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

WiSe 2016/17

Midterm Musterklausur Prozedurale Programmierung

12. Januar 2016

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.		

Es sind alle Aufgaben zu bearbeiten. Die maximal erreichbare Gesamtpunktzahl beträgt 60 Punkte. Zum Bestehen der Klausur sind 30 Punkte erforderlich.

Aufgabe	Punkte	Korrektur
1		
2		
3		
4		
5		
$\sum$		

Bonus	
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.
Programmieren Sie die Aufgaben in ISO-C oder 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)

Beantworten Sie die jeweiligen Fragen zu den Codezeilen.

Hinweise:

- Eine vollständig korrekt beantwortete Frage wird mit +1 Punkt, eine falsch beantwortete Frage mit −1 Punkt und eine unbeantwortete Frage mit 0 Punkten bewertet.
- Uneindeutige Antworten, insbesondere uneindeutige Korrekturen gelten als falsche Antwort.
- Die minimal mögliche Punktezahl ist 0 Punkte.

1. Öffnet die folgende Anweisung eine Datei zum Anhängen von Daten?

☑ Ja ☐ Nein

```
FILE* fp = fopen("data.txt", "a");
```

2. Bestimmt folgendes Makro die Summe zweier Integer-Zahlen?

☐ Ja ☑ Nein

```
#define SUM(int A, int B) (A) + (B)
```

3. Sind die Inhalte vom Array `f` am Ende folgender Codezeilen `{ 5.0, 0.0 }`?

☐ Ja ☑ Nein

```
char* str = "5.0";
float f[] = { 0.0, 0.0 };
sscanf(str, "%f", &f[1]);
```

4. Lautet die `printf`-Ausgabe „Autohof“ ?

☑ Ja ☐ Nein

```
char str[] = "Autobahnhof";
str[4] = '\0';
printf("%s%s", str, str+8);
```

5. Ist folgende Schleife eine Endlosschleife?

☑ Ja ☐ Nein

```
unsigned i = 0;
do {
  i--;
} while (i >= 0);
```

6. Ist folgende Array-Manipulation zulässig?

☐ Ja ☑ Nein

```
int arr[] = { 1, 2, 3, 4, 5 };
for (int i = 4; i >= 0; i--) {
  arr[i-1] = arr[i];
}
```

7. Ist folgende Struktur für die Erstellung einer einfach verketteten Liste anwendbar?

☐ Ja ☑ Nein

```
struct a {
  int x,
  int y,
  struct a* next;
};
```

8. Lautet die `printf`-Ausgabe „2 “ ?

☐ Ja ☑ Nein

```
int arr[] = { 1, 2, 3 };
printf("%d", *(arr+2));
```

9. Ist folgende String Initialisierung korrekt?

☑ Ja ☐ Nein

```
char str[] = {'H','a','l','l','o','\0'};
```

10. Hat `i` am Ende folgender Codezeilen den Wert „4 “ ?

☐ Ja ☑ Nein

```
int i = 2;
switch(i) {
  case 1: i++;
          break;
  case 2: i += 2;
  default: i += 3;
}
```

11. Gibt folgendes Programm den Programmnamen aus?

☑ Ja ☐ Nein

```
#include <stdio.h>
int main(int argc, char* argv[]) {
  printf("%s", argv[0]);
  return 0;
}
```

12. Ist der folgende Funktionszeiger syntaktisch korrekt und passend für eine Funktion mit der Signatur `int addDouble(int i, double d);` ?

☑ Ja ☐ Nein

```
int (*fcn_ptr)(int, double);
```

Aufgabe 2 (12 Punkte)

a) Schreiben Sie eine rekursive Funktion, die auf einem global definierten zweidimensionalen Array `FELD` der Größe $N \times N$ alle Buchstaben `'A'` findet und alle Positionen nur einmal auf dem Bildschirm ausgibt. Die Funktion soll zwei Startkoordinaten $0 \leq$ `x,y` $< N$ übergeben bekommen. Das Feld ist nur mit Großbuchstaben gefüllt und darf von Ihrer Funktion beliebig verändert werden.

