Studiengänge: Elektro- und Informationstechnik, Flug- und Fahrzeuginformatik, Informatik, Mechatronik

Prüfung
Grundlagen der Programmierung 1

Prüfer: S. Hahndel, F. Regensburger, U. Schmidt
Prüfungsdauer: 90 Minuten
Hilfsmittel: keine

Studiengang	Dozent	Matrikelnummer	Semester	Raum	Platz

Aufgabe	1	2	3	4	Σ	Note
Punkte						

Aufgabe 1: (Programmverständnis, ca. 25%)

a) Welche Ausgaben erzeugt das folgende Programm?

```c
#include <stdio.h>

int f(int a, int b) {
    printf("a: %d, b: %d\n", a, b);
    if (b == 0) return a;
    return f(b, a % b);
}

int main(void) {
    printf("%d\n", f(342, 162));
    return 0;
}
```

b) Geben Sie die Funktionsprototypen f1 bis f5 für folgendes Programm an:

```c
#include <stdio.h>

typedef struct wort {
    char        *text;
    struct wort *next;
} wort;

int main(void) {
    wort w = {"Wort", NULL}, *woerter = &w;
    w.text[0] = f1(woerter, "@");
    w.text    = f2(&woerter, w.next);
    w.next    = f3(woerter->text, &(woerter->next));
    woerter   = f4(w, woerter->text + 1);
    *woerter  = *f5((*woerter).next, *(w.text));
    return 0;
}
```

c) Die Funktion `kopiere` soll eine Kopie der übergebenen Zeichenkette zurückgeben. Die Funktion ist syntaktisch korrekt, funktioniert aber nicht. Warum nicht?

```c
#define MAX 100

char *kopiere(char s[MAX]) {
    char t[MAX];
    int i;
    for (i = 0; i < MAX; i++) t[i] = s[i];
    return t;
}
```

Aufgabe 2: (Reihungen, ca. 25 %)

In dieser Aufgabe addieren und multiplizieren wir Polynome. Ein Polynom

$$p(x) = p_0 + p_1x + p_2x^2 + \ldots + p_nx^n$$

stellen wir durch eine Reihung p seiner Koeffizienten dar, mit p[i] ≡ p_i. Um Polynome verschiedenen Grades darstellen zu können, markieren wir das Ende der Koeffizienten durch ein spezielles Symbol `ENDE`.

Wir lassen Polynome bis zum Grade `MAX-1` zu; die nachfolgend verlangten Funktionen brauchen aber nicht auf ein Überschreiten des maximales Grades zu prüfen und müssen auch keine Fehlerbehandlung vorsehen.

Das folgende Beispielprogramm definiert zwei Polynome

$$p(x) = 1 + 2x - 3x^3$$
$$q(x) = 2 - x$$

Das Programm bildet das Summenpolynom r = p + q und gibt dieses aus, berechnet dann den Funktionswert von p an der Stelle x = 2 und gibt diesen ebenfalls aus.

Zur Erinnerung: Polynome werden addiert, indem man Terme mit der gleichen Potenz von x addiert. Beispiel:

$$p(x) + q(x) = (1 + 2x - 3x^3) + (2 - x) = 3 + x - 3x^3$$

```
#include <stdio.h>

#define MAX 100
#define ENDE 99999999

int main(void) {
     double p[] = {1, 2, 0, -3, ENDE};
     double q[] = {2, -1, ENDE};
     double r[MAX];

     polyadd (r, p, q);
     polyprint(r);
     printf("p(2) = %.1f\n", f(p, 2));
     return 0;
}
```

`polyprint` gibt die Koeffizienten des übergebenen Polynoms mit einer Nachkommastelle aus:

```
3.0   1.0   0.0   -3.0
```
für $3 + x - 3x^3$

Die Funktion `f` berechnet den Funktionswert eines Polynoms an der Stelle x:

```
p(2) = -19.0
```
Funktionswert von p für x = 2

Implementieren Sie die Funktionen `polyadd`, `polyprint` und `f`.

Aufgabe 3: (Rekursion, ca. 20 %)

Das Pascalsche Dreieck ist die graphische Darstellung der Binomialkoeffizienten. Diese Koeffizienten sind im Dreieck so angeordnet, dass jeder Eintrag gleich der Summe der beiden darüberstehenden Einträge ist:

```
                1
             1     1
          1     2     1
       1     3     3     1
    1     4     6     4     1
 1     5    10    10     5     1
```

a) Schreiben Sie eine Funktion

```
int pascal(int z, int s)
```

die den Eintrag im Pascalschen Dreieck für Zeile z und Spalte s liefert. Die Indices beginnen bei 0. Die Spalten sind hierbei "schief" zu denken, so bildet z.B. der linke Dreiecksrand die Spalte 0.

Hinweis: hier bietet sich eine rekursive Implementierung an, da jeder Eintrag gleich der Summe der beiden im Dreieck über ihm stehenden Einträge ist. Die Einträge für die Randfelder bilden dabei die Rekursionsbasis, da bei ihnen keine zwei Einträge oberhalb ihrer Position vorhanden sind.

b) Schreiben Sie eine Funktion

```
void dreieck(int n)
```

die ein Pascalsches Dreieck der Höhe n in Dreiecksgestalt auf der Konsole ausgibt. Die oben gezeigte Ausgabe wird z.B. durch `dreieck(6)` erzeugt. Zur Erzeugung des Dreiecks ist es notwendig, zwischen zwei Einträgen jeweils einen Leerraum vorzusehen. Einträge und Leerräume sollen eine Feldbreite von 3 Zeichen haben.

Aufgabe 4: (Zeigerreihungen, ca. 30 %)

Das nachstehende Programm zerlegt einen Satz in eine Liste von Wörtern:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define MAX 100

int main(void) {
    char *woerter[MAX];
    int i;

    split(woerter,
          "Das ist ein (ganz) kleines, feines Beispiel!");

    for (i = 0; woerter[i] != NULL; i++)
        printf("%s\n", woerter[i]);

    return 0;
}
```

Implementieren Sie die Funktion `split`, der eine Zeichenkette übergeben wird und die eine Liste aller Wörter zurückgibt, die in dieser Zeichenkette enthalten sind. Für das gezeigte Beispiel liefert das obige Programm die folgende Ausgabe:

```
Das
ist
ein
ganz
kleines
feines
Beispiel
```

Ein `wort` sei hierbei definiert als jede Folge von Zeichen, für die `isalpha` einen Wert ungleich 0 liefert. Die an `split` übergebene Zeichenkette kann also mehrere Wörter, genau ein Wort oder auch gar kein Wort enthalten.

Sie dürfen unterstellen, dass die zu zerlegende Zeichenkette immer weniger als `MAX` Wörter enthält. Wie bei Zeigerreihungen üblich bildet `NULL` den Abschluss der Nutzinformation.

(Ende des Aufgabentextes) Viel Erfolg!