was schließen Sie aus der Ausgabe dieses Kommandos?