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

WiSe 2017/18

Midterm Musterklausur Prozedurale Programmierung

21. Dezember 2017

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 Lesen?

☑ Ja ☐ Nein

```
FILE* fp = fopen("input.txt", "r");
```

2. Lautet die `printf`-Ausgabe „2“?

☐ Ja ☑ Nein

```
int a[] = { 1, 2, 3, 4 };
printf("%d", *(a + a[3] / 2));
```

3. Stellt folgender Befehl Speicher für vier **double**-Werte bereit?

☐ Ja ☑ Nein

```
double* d = (double*) malloc (double * 4);
```

4. Ist folgende **while**-Schleife eine Endlosschleife?

☐ Ja ☑ Nein

```
unsigned i = 1;
while (i) {
  printf("loop %d", i++);
}
```

5. Ist `substr` ein Zeiger auf die null-terminierte Zeichenkette „bahn“?

☑ Ja ☐ Nein

```
char str[] = "autobahnhof";
char* substr = str + 4;
substr[4] = '\0';
```

6. Bestimmt folgende rekursive Funktion die Länge einer null-terminierten Zeichenkette?

☑ Ja ☐ Nein

```
int strlen(char* str) {
  return ( *str ? 1 + strlen(str + 1) : 0 );
}
```

☑ Ja ☐ Nein

7. Ist der Funktionszeiger `double (*fp)(double*, int);` für eine Funktion mit der Signatur `double f(double* a, int n);` anwendbar?

☑ Ja ☐ Nein

8. Ist folgende Struktur für die Erstellung einer einfach verketteten Liste geeignet?

```
struct List {
  int id;
  char name[50];
  struct List* next;
};
```

☐ Ja ☑ Nein

9. Gibt folgendes Programm das letzte Kommandozeilenargument aus?

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

☐ Ja ☑ Nein

10. Lautet die `printf`-Ausgabe „gerade “ ?

```
int i = 2;
switch (i % 2) {
  case 1: printf("ungerade");
    break;
  case 2: printf("gerade");
    break;
  default: printf("Fehler");
    break;
}
```

☑ Ja ☐ Nein

11. Wird durch die folgende Schleife jedes Array-Element von `a` um Eins verringert?

```
int a[] = { 1, 2, 3, 4 };
for (int i = 3; a[i]--; i--);
```

☐ Ja ☑ Nein

12. Werden die Werte der Zeiger `a` und `b` in der aufrufenden Funktion vertauscht?

```
void swap(int* a, int* b) {
  int* t = a;
  a = b;
  b = t;
}
```

Aufgabe 2 (12 Punkte)

Gegeben sei ein global definiertes zweidimensionales Array `FELD` der Größe $M \times N$. Jedes Element dieses Arrays sei mit einem Punkt `'.'` oder einem großen `'X'` gefüllt, wie folgende Abbildung zeigt:

```
X . . . . . . X
X X . X . . . X
. X . X . X X X
. X X X . X . .
. . . . . X . .
```

Oben, unten, links und rechts zusammenhängende `'X'`-Buchstaben bilden eine Kurve. Somit sind in der Abbildung zwei Kurven gegeben, wobei der linken Kurve die Länge 9 und der rechten Kurve die Länge 7 zugeordnet wird. Schließen Sie Ihre Implementierung an den untenstehenden Code an.

a) Schreiben Sie eine rekursive Funktion, welche die Länge einer zusammenhängenden Kurve durch das Zählen der `'X'`-Buchstaben ermittelt und zurück gibt. Die Kurve wird durch einen beliebigen Startpunkt (m, n), der sich auf einer Kurve selbst befindet, eindeutig bestimmt.

b) Schreiben Sie eine `main`-Funktion, welche Ihre rekursive Funktion mit einem geeigneten Startpunkt aufruft, um die Länge der **rechten** Kurve in der oberen Abbildung zu bestimmen.

Hinweis: Einträge im Array `FELD` dürfen geändert werden.

```
1  #include <stdio.h>
2  #define M 5
3  #define N 8
4
5  char FELD[M][N] = {
6    {'X','.','.','.','.','.','.','X'},
7    {'X','X','.','X','.','.','.','X'},
8    {'.','X','.','X','.','X','X','X'},
9    {'.','X','X','X','.','X','.','.'},
10   {'.','.','.','.','.','X','.','.'}
11 };
12
13 int arc_len(int m, int n) {
14   if ((m < 0) || (m >= M) || (n < 0) || (n >= N) || (FELD[m][n] != 'X')) {
15     return 0;
16   }
17   FELD[m][n] = 'x';
18   return 1 + arc_len(m-1, n) + arc_len(m+1, n) + arc_len(m, n-1) + arc_len(m, n+1);
19 }
20
21 int main() {
22   printf ("arc_len = %d\n\n", arc_len(0, N-1));
23   //  for (int m = 0; m < M; m++) { for (int n = 0; n < N; n++) {
24   //  printf("%c ", FELD[m][n]); } printf("\n"); }
25   return 0;
26 }
```

