Technische Universität Hamburg-Harburg
Institut für Zuverlässiges Rechnen
Prof. Dr. S.M. Rump

WS 2009/2010

Musterlösung der Klausur zur Zwischen-Prüfung Prozedurale Programmierung

(IIW / IT / CI)

8. Januar 2010

Sie haben 45 Minuten Zeit zum Bearbeiten der Klausur.

Tragen Sie bitte zunächst Ihren Namen, Ihren Vornamen, Ihre Matrikelnummer, Ihr Studienfach und Ihr Studiensemester in **DRUCKSCHRIFT** in die folgenden jeweils dafür vorgesehenen Felder ein.

Name:														
Vorname:														
Matr.-Nr.:								Fach				Sem.		

Es sind alle Aufgaben zu bearbeiten. Insgesamt können bis zu 30 Punkte erreicht werden.

Aufg.	Punkte	Korr.
1		
2		
3		

$\sum$	

Zur Beachtung:
Schreiben Sie bitte weder mit Bleistift noch mit Rotstift.
Es sind keinerlei Hilfsmittel in dieser Klausur zugelassen!
Vergessen Sie nicht den „Vorbehalt“ zu unterschreiben.
Hinweis zum Programmieren:
Programmieren sie die Aufgaben in ANSI-C

Vorbehalt
Ich bin darüber belehrt worden, dass die von mir zu erbringende Prüfungsleistung nur dann bewertet wird, wenn die Nachprüfung durch das Zentrale Prüfungsamt der TUHH meine offizielle Zulassung vor Beginn der Prüfung ergibt.

_____

(Datum, Unterschrift)

Aufgabe 1 (12 Punkte)

Folgendes Programm soll aus einer beim Programmstart angegebenen CSV-Datei (Comma-Separated Values) ganze Zahlen in ein dynamisches Array einlesen und sie auf dem Bildschirm ausgeben.

Leider sind dem Programmierer dabei 12 Fehler unterlaufen. Helfen Sie ihm, diese Fehler zu finden.

Hinweis: `fscanf` liefert `EOF` als Rückgabewert, wenn das Dateiende erreicht ist.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #def N 1000
4
5  void printcsv(FILE *file, int numbers)
6  {
7      int i=0; number;
8
9      if(file == NULL){
10         printf("nichts zum Ausgeben\n");
11         return 0;
12
13     while(fscanf(file,"%d,",&number) != EOF && i < N){
14         numbers[i] = number;
15         printf("%d ",number);
16         i+;
17     }
18     if (i >= N) printf("maximale Zeichenanzahl erreicht\n");
19     printf("\ninsgesamt %c Zahlen ausgelesen\n",i);
20 }
21
22 int main(int argc, char argv)
23 {
24     int *numbers;
25     FILE *file;
26
27     if(argc < 2 ) {
28         printf("Fehler: bitte eine CSV-Datei beim Programaufruf angeben!\n");
29         return -1;
30     }
31     if ((file=fopen(argv[1],"r")) == NULL){
32         printf("Fehler: konnte die Datei %c nicht oeffnen.\n", argv[1]);
33         return -1;
34     }
35     numbers = malloc(N*sizeof(numbers));
36     if (numbers = NULL) {
37         printf("Fehler: konnte benoetigten Speicher nicht reservieren.\n");
38         return -1;
39     }
40     printcsv(file, numbers);
41     fclose(file);
42     return 0
43 }
```

Der angegebene Programmcode enthält insgesamt 12 Fehler. Für jeden korrigierten Fehler wird ein Punkt vergeben. Es werden nur die ersten 12 Korrekturvorschläge bewertet.

Nachfolgend ist der korrekte Code zu sehen.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #define N 1000
4
5  void printcsv(FILE *file, int* numbers)
6  {
7      int i=0, number;
8
9      if(file == NULL){
10         printf("nichts zum Ausgeben\n");
11         return;
12     }
13     while(fscanf(file,"%d,",&number) != EOF && i < N){
14         numbers[i] = number;
15         printf("%d ",number);
16         i++;
17     }
18     if (i >= N) printf("maximale Zeichenanzahl erreicht\n");
19     printf("\ninsgesamt %d Zahlen ausgelesen\n",i);
20 }
21
22 int main(int argc, char **argv)
23 {
24     int *numbers;
25     FILE *file;
26
27     if(argc < 2 ) {
28         printf("Fehler: bitte eine CSV-Datei beim Programaufruf angeben!\n");
29         return -1;
30     }
31     if ((file=fopen(argv[1],"r")) == NULL){
32         printf("Fehler: konnte die Datei %s nicht oeffnen.\n", argv[1]);
33         return -1;
34     }
35     numbers = malloc(N*sizeof(*numbers));
36     if (numbers == NULL) {
37         printf("Fehler: konnte benoetigten Speicher nicht reservieren.\n");
38         return -1;
39     }
40     printcsv(file, numbers);
41     fclose(file);
42     return 0;
43 }
```

