Übungsblatt 1 zur Vorlesung Compilerbau

Abgabe: 27.10.2024, 23:59

Prof. Dr. Michael Kuhn (michael.kuhn@ovgu.de)
Michael Blesel (michael.blesel@ovgu.de)
Parallel Computing and I/O • Institut für Intelligente Kooperierende Systeme
Fakultät für Informatik • Otto-von-Guericke-Universität Magdeburg
https://parcio.ovgu.de

Dieses Übungsblatt ist als Einführung in die Benutzung der Programmiersprache C zu verstehen. Im Folgenden sollen Sie sich auf unserem Cluster einloggen, das Navigieren in einer Shell üben und erste Programmieraufgaben bearbeiten.

1. Gruppenbildung und Cluster-Login (60 Punkte)

Für die Abgabe der Aufgaben und das Bearbeiten im Team benötigt jede Abgabegruppe ein Git-Repository im GitLab der Universität. Diese müssen Ihnen manuell zugewiesen werden, weshalb Ihre erste Aufgabe darin besteht, eine Gruppe zu bilden und eine Liste der OVGU-Accountnamen der Mitglieder per E-Mail an michael.blesel@ovgu.de zu senden.

Wichtig: Sie müssen sich mindestens einmal im GitLab (https://code.ovgu.de/) eingeloggt haben, um Ihren Account zu aktivieren, bevor wir Ihnen dort ein Repository zuweisen können.

Tun Sie dies bitte so frühzeitig wie möglich, damit Sie Ihr Git-Repository rechtzeitig vor der Abgabefrist erhalten. Sollten Sie am ersten Übungstermin vor Ort keine Gruppe finden kann Mattermost für die Gruppensuche benutzt werden.

Zum Bearbeiten mancher Übungsaufgaben benötigen Sie gegebenenfalls Zugang zu unserem Cluster. Um sich dort einzuloggen, verbinden Sie sich mit dem OVGU-VPN und loggen Sie sich anschließend per SSH auf dem Cluster ein. Dort können Sie dann innerhalb der Shell arbeiten. Ein Shell-Tutorial finden Sie unter https://swcarpentry.github.io/shell-novice/.

Unter Linux, macOS oder dem Windows Subsystem for Linux können Sie sich mit folgendem Befehl auf den Login-Knoten verbinden, wobei <name> für Ihren OVGU-Accountnamen steht. Alternativ können Sie unter Windows einen grafischen SSH-Client wie PuTTY nutzen oder SSH über die Windows Shell verwenden. Sie können sich mit Ihrem üblichen OVGU-Account-Passwort einloggen.

```
$ ssh <name>@ants.cs.ovgu.de
```

Um nicht jedes Mal Ihren Benutzernamen angeben zu müssen, können Sie folgenden Eintrag in der SSH-Konfiguration unter ~/.ssh/config anlegen.

```
Host ants.cs.ovgu.de
    User <name>
```

Um außerdem nicht jedes Mal Ihr Passwort eingeben zu müssen, können Sie sich ein Schlüsselpaar generieren, was das Login erleichtert.

```
$ ssh-keygen
$ ssh-copy-id ants.cs.ovgu.de
```

Daten können auf den und vom Cluster mithilfe von SCP kopiert werden.

```
$ scp lokale/datei ants.cs.ovgu.de:entfernte/datei
$ scp ants.cs.ovgu.de:entfernte/datei lokale/datei
$ scp -r lokaler/ordner ants.cs.ovgu.de:entfernter/ordner
$ scp -r ants.cs.ovgu.de:entfernter/ordner lokaler/ordner
```

Nach dem Login müssen Sie folgenden Befehl ausführen, um die für die Übungen genutzte Softwareumgebung verfügbar zu machen. Beachten Sie das Leerzeichen zwischen `.` und `/`. Denken Sie daran, dies nach jedem Login vor der Bearbeitung der Aufgaben zu tun.

```
$ . /opt/spack/cb-202425/env.sh
```

Die Softwareumgebung enthält einen aktuellen Compiler in Form von GCC 14.1. Als Referenzsystem zur Bewertung der Aufgaben wird der Cluster genutzt. Die Übungsaufgaben werden dort korrigiert und müssen dementsprechend auch dort lauffähig sein.

