Einführung

I. Vorbereitung

Damit Sie die Aufgaben aus dem GIP1-Praktikum und der -Übung bearbeiten können, benötigen Sie Papier und Bleistift, einen C-Compiler, einen Editor mit Syntax-Highlighting und die Versionsverwaltung *Git*.

I.1. Papier und Bleistift

Um die Aufgaben im GIP1-Praktikum bearbeiten zu können, benötigen Sie Papier und (Blei-)stift. Legen Sie dies in Griffweite auf Ihrem Schreibtisch bereit. Beim Programmieren werden Sie sich immer wieder Notizen machen müssen um Ihre Gedanken zu ordnen, eine kurze To-do-Liste zu führen etc.

I.2. Installation des Compilers (GCC/Clang)

Für das GIP1-Praktikum benötigen Sie den C-Compiler *GCC* oder *Clang*. Diesen werden Sie aus der Kommandozeile bzw. aus dem Terminal bedienen. Verwenden Sie für das GIP1-Praktikum zunächst keine IDE, da Ihnen diese die Sicht auf die zugrunde liegenden Vorgänge nimmt. Ein Verständnis dafür ist allerdings immens wichtig und hilft Ihnen später, eine IDE effizient zu nutzen.

I.2.1 Linux

Falls noch nicht vorhanden, installieren Sie die Entwicklungswerkzeuge und -bibliotheken. Führen Sie dazu unter *Ubuntu* in der Kommandozeile folgenden Befehl aus:

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

Führen Sie unter *CentOS* in der Kommandozeile folgenden Befehl aus:

```
$ sudo {yum|dnf} groupinstall "Development Tools" "Development Libraries"
```

Ersetzen Sie dabei `{yum|dnf}` durch `yum` unter *CentOS 7* und durch `dnf` unter *CentOS 8*. In anderen Linux-Distributionen, können sich die Befehle unterscheiden.

I.2.2 macOS

Falls noch nicht vorhanden laden und installieren Sie folgende Entwicklungsumgebung:

- *Xcode* von `https://developer.apple.com/xcode/`.

Diese enthält den Compiler *Clang*.
Führen Sie nach der Xcode-Installation folgenden Befehl in der Kommandozeile aus, um die zugehörigen Kommandozeilen-Tools zu installieren:

```
$ xcode-select --install
```

I.2.3 Windows

Falls noch nicht vorhanden laden und installieren Sie eines der folgenden Pakete:

- *MingW-W64-builds* (nicht Win-Builds) von `http://mingw-w64.org` oder
- *LLVM/Clang pre-built Binary* für Windows (64-bit) von `https://clang.llvm.org`.

MingW
Wählen Sie während der Installation folgende Einstellungen (Settings):

- Version: 8.1.0
- Architecture: x86_64 (für 64-Bit)
- Threads: posix
- Exception: seh
- Build revision: 0

Clang
Wählen Sie während der Installation eine der folgenden Optionen:
- Add LLVM to the system PATH for all users
- Add LLVM to the system PATH for current user

Auf diese Weise können Sie `clang` aus der Kommandozeile direkt aufrufen.

I.3. Installation eines Editors

Für das GIP1-Praktikum benötigen Sie einen Texteditor mit Syntax-Highlighting für C-Code. Auch hier noch einmal der Hinweis: Verwenden Sie zunächst keine IDE.

I.3.1 Linux

Falls noch nicht vorhanden, installieren Sie folgenden Editor:
- Atom von `https://atom.io`

Eine ausführliche Installationsanleitung finden Sie hier: `https://flight-manual.atom.io/getting-started/sections/installing-atom/#platform-linux`

I.3.2 macOS / Windows

Falls noch nicht vorhanden laden und installieren Sie den folgenden Editor:
- Atom von `https://atom.io`

I.4. Installation von Git

Im GIP1-Praktikum lernen Sie die Grundzüge der Quellcodeverwaltung mit Versionskontrolle durch *Git* kennen. Dazu benötigen Sie Git-Werkzeuge, die Sie aus der Kommandozeile bedienen. Verwenden Sie auch hier zunächst keinen GUI-Client – es gilt dasselbe wie für die IDE.