Aufgabe 2 (12 Punkte)

a) Die sogenannten Legendre-Polynome erhalten ihre Bedeutung durch ihre vielseitige Anwendbarkeit in der Mathematik und in der theoretischen Physik. Für ein $n \in \mathbb{N}$ kann das n-te Legendre-Polynom $L_n(x)$ wie folgt rekursiv definiert werden:

$$L_n(x) = \begin{cases} 1 & \text{falls } n = 0, \\ x & \text{falls } n = 1, \\ \frac{2n-1}{n} x L_{n-1}(x) - \frac{n-1}{n} L_{n-2}(x) & \text{falls } n > 1. \end{cases}$$

Schreiben Sie eine **rekursive** Funktion f, die für einen Double-Wert `x` und einen Integer-Wert `n` den Funktionswert des n-ten Legendre-Polynoms an der Stelle x berechnet.

```
1   #include <stdio.h>
2
3   /* Definition der Funktion f mit Rueckgabetyp double und zwei
4      Parameter vom Typ double und int. Die Funktion berechnet den
5      Wert des n-ten Legendre-Polynoms an der Stelle x rekursiv.
6   */
7
8   double
9   f(double x, int n)
10  {
11      if (n < 0) {
12          printf("error: n must be >= 0\n");
13          return 1.0/0.0; /* entspricht Infinity */
14      }
15      if (n == 0) return 1; /* rekursiver Aufruf entsprechend */
16      if (n == 1) return x; /* der angegebenen Vorschrift */
17      return (2.0*n-1.0)/n*x*f(x,n-1) - (n-1.0)/n*f(x,n-2);
18  }
19
```

(6 Punkte)

b) Es gibt auch eine nicht-rekursive Darstellung der Legendre-Polynome:

$$L_n(x) = \sum_{k=0}^{kmax} (-1)^k \frac{(2n-2k)!\, x^{n-2k}}{(n-k)!\,(n-2k)!\,k!\,2^n}$$

mit

$$kmax = \begin{cases} \frac{n}{2} & \text{falls } n \text{ gerade} \\ \frac{n-1}{2} & \text{falls } n \text{ ungerade} \end{cases}$$

Schreiben Sie nun eine **nicht-rekursive** Funktion fn, die den Wert $L_n(x)$ berechnet.
Hinweis: zur Berechnung von Fakultäten und Potenzen seien die Funktionen `long fac(int n)` bzw. `double pow(double x, int n)` bereits gegeben.

```
1   #include <math.h>  /* fuer pow-Funktion der Stadard-Bibliothek  benoetigt */
2
3   /* Definition der Funktion fn mit Rueckgabetyp double und zwei
4      Parameter vom Typ double und int. Die Funktion berechnet den
5      Wert des n-ten Legendre-Polynoms an der Stelle x nicht
6      rekursiv.
7   */
8   double
9   fn(double x, int n)
10  {
11      int k, kmax;
12      double val = 0;
13
14      kmax = n%2? (n-1)/2 : n/2; /*bestimme kmax laut Vorgabe */
15
16      for (k = 0; k <= kmax; k++) { /* Verwende die angegebene Formel */
17          val = val + (pow(-1,k)*fac(2*n-2*k)*pow(x,n-2*k))/
18                      (fac(n-k)*fac(n-2*k)*fac(k)*pow(2,n));
19      }
20      return val;
21  }
22
```

(4 Punkte)

c) Erläutern Sie Vor- und Nachteile von rekursiven Funktionen im Vergleich zu nicht rekursiven Funktionen.

Nachteile:

Rekursive Funktion kann für große n sehr langsam werden. Das ist bedingt durch immer wiederkehrender Prozesse, die in Verbindung mit der Kontextvorbereitung zur Funtkionsausführung stehen. Lokale Variablen müssen auf dem Stack angelegt werden, Initialisierung der Register, lokaler Variablen und ev. weiterer Strukturen muss durchgeführt werden usw.. Außerdem wird der Stack stark belastet. Bei hoher Rekursiontiefe, kann es auch zu Stacküberlauf kommen.

Vorteile:

Meistens einfachere Berechnungsvorschrift. Klarere Lösungsstruktur.