Die Software wird in Form sogenannter Module verfügbar gemacht. Mithilfe von `module list` können Sie sich die aktuell geladenen Module anzeigen lassen. `module avail` gibt eine Übersicht aller verfügbaren Module aus.

Fragen: Welche Softwarepakete sind auf dem Cluster standardmäßig geladen und in welchen Versionen liegen sie vor? Welche Befehle wären notwendig, um ein fiktives Modul `foobar` zu laden und anschließend wieder zu entladen? Gibt es eine Möglichkeit, alle geladenen Module mit einem einzelnen Befehl auf einmal zu entladen? Wie können Sie sich weitere Informationen zu einem bestimmten Modul anzeigen lassen?

Zur Bearbeitung des Quelltextes sollten Sie sich außerdem einen geeigneten Editor für die Kommandozeile (z. B. nano, Vim, Emacs) oder Ihre grafische Desktopumgebung (z. B. Atom, gedit) installieren. Integrierte Entwicklungsumgebungen (z. B. Visual Studio Code, Eclipse) erleichtern die Arbeit noch weiter.[1]

2. Nutzung des Command Line Interface (30 Bonuspunkte)

Die folgenden konkreten Aufgaben haben Sie zu bewältigen:

1. Bewegen im CLI (Command Line Interface)
   a) Machen Sie sich mit der Verwendung von Manual-Pages vertraut:
      `$ man man`

[1] Visual Studio Code erlaubt das direkte Arbeiten über SSH: `https://code.visualstudio.com/docs/remote/ssh`

b) Lassen Sie sich den Namen des aktuellen Arbeitsverzeichnisses anzeigen:
`$ man pwd`

c) Lassen Sie sich den Inhalt Ihres Homeverzeichnisses anzeigen:
`$ man ls`

d) Erzeugen Sie ein neues Verzeichnis mit dem Namen `testdir`:
`$ man mkdir`

e) Ändern Sie das Arbeitsverzeichnis in das neue Verzeichnis:
`$ man cd`

f) Lassen Sie sich noch einmal das aktuelle Arbeitsverzeichnis anzeigen.

g) Erzeugen Sie eine leere Datei mit dem Namen `testfile`:
`$ man touch`

h) Benennen Sie die neue Datei um in `testfile2`:
`$ man mv`

i) Kopieren Sie die umbenannte Datei in `testfile3`:
`$ man cp`

j) Löschen Sie die Datei `testfile2`:
`$ man rm`

Frage: Erläutern Sie warum der Wechsel des Arbeitsverzeichnisses nur mit `cd` funktioniert, mit `/usr/bin/cd` aber nicht. (Tipp: `man bash`, `man builtin`)

2. Packen eines Archivs

a) Erstellen Sie ein Verzeichnis mit dem Namen `archiv`.

b) Erzeugen Sie darin eine Datei mit zufälligem Inhalt:
`$ dd if=/dev/urandom of=archiv/zufall bs=1k count=256`

c) Lassen Sie sich die Größe der Datei anzeigen:
`$ ls -lh archiv/zufall`

d) Lassen Sie sich die Größe des Verzeichnisses anzeigen:
`$ ls -ldh archiv`

e) Erzeugen Sie ein `tar`-Archiv, welches das Verzeichnis enthält:
`$ tar -cf archiv.tar archiv`

f) Lassen Sie sich die Größe des Archives `archiv.tar` ausgeben.

Frage: Was fällt Ihnen bezüglich der drei Größen auf?

g) Komprimieren Sie das Archiv:
`$ gzip archiv.tar`
Das Archiv ist nun erstellt. `gzip` hat das Archiv automatisch in `archiv.tar.gz` umbenannt.

h) Lassen Sie sich die Größe des gepackten Archives `archiv.tar.gz` ausgeben.

Frage: Ist es möglich, ein gepacktes Archiv (`.tar.gz`) mit einem Aufruf von `tar` zu erzeugen? Wie hätte dieser Aufruf lauten müssen?

i) Lassen Sie sich den Inhalt des gepackten Archives ausgeben.

