Studiengänge IB, FFI, CSI
WS 2023/24

Technische Hochschule Ingolstadt
Fakultät für Informatik

Praktikum zu „Grundlagen der Programmierung“

Blatt 1

Lernziele:	Erstellung, Übersetzung und Ausführung eines C-Programms mittels der Entwicklungsumgebung in der virtuellen Maschine
Erforderliche Kenntnisse:	Grundlegende Elemente und Aufbau von Programmen in der Sprache C
Voraussetzungen:	

- Die Entwicklungsumgebung in der virtuellen Maschine `ws23-AsmCpp2204-32bit` ist im PC-Pool und/oder zuhause aktiviert und ausprobiert worden.
- Förderlich ist es, wenn Sie schon die einführenden Dokumente
  11_WichtigeShellKommandos.pdf
  12_ArbeitenMitEinemRevisionControlSystem.pdf
  40_EditCompileRundDebug.pdf
  50_Makefiles.pdf
  bearbeitet haben
- Im Folgenden gehen wir davon aus, dass Sie die Entwicklungsumgebung in der virtuellen Maschine im PC-Pool benutzen.

Die Vorführung von Aufgaben zur Erlangung von Testaten muss auf Basis der virtuellen Maschine `ws23-AsmCpp2204-32bit` erfolgen. Für die Vorführung muss zwingend ein Pool-Rechner im PC-Pool G308 benutzt werden.

1 Aufgabe 1 (Orientierung in der Entwicklungsumgebung)

Melden Sie sich als Benutzer `lars` an der virtuellen Maschine an.
Öffnen Sie eine Shell und geben Sie folgenden Befehl zum Klonen der Daten aus dem Repository bei Bitbucket ein. Das Zeichen $ am Anfang der Zeile bitte nie tippen, das ist der Prompt der Shell!

```
$ cd                                          (Damit wechseln Sie ins Home-Verzeichnis)
$ git clone stud4_rw:s4_GdP1 GdP1             (Nutzt Alias aus ~/.gitconfig zum Klonen)
```

Hinweise: damit das Klonen funktioniert

- müssen Sie den URL-Alias `stud4_rw:s4_GdP1` durch Ihren Alias ersetzen!
- muss sich in Ihrer Datei ~/.gitconfig ein Eintrag folgender Bauart befinden
  (bitte Ihre Login-Daten und Ihren Alias einsetzen):

```
[url "https://KENNUNG:APPPWD@bitbucket.org/WORKSPACE/REPONAME.git"]
insteadOf = LABEL:DEFAULT_NAME
```

Nach dem erfolgreichen Klonen wechseln Sie in das Verzeichnis ~/GdP1 und erzeugen darin das Unterverzeichnis CodeExamples.

```
$ cd ~/GdP1
$ mkdir CodeExamples
```

Kopieren Sie nun das Unterverzeichnis ~/git_public_GdP1/CodeExamples/Count aus dem öffentlichen Repository der Vorlesung, welches in Ihrer VM bereits angelegt ist, in das lokale Unterverzeichnis CodeExamples.

```
$ cp -r ~/git_public_GdP1/CodeExamples/Count CodeExamples
```

Prüfen Sie den Inhalt des Verzeichnisses ~/GdP1/CodeExamples.
Sie können hierfür entweder den Dateimanager *Caja* verwenden oder, was einfacher ist, den nachfolgenden Befehl in der Shell.

```
$ ls -la ~/GdP1/CodeExamples
```

Sie sollten in etwa folgende Ausgabe erhalten:

```
insgesamt 12
drwxr-xr-x 3 lars lars 4096 24. Okt 08:31 .
drwxr-xr-x 4 lars lars 4096 24. Okt 08:31 ..
drwxr-xr-x 2 lars lars 4096 24. Okt 08:31 Count
```

Listen Sie die Dateien auf, die in Ihrer Arbeitsumgebung hinzugekommen sind, welche aber noch nicht im lokalen Repository gespeichert sind (Status).
Hinweis: das sind gerade die Dateien, die Sie oben aus dem Verzeichnis Count kopiert haben.

```
$ cd ~/GdP1
$ git status
```

```
Auf Branch master

Initialer Commit

Unversionierte Dateien:
(benutzen Sie "git add <Datei>...", um die Änderungen zum Commit vorzumerken)

        CodeExamples/

nichts zum Commit vorgemerkt, aber es gibt unversionierte Dateien (benutzen Sie
↪  "git add" zum Versionieren)
```

Merken Sie diese Dateien für den nächsten *Commit* vor und prüfen Sie danach abermals den Status:

```
$ git add .
$ git status
```

```
Auf Branch master

Initialer Commit

zum Commit vorgemerkte Änderungen:
(benutzen Sie "git rm --cached <Datei>..." zum Entfernen aus der Staging-Area)

neue Datei:     CodeExamples/Count/Makefile
neue Datei:     CodeExamples/Count/count.c
neue Datei:     CodeExamples/Count/count.gdb
```

