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

WiSe 2015/16

Midterm Musterklausur Prozedurale Programmierung

7. 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 Programm-Fragmenten.

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.

☐ Ja ☑ Nein

1. Öffnet die folgende Anweisung eine Datei zum Schreiben?

```
FILE* fp = fopen("data.dat", "s");
```

☑ Ja ☐ Nein

2. Ist folgende Strukturdeklaration und -initialisierung erlaubt?

```
struct a {
  int x;
  int y;
};
struct a b = { 1, 2 };
```

☐ Ja ☑ Nein

3. Gibt die `printf`-Anweisung das letzte Element von `a` aus?

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

☑ Ja ☐ Nein

4. Ist folgendes Programm korrekt?

```
int main(int argc, char* argv[]) {
  printf("%s", argv[argc-1]);
  return 0;
}
```

☑ Ja ☐ Nein

5. Wird durch folgende `for`-Schleife die Ausgabe „abc“ erzeugt?

```
for (char c = 'a'; c < 'd'; )
  printf("%c", c++);
```

☐ Ja ☑ Nein

6. Lautet die Ausgabe „20“?

```
int a = 0;
for (int i = 0; i < 10; i++)
  a++;
  a++;
printf("%d", a);
```

7. Kopiert folgende `for`-Schleife `str` in `buffer`?

☑ Ja ☐ Nein

```
char str[] = "Hallo";
char buffer[10];
for (int i = 0; (i < 10) && (buffer[i] = str[i]); i++);
```

8. Hat die Variable `id` am Ende von folgendem Programm den Wert 2?

☑ Ja ☐ Nein

```
unsigned id = 2;
switch (id) {
  case 3:  id /= id;
  case 2:  id *= 2;
  case 1:  id += 2;
  default: id  = 2;
}
```

9. Ist folgende `while`-Schleife eine Endlosschleife?

☐ Ja ☑ Nein

```
short i = 0;
while (i >= 0) {
  printf("Hallo!");
  i++;
}
```

10. Bestimmt folgendes Makro das arithmetische Mittel von `x` und `y`?

☐ Ja ☑ Nein

```
#define MITTEL(x,y) ( (x) + (y) / 2 )
```

11. Ist folgende `printf`-Ausgabe deterministisch?

☐ Ja ☑ Nein

```
int i;
while (i < 10)
  i++;
printf("%d", i);
```

12. Ist folgende dynamische Integer-Matrix-Allokation zulässig?

☑ Ja ☐ Nein

```
int** ptr = (int**) malloc(4 * sizeof(int*));
for (int i = 0; i < 4; i++)
  ptr[i] = (int*) malloc(3 * sizeof(int));
```

Aufgabe 2 (12 Punkte)

(A)

Schreiben Sie eine Funktion zur Berechnung von

$$\mathrm{f}(n) := \sum_{k=1}^{n} \left(k - \prod_{j=1}^{k} j \right), \qquad n \in \mathbb{N}.$$

Die Implementation soll ohne Verwendung von Funktionen aus Standardbibliotheken und ohne Schleifenkonstrukte erfolgen. Es dürfen jedoch Hilfsfunktionen implementiert werden um die Umsetzung zur vereinfachen.

```
1  int prod( int j ) {
2    if ( j == 1 )
3      return 1.0;
4    return j * prod( j - 1 );
5  }
6
7  int f( int k ) {
8    if ( k < 2 )
9      return 0.0;
10   return f( k - 1 ) + k - prod( k );
11 }
```

(B)

Für zwei positive ganze Zahlen a, b, wobei $a \leq b$ sei, soll die Modulo-Operation (formuliert durch: $b \bmod a$) den Rest der Ganzzahldivision bestimmen. Ergänzen Sie die folgende Rekursionsvorschrift zur Berechnung des kleinsten gemeinsamen Vielfachen von a und b um ein geeignetes Abbruchkriterium.

$$\mathrm{kgV}(a,b) = \begin{cases} \mathbf{b} & \text{, falls} \quad \mathbf{b} \bmod \mathbf{a} = \mathbf{0} \\ \frac{b}{b \bmod a}\ \mathrm{kgV}(b \bmod a, a) & \text{, sonst.} \end{cases}$$

Implementieren Sie die entsprechende Funktion in C-Code ohne den Gebrauch von Schleifenkonstrukten oder die Verwendung von Funktionen aus Standardbibliotheken. Beachten Sie hierbei, dass im Gegensatz zum Gesamtterm der Bruch $\frac{b}{b \bmod a}$ nicht immer ganzzahlig ist.

```
1 int kgV( int a, int b ) {
2   int r = b % a;
3   if ( r == 0 )
4     return b;
5   return b * kgV( r, a ) / r;
6 }
```

Aufgabe 3 (11 Punkte)

Betrachten Sie das Programm auf der folgenden Seite. Die enthaltene Funktion `sumup` (Zeile 12) soll ein Array von Integer-Zahlen folgendermaßen kumuliert addieren:

- Eingabe: $1, 1, 1, 2, 2, \ldots$
- Ausgabe: $1, 2, 3, 5, 7, \ldots$

Beim Kompilieren gibt es in der Zeile 31 eine `Compiler-Warning`, aber keine `Errors`, und eine Binärdatei wird erzeugt. Die Ausführung der Binärdatei endet mit einem `Segmentation Fault`.

a) Erklären Sie die in der Vorlesung behandelten Konzepte der Parameterübergabe und -rückgabe am Beispiel der Funktion `sumup`. Wo haben schreibende Zugriffe auf die Parameterinhalte Gültigkeit und wo nicht?