3. C-Grundlagen (60 Bonuspunkte)

Die Aufgabe dient dem besseren Kennenlernen der Sprache C. Es sollen grundlegende Konstrukte benutzt und geübt werden. In den Materialien finden Sie die Datei `map.c` mit dem dazugehörigen Makefile.[2] Mit `make map` wird die Anwendung kompiliert.

Es soll eine kleine Landkarte der Größe 3×3 simuliert werden. Legen Sie dazu ein globales statisches 3×3-Array mit dem Namen `map` an. Definieren sie außerdem einen Aufzählungsdatentypen (`enum`) `cardd` (cardinal direction) mit den vier Himmelsrichtungen N, E, S, W.

Implementieren Sie die vorgegebene Funktion `set_dir` so, dass an die übergebene Stelle mit Koordinaten `x` und `y` auf der Karte die entsprechende Himmelsrichtung gesetzt wird. Achten Sie dabei auf das adäquate Behandeln ungültiger Eingaben.

Implementieren Sie die Ausgabefunktion `show_map` mittels der Funktion `printf`. Verwenden Sie dazu die `switch`-Konstruktion. Die Ausgabe soll wie folgt aussehen.

```
0␣␣␣N␣␣␣0
W␣␣␣0␣␣␣E
0␣␣␣S␣␣␣0
```

3.1. Bitoperationen (30 Bonuspunkte)

Erweitern Sie die Karte um die Himmelrichtungen Nord-West (NW), Nord-Ost (NE), Süd-Ost (SE) und Süd-West (SW). Das `enum cardd` darf dabei modifiziert aber nicht erweitert werden. Die neuen Richtungen sollen mittels Bitoperationen gestaltet und wie folgt ausgegeben werden.

```
NW␣␣N␣␣NE
W␣␣␣0␣␣␣E
SW␣␣S␣␣SE
```

Als Einstieg in C ist beispielsweise The C Book (`https://publications.gbdirect.co.uk/c_book/`) empfehlenswert. Eine Liste weiterer Bücher und Tutorials zu C finden Sie unter `http://www.iso-9899.info/wiki/Books`.

4. C-Zeiger (60 Bonuspunkte)

In dieser Aufgabe sollen Sie sich mit dem grundlegenden Konzept der Zeiger in C vertraut machen. In der Datei `pointer.c` finden Sie einige Funktionen. An einigen Stellen verrät die Ausgabe mittels `printf` das erwartete Ergebnis. An anderen Stellen verraten die Variablennamen oder Kommentare, was gemeint ist.

[2] Ein Tutorial zu Makefiles finden Sie unter `https://swcarpentry.github.io/make-novice/`.

Ihre Aufgabe ist es, die fehlenden Einträge zu vervollständigen, sodass die beschriebene Ausgabe korrekt erfolgt. Beachten Sie: Es darf nichts anderes am Programm geändert werden, außer die mit `TODO` gekennzeichneten Stellen. Das Programm muss am Ende fehler- und warnungsfrei kompilieren und eine semantisch korrekte Ausgabe produzieren.

5. Debugging einer einfachen Anwendung (150 Bonuspunkte)

Im Verzeichnis `simple` ist ein primitives Programm enthalten, welches mit `make` kompiliert werden kann. Dieses Programm dient dazu, dass Sie sich ein wenig mit GDB und Valgrind beschäftigen. Es enthält lediglich vier Funktionen, welche jeweils einen Zeiger auf eine Zahl oder ein Array mit einer Zahl enthalten und gibt diese dann in der `main`-Funktion aus. Leider enthält dieses Programm diverse Fehler.