Aufgabe 3 (12 Punkte)

a) Schreiben Sie eine Funktion, die einen Vektor und dessen Länge $n \geq 2$ als Eingabeparameter besitzt. Ihre Funktion soll alle Elemente des Vektors mit dem quadrierten Wert überschreiben. Schreiben Sie außerdem eine `main`-Funktion, die Ihre Funktion mit geeigneten Parametern aufruft.

```
1  #include <stdio.h>
2
3  void square(double *vektor, int n) {
4    for (int i = 0; i < n; i++) {
5      vektor[i] = vektor[i] * vektor[i];
6    }
7  }
8
9  int main() {
10   int n = 3;
11   double vektor[] = {1.0, 2.0, 2.5};
12   square(vektor, n);
13   // Ausgabe nicht gefordert!
14   for (int i = 0; i < n; i++) printf("vektor[%d] = %f\n", i, vektor[i]);
15   return 0;
16 }
```

b) Welche der zwei Möglichkeiten der Parameterübergabe haben Sie jeweils für Ihre Funktionsparameter gewählt? Erläutern Sie, was genau übergeben wird und wie Einfluss auf die aufrufende Funktion genommen wird.

1. Call-by-value: Angewendet für den Parameter `n`. Eine Kopie des Inhalts der Variable (Array-Länge) wird übergeben. In der aufrufenden Funktion bleibt der Inhalt unverändert.
   (Die Gültigkeit der Kopie endet mit dem Funktionsrücksprung.)
2. Call-by-Reference: Angewendet für den Parameter `vektor`. Eine Kopie der Adresse des ersten Array-Elements aus der aufrufenden Funktion wird übergeben. Der Inhalt bei der Adresse wird in der aufrufenden Funktion durch das Quadrieren nachhaltig verändert.
   (Mehrere Rückgabewerte oder Array-Parameter können so in C realisiert werden).

c) Schreiben Sie analog zum Aufgabenteil a) eine Funktion, die einen Vektor und dessen Länge $n \geq 2$, sowie einen Funktionszeiger als Eingabeparameter besitzt. Der Funktionszeiger soll kompatibel mit einer der bereits durch die `math.h` bereitgestellten trigonometrischen Funktion sein, z.B. **double** `sin(`**double** `x)` oder **double** `cos(`**double** `x)`. Ihre Funktion soll alle Elemente des Vektors mit der Ausgabe der referenzierten Funktion überschreiben. Schreiben Sie außerdem eine `main`-Funktion, die Ihre Funktion mit geeigneten Parametern aufruft.

```
1  #include <stdio.h>
2  #include <math.h>
3
4  void eval(double (*fp)(double), double* vektor, int n) {
5    for (int i = 0; i < n; i++) {
6      vektor[i] = fp(vektor[i]);
7    }
8  }
9
10 int main() {
11   int n = 3;
12   double vektor[] = {0.0, 1.57, 3.14};
13   eval(sin, vektor, n);
14   // Ausgabe nicht gefordert!
15   for (int i = 0; i < n; i++) printf("vektor[%d] = %f\n", i, vektor[i]);
16   return 0;
17 }
```

Aufgabe 4 (12 Punkte)

Schreiben Sie ein Programm, dem zwei positive ganze Zahlen n, k und ein Ausgabedateiname als Kommandozeilenparameter übergeben werden, und welches dann k aus dem Buchstaben `'X'` zusammengesetzte, große X, die sich über $m := 2n+1$ Zeilen erstrecken, hintereinander, wie unten angegeben in die Ausgabedatei schreibt.