I.4.1 Linux

Falls noch nicht vorhanden, installieren Sie Git. Führen Sie dazu in der Kommandozeile folgenden Befehl aus:

```
$ sudo {apt|yum|dnf} install git-all
```

Ersetzen Sie dabei `{apt|yum|dnf}` durch `apt` unter *Ubuntu*, durch `yum` unter *CentOS 7* und durch `dnf` unter *CentOS 8*. In anderen Linux-Distributionen, können sich die Befehle unterscheiden.

I.4.2 macOS

Falls noch nicht vorhanden laden und installieren Sie Git als Teil der
- Command Line Tools.

Sollten diese nicht wie zuvor beschrieben installiert worden sein, erhalten Sie diese auf den Entwicklerseiten von Apple (`http://developer.apple.com/downloads/`), auf die Sie mit Ihrer Apple-ID kostenlosen Zugang erhalten können.

I.4.3 Windows

Falls noch nicht vorhanden laden und installieren Sie das folgende Paket von `https://git-scm.com/download/win`:
- *Git for Windows Setup* in 64-bit.

Wählen Sie bei der Installation als Standard-Editor für Git
- Use Atom as Git's default editor.

Wählen Sie als Einstellung für die `PATH`-Variable
- Git from the command line and also from 3rd-party software.

Wählen Sie als Einstellung für Zeilenenden
- Checkout Windows-style, commit Unix-style line endings.

Wählen Sie als Terminal-Emulator
- MinTTY

Wählen Sie als Standardverhalten für `git pull`
- Default (fast-forward or merge).

Viele davon sind bereits die Standardeinstellungen, kontrollieren Sie diese dennoch. Alle übrigen Einstellungen bei der Installation bleiben auf den Standardwerten.

I.4.4 für alle Betriebssysteme nach der Installation

Zuletzt müssen Sie Ihren Namen und Ihre E-Mail-Adresse in der Git-Konfiguration eintragen. Öffnen Sie dazu eine Kommandozeile und führen Sie folgende Befehle aus:

```
$ git config --global user.name "VORNAME NACHNAME"
$ git config --global user.email "youraddress@example.com"
```

Ersetzen Sie dabei `VORNAME`, `NACHNAME` und `youraddress@example.com` durch Ihre Angaben.

II. Hintergrundinfos

In diesem Praktikum erstellen Sie eine einfache Konsolenanwendung und stellen diese in einem *Git-Repository* unter Versionsverwaltung. Sie starten Ihre Anwendung von der Konsole und sehen sich den Rückgabewert an. Schließlich werfen Sie einen Blick auf den Assemblercode, der als Zwischenschritt während der Anwendungserstellung durch Compiler und Linker entsteht.

II.1. Konsolenanwendung

Eine *Konsolenanwendung* ist ein Programm, das von der Kommandozeile aufgerufen werden kann und das – wenn überhaupt – lediglich ein Text-basiertes Interface zur Verfügung stellt. Die Bedienung erfolgt über die Tastatur.
Konsolenanwendungen geben am Ende eine Ganzzahl an die Kommandozeile zurück, die ausgewertet werden kann. Diese Ganzzahl informiert in der Regel über aufgetretene Fehler. Der Rückgabewert `0` signalisiert üblicherweise „keine Fehler“. Die Bedeutung anderer Rückgabewerte ist anwendungsabhängig.
Unter Linux – zumindest in den Shells `bash`, `ksh`, `tcsh` und `sh` – ist der zuletzt zurückgegebene Wert in der Variable ? gespeichert. Er kann durch folgendes Kommando ausgegeben werden:

```
$ echo $?
```

In der Windows-Kommandozeile `cmd` ist der zuletzt zurückgegebene Wert in der Variable `errorlevel` gespeichert. Er kann durch folgendes Kommando ausgegeben werden:

```
> echo %errorlevel%
```

Da wir die Standardausgabe erst in den kommenen Vorlesungen behandeln werden, nutzen wir in diesem Praktikum lediglich die „Ausgabe“ über den Rückgabewert der Anwendung.

