Institut für Informatik

der Ludwig-Maximilians-Universität München

Systempraktikum – Wintersemester 2011/2012

Prof. Dr. Dieter Kranzlmüller
Dr. Nils gentschen Felde, Stephan Reiter, Johannes Watzl

Blatt 2— Grundlagen I: Funktionen, Kontrollstrukturen, E/A-Konzepte

Abgabedatum theor. Aufgaben	Abgabedatum prakt. Aufgaben	Deadline Projektaufgaben
30.10.	30.10.	—

Theoretische Aufgaben (Blatt 2)

Hinweise

- Die Abgabe der theoretischen Aufgaben erfolgt über UniWorX unter der Adresse `https://uniworx.ifi.lmu.de/`.
- Gültige Dateiformate für die Abgabe Ihrer Lösungen zu den Theorieaufgaben sind ausschließlich PDF und Plain-Text.

Aufgabe T-2-1

Der GNU C-Compiler bietet eine Reihe verschiedener Aufrufparameter (Flags/Optionen). In der Manpage (`man gcc`) werden diese Parameter in verschiedene Kategorien eingeteilt, zu finden im Abschnitt „OPTIONS“, Unterabschnitt „Option Summary“. Erstellen Sie eine Liste dieser Kategorien. (Unterkategorien der Kategorie „Machine Dependent Options“ sollen nicht aufgelistet werden.) Geben Sie zu jeder gefundenen Kategorie an, für welche Arten von Aufgaben diese Kategorie passende Aufrufparameter anbietet. Suchen Sie sich aus jeder Kategorie einen beliebigen Aufrufparameter (Flag) aus, und beschreiben Sie kurz (in deutscher Sprache) seine Funktion.

Aufgabe T-2-2

Der GNU C-Compiler lässt sich unter anderem mit den Flags `-ansi` und/oder `-pedantic` aufrufen. Finden Sie heraus, wofür diese Zusatzoptionen jeweils stehen.

Aufgabe T-2-3

a. Um aus C-Quelltext ein ausführbares Programm zu generieren, kommen verschiedene Werkzeuge/Tools zum Einsatz. Welche und in welcher Reihenfolge?

b. Der eigentliche Übersetzer (Compiler) ist nur eines dieser Werkzeuge. Dennoch besteht auch die Übersetzung wieder aus verschiedenen Teilphasen, nämlich: Code-Optimierung, Code-Erzeugung, Lexikalische Analyse, Syntaktische Analyse, Semantische Analyse. Stellen Sie auch hier die korrekte Reihenfolge her.

c. Was geschieht in diesen einzelnen Teilphasen der Übersetzung? (Kurze Stichpunkte genügen.)

Praktische Aufgaben (Blatt 2)

Hinweise

- Die Abgabe der praktischen Aufgaben erfolgt ebenfalls über UniWorX unter der Adresse `https://uniworx.ifi.lmu.de/`.
- Geben Sie Ihren Programmen eindeutige Namen, z.B. `prog-2-3-d.c` für das Programm aus Aufgabe P-2-3, Teilaufgabe d.
- Verwenden Sie die Programmrahmen, soweit diese vorgegeben sind (betrifft künftige Aufgabenblätter).
- Übersetzen Sie Ihre Programme, und testen Sie sie ausführlich.
- Erstellen und verwenden Sie **grundsätzlich in jeder Aufgabe (auch wenn nicht explizit erwähnt) ein Makefile**, mit dem sich Ihre Programme übersetzen lassen. Wenn Sie also beispielsweise die Programme zur Aufgabe P-2-3 abgeben, geben Sie zusätzlich ein Makefile ab, welches in jedem Fall das Ziel `all` enthält. Durch Aufruf von `make all` sollen alle Programme einer Aufgabe übersetzt werden. Bitte halten Sie sich unbedingt an diese Spezifikation, da einige Aufgaben automatisiert getestet werden.
- Falls Teilaufgaben aufeinander aufbauen, speichern Sie bitte den Stand nach jeder Teilaufgabe als eigenes Programm ab und arbeiten dann an einer Kopie weiter, sodass sich die Entwicklung auch im Nachhinein nachvollziehen lässt. Es genügt also nicht, nur das letzte (finale) Programm abzugeben.
- Zulässige Dateiformate für die Abgabe Ihrer Lösungen zu den praktischen Aufgaben sind C-Quelldateien (*.c), Headerdateien (*.h) und zugehörige Makefiles. Objektdateien (*.o) und ausführbare Programmdateien müssen **grundsätzlich nicht** abgegeben werden. Für die Beantwortung von Fragen im Rahmen der praktischen Aufgaben (z.B. Aufgabe P-2-1 d, e, f) erstellen Sie bitte eine Plain-Text-Datei (*.txt).

