KLAUSUR im FACH ***Ingenieur-Informatik***

im Wintersemester 2016/2017

Name, Vorname: …….

Studiengang/Semester: ……

Prüfer:
Dipl.-Ing. (FH) Sebastian Balz, Dipl.-Ing. (FH) Sebastian Stelzl,
Dipl.-Ing. (FH) Rüdiger Ehret, Prof. Dr.-Ing. Daniel Fischer

Bearbeitungshinweise

1. Tragen Sie auf jeder Seite in der Kopfzeile Ihre Matrikelnummer ein.

2. Der Aufgabensatz (inkl. Deckblatt und Anhang), der aus 11 Seiten besteht (Seite 1 bis 11), ist auf Vollständigkeit zu überprüfen.

3. Der Aufgabensatz ist mit den Lösungsblättern abzugeben.

4. Lösungen auf selber mitgebrachten Lösungsblättern werden nicht ausgewertet. Verwenden Sie die Ihnen ausgeteilten Lösungsblätter und tragen Sie auch dort Ihre Matrikelnummer ein.

5. Bei Rechenaufgaben muss der Lösungsweg ersichtlich und lesbar sein, sonst erfolgt keine Bewertung der Aufgabe oder des Aufgabenteils.

6. Die Bearbeitungszeit beträgt 90 Minuten.

7. Es wird hiermit darauf hingewiesen, dass vom Prüfungsamt nicht vorher geprüft wurde, ob Sie das Recht bzw. die Pflicht zur Teilnahme an dieser Klausur haben. Die Teilnahme erfolgt auf eigene Gefahr, gleichzeitig bekundet die Teilnahme die Zustimmung zu diesem Passus.

8. Hilfsmittel:
   - handgeschriebene Formelsammlung (1 DIN A4 Blatt)
   - Taschenrechner sind **nicht** erlaubt

9. Bewertung:
   Gesamtpunktzahl = 100 Punkte

   Note 1,0 = 90 Punkte
   Note 4,0 = 45 Punkte

Aufgabe	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**SUMME**	
Punkte	**10**	**10**	**10**	**10**	**15**	**25**	**20**	**100**	**NOTE**
Erreichte Punkte									

Aufgabe 1: (10 Punkte)
Es ist jeweils eine Antwort richtig. Falsche Antworten führen **nicht** zu Punktabzug! Kreuzen Sie auf alle Fälle immer eine Antwort an!

1.1 Informationen werden durch …. repräsentiert.
a) Adam und Eva
b) Komplexität
c) Brute Force
d) Any Keys
e) **Nullen und Einsen**

1.2 Welche Programme werden **meist** noch teilweise in C geschrieben?
a) Betriebswirtschaftliche Software (SAP R3)
b) Betriebssysteme
c) Webseiten
d) Betriebsarten
e) Konjunkturprogramme

1.3 IDE steht für
a) **Integrated Development Environment**
b) Integral Direction Expansion
c) Integrated Development Evolution
d) Integrated Direction Environment
e) