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

WS 2014/2015

Midterm Klausur Prozedurale Programmierung

8. Januar 2015

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		
$\sum$		

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.
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)

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.

1. Öffnet die folgende Anweisung eine Datei zum Schreiben?

☑ Ja ☐ Nein

```
FILE* out_file = fopen("output.txt", "w");
```

2. Kreuzen Sie die fehlerfreie Codezeile an.

☐
```
double arr(6);
```
☐
```
double[4 + 2] arr;
```
☑
```
double* arr = (double*) malloc(4 * sizeof(double));
```

3. Ist folgendes Programm-Fragment korrekt?

```
int a[10], i;
for (i = 0; i <= 10; i++)
  *(a + i) = i;
```

☐ Ja ☑ Nein

4. Ist folgendes Programm-Fragment korrekt?

```
int main(int argc, char* argv[]) {
  int a[4], i;
  for (i = 0; ((i < argc) && (i < 4)); i++)
    sscanf(argv[i+1], "%d", &a[i]);
  return 0;
}
```

☐ Ja ☑ Nein

5. Werden durch folgende `for`-Schleife die natürlichen Zahlen 1 bis 10 (in beliebiger Reihenfolge) auf der Konsole ausgegeben?

```
int i;
for (i = 20; i; i -= 2)
  printf("%d ", i / 2);
```

☑ Ja ☐ Nein

6. Ist die Sortierfunktion korrekt?

```
void sort (int* a, int* b) {
  *a = (*a < *b ? *a : *b);
  *b = (*a < *b ? *b : *a);
}
```

☐ Ja ☑ Nein

7. Ein Array `arr` sei folgendermaßen definiert: `int arr[3];` Kreuzen Sie an, in welcher Codezeile korrekt auf den Inhalt des letzten Elements zugegriffen wird.

- ☐ `arr[3];`
- ☐ `*(arr + 3);`
- ☐ `&arr + 3;`
- ☑ `*(&arr[2]);`
- ☐ `*arr + 2;`

8. Ist folgende `switch`-Anweisung korrekt?

```
unsigned id = 2;
switch (id) {
  case 0:
  case 1:  printf("< 2"); break;
  case 2:  printf("= 2"); break;
  default: printf("> 2"); break;
}
```

☑ Ja ☐ Nein

9. Tritt ein Wrap-Around auf?

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

☑ Ja ☐ Nein

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

```
#define MAX(x,y) ( (x)>=(y) ? (y) : (x) )
```

☐ Ja ☑ Nein

11. Ist `d1 == d2` wahr?

```
double d1 = 1e60;
long l = (long) d1;
double d2 = (double) l;
```

☐ Ja ☑ Nein

12. Kreuzen Sie die korrekte Erklärung der Definition `int (*ptr)();` an.

- ☐ Ein Zeiger auf ein Array von Integer-Variablen.
- ☑ Ein Zeiger auf eine Funktion, die einen Integer zurück gibt.
- ☐ Ein Zeiger auf eine noch nicht initialisierte Integer-Variable.
- ☐ Eine ungültige Definition.

Aufgabe 2 (12 Punkte)

Implementieren Sie eine rekursive Funktion, welche dazu dient die ersten n Zahlen der Fibonacci-Folge aufsteigend auf dem Bildschirm auszugeben. Realisieren Sie eine zweite rekursive Funktion zur Ausgabe der entsprechenden Zahlen in absteigender Reihenfolge. Worin unterscheiden sich die beiden Funktionen? Welche Formen der Rekursion verwenden Sie (Stichwörter: linear, wechselseitig, kaskadenförmig)?

Schreiben Sie ein vollständiges Programm, in welchem beide Funktionen aufgerufen werden. Die Anzahl der auszugebenden Fibonacci-Zahlen n soll über die Tastatur eingelesen werden.