Antwort:

(6 Punkte)

- Der erste Übergabeparameter ist ein Zeiger auf ein Array von Integer-Zahlen, es handelt sich also um Call-by-Reference. Änderungen an den Array-Inhalten gelten in `sumup`, der Aufrufenden Funktion und überall, wo Zugriff auf das Array existiert.
- Der zweiter Übergabeparameter ist eine Kopie eines Integer, der die Länge des Arrays angibt, es handelt sich also um Call-by-Value. Alle Wertänderungen gelten nur lokal in der Funktion `sumup`.
- Die Rückgabe ist eine Kopie des letzten Array-Elementes, die in der aufrufenden Funktion genutzt werden kann.

b) Korrigieren Sie den `sumup`-Funktionsaufruf direkt im Programm auf der nächsten Seite in Zeile 31. Tritt dieses Problem an anderen Stellen ebenfalls auf, so korrigieren Sie diese ebenfalls!

(2 Punkte)

c) Nach der Korrektur im Aufgabenteil b) ergibt sich eine merkwürdige Ausgabe für `array3`:

```
Summe =  1, Array = 1
Summe = 10, Array = 1 3 6 10
Summe =  3, Array = 1 3
```

Wie würde die korrekte Ausgabe für `array3` lauten, wie kommt es zu dieser Ausgabe?

Hinweis: Auf dem genutzten System belegt ein Integer-Wert 4 Byte und ein Zeiger 8 Byte.

Antwort:

(3 Punkte)

- Korrekte Ausgabe `Summe = 36, Array = 1 3 6 10 15 21 28 36`
- Die Berechnung von `n3` ist fehlerhaft: `sizeof(array3)/sizeof(int) = 8/4 = 2` und nicht `8`.
- (Besser Makro-Konstante für `#define N3 8` einführen!)

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

void printArray(int* array, int n, int sum) {
  printf("Summe = %2d, Array = ", sum);
  for (int i = 0; i < n; i++) {
    printf("%d ", array[i]);
  }
  printf("\n");
}

int sumup(int* array, int n) {
  for (int i = 0; i < (n-1); i++, array++) {
    *(array+1) += *array;
  }
  return *array;
}

int main () {
  int  array1   = 1;
  int  array2[] = { 1, 2, 3, 4 };
  int* array3   = (int*) malloc (8 * sizeof(int));
  for (int i = 0; i < 8; i++) {
    array3[i] = i+1;
  }

  int n1 = sizeof(array1) / sizeof(int);
  int n2 = sizeof(array2) / sizeof(int);
  int n3 = sizeof(array3) / sizeof(int);

  printf("int   = %lu\n", sizeof(int));
  printf("int*  = %lu\n", sizeof(int*));
  printf("char  = %lu\n", sizeof(char));
  printf("char* = %lu\n", sizeof(char*));

  printf("n1 = %d\n", n1);
  printf("n2 = %d\n", n2);
  printf("n3 = %d\n", n3);

  int sum1 = sumup(&array1, 1);
  int sum2 = sumup( array2, 4);
  int sum3 = sumup( array3, 8);

  printArray(&array1, 1, sum1);
  printArray( array2, 4, sum2);
  printArray( array3, 8, sum3);

  return 0;
}
```

Aufgabe 4 (12 Punkte)

(A)

Schreiben Sie ein vollständiges Programm, welches die Pfade zu zwei Textdateien von der Tastatur einliest und anschließend den Text der einen Datei an den der anderen anfügt.