Beispiele: `n=1, k=1, m=3` `n=3, k=2, m=7` `n=2, k=3, m=5`

```
X X
 X
X X
```

```
X     X     X
 X   X X   X
  X X   X X
   X     X
  X X   X X
 X   X X   X
X     X     X
```

```
X   X   X   X
 X X X X X X
  X   X   X
 X X X X X X
X   X   X   X
```

Hinweis: Überlegen Sie sich zuerst den Fall $k = 1$, bei dem nur ein einziges großes X ausgegeben wird, und erweitern Sie diesen dann auf beliebige k.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  int main (int argc, char* argv[]){
5      int i,j,n,k,m,l,j1;
6      FILE* fp;
7      if(argc < 4){
8          printf("error: not enough program arguments \n"); exit(1);
9      }
10     sscanf(argv[1],"%d",&n);
11     sscanf(argv[2],"%d",&k);
12     if(strcmp(argv[3],"stdout") != 0)
13         fp = fopen(argv[3],"w");
14     else
15         fp = stdout;
16     if(fp == NULL){
17         printf("error: fopen \n"); exit(1);
18     }
19     m = 2*n+1;
20     fprintf(fp,"\n");
21     for(i = 1; i <= m; i++){
22         j1 = 1;
23         for(l = 1; l <= k; l++){
24             for(j = j1; j <= m; j++)
25                 if ( j == i || j == m-i+1 )
26                     fprintf(fp,"X");
27                 else
28                     fprintf(fp," ");
29             j1 = 2;
30         }
31         fprintf(fp,"\n");
32     }
33     fprintf(fp,"\n");
34     fclose(fp);
35     return(0);
36 }
```

Aufgabe 5 (12 Punkte)

Man betrachte n Spieler, die an einem runden Tisch sitzen und von 1 bis n im Uhrzeigersinn durchnumeriert sind. Weiterhin sei eine positive ganze Zahl m vorgegeben. Beginnend mit Spieler 1 wird nun solange im Uhrzeigersinn jeder m-te Spieler entfernt bis nur noch ein Spieler übrigbleibt. Die Nummer dieses Spielers wird mit $L := L(n, m)$ bezeichnet und soll von einem Programm ermittelt werden.

a) Die Spieler sollen in einer zyklischen, einfach verketteten Liste gespeichert werden. Legen Sie dazu eine Struktur `s_ring` mit folgenden Komponenten an:

   - `nr` : Spielernummer
   - `next` : Zeiger auf Nachfolger, der noch am Tisch sitzt

b) Schreiben Sie eine Funktion `create`, welche zum Eingabeparameter n eine zyklische Liste mit n Spielern erzeugt und einen Zeiger auf Spieler n zurückgibt. Zyklisch bedeutet dabei nur, dass Spieler n den Nachfolger 1 hat.

c) Schreiben Sie eine Funktion `delete`, die als Eingabeparameter einen Zeiger auf eine zyklische Liste gemäß b) und eine positive ganze Zahl m erhält. Die Funktion soll dann die Liste durchlaufen und jeweils das nach $m-1$ Schritten erreichte Element löschen bis die Liste nur noch ein einziges Element enthält. Die Nummer $L = L(n, m)$ dieses Elements soll dann von der Funktion zurückgegeben werden.

   Beispiele: L(7,4) = 2, L(5,2) = 3, L(6,3) = 1.

```c
#include <stdio.h>
#include <stdlib.h>
/* zu a) */
struct s_ring{
    unsigned int nr;
    struct s_ring* next;
};
/* zu b) */
struct s_ring* create(unsigned int n){
  unsigned int i;
  struct s_ring *p = NULL, *q = NULL;
  p = (struct s_ring*)malloc(sizeof(*p));
  if(p == NULL){ printf("error malloc \n"); exit(1); }
  p->nr = 1;
  q = p;
  for(i = 2; i <= n; i++){
    q->next = (struct s_ring*)malloc(sizeof(*p));
    if(q->next == NULL){ printf("error malloc \n"); exit(1); }
    q = q->next;
    q->nr = i;
  }
  q->next = p;
  return q;
}
/* zu c) */
unsigned int delete(struct s_ring* p, unsigned int m){
    unsigned int steps = 0,L;
    struct s_ring *q = p->next; /* q ist der Zeiger auf die aktuell betrachtete Person und
                                   p ist der Zeiger auf deren Vorgaenger.*/
    while(p->nr != q->nr){
      if(steps == m-1){
        p->next = q->next;
        free(q);
        steps = 0;
      }
      else{
        p = p->next;
        steps++;
```

```
39        }
40        q = p->next;
41      }
42      L = p->nr;
43      free(p);
44      return L;
45  }
46  /* Hauptprogramm, nicht Teil der Aufgabe */
47  int main(){
48    unsigned int m,n,L;
49    struct s_ring *p = NULL;
50    printf("\n n = ");
51    scanf("%u",&n);
52    printf("\n m = ");
53    scanf("%u",&m);
54    p = create(n);
55    L = delete(p,m);
56    printf("\n L(%u,%u) = %u \n",n,m,L);
57    return 0;
58  }
```