II.2. Git-Repository

Git ist eine eine verteilte Quellcodeverwaltung. Damit können Änderungen an Dateien erfasst und einzelne Zwischenschritte (*commit*) abgelegt und wiederhergestellt werden. Grundsätzlich kann von jedem abgelegten Zwischenschritt abgezweigt werden (*branch*). Durch die Verteilung des *Repositories* – des Aufbewahrungsortes der Änderungshistorie – können mehrere Entwickler gleichzeitig an verschiedenen Zweigen arbeiten und die Änderungen anschließend zusammenführen (*merge*). Weiterhin benötigt *Git* dadurch keinen zentralen Server; das komplette Repository liegt lokal bei jedem einzelnen Entwickler.
In diesem Praktikum werden Sie lediglich mit einem eigenen, lokalen Repository arbeiten. Änderungen legen Sie nacheinander ab. Verzweigungen werden Sie noch nicht nutzen.

III. Aufgaben – Teil 1

1. Hello 42

In dieser Aufgabe erstellen Sie eine Konsolenanwendung, die lediglich einen einzelnen Wert zurückgibt.

(a) Legen Sie auf Ihrer Festplatte einen Ordner für dieses Praktikum an. Öffnen Sie die Kommandozeile und wechseln Sie in diesen Ordner, indem Sie folgenden Befehl eingeben:

```
$ cd PROJECT_DIRECTORY
```

`PROJECT_DIRECTORY` steht dabei für den Pfad des zuvor angelegten Ordners.
Erzeugen Sie in diesem Ordner ein leeres *Git-Repository* mit folgendem Befehl:

```
$ git init DIRECTORY
```

Dieser legt das Repository im Unterverzeichnis `DIRECTORY` an. Wählen Sie dafür einen geeigneten Namen.

(b) Wechseln Sie in dieses Verzeichnis:

```
$ cd DIRECTORY
```

(c) Erstellen Sie in Ihrem Texteditor in diesem Verzeichnis eine Datei `main.c` mit folgendem Inhalt:

```
int main() {
  return 42;
}
```

(d) Legen Sie das sogenannte *Build-Directory* an, das die erzeugte Konsolenanwendung enthalten wird, indem Sie in der Kommandozeile folgenden Befehl eingeben:
Linux / macOS:

```
$ mkdir build
```

Windows:

```
> md build
```

(e) Erzeugen Sie Ihre Konsolenanwendung aus oben angelegter Datei `main.c`, indem Sie in der Kommandozeile folgenden Befehl eingeben:

```
$ gcc -o build/main main.c
```

Dieser Befehl hat die Form

```
$ gcc -o OUTPUT_FILE INPUT_FILE
```

`INPUT_FILE` gibt die zu übersetzende Quellcodedatei an, `OUTPUT_FILE` die zu erzeugende, ausführbare Datei – die Konsolenanwendung.
Ersetzen Sie `gcc` durch `clang`, falls Sie Clang statt GCC verwenden.

(f) Welchen Rückgabewert erwarten Sie, wenn Sie Ihre Konsolenanwendung ausführen? Schreiben Sie diesen an den Rand dieses Blattes oder auf einen Zettel.

(g) Führen Sie Ihre Konsolenanwendung aus, indem Sie in der Kommandozeile folgenden Befehl eingeben:

```
$ build/main
```

(h) Überprüfen Sie, ob der Rückgabewert korrekt ist, indem Sie diesen ausgeben. Geben Sie dazu in der Kommandozeile folgenden Befehl ein:
Linux / macOS:

```
$ echo $?
```

Windows:

```
> echo %errorlevel%
```

Entspricht der Rückgabewert nicht Ihren Erwartungen, korrigieren Sie mögliche Fehler.

(i) Funktioniert Ihre Konsolenanwendung fehlerfrei, merken Sie die Änderungen an Ihrem Projekt zur Ablage im Repository vor. Stellen Sie dazu die neu angelegte Datei `main.c` unter Versionskontrolle, indem Sie in der Kommandozeile den folgenden Befehl ausführen:

```
$ git add main.c
```

Überprüfen Sie dann den Zustand Ihres Repositories – genauer gesagt der *Arbeitskopie* des Repositories – indem Sie in der Kommandozeile folgenden Befehl ausführen:

```
$ git status
```

Die Ausgabe sollte folgender Ausgabe prinzipiell entprechen:

```
1  On branch master
2
3  No commits yet
4
5  Changes to be committed:
6    (use "git rm --cached <file>..." to unstage)
7          new file:   main.c
8
9  Untracked files:
10   (use "git add <file>..." to include in what will be committed)
11         build/
```

Zeile 3 teilt Ihnen mit, dass in Ihrem Repository noch keine Änderungen abgelegt wurden. Zeile 5 ff. teilt Ihnen mit, welche Änderungen zur Ablage im Repository vorgemerkt wurden. Hier sollte lediglich die neu angelegte Datei `main.c` aufgelistet sein. Zeile 9 ff. teilt Ihnen mit, welche Dateien und Verzeichnisse noch nicht unter Versionskontrolle stehen. Das *Build-Directory* samt Inhalt soll nicht unter Versionskontrolle stehen, da die Dateien dort lokal erzeugt werden können und in der Regel auch lokal erzeugt werden müssen.

