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

WS 2010/2011

Musterlösung zu Probeklausur Prozedurale Programmierung

(IIW / ET / CI / MC)

6. Januar 2011

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 eines Studenten soll bis zu 100 aus maximal 20 Zeichen bestehende Wörter von der Tastatur einlesen, sie alphabetisch nach der Insertionsort-Methode sortieren und anschließend am Bildschirm ausgeben. Frühzeitiges Eingabeende soll durch die Eingabe von `'!'` oder `'@'` erfolgen. Den Fehlermeldungen des Compilers können wir entnehmen, dass in dem Programm genau 12 syntaktische oder semantische Fehler vorhanden sind. Versuchen Sie alle Fehler zu finden. Für jeden korrigierten Fehler wird ein Punkt vergeben. Es werden nur die ersten 12 Korrekturvorschläge bewertet.

```
1  #include <stdio.h>      /* Korrigierter Code */
2  #define N 100
3  #define L 21
4
5  int strcompare(char *a, char *b) {
6      int i = 0;
7      while (a[i] != '\0' && b[i] != '\0' && a[i] == b[i]) ++i;
8      return (a[i] > b[i]);
9  }
10
11 void strcopy(char *s, char *scpy) {
12     int i = -1;
13     do{
14         ++i;
15         scpy[i] = s[i];
16     } while (s[i] != '\0');
17 }
18
19 void insert_sort(char a[][L]) {
20     char tmp[L];
21     int i, j;
22     for (i = 1; i < N && a[i][0] != '\0'; ++i) {
23         j = i;
24         strcopy(a[i],tmp);
25         while ( j > 0 && strcompare(a[j-1],tmp)) {
26             strcopy(a[j-1],a[j]);
27             --j;
28         }
29         strcopy(tmp,a[j]);
30     }
31 }
32
33 void print_array(char a[][L]) {
34     int i;
35     for (i=0; i < N && a[i][0] != '\0'; ++i) printf("%s\n",a[i]);
36 }
37
38 int main(void) {
39     char input[N][L] = {{0}};
40     int i = 0;
41     while (i < N && scanf("\n%20s",input[i]) && input[i][0] != '!' && input[i][0] != '@') ++i;
42     input[i][0]='\0';
43     insert_sort(input);
44     print_array(input);
45     return 0;
46 }
```

Zeile	Schreiben Sie die korrigierten Zeilen hier rein
3	`#define L 21` (20 Zeichen plus 1 für das Stringende-Zeichen)
11	`void strcopy(char *s, char *scpy)` (es soll hier ein String übergeben werden)
20	`char tmp[L];` (Arrays nicht definierter Länge sind in C nicht erlaubt)
21	`int i, j;` (eine Deklaration wird mit Semikolon abgeschlossen)
22	`for (i = 1; i < N && a[i][0] != '\0'; ++i)` ("\0" ist kein Zeichen)
27	`--j;` (`-j-;` (ist ein nicht vollständiger Ausdruck)
29	`strcopy(tmp,a[j]);` (wohl ein Tippfehler gewesen)
35	`for (i=0; i < N && a[i][0] != '\0'; ++i) printf("%s\n",a[i]);` (String-Ausgabe)
36	`}` (Funktionsende)
43	`insert_sort(input);` (`insert_sort` erwartet einen Zeiger auf ein Array der Länge L)
45	`return 0;` (außer bei `void`-Funktion ist `return` ohne Rückgabewert nicht erlaubt)

Aufgabe 2 (12 Punkte)

Eine Maus befindet sich in der Mitte eines quadratischem Labyrinths und sucht einen Ausgang. Sie kann dabei nur senkrechte und waagererechte Schritte machen, ist jedoch in der Lage, Felder zu markieren und Markierungen wieder zu entfernen. Das Labyrinth wird in einem Rahmenprogramm durch ein globales zweidimensionales `char`-Array `L` mit 11 Zeilen und Spalten dargestellt. Mauern werden mit `'*'`-Symbolen und Wege durch Leerzeichen gekennzeichnet, siehe Abbildung. Das `'o'` in der Mitte symbolisiert die Maus.