Aufgabe P-2-1

Für diese Aufgabe legen Sie bitte die folgenden drei Dateien an: p1_main.c, p2_main.c, printfunctions.c

a. Definieren Sie in printfunctions.c einige (mindestens drei verschiedene) Funktionen zur Ausgabe irgendwelcher Texte oder Werte auf der Konsole. Die Datei printfunctions.c soll keine `main(...)`-Funktion enthalten.

b. Definieren Sie in p1_main.c und p2_main.c jeweils eine `main(...)`-Funktion, in der (beliebige) Aufrufe von Funktionen aus printfunctions.c vorkommen.

c. Schreiben Sie ein Makefile, das es möglich macht, durch einen Aufruf entweder nur Programm 1 (`make p1`), nur Programm 2 (`make p2`) oder beide Programme (`make all`) zu übersetzen. Achten Sie vor allem darauf, die Abhängigkeiten für die verschiedenen Ziele korrekt zu definieren.

d. Was passiert, wenn Sie zwei mal direkt hintereinander (`make all`) aufrufen?

e. Öffnen Sie die Datei p1_main.c, fügen Sie an geeigneter Stelle eine weitere Konsolenausgabe hinzu und speichern Sie die Änderungen. Was passiert, wenn Sie jetzt erneut ein (`make all`) aufrufen?

f. Ändern Sie das Compiler-Aufruf-Makro in Ihrem Makefile so, dass beim Übersetzen der Module alle Warnungen ausgegeben werden. Falls Warnungen ausgegeben werden: Um welche Warnungen handelt es sich, und worin liegen jeweils die Ursachen? Verändern Sie die Quelldateien so, dass keine einzige Warnung mehr ausgegeben wird.

Aufgabe P-2-2

Diese Aufgabe soll die Übergabe von Komandozeilenparametern an ein Programm verdeutlichen. Schreiben Sie ein C-Programm, das die eingegebenen Kommandozeilenparameter auf der Konsole formatiert wieder ausgibt. Es soll jeder Parameter mit vorangestellter Nummerierung ausgegeben werden (z.B: `Parameter 1: param1`). Bauen Sie auch eine Fehlerbehandlung ein.

Überlegen Sie sich hierzu Lösungen für die folgenden Teilschritte:

- Wie werden Komandozeilenparameter an ein Programm übergeben?
- Wie kann eine formatierte Ausgabe realisiert werden? In welcher C-Standardbibliothek findet sich eine passende Funktion?
- Wie muss ein Konstrukt aussehen, das immer alle Parameter ausgibt, unabhängig von der tatsächlichen Anzahl?
- Welche Fehler können auftreten, und wie müssen diese behandelt werden?

Bitte denken Sie an ein Makefile!

Aufgabe P-2-3

In dieser Aufgabe sollen Sie mehrere verschiedene Programme entwerfen. Sie können jedoch häufig ein Programm aus einer vorherigen Teilaufgabe wiederverwenden, um es an die neue geforderte Funktionalität anzupassen. Testen Sie alle Ihre Programme sorgfältig!

a. Schreiben Sie ein Programm, das beim Aufruf das Alphabet als Zeichenkette auf der Konsole ausgibt: `abcdefghijklmnopqrstuvwxyz`. Fügen Sie außerdem am Ende einen Zeilenumbruch ein. Verwenden Sie die Funktion `printf()`.

b. Schreiben Sie ein Programm, das die gleiche Ausgabe erzeugt wie das Programm aus Teilaufgabe a, verwenden Sie jedoch diesmal die Funktion `fprintf()`. Tipp: Der Name für den Ausgabe-Strom der Konsole ist `stdout`.

c. Schreiben Sie ein weiteres Programm, welches das Alphabet ausgibt – jedoch diesmal, ohne irgendeine Funktion aus der `*printf()`-Familie zu verwenden (gilt nur für diese Teilaufgabe). Benutzen Sie stattdessen die Funktion `putc()` in einer Schleife mit 26 Durchläufen. Tipp: Sehen Sie in einer ASCII-Tabelle nach, welchen Dezimalwert das Zeichen `'a'` hat. Auch der Zeilenumbruch soll auf diese Weise realisiert werden.

d. Das Alphabet soll nun nicht mehr auf der Konsole ausgegeben werden, sondern in eine neue, durch das Programm selbst anzulegende Datei geschrieben werden, deren Name dem Programm als Aufrufparameter übergeben wird. Beispiel für einen Aufruf: `progname newfile.txt`. Verwenden Sie zum Anlegen der Datei die Systemfunktion `open()`. Sie liefert als Rückgabewert einen Filedeskriptor, den Sie beim Schreiben mittels `write()` benutzen können. Achten Sie jetzt auch auf eine sinnvolle Fehlerbehandlung, und prüfen Sie zu Beginn, ob tatsächlich ein Argument übergeben wurde. Falls nicht, lassen Sie eine entsprechende Meldung (auf der Konsole) ausgeben.

