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

WS 2012/2013

Musterklausur Prozedurale Programmierung

6. März 2013

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)

Folgendes Programm dient zum Einlesen einer unteren Dreiecksmatrix aus einer Datei, in deren ersten Zeile die Dimension n der Matrix steht, woran dann die Matrixzeilen anschließen. Dem Programmierer sind 12 syntaktische und semantische Fehler unterlaufen. Finden Sie diese!

Hinweis: Die Fehler befinden sich in den Zeilen 1-3, 12-18, 22-27, 35-45. Nehmen Sie die anderen Zeilen trotzdem zum Verständnis des Programmes zur Kenntnis.

```
1  #include <stdio.h>
2  #include <stdlab.h>
3  #define A(i,j)  A[((i)*((i)-1)/2 + (j)-1)];
4
5  /* read_symmatrix reads a lower triangular matrix of order n=*pn from a textfile "filename".
6   * All matrix elements of the lower triangle are stored in a one-dimensional array
7   * of length n*(n+1)/2. */
8  float *read_symmatrix(char filename[], int * pn) {
9      int n = 0, i = 0;  /* index value */
10     float *matrix = NULL;  /* pointer to matrix */
11     FILE *matrixfile = NULL;  /* pointer to textfile */
12     matrixfile = fopen(*filename,"r"); /* open textfile - read-only */
13     if(matrixfile == NULL){
14         printf("Error: Could not open \"%s\"\n", filename);
15         return NULL}
16     fscanf(matrixfile, "%d", &pn); /* read dimension n */
17     n = *pn;
18     matrix = (float *) malloc(n*sizeof(float)); /* allocate memory for matrix */
19     if(matrix == NULL){
20         printf( "Error: Could not allocate memory" );
21         return NULL;}
22     for(i = 0; i > (n*(n+1))/2; i++)
23         fscanf(matrixfile, "%d", matrix+i); /* read elements of matrix */
24     return matrix; /* return pointer to matrix */
25 }  /* read_symmatrix */
26
27 int main( int argc, double *argv[] ){
28     int i = 0, j = 0;  /* indices */
29     int n = 0;  /* dimension of matrix */
30     float *A = NULL;  /* pointer to matrix */
31
32     if( argc < 2 ){
33         printf("Error: Not enough input parameter!\n"); /* check number of arguments */
34         return 1;}
35     /* read symmetric matrix */
36     if(A = read_symmatrix(argv[1], &n) == NULL)
37         return 2;
38
39     /* display matrix elements */
40     for(i = 1; i <= n; i++){
41         for(j = 1; j <= i; i++)
42             printf("%f ", A(i,j));
43         printf("\n");
44     return 0;
45 }  /* main */
```

Zeile	Notieren Sie die korrigierten Zeilen (maximal 12) hier!
2	`#include <stdlib.h>`
3	`#define A(i,j)  A[((i)*((i)-1)/2 + (j)-1)]`
12	`matrixfile = fopen(filename,"r");`
15	`return NULL;}`
16	`fscanf(matrixfile, "%d", pn);`
18	`matrix = (float *) malloc(n*(n+1)/2*sizeof(float));`
22	`for(i = 0; i < (n*(n+1))/2; i++)`
23	`fscanf(matrixfile, "%f", matrix+i);`
27	`int main( int argc, char * argv[] ){`
36	`if((A = read_symmatrix(argv[1], &n)) == NULL)`
41	`for(j = 1; j <= i; j++)`
43	`printf("\n");}`

Aufgabe 2 (12 Punkte)

Schreiben Sie ein vollständiges Programm, das für reelle x und $y \neq 0$ und ganze nicht-negative Zahlen m und n den Wert

$$f(x, y, m, n) := x^m y^{-n}$$

rekursiv berechnet und auf dem Bildschirm ausgibt. Geben Sie dazu zunächst eine rekursive Formel für die Funktion f an:

$$f(x, y, m, n) = \begin{cases} 1 & \text{, falls } m = 0 = n, \\ f(x, y, m-1, n) \cdot x & \text{, falls } m > 0, \\ f(x, y, m, n-1)/y & \text{, falls } n > 0. \end{cases}$$

3 Punkte

Die Parameter x, y, m, n sollen im Hauptprogramm von der Tastatur eingelesen werden.

```
1   #include <stdio.h>
2
3   float f(float x, float y, int m, int n){  /* 1 Punkte */
4     if(m==0 && n==0) return 1;              /* 1 Punkte */
5     if(m>0) return(f(x,y,m-1,n)*x);         /* 2 Punkte */
6     return(f(x,y,m,n-1)/y);                 /* 2 Punkte */
7   }
8
9   int main(void){
10    int m,n;
11    float x,y;
12    printf("\n x = "); scanf("%f",&x);       /* 1 Punkt */
13    printf("\n m = "); scanf("%d",&m);       /* 1 Punkt */
14    printf("\n y = "); scanf("%f",&y);
15    printf("\n n = "); scanf("%d",&n);
16    printf("\n %f^%d*%f^-%d = %f \n",x,m,y,n,f(x,y,m,n));  /* 1 Punkte */
17    return(0);
18  }
```

