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

SS 2009

Klausur zur Vorlesung Programmiermethodik

29. Juli 2009

Sie haben 90 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.	

Bei jeder der fünf Aufgaben können 12 Punkte erreicht werden, insgesamt also 60 Punkte. Es sind alle Aufgaben zu bearbeiten. Zum Bestehen der Klausur sind 30 Punkte erforderlich.

Aufg.	Punkte	Korr.
1		
2		
3		
4		
5		
$\sum$		

Note	

Zur Beachtung:
Schreiben Sie bitte weder mit Bleistift noch mit Rotstift. Es sind keinerlei Hilfsmittel in dieser Klausur zugelassen! Das Schreiben vor dem Startsignal und auch das Schreiben nach dem Endsignal führt ohne weitere Warnung sofort zur Ungültigkeit der Klausur. Dies gilt auch für das Schreiben von Namen und Matrikelnummer nach dem Endsignal. 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)

Ein Student sollte ein C-Programm schreiben, mit dem sich die Häufigkeit eines vom Benutzer eingegebenen Wortes in der übergebenen Textdatei zählen lässt. Der Student hat folgendes Programm geschrieben. Nur leider lässt es sich nicht kompilieren. Sein Tutor fand genau 12 syntaktische bzw. semantische Fehler im Programm. Finden Sie diese Fehler.

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

```
1   #def N 128
2   #include <stdio.h>
3   #include <stdlib.h>
4   *include <string.h>
5
6   int main(int argc, char **argv[]) {
7           char *inword, *sword;
8           int i = 0; sum = 0;
9           FILE wordfile;
10
11          if ( argc < 2 ) {
12                  printf("Falscher Programmaufruf: searchword(filename) erwartet.\n");
13                  return -1;
14
15          inword = malloc(sizeof(char)*N);
16          sword = malloc(sizeof(char)*N);
17          wordfile = fopen(argv[1],'r');
18          if (wordfile == NULL) {
19                  printf("Fehler: konnte die Datei %s nicht oeffnen.\n",argv[1]);
20                  return -2;
21          }
22          printf("Bitte geben Sie das gesuchte Wort ein: >>");
23          scanf("%s",*sword);
24          while ( i < (N-1) && (inword[i] = fgetc(wordfile)) != EOF ) {
25                  if ( inword[i] == ' ' || inword[i] == '\n' ) {
26                          inword[i] = '\0';
27                          i = 0;
28                          if ( strcmp(inword,sword) = 0 ) {
29                                  sum+;
30                          }
31                  }
32                  else i++;
33          }
34          if ( i >= N ) {
35                  printf("Fehler:\n");
36                  printf("die Datei enthaelt ein Wort laenger als %d Zeichen.\n",  );
37                  return -3;
38          }
39          else {
40                  printf("Das Wort %c kommt %d Mal in der Datei vor.\n",sword,sum);
41          }
42
43          return 0;
44  }
45
46
```

Zeile	Notieren Sie die korrigierten Zeilen (maximal 12) hier !

Aufgabe 2 (12 Punkte)

Die „Ackermannfunktion“, benannt nach ihrem Erfinder Wilhelm Ackermann[1], dient in der theoretischen Informatik als ein Beispiel für eine Klasse von Funktionen, die berechenbar aber nicht primitiv-rekursiv sind. Eine wichtige Eigenschaft dieser Funktion ist, dass sie extrem schnell wächst. In vereinfachter Version wird die Ackermannfunktion wie folgt definiert:

$$A(m,n) = \begin{cases} n+1 & \text{falls } m = 0 \\ A(m-1,1) & \text{falls } m > 0 \text{ und } n = 0 \\ A(m-1, A(m,n-1)) & \text{sonst.} \end{cases}$$

a) Schreiben Sie eine **rekursive** Funktion `A(m,n)`, die für gegebene Integer-Werte m und n den Funktionswert der Ackermannfunktion berechnet. Beachten Sie bei der Wahl des Datentyps für die Ein- und Rückgabeparameter, dass die Funktion sehr schnell große Werte annehmen kann.

Notieren Sie die Funktion hier !

b) Begründen Sie Ihre Wahl des in a) gewählten Datentyps. Gehen Sie dabei insbesondere auf den Wertebereich und gegebenenfalls die Genauigkeit des Datentyps ein.

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

[1] Wilhelm Ackermann (*29.3.1896 - †24.12.1962) war ein deutscher Mathematiker. Er studierte von 1914 - 1924 mit Unterbrechungen durch den Ersten Weltkrieg Mathematik, Physik und Philosophie an der Universität Göttingen. Er war ein Schüler von David Hilbert in Göttingen und wurde berühmt durch die nach ihm benannte Ackermannfunktion.