- Führen Sie folgende kleinere Tests durch, um GDB kennen zu lernen. Dokumentieren Sie die genutzten Eingabebefehle und die Ausgabe in einer Textdatei. Ein kurzes Tutorial für GDB finden Sie beispielsweise unter `https://www.cs.cmu.edu/~gilpin/tutorial/`. Die wichtigsten Befehle sind außerdem in `https://darkdust.net/files/GDB%20Cheat%20Sheet.pdf` zusammengefasst.
  - Starten Sie das Programm. Dafür übergeben Sie am besten direkt den Programmnamen an GDB (`gdb ./simple`).
  - Platzieren Sie einen Breakpoint auf der Funktion `mistake1`, starten Sie das Programm, geben Sie den Wert von `buf` und `buf[2]` aus. Gehen Sie zur nächsten Zeile und geben Sie beide Werte wieder aus. Von welchem Typ ist `buf`?
  - Platzieren Sie einen Breakpoint in der Funktion `mistake2`, setzen Sie den Programmlauf fort, welchen Typ hat `buf`?
  - Setzen Sie den Programmlauf fort, welcher Text wird ausgegeben? Lassen Sie sich den Code um diese Stelle herum ausgeben. Welche Frames sind auf dem Stack? Wechseln Sie zu Frame 1. Geben Sie den Inhalt von `p` aus.
  - Rufen Sie innerhalb von GDB die Funktion `mistake4` auf (schauen Sie nach, wie man in GDB Funktionen direkt aufrufen kann).
- Modifizieren Sie das Programm zunächst so, dass es nicht mehr abstürzt. Versuchen Sie die Modifikationen möglichst gering zu halten. Verwenden Sie zunächst GDB, um die Fehlerstellen aufzuspüren. Die Ausgabe soll wie folgt aussehen:

```
1: 1
2: 2
3: 3
4: 4
```

- Nun läuft das Programm, leider enthält es jedoch noch weitere Speicherfehler, die je nach Umgebung (mehr oder weniger zufällig) auftreten können. Modifizieren Sie das Programm unter Zuhilfenahme von Valgrinds `memcheck` so, dass jede Methode Speicher korrekt reserviert und dass am Ende der Speicher korrekt freigegeben wird. Rufen Sie dafür das Programm mit `valgrind ./simple` auf.

  Hinweis: Den Speicher einfach mit `static` zu allokieren ist *nicht* erlaubt. Verzichten Sie darüber hinaus auf globale Arrays und Variablen. Eine kurze Anleitung für Valgrind finden Sie beispielsweise unter `https://www.valgrind.org/docs/manual/quick-start.html`.

Dokumentieren Sie die Fehler, die zu den Abstürzen und Speicherfehlern führen. Notieren Sie hierfür für jeden vorhandenen Fehler die Code-Zeilen, welche fehlerhaft sind und den genauen Grund der Ursache (z. B. Speicher mehrfach freigegeben).

Abgabe

Als Abgabe werten wir den letzten Commit vor der Abgabefrist in Ihrem Git-Repository. Im Hauptverzeichnis des Repositories wird ein Verzeichnis `CB-2024-Uebung-01-Materialien` mit folgendem Inhalt erwartet:

- Eine Datei `gruppe.md` mit den Gruppenmitgliedern (eines je Zeile) im folgenden Format:

  ```
  Erika Musterfrau <erika.musterfrau@example.com>

  Max Mustermann <max.mustermann@example.com>
  ```

- Eine Datei `antworten.md` mit Ihren Antworten (Aufgaben 1 und 2)
- Der überarbeitete Code des `map`-Programms (Aufgabe 3)
  - Alle Quellen, aus denen Ihr Programm besteht (`map.c`); gut dokumentiert
  - Ein `Makefile` derart, dass `make map` und `make clean` erwartungsgemäße Binärdateien erzeugen bzw. löschen.
- Der überarbeitete Code des `pointer`-Programms (Aufgabe 4)
  - Alle Quellen, aus denen Ihr Programm besteht (`pointer.c`); gut dokumentiert
  - Ein `Makefile` derart, dass `make pointer` und `make clean` erwartungsgemäße Binärdateien erzeugen bzw. löschen.
- Eine Textdatei mit den Ein-/Ausgaben von GDB mit dem Namen `simple-gdb.md`, eine Textdatei namens `simple-error.md` mit Fehlerbeschreibungen (Ursache, Code-Zeilen) und der überarbeitete Code im Verzeichnis `simple` (Aufgabe 5)