Aufgabe 3 (12 Punkte)

a) Schreiben Sie eine Funktion `fib_array`, die einen Zeiger auf ein `int`-Array der Länge `n` zurückgibt. Der einzige Eingabeparameter der Funktion soll eine `int`-Variable sein, welche die Länge des Arrays festlegt. Zusätzlich soll die Funktion das Array mit den ersten `n` Zahlen der Fibonacci-Folge initialisieren, also: $a_0 = 0$, $a_1 = 1$ und $a_i = a_{i-2} + a_{i-1}$. Achten Sie auch darauf, falsche Eingabewerte und Fehler bei der Speicherreservierung abzufangen!

```
1   #include <stdlib.h>
2
3   int* fib_array(int n) {                        /* 1 Punkt */
4     int i = 0; int *fibs = NULL;
5     if (n < 1) return NULL;
6     fibs = (int *) malloc(sizeof(int) * n);      /* 1 Punkt */
7     if (fibs == NULL) return fibs;
8     fibs[0] = 0;
9     if (n > 1)  fibs[1] = 1;                     /* 1 Punkt */
10    for (i = 2; i < n; i++)
11      fibs[i] = fibs[i-2] + fibs[i-1];           /* 1 Punkt */
12    return fibs;
13  }
```

b) Die Variable `A` sei folgendermaßen deklariert: `struct S A[8]`. Dabei sei `S` eine nicht näher spezifizierte Struktur, die eine Komponente `keys` vom Typ `int *` besitze. Geben Sie vier verschiedene Zugriffsmöglichkeiten an, die äquivalent zum folgenden Aufruf sind: `A[3].keys[2]`

```
*(A[3].keys+2)      (A+3)->keys[2]       *((A+3)->keys+2)
        (*(A+3)).keys[2]       *((*(A+3)).keys+2)
```

c) Welche Ausgabe erzeugt das folgende Programm?

```
1   #include <stdio.h>
2   #include <stdlib.h>
3
4   int main(void) {
5     int i = 0, pos = 0;
6     char str[18] = "Mit dem Metronom!";
7
8     for (pos = 0; str[pos] != 'd'; pos++) ;
9
10    for (i = 0; i < 4; i++)
11      str[pos+i] = str[15-i];
12                                                  /* Ausgabe:     */
13    for (i = 1; i < 5; i++) {                     /*              */
14      printf("out[%d] = %c\n", i, str[i/2]); i++; /* out[1] = 'M' */
15      printf("out[%d] = %c\n", i, str[pos+i/2]);  /* out[2] = 'o' */
16    }                                             /* out[3] = 'i' */
17    return 0;                                     /* out[4] = 'n' */
18  }
```

Aufgabe 4 (12 Punkte)

a) Stellen Sie die Dezimalzahl -74.375 als Fließkommazahl im 32 Bit IEEE 754 Standard dar.

Vorzeichenbit:

1

Exponent: (mit dem Shift 127)

1	0	0	0	0	1	0	1

Mantisse: (Vergessen Sie nicht die implizite Eins!)

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

b) Schreiben Sie eine Funktion `str2int`, die einen numerischen String in eine *int*-Variable konvertiert. Es sollen keine Funktionen der C-Standardbibliotheken verwendet werden, dafür darf der Eingabestring ohne zusätzliche Fehlerabfragen als rein numerischer String angenommen werden.

Beispiel: Der Aufruf `str2int("4711")` liefert eine Integervariable mit dem Wert 4711 zurück.

```
1  int str2int(char *s) {                  /* 1 Punkt */
2    int ret = 0;
3    while (*s != '\0')                    /* 1 Punkt */
4      ret = ret * 10 + (*s++ - '0');      /* 2 Punkte */
5    return ret;
6  }
```

c) Welche Ausgaben erzeugt die folgende Code-Sequenz ?

```
1    short int x = 256 + 1, y = 256 - 1;
2    printf("%hd\n", x * y);            /* 1. Ausgabe:  -1  */
3    printf("%hd\n", x / y);            /* 2. Ausgabe:   1  */
4    printf("%c\n", 'C' + x);           /* 3. Ausgabe:  'D' */
5    printf("%hd\n", x & y);            /* 4. Ausgabe:   1  */
```