Aufgabe 3 (12 Punkte)

a) Betrachten Sie folgendes Programm:

```
1   #include <stdio.h>
2
3   void swp1(int a, int b) {
4           int tmp = a;
5           a = b;
6           b = tmp;
7   }
8
9   void swp2(int *a, int *b) {
10          int tmp = *a;
11          *a = *b;
12          *b = tmp;
13  }
14
15  int main() {
16          int a = 0, b = 1;
17          printf("a = %d, b = %d\n",a,b);
18          swp2(&a,&b); printf("swp2(a,b) ergibt: a = %d, b = %d\n",a,b);
19          swp1(a,b);   printf("swp1(a,b) ergibt: a = %d, b = %d\n",a,b);
20          return 0;
21  }
22
```

Ergänzen Sie die Ausgabe des Programms an den angegebenen Stellen.

Ausgabe:

```
17  a = 0, b = 1
18  swp2(a,b) ergibt: a = __, b = __
19  swp1(a,b) ergibt: a = __, b = __
```

b) Erklären Sie den Unterschied zwischen Call-by-Value und Call-by-Reference.

c) Definieren Sie einen *neuen* strukturierten Datentyp `znumber` mit zwei Datenfeldern `real` und `imag` vom Typ `double` zur Speicherung von komplexen Zahlen:

```
1   ______________ _____________________________________ /* Typdefinition/Struktur */
2                  _____________________________________ /* Realteil */
3                  _____________________________________ /* Imaginaerteil */
4   } znumber;
```

Vervollständigen Sie die nachfolgende Funktion zur Summation zweier komplexen Zahlen vom Typ `znumber`.

```
0   znumber *sum(znumber *z1, znumber *z2) {
1           znumber *newz = ____________________________ /* reserviere Speicherplatz   */
2           newz->real =    ____________________________ /* summiere die Realteile    */
3           newz->imag =    ____________________________ /* summiere die Imaginaerteile */
4           return ___________                           /* gebe das Ergebnis zurueck  */
5   }
```

Aufgabe 4 (12 Punkte)

a) Stellen Sie in den ersten drei Zeilen die Dezimalzahlen -777, 345 und -102 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.

-777 :																
$+345$:																
-102 :																
	:	1	1	1	1	1	1	1	1	1	0	1	0	1	1	0	0
$\sum$	:																

b) Welche Ausgabe erzeugt die folgende Code-Sequenz?

```
100  short int x = 2, y = 0, z = 4; float f1, f2; double d1, d2;
101  while ( x > 0 ) {x = x*x; y += 1;}
102  printf("%d \n",x);                          /* 1. Ausgabe: _________________ */
103  printf("%d \n",y);                          /* 2. Ausgabe: _________________ */
104  x = 100; y = 11; x = x/y/z;
105  printf("%d \n",x);                          /* 3. Ausgabe: _________________ */
106  f1 = d1 = 0.1; d2 = f2 = 0.1;
107  printf("%u \n",d2 != d1);                   /* 4. Ausgabe: _________________ */
```

Begründen Sie Ihre Antwort:

_____

_____

_____

_____

_____

_____

_____

_____

_____

c) Stellen Sie die Dezimalzahl -34.125 im IEEE-754 Standard dar:

Vorzeichenbit:

Exponent: (mit dem Shift B=127)

Mantisse: Denken Sie an die implizite Eins !

Aufgabe 5 (12 Punkte)

a) Ein Unternehmen will die Personalnummern seiner Mitarbeiter in einer verketteten Liste verwalten. Definieren Sie hierfür eine Struktur `sEmpList`, welche die Komponenten `PersNr` für die ganzzahlige Personalnummer und `next` für den Zeiger auf den Nachfolger besitzt.

Notieren Sie die Definition der Struktur `sEmpList` hier!

b) Schreiben Sie eine Funktion `NewEmployee`, die ein neues Element mit einer neuen Personalnummer am Anfang einer verketteten Liste mit der Struktur aus a) einfügt. Eingabeparameter sind ein Zeiger `E` auf den Listenanfang und die neue Personalnummer `PersNr`. Rückgabewert soll ein Zeiger auf den Anfang der erweiterten Liste sein.

Notieren Sie die Funktion `NewEmployee` hier!

c) Eine Methode um Zugriffszeiten auf häufig angefragte Listenelemente zu verringern besteht bei so genannten adaptiven Listen darin, bei jedem Zugriff auf ein Element dieses mit seinem Vorgänger zu vertauschen. Schreiben Sie eine Funktion `swap`, die ein Listenelement mit seinem Vorgänger vertauscht, ohne schreibend auf Felder `PersNr` zuzugreifen. Die Funktion besitze die gleichen Ein- und Ausgabeparameter wie `NewEmployee` aus b).

Notieren Sie die Funktion `swap` hier!