Norden

*	*	*	*	*	*	*	*	*	*	*
*	*				*		*			*
*		*	*				*			*
*			*		*			*		*
*	*		*			*				*
*					o	*		*	*	*
*		*		*	*	*		*		
*		*	*					*		*
*		*		*		*	*	*		*
		*								*
*	*	*	*	*	*	*	*	*	*	*

Westen — Osten

Süden

Schreiben Sie eine rekursive Funktion `int maus(int x, int y)`, welche die im Folgenden beschriebene Suchstrategie der Maus implementiert. Die Eingabeparameter `x,y` sind die Zeilen- und Spaltennummer des Feldes, auf dem sich die Maus befindet. Dieses Feld wird anfangs mit einem `'o'` markiert. Ist es ein Ausgang aus dem Labyrinth, so werden `x` und `y` auf dem Bildschirm ausgegeben und die Funktion mit dem Rückgabewert `1` ("Ausgang gefunden") beendet. Andernfalls versucht die Maus nacheinander, nach Osten, Norden, Westen, Süden zu laufen. Stößt sie dabei auf ein bisher unbesuchtes freies Feld, so läuft sie in diese Richtung durch einen rekursiven Aufruf. Führt sie dies zu einem Ausgang, so werden `x` und `y` ausgegeben und die Funktion mit dem Rückgabewert `1` beendet. Wenn kein rekursiver Aufruf zu einem Ausgang führt oder die Maus kein freies unbesuchtes Feld findet, so wird die anfängliche Markierung des Feldes wieder gelöscht und die Funktion mit dem Rückgabewert 0 ("Ausgang nicht gefunden") beendet.

```
int maus(int x, int y){
  int a=0; /* lokale Variable zum Speichern von Rueckgabewerten rekursiver Aufrufe von maus */
  L[x][y]='o'; /* aktuelles Feld anfangs markieren */
  if(x==0 || y==0 || x==10 || y==10){ /* Randpunkt erreicht -> Ausgang gefunden */
    printf("(%d,%d)<-",x,y);
    return 1;
  }
  if(L[x][y+1]==' ') a = maus(x,y+1); /* nach Osten gehen */
  if(a==0 && L[x-1][y]==' ') a = maus(x-1,y); /* nach Norden gehen */
  if(a==0 && L[x][y-1]==' ') a = maus(x,y-1); /* nach Westen gehen */
  if(a==0 && L[x+1][y]==' ') a = maus(x+1,y); /* nach Sueden gehen */
  if(a) printf("(%d,%d)<-",x,y); /* Pfad von (x,y) aus Labyrinth gefunden -> Ausgabe (x,y) */
  else L[x][y]=' '; /* kein Pfad von (x,y) aus Labyrinth gefunden -> (x,y) entmarkieren */
  return a; /* a=1 falls Ausgang gefunden, a=0 sonst */
}
```

Aufgabe 3 (12 Punkte)

a) Stellen Sie in den ersten drei Zeilen die Dezimalzahlen 543, −177, −233 im 2-er Komplement (`short int`) binär dar. Die vierte 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.

$+543$:	0	0	0	0	0	0	1	0	0	0	0	1	1	1	1	1
$+177$:	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	1
-177 :	1	1	1	1	1	1	1	1	0	1	0	0	1	1	1	1
$+233$:	0	0	0	0	0	0	0	0	1	1	1	0	1	0	0	1
-233 :	1	1	1	1	1	1	1	1	0	0	0	1	0	1	1	1
$\boxed{-300}$:	1	1	1	1	1	1	1	0	1	1	0	1	0	1	0	0
$\sum = -167$:	1	1	1	1	1	1	1	1	0	1	0	1	1	0	0	1

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

b) Welche Ausgabe erzeugt folgende Code-Sequenz ?

```
101   short int k=-32;
102   k=k*k*k-1;
103   printf("%d \n",k);          /* 1. Ausgabe: 32767     1 Punkt */
104   k=3*k;
105   printf("%d \n",k);          /* 2. Ausgabe: 32765     1 Punkt  */
106   k=65;
107   k=(k/(k/2))*(k/2);
108   printf("%d \n",k);          /* 3. Ausgabe: 32766     1 Punkt  */
```

Begründen Sie Ihre Antwort:

1. Ausgabe: $(-32)^3 - 1 = ((-2)^5)^3 - 1 = -2^{15} - 1 = -32768 - 1 =$ (wrap around) 32767
2. Ausgabe: $3 * 32767 = 32767 + 32767 + 32767 = (32767 + 1) + 32766 + 32767 =$
   (wrap around) $-32768 + 32766 + 32767 = -2 + 32767 = 32765$
3. Ausgabe: $(65/(65/2)) * (65/2) =$(int-Division)$(65/32) * 32 = 2 * 32 = 64$

(3 Punkte)

c) Bestimmen Sie die Dezimalzahl, die durch folgende Binärdarstellung im IEEE-754 Standard angegeben ist.

Vorzeichenbit:

1

Exponent: (mit dem Shift 127)

1	0	0	0	0	0	1	0

Mantisse: Denken Sie an die implizite Eins !

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

Dezimalzahl : $-12,25$

(4 Punkte)