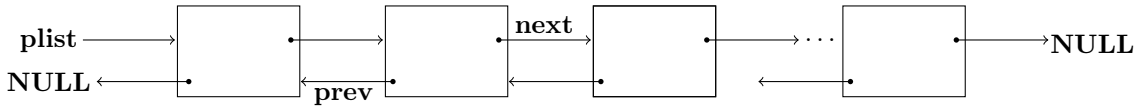
Begründen Sie Ihre Antwort:

1) $xy = (2^8 + 1)(2^8 - 1) = 2^{16} - 1$, nach `wrap around` bleibt -1 als Ergebnis.
2) $257/255 = 1$ modulo 2, bei der Integerdivision fällt der Rest weg.
3) `'C'`$+2^8$ ist aufgrund des `wrap around` immer noch `'C'`.
   Es folgt: `'C'`$+2^8 + 1 =$ `'C'`$+1 =$ `'D'`
4) Wegen $256 + 1 = 2^8 + 2^0$ und $256 - 1 = 2^7 + 2^6 + \ldots + 2^0$ ist 2^0 die einzige Zweierpotenz, die in beiden Variaben gesetzt ist. Damit ist `x & y` $= 2^0 = 1$.

Aufgabe 5 (12 Punkte)

a) Die Parameter eines mathematischen Problems sollen in einer doppelt verketteten linearen Liste abgelegt werden, deren Elemente die Struktur `lp_list` haben, welche folgende Komponenten besitzt:

- `name` : Name des Parameters (maximal 50 Zeichen)
- `value` : Wert des Parameters (doppelt-genaue Fließkommazahl)
- `type` : Typ des Parameters (nicht-negative Ganzzahl)
- `next` : Zeiger auf das nachfolgende Listenelement
- `prev` : Zeiger auf das vorherige Listenelement



Definieren Sie die Struktur `lp_list`:

```
1  struct lp_list{
2    char name[51];     /* oder : char* name; */
3    double value;
4    unsigned int type;
5    struct lp_list *prev;
6    struct lp_list *next;
7  };
```

b) Zum Sortieren der Liste schrieb ein Programmierer folgende Hilfsfunktion `insert_before`, die eine Instanz `*elem` der Struktur `lp_list` vor einem Listenelement `*pos` einfügen soll:

```
1  void insert_before(struct lp_list *elem, struct lp_list *pos) {
2    elem->next = pos->next;
3    elem->prev = pos->prev;
4    pos->prev = elem;
5    if (elem->prev != NULL)
6      (elem->prev).next = elem;
7    return ;
8  }
```

Dem Programmierer sind dabei zwei Fehler unterlaufen. Korrigieren Sie die entsprechenden Zeilen!

Zeile 2	`elem->next = pos;`
Zeile 6	`(elem->prev)->next = elem;`

c) Schreiben Sie eine Funktion `insert_after`, die in ähnlicher Weise wie `insert_before` eine Instanz der Struktur `lp_list` in die Liste einfügt, jetzt allerdings hinter dem Listenelement.

```
1  void insert_after(struct lp_list *elem, struct lp_list *pos) {
2    elem->next = pos->next;
3    elem->prev = pos;
4    pos->next = elem;
5    if (elem->next != NULL)
6      (elem->next)->prev = elem;
7    return ;
8  }
```

d) Schreiben Sie eine Funktion `sort_lp`, welche die Listenelemente nach Namen in alphanumerischer Reihenfolge aufsteigend sortiert! Eingabeparameter ist ein Zeiger auf den Listenanfang der zu sortierenden Liste. Zurückgegeben werden soll ein Zeiger auf den Anfang der sortierten Liste.

Hinweise:

- Verwenden Sie eine der Funktionen `insert_before` oder `insert_after` aus b) und c) !
- Benutzen Sie die Standardfunktion `int strcmp(const char *s1, const char *s2)` zum Vergleichen der Namen! Diese liefert negative Werte zurück, falls `s1` in alphanumerischer Reihenfolge vor `s2` steht, positive Werte im umgekehrten Fall, und 0 bei identischen Strings.

```
1  struct lp_list * sort_lp(struct lp_list *plist) {
2    struct lp_list *pos, *pn, *next;
3    pn = plist->next;
4
5    while (pn != NULL) {
6      pos = pn;
7      next = pn->next;
8      while (pos != plist && strcmp(pn->name, (pos->prev)->name) < 0)
9        pos = pos->prev;
10     if (pos != pn) {
11       (pn->prev)->next = next;
12       if (next != NULL)  next->prev = pn->prev;
13       insert_before(pn, pos);
14       if (pos == plist)  plist = pn;
15     }
16     pn = next;
17   }
18   return plist;
19 }
```