Schließen Sie Ihre Implementation an den folgenden Code an.

b) Schreiben Sie eine `main`-Funktion, die Ihre Funktion mit geeigneten Startkoordinaten aufruft.

```
1  #include <stdio.h>
2
3  #define N 5
4
5  char FELD[N][N] = {
6    {'D','T','U','B','A'},
7    {'L','A','I','R','F'},
8    {'P','E','M','K','V'},
9    {'F','P','A','G','Z'},
10   {'Q','K','J','Y','A'}
11 };
12
13 void findA(int x, int y) {
14   if (FELD[x][y] == 'A') {
15     printf("%d - %d\n", x, y);
16   }
17   FELD[x][y] = 'x';
18   if (((x-1) >= 0) && (FELD[x-1][y] != 'x')) {
19     findA(x-1, y);
20   }
21   if (((x+1) < N) && (FELD[x+1][y] != 'x')) {
22     findA(x+1, y);
23   }
24   if (((y-1) >= 0) && (FELD[x][y-1] != 'x')) {
25     findA(x, y-1);
26   }
27   if (((y+1) < N) && (FELD[x][y+1] != 'x')) {
28     findA(x, y+1);
29   }
30 }
31
32 int main() {
33   findA(0, 0);
34   return 0;
35 }
```

Aufgabe 3 (11 Punkte)

a) Schreiben Sie eine Funktion, die zu einer gegebenen Kantenlänge den Umfang und den Flächeninhalt eines Quadrats berechnet und an die aufrufende Funktion zurückgibt. Schreiben Sie eine `main`-Funktion, die Ihre Funktion mit geeigneten Parametern aufruft. Welche zwei Möglichkeiten der Parameterübergabe haben Sie? Erklären Sie anhand der Funktion beide Möglichkeiten!

```
1  #include <stdio.h>
2
3  void quadrat(double l, double* U, double* F) {
4    *U = 4*l;
5    *F = l*l;
6  }
7
8  int main() {
9    double l = 3.0, U, F;
10   quadrat(l, &U, &F);
11   printf("Ein Quadrat mit Kantenlaenge l=%f ", l);
12   printf("hat den Umfang U=%f und den Flaecheninhalt F=%f\n", U, F);
13   return 0;
14 }
```

1. Call-by-value: eine Kopie des Wertes der Variable wird übergeben. Der Wert in der aufrufenden Funktion bleibt unverändert. Die Gültigkeit der Kopie endet mit dem Funktionsrücksprung. Angewendet in der Funktion `quadrat` für die Variable l.
2. Call-by-Reference: eine Kopie der Adresse der Variable in der aufrufenden Funktion wird übergeben. Der Wert der Variable kann in der aufrufenden Funktion verändert werden. Auch mehrere Rückgabewerte können so in C realisiert werden, wie in der Funktion `quadrat` für die Parameter `U` und `F`.

b) Implementieren Sie unter dem folgenden Code die fehlende Funktion `print_array`. Diese hat als Eingabeparameter einen Zeiger auf das auszugebende Array, die Arraylänge und einen Zeiger auf eine Funktion (z.B. `print_reverse`), welche die ersten beiden Eingabeparameter selbst als Eingabeparameter hat.

Hinweis: die geforderte Implementation von `print_array` ist mit drei Codezeilen realisierbar!

```
1  #include <stdio.h>
2
3  void print_reverse(int* a, int n) {
4    for (int i = n-1; i >= 0; i--) {
5      printf("%d ", a[i]);
6    }
7  }
8
9  void print_array(int* a, int n, void (*print)(int*, int)) {
10   print(a, n);
11 }
12
13 int main() {
14   int a[] = { 1, 2, 3, 4, 5 };
15   print_array(a, 5, print_reverse);
16   return 0;
17 }
```

Aufgabe 4 (12 Punkte)

Schreiben Sie ein vollständiges Programm, das aus den Eingabedaten, X-Koordinate, Y-Koordinate und Symbol, eine quadratische Karte erstellt. Die Eingabedaten werden aus einer Textdatei eingelesen und die vollständige Karte wird in eine zweite Textdatei geschrieben. Beide Dateinamen werden als Kommandozeilenargumente übergeben, z.B.

```
prog.exe input.txt output.txt
```

Da die Eingabedaten nicht sortiert sind und die Kartengröße variabel ist, bietet es sich an, die Karte zunächst in ein dynamisches zweidimensionales Array zwischenzuspeichern. Achten Sie auf die korrekte Index-Nutzung! Die ersten beiden Werte der Eingabedaten sind die Anzahl der Symbole und die Kartengröße.