Fügen Sie die vorgemerkten Dateien dem lokalen Repository hinzu *(Commit)*:

```
$ git commit -m "Beispiel Count aus CodeExamples-C-CrashCourse hinzugefuegt"
$ git status
```

Nun ist das kopierte Beispiel `Count` auch im lokalen Repository der virtuellen Maschine.
Als nächstes synchronisieren wir das lokale Repository mit dem Repository bei Bitbucket.
Alle für die Synchronisation vom lokalen Repository ins entfernte Repository anstehenden Änderungen werden über den Versuch ausgeben, den sog. *dry-run*:
Hinweis: Ihre Ausgabe wird sich geringfügig unterscheiden

```
$ git push --dry-run
```

```
To https://bitbucket.org/student4\_fabr/s4\_gdp1.git
* [new branch]      master -> master
```

Synchronisation ausführen:

```
$ git push
```

```
Zähle Objekte: 100% (8/8), fertig.
Komprimiere Objekte: 100% (6/6), fertig.
Schreibe Objekte: 100% (7/7), 1.25 KiB | 1.25 MiB/s, fertig.
Gesamt 7 (Delta 0), Wiederverwendet 0 (Delta 0), Pack wiederverwendet 0
To https://bitbucket.org/student4_fabr/s4_gdp1.git
   53c0091..b640e56  master -> master
```

Nun sind die neuen Dateien auch im Repository bei Bitbucket.

```
$ git push --dry-run
```

```
Everything up-to-date
```

Hinweis:

Die virtuellen Maschinen in den PC-Pools werden im sogenannten Kiosk-Mode betrieben. Daher gehen nach dem Herunterfahren der Maschinen alle während der Sitzung gemachten Änderungen verloren. Aus diesem Grund ist es **sehr wichtig**, dass Sie Ihre Daten (Programme) vor dem Herunterfahren der virtuellen Maschinen **zuerst ins lokale Repository** innerhalb der Kennung `lars` sichern (*git commit*) und **danach** dieses **lokale Repository mit dem Repository bei Bitbucket synchronisieren** (*git push*).

2 Aufgabe 2 (C-Programm count.c kompilieren und ausführen)

Öffnen Sie zwei Shells und wechseln Sie in jeder dieser Shells ins Verzeichnis
`~/GdP1/CodeExamples/Count`
Hinweis: die Tabulator-Taste sorgt für die praktische Vervollständigung von Pfad- und Dateinamen.
`$ cd ~/GdP1/CodeExamples/Count`

Öffnen Sie in einer dieser beiden Shells mit dem Editor vim die C-Datei count.c
`$ vim count.c`

In der zweiten Shell möchten wir nun den Compiler `gcc` so ausführen, dass die C-Datei `count.c` übersetzt wird. Bevor wird dies tun, legen wir unter `~/GdP1/CodeExamples/Count` noch ein Unterverzeichnis bin an, in das die vom Compiler erzeugte ausführbare Datei kopiert werden soll.
`$ mkdir -p bin`
Das Option `-p` sorgt dafür, dass das Verzeichnis bin nur dann angelegt wird, wenn es noch nicht existiert.

Kompilieren Sie nun die C-Datei `count.c` wie folgt:
`$ gcc -g count.c -o bin/count`

Das Kommando `gcc` übersetzt die Quell-Datei `count.c` und hinterlegt dabei zusätzliche *Debugging-Information* im Objekt-Code für spätere Sitzungen mit dem Debugger `gdb` (Option `-g`).

Die Option `-o bin/count` sorgt dafür, dass der Compiler die fertig gebundene ausführbare Datei unter dem Namen `count` (unter Windows würde automatisch ein `.exe` angehängt) im Unterverzeichnis `bin` ablegt. Wahlweise können Sie nach `-g` noch die Option `-Wall` angeben, die für mehr Überprüfungen und Warnungen sorgt.