e. Verändern Sie Ihr Programm aus Teilaufgabe d jetzt so, dass das Alphabet nicht nur ein einziges Mal in die neu angelegte Datei geschrieben wird, sondern so oft, bis eine Dateigröße von 1 MByte (= 1048576 Bytes) überschritten wurde. Achtung: Verwenden Sie keine `for`-, sondern eine `while`-Schleife, und zählen Sie die bereits geschriebenen Bytes ständig mit. Geben Sie am Ende auf der Konsole aus, wie viele Zeilen (also wie viele Aphabete) in die Datei geschrieben wurden.

f. Modifizieren Sie das Programm aus Teilaufgabe e so, dass die Dateigröße, bis zu der die neue Datei mit Alphabeten gefüllt wird, nicht statisch 1 MByte beträgt, sondern als frei wählbarer Kommandozeilenparameter an das Programm übergeben werden kann. Ein möglicher Aufruf wäre also `progname newfile.txt 8388608`, um eine 8 MBytes große Datei zu erzeugen. Achtung: Begrenzen Sie die maximale Dateigröße auf einen sinnvollen Wert (z.B. 500 MBytes). Würde der übergebene Parameter diese Grenze überschreiten, soll das Programm mit einer Fehlermeldung beendet werden, ohne dass eine Datei erstellt und geschrieben wird.

Bitte denken Sie an ein Makefile!

Aufgabe P-2-4

Um die folgenden beiden Teilaufgaben zu lösen, können Sie eventuell Teile der Programme aus Aufgabe P-2-3 wiederverwenden.

a. Schreiben Sie ein Programm, welches eine vorhandene Datei kopiert, indem es eine neue Datei erstellt, den Inhalt der vorhandenen Quelldatei gepuffert liest und in die neu zu erstellende Zieldatei schreibt. Ein Aufruf des Programms könnte also so aussehen: `progname sourcefile.txt destfile.txt`. Tipp: Um die Größe der Quelldatei zu ermitteln, inkludieren Sie bitte die Headerdatei `<sys/stat.h>` und verwenden im Programm den folgenden Code:

```
struct stat filestat;
stat(sourcefilename, &filestat);
long filesize = filestat.st_size;
```

Anschließend enthält die Variable `filesize` die Größe der Datei mit dem Dateinamen `sourcefilename` in Bytes.

b. Nun soll folgende Änderung (ausgehend von der vorigen Teilaufgabe) implementiert werden: Statt die Datei einfach zu kopieren, sollen in der Kopie alle Buchstaben durch den nächsten Buchstaben im Alphabet ersetzt werden, also aus a wird b, aus b wird c, ..., aus y wird z und aus z wird wieder a. Analog für Großbuchstaben! Achten Sie darauf, dass alle anderen Zeichen (insb. Ziffern, Leerzeichen und Zeilenumbrüche) unverändert bleiben. Schreiben Sie ein Programm, welches eine vorhandene Quelldatei in der beschriebenen Weise zu einer neuen Datei „verschlüsselt“. Tipp: Eine Möglichkeit ist die Verwendung der Funktion `putc()` beim Schreiben aus dem Puffer in die neue Datei (siehe Aufgabe P-2-3 c).

Bitte denken Sie an ein Makefile!

Aufgabe P-2-5

Schreiben Sie ein C-Programm, das den Inhalt einer **oder mehrerer** Dateien auf dem Bildschirm ausgibt – ähnlich dem Dienstprogramm `cat` unter Linux. Die Namen der auszugebenden Dateien erhält das Programm dabei als Aufrufparameter über die Kommandozeile. Verwenden Sie hierzu wieder die Bibliotheksfunktionen `open()`, `close()`, `read()` und `write()` (elementare E/A-Funktionen). Stellen Sie sicher, dass die Dateien mit `open()` nur zum Lesen geöffnet werden.

Testen Sie das Laufzeitverhalten ihres Programms mit verschiedenen Puffergrößen (1, 2, 4, 16, 64, 512, 4096, 131072 und dem Standardwert `BUFSIZ`), indem Sie eine große Datei (am besten mehrere Megabytes) ausgeben lassen. Verwenden Sie dazu den Befehl `time` und lenken Sie die Ausgabe nach `/dev/null` um (Aufruf: `time prog-2-4 datei > /dev/null`).

Bitte denken Sie an ein Makefile!