(2 Punkte)

Aufgabe 3 (12 Punkte)

a) Nehmen Sie bei dieser Aufgabe an, dass die Fließkommazahlen entsprechend dem IEEE-754 Standard kodiert werden. (Der IEEE-754 Standard beschreibt die in der Vorlesung und in den Übungsaufgaben behandelte Binärdarstellung der Fließkommazahlen.)

Wandeln Sie die angegebene „binäre“ Fließkommazahl in eine Dezimalzahl um. Gegeben sind:

das **Vorzeichenbit:**

1

der **Exponent:** (mit dem Shift 127)

1	0	0	0	0	0	1	0

und die **Mantisse:** (vergessen Sie nicht die implizite Eins)

0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Dezimalzahl: -10.625

Erläuterung:

$$
\begin{aligned}
x &= (-1)^s \, 2^E \, (1 + \sum_{i=1}^{23} \frac{m_i}{2^i}) \\
s &= 1 \\
e &:= E + B = 128 + 2 = 130 \\
E &= e - B = 130 - 127 = 3 \\
m &= \frac{1}{4} + \frac{1}{16} + \frac{1}{64} \\
x &= (-1)^1 \, 2^3 \, (1 + \frac{1}{4} + \frac{1}{16} + \frac{1}{64}) \\
&= -(8 + 2 + \frac{1}{2} + \frac{1}{8}) = -10.615
\end{aligned}
$$

(3 Punkte)

b) Warum können folgende Code-Zeilen zu einer Endlosschleife führen?

```
1    #define N 100
2    ...

13   int i,carray[N];
14   ...

55   for(i=0;i<=N;i++)
56       carray[i]=0;
57   ...
```

Begründung:

Durch die falsche Abbruchbedingung der `for`-Schleife wird hier über Array-Grenzen von `carray` geschrieben. Abhängig vom System und vom Kompiler kann die Adresse der Variablen `i` gerade direkt nach `carray[N-1]` folgen. In solchem Fall wird der Wert in der Variablen `i` durch die Zuweisung `carray[N] = 0` jedes Mal wenn `i==N` mit 0 überschrieben. Die Schleife wird somit nie enden, da die Bedingung `i<=N` bleibt für immer erfüllt.

(1 Punkt)

Welche Ausgabe erzeugt folgende Code-Sequenz?

```
10  short int x = 2, y = 0, z = 4;
11  float f1, f2; double d1, d2;
12  while ( x > 0 ) {x = x*x; y += 1;}
13  printf("%d \n",x); /*Ausgabe: 0 */
14  printf("%d \n",y); /*Ausgabe: 4 */
15  x = 100; y = 11; x = x/y/z;
16  printf("%d \n",x); /*Ausgabe: 2 */
17  f1 = d1 = 0.1; d2 = f2 = 0.1;
18  printf("%u \n",d2 != d1);
19                     /*Ausgabe: 1 */
```

Begründung:

1 -te Ausgabe: $x = 2^{2^y}$, wrap-around bei $y = 4$, $x = 2^{16} = (2^{15}+1)+2^{15}-1 \Rightarrow x = -32767+32767 = 0$

2 -te Ausgabe: s.o. (1 Punkt)

3 -te Ausgabe: Auswertung von links nach rechts: $x/y/z = (100/11)/4 = 9/4 = 2$ (1 Punkt)

4 -te Ausgabe: Wegen $0.1_d = 0.00011001100..._b$, ist 0.1 binär nicht exakt darstellbar. Der Wert wird im jeweiligen Fließkomma-Format nur durch einen Näherungswert best möglich approximiert. Da `f2` als `float` eine niedrigere Genauigkeit als `d1` besitzt, enthält `f2` und somit auch `d2` eine schlechtere Approximation von 0.1 als `d1`. Die beiden Werte sind auf jedenfall nicht gleich! (1 Punkt)

c) Stellen Sie in den ersten drei Zeilen die Dezimalzahlen -432, 234 und -256 im 2-er Komplement (`short int`) binär dar. Die dritte Zeile enthält bereits eine solche Binär-Darstellung. Geben Sie links den entsprechenden Dezimalwert an. Addieren Sie abschließend in der letzten Zeile **alle vier** Zahlen binär.

-432 :	1	1	1	1	1	1	1	0	0	1	0	1	0	0	0	0
$+234$:	0	0	0	0	0	0	0	0	1	1	1	0	1	0	1	0
-256 :	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
-110 :	1	1	1	1	1	1	1	1	1	0	0	1	0	0	1	0

(1 Punkt) (1 Punkt) (1 Punkt) (1 Punkt)

$\sum = -432 + 234 - 256 - 110 = -564$:

1	1	1	1	1	1	0	1	1	1	0	0	1	1	0	0

(1 Punkt)