Hinweis: Fibonacci-Folge ist durch das Bildungsgesetz $a_i = a_{i-1} + a_{i-2}$ für $i > 2$ und die Anfangswerte $a_1 = a_2 = 1$ definiert.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /**
5   * Beispiel (a) fuer Ausgabe aufsteigend
6   *   - Rekursion: linear
7   *   - 'curr' und 'prec' entsprechen jeweils der n-ten
8   *     und (n-1)-ten Fibonacci-Zahl
9  **/
10 void fib_inc_a ( int n, int * curr, int * prec ) {
11   if ( n > 1 ) {
12     fib_inc_a ( n - 1, prec, curr );
13     *curr += *prec;
14   }
15   else {
16     *curr = 1;
17     *prec = 0;
18   }
19   printf ( "%d\n", *curr );
20 }
21
22 /**
23  * Beispiel (b) fuer Ausgabe aufsteigend
24  *   - Rekursion: kaskadiert  (hier ineffizient)
25  *   - 'print' zum Festlegen der Ausgabe
26 **/
27 int fib_inc_b ( int n, int print ) {
28   int fib = n > 0;
29   if ( n > 1 ) {
30     fib = fib_inc_b ( n - 2, 0 );
31     fib += fib_inc_b ( n - 1, 1 && print );
32   }
33   if ( print )
34     printf ( "%d\n", fib );
35   return ( fib );
36 }
37
38
39 /**
40  * Beispiel (a) fuer Ausgabe absteigend
41  *   - Rekursion: linear
42  *   - Parameter: Folenlaenge + 1. und 2. Element
43 **/
```

```
44  void fib_dec_a ( int n, int f1, int f2 ) {
45    if ( n > 1 )
46      fib_dec_a ( n - 1, f2, f1 + f2 );
47    printf ( "%d\n", f1 );
48  }
49
50
51
52  int main () {
53    int n, fib, prec;
54
55    printf ( "Die Laenge der auszugebenen Fibonacci-Folge: " );
56    scanf ( "%d", &n );
57
58    printf ( "\nIn aufsteigender Reihenfolge (a):\n" );
59    fib_inc_a ( n, &fib, &prec );
60
61    printf ( "\nIn aufsteigender Reihenfolge (b):\n" );
62    fib = fib_inc_b ( n, 1 );
63
64    printf ( "\nIn absteigender Reihenfolge (a):\n" );
65    fib_dec_a ( n, 1, 1 );
66
67
68    return 0;
69  }
```

Aufgabe 3 (12 Punkte)

a) Erklären Sie den Unterschied zwischen Call-by-Value und Call-by-Reference anhand eines kurzen Funktionsbeispiels.

Call-by-Value: Werte werden als lokale Kopie übergeben. Änderungen der Werte beeinflussen nicht die aufrufenden Funktion. `void swap(int a, int b) {int t = a; a = b; b = t; }`

Call-by-Reference: Die Addressen der Werte werden als Kopie übergeben. Änderungen der Werte beeinflussen die aufrufenden Funktion. `void swap(int* a, int* b) {int t = *a; *a = *b; *b = t; }`

b) Ergänzen Sie das folgende Programm, indem Sie in Programmzeile 11 den Radius `r` mit Hilfe der Funktion `berechne_radius` und dem gegebenen Durchmesser `d` ermitteln.

c) Implementieren Sie die Funktion `berechne` aus Programmzeile 12 in Aufgabenteil b), die zu einem gegebenen Radius `r` den Kreisumfang `u` $= 2\pi r$ und Flächeninhalt `a` $= \pi r^2$ berechnet. Das Makro `PI` darf verwendet werden.

```
1  #include <stdio.h>
2  #define PI 3.1415926536
3
4  double berechne_radius(double d) {
5    return d/2;
6  }
7
8  int main () {
9    double d = 8, r, u, a;
10
11   r = berechne_radius(d);
12   berechne(r, &u, &a);
13
14   printf("Kreisradius  = %lf\n", r);
15   printf("Kreisumfang  = %lf\n", u);
16   printf("Kreisflaeche = %lf\n", a);
17   return 0;
18 }
19
20 void berechne(double r, double* u, double* a) {
21   *u = 2 * PI * r;
22   *a = PI * r * r;
23 }
```

d) Schreiben Sie eine Funktion, die dynamisch Speicher für n `double`-Vektoren der Länge n bereitstellt und geben Sie den Anfangszeiger auf dieses zweidimensionale Feld zurück. Alle Elemente der Vektoren sollen mit 1.0 initialisiert sein.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  double** myAlloc(int n) {
5    int i = 0, j = 0;
6    double** m = (double**) malloc (n * sizeof(double*));
7    for (; i < n; i++) {
8      m[i] = (double*) malloc (n * sizeof(double));
9      for (; j < n; j++) {
10       m[i][j] = 1.0;
11     }
12   }
```

```
13    return m;
14  }
15
16  int
17  main ()
18  {
19    int n = 3, i = 0, j = 0;
20    double** A = myAlloc(n);
21
22    for (;i < n; i++) {
23      for (;j < n; j++) {
24        printf("%lf ", A[i][j]);
25      }
26      printf("\n");
27      j = 0;
28    }
29
30    return 0;
31  }
```