```c
#include <stdio.h>

int main( void ) {

  char path1[ 100 ], path2[ 100 ], c;
  FILE *pf1, *pf2;

  scanf( "%s", path1 );
  scanf( "%s", path2 );

  pf1 = fopen( path1, "r" );
  pf2 = fopen( path2, "a" );

  while ( fscanf( pf1, "%c", &c ) != EOF )
    fprintf( pf2, "%c", c );

  fclose( pf1 );  fclose( pf2 );

  return 0;
}
```

(B)

Schreiben Sie ein vollständiges Programm, welches einen mathematischen Term als Kommandozeilenargument übergeben bekommt und diesen auswertet. Das Programm soll mit den elementaren Operationen $\{+, -\}$ umgehen können und den übergebenen Term von links nach rechts auswerten. Der Aufruf

```
programm.exe  4.5 - 1.5 + 1.0
```

soll demnach "4" [1] auf dem Bildschirm ausgeben. Mögliche Fehler bei der Eingabe dürfen unberücksichtigt bleiben. [2]

```
1  #include <stdio.h>
2
3
4  int main( int argc, char *argv[] ) {
5    double res = 0.0,  temp;
6
7    for ( int i = 1; i < argc; i += 2 ) {
8      sscanf( argv[ i ], "%lf", &temp );
9      switch ( argv[ i - 1 ][ 0 ] ) {
10       case '-':   res -= temp;  break;
11       default:    res += temp;  break;
12     }
13   }
14
15   printf( "\n res = %f\n", res );
16
17   return 0;
18 }
```

[1] $= ((4.5 - 1.5) + 1)$.

[2] Es bietet sich die Verwendung von `argv` und `sscanf` an.

Aufgabe 5 (13 Punkte)

a) Definieren Sie eine Struktur für die Organisation von Mitarbeitern in einer linearen einfach verketteten Liste. Neben dem Mitarbeiternamen und seiner Identifikationsnummer enthalten die Strukturelemente auch eine Projektkennziffer. Erzeugen Sie eine entsprechende Liste mit zwei dynamisch allozierten Elementen.

b) Schreiben Sie eine Funktion, welche in einer solchen Liste alle diejenigen Mitarbeiter sucht, die an einem gegebenen Projekt arbeiten. Die Name dieser Mitarbeiter sollen auf dem Bildschirm ausgegeben werden.

c) Implementieren Sie eine Funktion, welche einen Mitarbeiter am Anfang einer solchen Liste hinzufügt. Erläutern Sie kurz die Verwendung Ihrer Funktion.

d) Schreiben Sie eine Funktion, die zwei gegebene Listen zusammenführt. Die Elemente der Eingangslisten seien jeweils alphabetisch sortiert. Realisieren Sie die Zusammenführung der Listen so, dass die resultierende Liste diese Sortierung beibehält. Die Originallisten müssen nicht erhalten bleiben und Sie dürfen vereinfacht davon ausgehen, dass die Listen nicht leer sind.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  /* Aufgabe 1 */
6  typedef struct mitarbeiter{
7    char name[ 100 ];
8    int id;
9    int project;
10   struct mitarbeiter *next;
11 } sma;
12
13 /* Aufgabe 2 */
14 void print_proj_sma( sma *lst, int proj ) {
15   while ( lst != NULL ) {
16     if ( lst->project == proj )
17       printf( "%s\n", lst->name );
18     lst = lst->next;
19   }
20 }
21
22 /* Aufgabe 3 */
23 sma * insert_first( sma *lst, sma *elem ) {
24   elem->next = lst;
25   return elem;
26 }
27
28 /*  Aufgabe 4 */
29 sma * merge_lists( sma *lst1, sma *lst2 ) {
30   sma dummy, *tmp;
31   dummy.next = lst1;
32   lst1 = &dummy;
33   while ( lst2 != NULL ) {
34     if ( lst1->next == NULL || strcmp( lst1->next->name, lst2->name ) > 0 ) {
35       tmp = lst1->next;
36       lst1->next = lst2;
37       lst2 = tmp;
```

```
38      }
39      else
40        lst1 = lst1->next;
41    }
42    return dummy.next;
43  }
44
45
46  int main( void ) {
47
48    // 1)
49    sma *lst, *lst2, *elem;
50    lst = ( sma * ) malloc( sizeof( sma ) );
51    lst->next = ( sma * ) malloc( sizeof( sma ) );
52    lst->next->next = NULL;
53    strcpy( lst->name, "Hans Eins" );
54    strcpy( lst->next->name, "Thomas Zwei" );
55    lst->id = 101;
56    lst->next->id = 99;
57    lst->project = 9;
58    lst->next->project = 7;
59
60    // 2) ab hier keine Relevanz fuer Klausur
61    print_proj_sma( lst, 7 );
62
63    // 3)
64    elem = ( sma * ) malloc( sizeof( sma ) );
65    elem->next = NULL;
66    strcpy( elem->name, "Anna Drei" );
67    elem->id = 100;
68    elem->project = 7;
69    lst2 = insert_first( lst, elem );
70
71    // 4)
72    lst = ( sma * ) malloc( sizeof( sma ) );
73    lst->next = ( sma * ) malloc( sizeof( sma ) );
74    lst->next->next = NULL;
75    strcpy( lst->name, "Johannes Eins" );
76    strcpy( lst->next->name, "Yvonne Zwei" );
77    lst->id = 101;
78    lst->next->id = 99;
79    lst->project = 9;
80    lst->next->project = 7;
81
82    lst = merge_lists( lst2, lst );
83
84    return 0;
85  }
```