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

WS 2007/2008

Musterlösung der Klausur zur Zwischen-Prüfung Programmiermethodik

(Informatik-Ingenieurwesen / Informationstechnologie)

8. Januar 2009

Sie haben 40 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. Insgesamt können bis zu 28 Punkte erreicht werden.

Aufg.	Punkte	Korr.
1		
2		
3		

$\sum$	

Zur Beachtung:
Schreiben Sie bitte weder mit Bleistift noch mit Rotstift.
Es sind keinerlei Hilfsmittel in dieser Klausur zugelassen!
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 (8 Punkte)

Ein Student (Erstsemestler) sollte ein Programm schreiben, welches vom Benutzer zwei `float` Zahlen und einen Operator $(+, -, *, /)$ einliest, die entsprechende Operation ausfi£¡hrt und das Ergebnis auf dem Bildschirm ausgibt. Er sollte keine Fehlerbehandlung, auï£¡er Division durch Null, implementieren. Das Einlesen und die Berechnung sollten solange wiederholt werden, bis der Benutzer 'n' oder 'N' eingibt. Durch den schlechten Programmierstil des Studenten sind ihm einige Fehler unterlaufen. Den Meldungen des Compilers können wir entnehmen, dass genau 8 Fehler vorhanden sind.

Versuchen Sie alle Fehler zu finden.

(**Falsch erkannte Fehler werden negativ bewertet!**)

```
1   #include <stdio.c>
2
3   int main(void)
4   {
5     float a, b;
6     char again; op;
7
8     do {
9       printf('1. Operand: '); scanf("%f", &a);
10      printf("Operator: "); scanf("\n%c",&op);
11      printf("2. Operand: "); scanf("%f", b);
12
13      switch (&op) {
14      case '+':
15      printf("%f + %f = %f\n", a, b, a + b);
16      break;
17
18      case '-':
19      printf("%f - %f = %f\n", a, b, a - b);
20      break;
21
22      case '*':
23      printf("%f * %f = %f\n", a, b, a * b);
24
25      case '*/':
26      if (b == 0) {
27        printf("Division durch 0.\n");
28      }
29      else {
30        printf("%f / %f = %f\n", a, b, a / b);
31      }
32      break;
33      }
34
35      printf("Nochmal [j/n]? ");
36      scanf("\n%c", &again);
37    } while ((again != "n") && (again != 'N'));
38
39    return 0;
40  }
```

Lösung:

Der angegebene Programmcode enthält insgesamt 8 Fehler. Für jeden korrigierten Fehler wird ein Punkt vergeben. Für falsch erkannte Fehler wird jeweils ein Punkt abgezogen. (Negative Punkte übertragen sich nicht auf andere Aufgaben)

Nachfolgend ist der korrekte Code zu sehen. Die Anzahl der rechts angegebenen Punkte entspricht dabei der Anzahl vorhandener Fehler in der jeweiligen Zeile.

```
1   #include <stdio.h>                                          /* 1 Punkt */
2
3   int main(void)
4   {
5     float a, b;
6     char again, op;                                          /* 1 Punkt */
7
8     do {
9       printf("1. Operand: ");                                /* 1 Punkt */
10      scanf("%f", &a);
11      printf("Operator: ");
12      scanf("\n%c",&op);
13      printf("2. Operand: ");
14      scanf("%f", &b);                                       /* 1 Punkt */
15      switch (op) {                                          /* 1 Punkt */
16      case '+':
17        printf("%f + %f = %f\n", a, b, a + b);
18        break;
19      case '-':
20        printf("%f - %f = %f\n", a, b, a - b);
21        break;
22      case '*':
23        printf("%f * %f = %f\n", a, b, a * b);
24        break;                                               /* 1 Punkt */
25      case '/':                                              /* 1 Punkt */
26        if (b == 0) {
27          printf("Division durch 0.\n");
28        }
29        else {
30          printf("%f / %f = %f\n", a, b, a / b);
31        }
32        break;
33      }
34
35      printf("Nochmal [j/n]? ");
36      scanf("\n%c", &again);
37    } while ((again != 'n') && (again != 'N'));              /* 1 Punkt */
38
39    return 0;
40  }
```

Aufgabe 2 (8 Punkte)

Nehmen Sie bei dieser Aufgabe an, dass vorzeichenbehaftete Datentypen im 2-er Komplement dargestellt werden.

a) Stellen Sie die Zahlen 283, −475, 123, −3000, 728, −4356 als `short int` (binär) dar und addieren Sie diese.