Aufgabe 4 (12 Punkte)

Schreiben Sie ein vollständiges Programm, welches die Einträge einer quadratischen Matrix aus einer Textdatei liest, und diese anschließend auf dem Bildschirm ausgibt. Der Pfad zur Textdatei soll als Kommandozeilenparameter übergeben werden. Das Speicherformat sei durch folgendes Beispiel veranschaulicht.

```
4
0  1  3.5
2  1  -5.0
3  0  4.4
3  3  0.5
```

Der erste Wert liefert die Dimension der Matrix. Jede weitere Zeile beinhaltet die Zeile, die Spalte und den Wert eines Matrixelements in eben dieser Reihenfolge. In der Textdatei sind nur Nichtnulleinträge gespeichert. Bei der Bildschirmausgabe sollen alle Matrixelemente, also auch die Nulleinträge, angezeigt werden.

Hinweis: Verwenden Sie `argv` und `sscanf`. Mögliche Fehler bei der Eingabe oder der Speicherallokation dürfen unberücksichtigt bleiben.

```
1   #include <stdio.h>
2   #include <stdlib.h>
3
4   int main ( int argc, char *argv[] ) {
5     int n = 0, i = 0, j = 0;
6     double elem = .0, *mat = NULL;
7     FILE *finput = NULL;
8
9     finput = fopen ( argv[1], "r" );
10    fscanf ( finput, "%d", &n );
11
12    mat = (double *) malloc ( n * n * sizeof ( double ) );
13
14    for ( i = 0; i < n * n; i++ )
15      mat[ i ] = .0;
16
17    while ( fscanf ( finput, "%d %d %lf", &i, &j, &elem ) != EOF )
18      mat[ i * n + j ] = elem;
19
20    for ( i = 0; i < n; i++ ) {
21      for ( j = 0; j < n; j++ )
22        printf ( "%7.2f", mat[ i * n + j ] );
23      printf ( "\n" );
24    }
25
26    return 0;
27  }
```

Aufgabe 5 (12 Punkte)

a) Definieren Sie eine Struktur zur Verwaltung von Telefonnummern in einer linearen einfach verketteten Liste. Erzeugen Sie eine entsprechende Liste mit zwei dynamisch allozierten Elementen.

b) Schreiben Sie eine Funktion, welche zu einem Namen in einer solchen Liste die zugehörige Telefonnummer sucht.

c) Implementieren Sie eine Funktion, welche den Nachfolger eines gegebenen Listenelements aus der Liste entfernt.

d) Schreiben Sie eine Funktion, die eine solche Liste an einer bestimmten Position in zwei Listen auftrennt. Erläutern Sie kurz das Interface der Funktion.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  struct stel {
6    char name[ 50 ];
7    unsigned long tel_nr;
8    struct stel *next;
9  };
10
11
12 unsigned long search_tel ( struct stel *lst, char *name ) {
13   while ( lst != NULL && strcmp ( name, lst->name ) != 0 )
14     lst = lst->next;
15   if ( lst == NULL )
16     return ( 0 );
17   return ( lst->tel_nr );
18 }
19
20 void del_next ( struct stel *elem ) {
21   struct stel *tt;
22   if ( elem == NULL || elem->next == NULL )
23     return;
24   tt = elem->next;
25   elem->next = tt->next;
26   free ( tt );
27 }
28
29
30 /* Variante 1 */
31 void split_at_pos ( struct stel *lst, struct stel **lst2, int pos ) {
32   while ( lst != NULL && pos > 0 ) {
33     pos--;
34     lst = lst->next;
35   }
36   if ( lst != NULL && pos == 0 ) {
37     *lst2 = lst->next;
38     lst->next = NULL;
39   }
40 }
41
42 /* Variante 2 */
43 void split_by_elem ( struct stel *lst, struct stel *elem ) {
```

```
44    while ( lst != NULL && lst->next != elem )
45      lst = lst->next;
46    if ( lst != NULL )
47      lst->next = NULL;
48  }
49
50
51  int main () {
52    struct stel *tlst = NULL;
53
54    tlst = ( struct stel * ) malloc ( sizeof ( struct stel ) );
55    strcpy ( tlst->name, "Hans Mueller" );
56    tlst->tel_nr = 1705638754;
57
58    tlst->next = ( struct stel * ) malloc ( sizeof ( struct stel ) );
59    strcpy ( tlst->next->name, "Frida Bauhofer" );
60    tlst->next->tel_nr = 1602108941;
61    tlst->next->next = NULL;
62
63    return 0;
64  }
```