Überprüfung des Erfolgs:
`$ ls bin`
Sie sollten die folgende Ausgabe erhalten:

```
count
```

Führen Sie das Programm count aus.
`$ bin/count`

Sie sollten die folgende Ausgabe erhalten:

```
i=0
i=1
i=2
i=3
i=4
```

Ändern Sie nun im Editor das Programm count.c so ab, dass

- die Schleifenvariable `k` heißt
- die Schleife von `k=10` bis `k=19` läuft

Nach der Änderung speichern, kompilieren und dann ausführen.
`$ bin/count`

```
k=10
k=11
...
k=17
k=18
k=19
```

Nachdem Sie diese Übung beendet haben, betrachten Sie die Code-Differenz, die sich aufgrund Ihrer Änderungen (`i` abgeändert in `k`) zwischen Ihrer Arbeitskopie und dem Stand im lokalen Repository ergeben haben.
`$ git status`

```
...
        geändert:       count.c

Unversionierte Dateien:
(benutzen Sie "git add <Datei>...", um die Änderungen zum Commit vorzumerken)

.count.c.swp
bin/
...
```

`$ git diff`

```
diff --git a/CodeExamples/Count/count.c b/CodeExamples/Count/count.c
index f857432..46580d6 100644
```

```
--- a/CodeExamples/Count/count.c
+++ b/CodeExamples/Count/count.c
@@ -2,7 +2,7 @@

int main() {
int i;
-    for (i = 0; i<5; i++) {
+    for (i = 10; i < 20; i++) {
printf("i=%d\n",i);
}
return 0;
```

Wenn Sie die obige Ausgabe analysieren, erkennen Sie die von ihnen ausgeführten Änderungen. Diese Änderungen wollen wir aber nicht im Repository speichern, und machen Sie daher in der Arbeitskopie rückgängig. Wir setzen automatisch auf den Stand im lokalen Repository zurück (reset)

```
$ git reset --hard
```

Wir übersetzen neu und testen:

```
$ gcc -g -Wall count.c -o bin/count
$ bin/count
```

```
i=0
i=1
i=2
i=3
i=4
```

Der Code des Beispiels Count ist nun wieder auf dem originalen Stand.

3 Aufgabe 3 (Das erste eigene C-Programm: myname.c)

Öffnen Sie eine Shell, legen Sie im Verzeichnis ~/GdP1 ein Unterverzeichnis Praktikum an, darunter ein Verzeichnis Code und darin wiederum ein Verzeichnis mit dem Namen MyName.
Ergebnis: Die Verzeichnishierarchie ~/GdP1/Praktikum/Code/MyName
Hinweis: statt mühsam die Verzeichnisse einzeln anzulegen genügt der Befehl

```
$ mkdir -p ~/GdP1/Praktikum/Code/MyName
```

Er legt alle fehlenden Zwischenverzeichnisse ebenfalls an.

Wechseln Sie in das Verzeichnis ~/GdP1/Praktikum/Code/MyName

```
$ cd ~/GdP1/Praktikum/Code/MyName
```

Erzeugen Sie mit dem Kommando

```
$ vim myname.c
```

eine neue Datei und geben Sie in die leere Datei C-Code ein, der in der main-Funktion unter anderem die folgenden Anweisungen enthält:

```
printf("Mein Name ist %s\n",name);
printf("Mein Geburtsdatum ist der %02d.%02d.%04d\n",day,month,year);
```

Ergänzen Sie dieses Code-Fragment zu einem kompletten C-Programm, das die noch fehlenden Deklarationen und Anweisungen enthält.

Hinweis: Sie müssen Variablen `name, day, month, year` mit geeigneten Typen definieren und dabei gleich initialisieren.

Übersetzen und testen Sie das Programm. Ein Testlauf könnte das folgende Ergebnis liefern:
`$ bin/myname`

```
Mein Name ist Lars Tragl
Mein Geburtsdatum ist der 21.07.1984
```

Wenn Sie mit der Ausgabe Ihres Codes zufrieden sind, fügen Sie diesen Ihrem lokalen Repository hinzu und synchronisieren Sie dann das lokale Repository mit dem Repository bei Bitbucket.
Hinweis: folgende Befehle könnten zielführend sein:

```
$ git status
$ git add myname.c
$ git commit -u lars -m "Mein erstes Programm MyName"
$ git push --dry-run
$ git push
```

Bemerkung: die Angabe von `-u lars` kann entfallen, wenn die Identität des ausführenden Benutzers in der Datei `~/.gitconfig` unter der Rubrik [user] im Attribut name hinterlegt ist.

4 Aufgabe 4 (Debugging: break/run/next)

Öffnen Sie zwei Shells und wechseln Sie jeweils in das Verzeichnis mit dem Beispiel
`~/GdP1/CodeExamples/Count`
Im ersten Fenster öffnen Sie mit einem Editor (`vim`) den Source-Code `count.c`
Im zweiten Fenster starten Sie den GNU-Debugger `gdb` und laden die ausführbare Datei `bin/count`.
Führen Sie das Programm `count` im Debugger schrittweise aus und untersuchen Sie nach jedem Schritt die Belegung der Variable, die die Schleife kontrolliert.

Hinweis: Studieren Sie zunächst das einführende Dokument
`40_EditCompileRunDebug.pdf`
und konsultieren Sie (nur bei Bedarf) die ausführliche Dokumentation im Referenzmanual des Debuggers `gdb.pdf`