Im Beispiel: 3 Symbole mit Positionen, aus der eine 5×5 Karte erstellt werden soll.

```
input.txt:          output.txt (ohne Rahmen):

   3 5                 +-----+
   1 5 S               |    S|
   4 2 F               |  X  |
   2 3 X               |     |
                       | F   |
                       |     |
                       +-----+
```

```c
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char* argv[]) {
  FILE* fp = fopen(argv[1], "r");
  int zeilen = 0;
  int n = 0;
  fscanf(fp, "%d %d", &zeilen, &n);

  // optional: eindimensional
  char** karte = (char**) malloc (n * sizeof(char*));
  for (int i = 0; i < n; i++) {
    karte[i] = (char*) malloc (n * sizeof(char));
    for (int j = 0; j < n; j++) { // optional
      karte[i][j] = ' ';          // optional
    }                             // optional
  }

  for (int i = 0; i < zeilen; i++) {
    int x = 0;
    int y = 0;
    char c = 0;
    fscanf(fp, "%d %d %c", &x, &y, &c);
    karte[x-1][y-1] = c;
  }

  fclose(fp);

  fp = fopen(argv[2], "w");
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      fprintf(fp, "%c", karte[i][j]);
    }
    fprintf(fp, "\n");
  }
  fclose(fp);

  for (int i = 0; i < n; i++) {
    free(karte[i]);
  }
  free(karte);

  return 0;
}
```

Aufgabe 5 (13 Punkte)

a) Definieren Sie eine Struktur für die Speicherung eines Highscores in Form einer doppelt verketteten Liste. Die Struktur soll den Spielernamen, die erreichten Punkte und die Spielzeit speichern. Schreiben Sie eine `main`-Funktion und allozieren Sie dynamisch Speicher für zwei Elemente. Initialisieren Sie alle Zeiger der Liste mit geeigneten Werten!

b) Schreiben Sie eine Funktion `position_up`, die einen Zeiger auf ein Element der doppelt verketteten Liste bekommt und dieses Element eine Position in Richtung des Listenanfangs verschiebt. Die Funktion soll den neuen Listenanfang zurückgeben.

c) Schreiben Sie eine Funktion, welche die doppelt verkettete Liste nach Punktestand absteigend sortiert. Die Funktion `position_up` aus Aufgabenteil b) kann als korrekt implementiert angenommen werden und ermöglicht eine sehr kurze Implementation!

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  // zu a)
5  typedef struct highscore {
6    char name[51];
7    int punkte;
8    int zeit;
9    struct highscore* next;
10   struct highscore* prev;
11 } hs;
12
13 // zu b)
14 hs* position_up(hs* elem) {
15   if ((elem != NULL) && (elem->prev != NULL)) { // Leer oder Listenanfang?
16     hs* elem1 = elem->prev->prev;
17     hs* elem2 = elem;
18     hs* elem3 = elem->prev;
19     hs* elem4 = elem->next;
20     if (elem1 != NULL) {
21       elem1->next = elem2;
22     }
23     elem2->prev = elem1;
24     elem2->next = elem3;
25     elem3->prev = elem2;
26     elem3->next = elem4;
27     if (elem4 != NULL) {
28       elem4->prev = elem3;
29     }
30     for (; elem->prev != NULL; elem = elem->prev); // Gehe zum Listenanfang
31   }
32   return elem;
33 }
34
```

```
35 // zu c)
36 void sort(hs* list) {
37   while (list->next != NULL) {
38     if (list->punkte < list->next->punkte) {
39       list = position_up(list->next);
40     } else {
41       list = list->next;
42     }
43   }
44 }
45
46 // zu a)
47 int main () {
48   hs* start = (hs*) malloc (sizeof(hs));
49   start->prev = NULL;
50   start->next = (hs*) malloc (sizeof(hs));
51   start->next->prev = start;
52   start->next->next = NULL;
53
54   return 0;
55 }
```