283 :	0	0	0	0	0	0	0	1	0	0	0	1	1	0	1	1
475 :	0	0	0	0	0	0	0	1	1	1	0	1	1	0	1	1
−475 :	1	1	1	1	1	1	1	0	0	0	1	0	0	1	0	1
123 :	0	0	0	0	0	0	0	0	0	1	1	1	1	0	1	1
3000 :	0	0	0	0	1	0	1	1	1	0	1	1	1	0	0	0
−3000 :	1	1	1	1	0	1	0	0	0	1	0	0	1	0	0	0
728 :	0	0	0	0	0	0	1	0	1	1	0	1	1	0	0	0
4356 :	0	0	0	1	0	0	0	1	0	0	0	0	0	1	0	0
−4356 :	1	1	1	0	1	1	1	0	1	1	1	1	1	1	0	0

$283 - 475 + 123 - 3000 + 728 - 4356 = -6697$

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

b) Geben Sie die kleinste und größte im 14-Bit-Zweierkomplement darstellbare Zahl in Dezimalsschreibweise an.

kleinste Zahl: $-2^{13} = -2^{15}/4 = -32768/4 = -8192$

größte Zahl: $2^{13} - 1 = 8191$

c) Welche Ausgabe erzeugt folgende Code-Sequenz:

```
101    short int k=-32768;
102    k=2*k;
103    printf("%d \n",k);
104    k=k-1;
105    printf("%d \n",k);
106    k=-32768;
107    k=k*(k+1)*(k+2);
108    printf("%d \n",k);
```

Programmausgabe:
1) 0
2) −1
3) 0

Begründen Sie Ihre Antwort:

1) Im 16-Bit-Zweierkomplement mit wrap-around gilt $-32768 - 1 = 32767$, also

$$2 * k = 2 * (-32768) = -32768 + -32768 = (-32768 - 1) - 32767 = 32767 - 32767 = 0.$$

2) $k = 0 - 1 = -1$

3) Weil $k + 2$ gerade ist und $2 * k = 0$ gemäß 1), folgt

$$k * (k + 1) * (k + 2) = (2 * k) * (k + 1) * [(k + 2)/2] = 0 * (k + 1) * [(k + 2)/2] = 0.$$

Aufgabe 3 (12 Punkte)

Schreiben Sie ein Programm, das in einer Endlosschleife zwei ganze Zahlen n, k mit $0 \leq k \leq n$ von der Tastaur einliest, den Binomialkoeffizienten $\binom{n}{k}$ rekursiv in einer Funktion `binomial` berechnet und diesen abschließend ausgibt. Das Ergebnis von `binomial` soll vom Typ `int` sein. Dabei gilt folgende Rekursionsvorschrift:

$$\binom{n}{0} = 1, \quad \binom{n}{n} = 1, \quad \binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k} \quad \text{für } 0 < k < n.$$

Um Zeit zu sparen, sollen fi£¡r $n \leq 100$ bereits berechnete Binomialkoeffizienten $\binom{n}{k}$ in einem globalen Array `BiKo` abgelegt und sofern möglich in der Funktion `binomial` effizient verwendet werden. Das Feld `BiKo` soll im Hauptprogramm zu Beginn mit Nullen initialisiert werden. Eine Prüfung der Grenzen für `n` und `k` sei nicht erforderlich.

```
1   #include <stdio.h>                                                    /* 1 Punkt */
2   /*obere Schranke fï£¡r n*/
3   #define NMAX 100
4   /* Die Anzahl der Binomialkoeffizienten (n ï£¡ber k),
5   fï£¡r die n<=NMAX=100 ist, betrï£¡gt ((NMAX)+1)*((NMAX)+2)/2=20604*/
6   double BiKo[(NMAX+1)*(NMAX+2)/2];
7   double binomial(int n, int k){                                        /* 1 Punkt */
8       short unsigned int index;
9       /* In den Sonderfï£¡llen 1 zurï£¡ckgeben */
10      if ((k==0) || (n==k))  return(1);                                 /* 2 Punkte */
11      index=n*(n+1)/2+k; /*Position, an der (n ï£¡ber k) in BiKo steht*/
12      if (n <= NMAX ){                                                  /* 1 Punkt */
13          /* Wenn der Binomialkoeffizient nicht im Array gespeichert ist,
14          wird dieser rekursiv berechnet und im Array abgelegt. */
15          if(BiKo[index]==0){BiKo[index]= binomial(n-1,k-1)+binomial(n-1,k);} /* 2 Punkte */
16          return(BiKo[index]);
17      }else
18      /* Input Argumente sind grï£¡sser als die Arraygrenzen: rekursive Berechnung */
19          return(binomial(n-1,k-1)+binomial(n-1,k));                    /* 1 Punkt */
20  }
21  int  main(void){
22      int k,n;                                                           /* 1 Punkt */
23      /* Das globale Array mit Nullen initialisieren */
24      for(n= 0; n<= (NMAX+1)*(NMAX+2)/2-1; n++) {BiKo[n]= 0;}            /* 1 Punkt */
25      printf("Binomialkoeffizienten - Berechnung\n\n");
26      while(1) {                                                         /* 1 Punkt */
27      printf("Wert von n: ");    scanf ("%d", &n);
28      printf("Wert von k: ");    scanf ("%d", &k);
29      printf("Ergebnis = %g \n", binomial(n,k));
30      }
31      return(0);
32  }
```