(j) Ist die Ausgabe fehlerfrei, legen Sie die vorgemerkten Änderungen im Repository ab, indem Sie in der Kommandozeile folgenden Befehl ausführen:

```
$ git commit
```

Es öffnet sich ein Texteditor, in dem Sie die durchgeführten Änderungen kurz beschreiben – die *Commit-Message* anlegen. Ändern Sie nur den Teil der Datei vor den Zeilen, die mit dem Kommentarzeichen # beginnen. Speichern Sie anschließend die Datei, ohne den Namen zu ändern, und beenden Sie den Texteditor. Nun sind die Änderungen im Repository abgelegt und durch die *Commit-Message* beschrieben.

Alternativ kann eine kurze *Commit-Messsage* direkt über die Kommandozeile angegeben werden:

```
$ git commit -m "MESSAGE"
```

Anstelle von `MESSAGE` geben Sie dann die *Commit-Message* an.

(k) Sehen Sie sich die Änderungshistorie an, indem Sie folgenden Befehl in der Kommandozeile ausführen:

```
$ git log
```

Die Ausgabe sollte folgender Ausgabe prinzipiell entprechen:

```
1 commit 4f762a842b7c4beff03b14ee6f8bc842160a14c4 (HEAD -> master)
2 Author: Jane Appleseed <jane.appleseed@w-hs.de>
3 Date:   Tue Aug 1 00:00:01 1992 +0200
4
5     Add main.c
```

Zeile 1 teilt Ihnen den eindeutigen *Commit-Hash* mit. Zeile 2 teilt Ihnen mit, wer diesen *Commit* erzeugt hat; Zeile 3 das zugehörige Datum. Die *Commit-Message* finden Sie in Zeile 5.

2. Hello other number

In dieser Aufgabe ändern Sie den Rückgabewert und legen diese Änderung im Repository ab.

Arbeiten Sie mit dem Ergebnis der vorherigen Aufgabe weiter. Öffnen Sie dazu eine Kommandozeile und wechseln Sie in Ihr Projektverzeichnis – nicht das Build-Directory. Verwenden Sie bei Bedarf die detaillierten Beschreibungen der vorherigen Aufgabe als Anleitung zu den einzelnen Schritten.

(a) Überlegen Sie sich selbst einen Rückgabewert zwischen 0 und 255 und schreiben Sie diesen an den Rand dieses Blattes oder auf einen Zettel.

(b) Ändern Sie die Datei `main.c` entsprechend.

(c) Erzeugen Sie Ihre Konsolenanwendung erneut.

(d) Führen Sie Ihre Konsolenanwendung aus, überprüfen Sie den Rückgabewert und beheben Sie ggf. auftretende Fehler.

(e) Sehen Sie sich den Zustand der Arbeitskopie Ihres Repositories an. Die Ausgabe des Befehls

```
$ git status
```

sollte `main.c` als geänderte Datei (modified) aufführen, die noch nicht vorgemerkt wurde (not staged).

(f) Sehen Sie sich die Änderungen in `main.c` an, indem Sie folgenden Befehl eingeben:

```
$ git diff main.c
```

Die Ausgabe sollte folgender Ausgabe prinzipiell entprechen:

```
1 diff --git a/main.c b/main.c
2 index 45269d1..d46ce27 100644
3 --- a/main.c
4 +++ b/main.c
5 @@ -1,3 +1,3 @@
6  int main() {
7 -  return 42;
8 +  return 17;
9  }
```

