Übungsblatt 1

Willkommen zum Praktikum zu Programmieren 3.

Aufgabe 1. Öffnen Sie in Linux eine Shell und lesen Sie die Man-Page zu `man`, `ls` und `cp`. Erstellen Sie mit Emacs eine Datei `hello.c` mit folgendem Inhalt:

```
1 #include <stdio.h>
2
3 int main(void) {
4   printf("Hello C World\n");
5   return 0;
6 }
```

Übersetzen Sie das Programm. Erstellen Sie eine ausführbare Datei `hello` und führen Sie diese aus. Erstellen Sie ein `Makefile` und erzeugen Sie die ausführbare Datei mit `make`. Unterstützen Sie auch sinnvoll die Ziele `run` und `clean`.

Aufgabe 2. Schreiben Sie ein Programm `rot13`, das von Standardeingabe einen beliebigen Text liest und diesen ROT13-verschlüsselt und auf der Standardausgabe ausgibt. ROT13[1] (rotiere um 13 Stellen) ist ein sehr einfacher Verschlüsselungsalgorithmus bei dem alle Zeichen im Bereich `'A'` bis `'Z'` und im Bereich `'a'` bis `'z'` um jeweils 13 Zeichen verschoben werden. Zum Beispiel wird aus `a` das Zeichen `n` und aus `X` wird `K`. Aus dem Text `C ist toll!` wird `P vfg gbyy!`.

Testen Sie Ihr Programm mit einigen Texten indem Sie den Inhalt einer Datei als Eingabe des Programms verwenden und das Ergebnis in eine andere Datei speichern. Beachten Sie, dass die zweimalige Anwendung von rot13 wieder den ursprünglichen Text ergeben sollte. Mit dem Unix-Tool `diff` können Sie Unterschiede zwischen zwei Texten feststellen.

```
cat datei.txt | rot13 | rot13 > datei.rot13
```

Testen Sie Ihr Programm auch mit der Datei `/usr/share/dict/words`.

Aufgabe 3. Schreiben Sie ein Programm `io`, das eine positive ganze Zahl von der Standardeingabe liest und sowohl die Binärdarstellung der Zahl als auch die Dezimaldarstellung ausgibt. Verwenden Sie nur `putchar` zur Ausgabe. Verwenden Sie für die Binärdarstellung Bitoperationen und Iteration. Verwenden Sie für die Dezimaldarstellung Modulo und Rekursion. Testen Sie Ihr Programm mit 0, 1, 16, 255, 4294967295 und 4294967296.

[1] EBG13 jheqr vz Hfrarg va qra seüura 80re Wnuera rvatrfrgmg. Mvry jne avpug qvr Irefpuyüffryhat, fbaqrea qnff zna avpug hatrjbyyg riraghryy "bssrafvir" (oryrvqvtraqr?) Grkgr yrfra zhffgr. Rf jne rva Gnfgraqehpx abgjraqvt qvrfr mh yrfra.

Aufgabe 4. Erstellen Sie folgende C-Quelle `zahlen.c` mit Emacs.

```
#include <stdio.h>
int main(void) {
  int i, a, b, c=0;
  for (i=0; i < 17; i++) {
    a = i ^ 10;
    b = 17 / a + 1;
    c = a + b + c;
  }
  printf("Fertig! Ergebnis = %d\n",c);
  return 0;
}
```

Übersetzen Sie es zu `zahlen` und führen Sie es aus. Es wird nicht funktionieren. Führen Sie es mit `valgrind` aus. Übersetzen Sie es so, dass es fürs Debuggen geeignet ist und debuggen Sie es von Emacs aus. Wo geht es schief? Was könnte der Fehler sein?

Aufgabe 5. Schreiben Sie ein Programm, dass die Quadratwurzel einer `double`-Zahl x annäherungsweise mit dem Heron-Verfahren berechnet und ausgibt. Das Heron-Verfahren berechnet eine Folge von Zahlen a_i, die sich immer stärker an $\sqrt{x}$ annähert. Dabei gilt

$$a_0 = \frac{1+x}{2} \qquad\qquad a_{i+1} = \frac{a_i + \frac{x}{a_i}}{2}$$

Man nähert meist bis zu einem Wert a_{i+1} an, so dass sich a_i und a_{i+1} nur um einen Wert kleiner ε unterscheidet.

Lesen Sie einen Wert x von der Standardeingabe ein mit `scanf` und verwenden Sie für ε den Wert 10^{-7}. Automatisieren Sie mit Ihrem Makefile die Ausführung für die Werte 2.0, 4.0, 7.0, 9.0, 16.0, und 17.0.

Aufgabe 6. Erweitern Sie Ihr Programm der letzten Aufgabe so, dass Sie den Wert für x auch als ersten Parameter von der Kommandozeile akzeptieren. Verwenden Sie dazu folgendes C-Fragment:

```
int main(int argc, char *argv[]) {
  double x;
  if (argc > 1) { /* es gibt Parameter */
    sscanf(argv[1], "%lf", &x);
  } else { /* mit scanf */
```

Hinweis 1. Erstellen Sie für jedes C-Übungsblatt einen eigenen Unterordner in dem Sie ein `Makefile` pflegen, das alle Aufgaben in diesem Ordner aktualisiert.

Hinweis 2. Lesen Sie auszugsweise die Dokumentation in dem Informationsblatt "Umgebung für Entwicklung in C".

`http://www.mi.hs-rm.de/~barth/hsrm/prog3`