Vorlesung Systemnahe Programmierung – WS 2014/15

Prof. Dr. Matthew Smith, Dr. Matthias Frank, Sergej Dechand

1. Übungszettel

Ausgabe: Mi 08.10.2014; Abgabe bis zum (Sonntag) 19.10.2014, 23:59 Uhr
(Die folgenden Abgaben ab Blatt 2 liegen wie angekündigt jeweils Freitags bis 23.59 Uhr)
Die Vorführung erfolgt in der Woche vom 20.10.2014 bis zum 24.10.2014 in Ihrer Übungsgruppe.

Alle Programme müssen im Poolraum unter Linux kompilierbar bzw. lauffähig und ausreichend kommentiert sein. Die Lösungen sind bei Ihrem Tutor während Ihrer Übungsgruppen vorzuführen.

Aufgabe 1: Dynamische Listen (4 Punkte)

In dieser Aufgabe soll eine zeigerverkettete Liste implementiert werden, in der Personendaten gespeichert werden können. Die Daten sollen in alphabetischer Reihenfolge eingefügt werden. Die Sortierung erfolgt nach dem Nachnamen der Personen. Im Falle von gleichen Nachnamen wird der Vorname als zweites Kriterium hinzugezogen. Stimmen sowohl Nach- als auch Vornamen überein, ist die Reihenfolge der Speicherung irrelevant. Bei den Daten handelt es sich um Vor- und Nachnamen, die getrennt durch ein Leerzeichen in einer Textdatei vorliegen. Eine Zeile besteht aus einem Vornamen und einem Nachnamen. Sowohl Vor- als auch Nachnamen sind jeweils maximal 100 Zeichen lang. Eine solche Namensdatei könnte beispielsweise wie folgt aussehen:

Victor Hugo
Albert Einstein
Johann-Wolfgang Von-Goethe

Der Name der Datei soll während der Laufzeit abgefragt werden. Das Programm liest alle Daten aus der Datei ein und gibt diese anschließend in sortierter Form auf dem Bildschirm aus. Der vom Programm reservierte Speicher soll vor Beendigung wieder freigegeben werden.

Um nicht unnötig viel Speicherplatz für die einzelnen Einträge zu belegen, soll für die jeweiligen Einträge nur jeweils der wirklich benötigte Speicherplatz reserviert werden. Schauen Sie sich in diesem Zusammenhang die beiden Funktionen `malloc` und `calloc` an.

Ein Listenelement könnte beispielsweise folgende Struktur haben:

```
struct node {
  struct node *succ;
  char *vorname;
  char *nachname;
};
```

Bei der Implementierung sollen mindestens zwei Header-Dateien verwendet werden. Eine Headerdatei, z.B. personen_liste.h, definiert die benötigten Funktionsköpfe für die verkettete Liste und die andere, z.B. mystring.h definiert Funktionen für die Bearbeitung von Zeichenketten. Auf die Standard string.h soll hier bewusst verzichtet werden.

Überprüfen Sie mittels des Tools „Valgrind“ (http://valgrind.org/), das auf den Rechnern im CIP-Pool installiert ist, ob der gesamte von Ihnen reservierte Speicher wieder freigegeben wurde („valgrind --tool=memcheck --leak-check=yes ./myprog“).

Aufgabe 2: Makefiles (4 Punkte)

Das Programm „make“ wird benutzt, um das Kompilieren eines Projekts zu vereinfachen. So werden beim Aufruf von „make“ nur diejenigen Dateien (z.B. .c Dateien) kompiliert, deren Änderungszeitpunkt älter ist als der des Produkts (z.B. .o Dateien). Dabei arbeitet „make“ nach Regeln, die in einer Datei namens „Makefile“ definiert sind. Diese Regeln sind nach dem folgenden Schema aufgebaut:

```
Ziel: Abhängigkeiten
 <TAB>  Kommando
 <TAB>  ...
```

Beispiel:

```
ftpd: main.o commands.o
        gcc -o ftpd main.o commands.o

main.o:
        gcc -c main.c

commands.o:
        gcc -c commands.c
```

Das Programm „make“ liest das Makefile ein und startet bei der ersten Regel, die auch „default goal“ genannt wird.

Oft ist es sinnvoll, Variablen in Makefiles zu deklarieren, die später in den Regeln benutzt werden können. Dabei werden einfache Textersetzungsregeln benutzt, ähnlich dem C-Präprozessor.

Beispiel:

```
VORLESUNG = Systemnahe Programmierung
SEMESTER = Wintersemester
JAHR = 2012

default:
        echo $(VORLESUNG) $(SEMESTER) $(JAHR)
```

Trotz der Variablen kann die Anzahl der Regeln je nach Anzahl der zu kompilierenden Dateien sehr lang werden. GNU make unterstützt eine Reihe von automatischen Variablen, die zur Laufzeit gesetzt werden, z.B.:

$@ : Dateiname des Ziels der Regel

$< : Name der ersten Abhängigkeit

Beispiel (vorher müssen die Dateien a.0, b.0 und c.0 angelegt werden, z.B. mit „touch a.0“...):

```
OK: a.1 b.1 c.1
	echo $@

%.1: %.0
	mv $< $@
```

Eine kurze Einführung ist auf http://www.student.cs.uwaterloo.ca/~isg/res/unix/make/tutorial/ zu finden.