Zeile 1 gibt an, dass sich der Name der Datei `main.c` von Version `a` (Originaldatei im Repository) zur Version `b` (aktuelle Änderung) nicht geänder hat. Zeile 2 führt die Git-internen IDs der beiden Zustände von `main.c` auf; die erste ID bezeichnet den Zustand vor der Änderung, die zweite ID den Zustand nach der Änderung. Abschließend folgt als Zahlenfolge kodiert die Angabe des Typs und der Zugriffsrechte von `main.c`. Zeile 3 und 4 enthalten den sogenannten *Unified-Diff-Header*, der gewissermaßen Zeile 1 in einem anderen Format wiederholt. In Zeile 5 folgt ein *Change-Hunk-Header*, der eine Gruppe von Änderungen (*change hunk*) einleitet: `-1,3` gibt dabei an, dass der folgende *Hunk* die Zeilen 1 bis 3 der Originaldatei beinhaltet, `+1,3` gib dabei an, dass dies den Zeilen 1 bis 3 der geänderten Datei entspricht. In der dann folgenden Ausgabe des *Hunks* beginnen unveränderte Zeilen mit einem Leerzeichen, aus der Originaldatei gelöschte Zeilen beginnen mit einem `-`, in der geänderten Version hinzugefügte Zeilen beginnen mit einem `+`.

(g) Entspricht die angezeigte Änderung Ihren Erwartungen, verwenden Sie den Befehl

```
$ git add main.c
```

um die Änderungen an `main.c` zur Ablage im Repository vorzumerken.

(h) Sehen Sie sich den veränderten Zustand Ihres Repositories an:

```
$ git status
```

(i) Legen Sie die vorgemerkten Änderungen mit

```
$ git commit
```

tatsächlich ab. Vergeben Sie eine aussagekräftige Commit-Message.

(j) Überprüfen Sie mit

```
$ git log
```

ob die Änderungen abgelegt sind.

(k) Sehen Sie sich den veränderten Zustand Ihres Repositories an:

```
$ git status
```

IV. Aufgaben – Teil 2

3. Hello Assembler

In dieser Aufgabe sehen Sie sich den Assemblercode an, der während des Kompilierens erzeugt wird.

Arbeiten Sie mit dem Ergebnis der vorherigen Aufgabe weiter. Öffnen Sie dazu eine Kommandozeile und wechseln Sie in Ihr Projektverzeichnis – nicht das Build-Directory.

(a) Schreiben Sie den in der vorherigen Aufgabe gewählten Rückgabewert an den Rand dieses Blattes oder auf einen Zettel.

(b) Erzeugen Sie den zugehörigen Assemblercode mit folgendem Befehl:

```
$ gcc -S -o build/main.asm main.c
```

Das *Flag* `-S` beendet `gcc` nach dem Kompilieren und damit vor Aufruf des Linkers. GCC gibt in diesem Fall Assemblercode aus.

(c) Öffnen Sie die Datei `build/main.asm` im Texteditor. Finden Sie den Codeblock der Funktion `main()`.
Finden Sie anschließend die Stelle, an welcher der von Ihnen festgelegte Rückgabewert in das Ausgaberegister „verschoben“ (*move*) wird.
Hinweis: Auf der *x86-* bzw. der *x86-64-*Architektur (z. B. Intel-basierter PC) erfolgt die Rückgabe über das Register `eax`, auf der ARM-Architektur (z. B. Raspberry Pi) erfolgt die Rückgabe über das Register `r0`.

4. Assemblercode editieren

In dieser Aufgabe editieren Sie den entstandenen Assemblercode, und erzeugen die Konsolenanwendung daraus.

Arbeiten Sie mit dem Ergebnis der vorherigen Aufgabe weiter. Öffnen Sie dazu eine Kommandozeile und wechseln Sie in Ihr Projektverzeichnis – nicht das Build-Directory.

(a) Überlegen Sie sich selbst einen weiteren Rückgabewert zwischen 0 und 255 und schreiben Sie diesen an den Rand dieses Blattes oder auf einen Zettel.

(b) Ändern Sie die Datei `build/main.asm` entsprechend.

(c) Erzeugen Sie die Konsolenanwendung mit folgendem Befehl:

```
$ gcc -o build/main build/main.asm
```

gcc erkennt automatisch, das die Eingabedatei Assemblercode enthält.

(d) Führen Sie Ihre Konsolenanwendung aus, überprüfen Sie den Rückgabewert und beheben Sie ggf. auftretende Fehler.