Eine ausführliche Anleitung zu GNU make gibt es z.B. auf http://www.gnu.org/software/make/manual/make.html.

Aufgabe: Schreiben Sie ein Makefile für Aufgabe 1, das

a) folgende Variablen definiert und benutzt:

   OBJECTS (alle .o Dateien),

   CPPFLAGS, das jedem gcc Aufruf übergeben werden soll und auf -g -Wall gesetzt werden soll

b) alle .c Dateien in .o Dateien unter Verwendung von automatischen Variablen kompiliert (gcc -c)

Achten Sie darauf, dass alle „.c“-Dateien, die eine „.h“-Datei einbinden, neu kompiliert werden falls die „.h“-Datei geändert wird.

Alternativ kann auch ein anderer Compiler wie z.B. „clang“ verwendet werden. Im Vergleich zu GCC bietet „clang“ umfangreichere Fehlermeldungen. Die Optionen orientieren sich hierbei an gcc, d.h. aus „gcc -o output file.c“ wird „clang -o output file.c“.

WICHTIG: Künftig sollen alle C Aufgaben mit Makefile abgegeben werden.

Aufgabe 3: Subversion (4 Punkte)

Subversion (http://subversion.apache.org/) ist ein Versions-Verwaltungssystem, das auch zum kollaborativen Arbeiten genutzt wird. Es soll im Folgenden sowohl für Ihr gemeinsames Arbeiten an den Aufgaben, als auch als Medium zur Abgabe der Übungsaufgaben an Ihren Tutor genutzt werden. Subversion verwaltet die Projektarchive (Repositories) als Dateisysteme und protokolliert Veränderungen an den Dateien, sodass auch auf ältere Versionen von Dateien zugegriffen werden kann.

Jeder Abgabegruppe ist ein Repository zugeordnet, auf das per Internet zugegriffen werden kann (für Zugriffe von außerhalb des Campus verwenden Sie einen VPN-Zugang):

https://nessos.informatik.uni-bonn.de/svn/SystemnaheProgrammierung/GruppeXY/

Benutzername und Passwort werden allen Dreiergruppen nach Schließung der Übungsanmeldung (mit dem TVS) per Mail zugesandt.

Hinweis: *Falls Sie sich noch nicht zu dritt in Ihrer Übungsgruppe angemeldet haben holen Sie das bitte noch nach. Sollten Sie noch keinen Abgabepartner haben suchen Sie einen*

Partner z.B. über unsere Mailingliste. Nach Schließung von TVS werden dann die verbleibenden Einzelpersonen und Zweiergruppen eines Abgabetermins in Dreier-Abgabegruppen zusammengefasst.

Ihr Repository soll nach dieser Aufgabe einen Ordner „trunk“ und einen Ordner „tag“ enthalten. Der Ordner „trunk“ ist Ihr persönlicher Arbeitsbereich und „tag“ dient zur Abgabe der Ergebnisse. Zum Bearbeiten dieser Aufgabe sind die Links http://svnbook.red-bean.com/ und http://www.gnu.org/software/indent/indent.html hilfreich.

Aufgabe: Importieren Sie die in Aufgabe 1 und 2 erstellten Quellcode Dateien und das Makefile, in Ihr Repository unter
https://nessos.informatik.uni-bonn.de/svn/SystemnaheProgrammierung/GruppeXY/trunk/Aufgabe03
Benutzen Sie hierzu den Befehl „import“. Checken Sie nun Ihr gesamtes Repository aus und modifizieren Sie den Programmcode mit dem Tool „GNU indent“. Dieses sorgt für eine gut lesbare Formatierung. Der Aufruf von „indent“ inklusive der verwendeten Parameter zur Formatierung soll in einer „README“ Datei protokolliert und ebenfalls mittels „add“ ins Repository aufgenommen werden. Diese Änderungen werden mittels „commit“ im Repository eingepflegt. Anschließend soll das Verzeichnis „tag“ lokal angelegt werden. Mittels des Kommandos „add“ kann das Verzeichnis in die Versionskontrolle aufgenommen werden. Mittels „commit“ werden dann die Änderungen von Ihrer lokalen Kopie ins Repository überführt. Die zur Abgabe bereite Version Ihrer Sourcen soll nun mittels des Befehls „copy“ nach https://nessos.informatik.uni-bonn.de/svn/SystemnaheProgrammierung/GruppeXY/tag/Aufgabe03 kopiert werden.

WICHTIG: Zukünftig geben Sie bitte analog zu dieser Aufgabe unaufgefordert alle Aufgaben der folgenden Übungsblätter in einem jeweils separaten Verzeichnis mittels SVN ab.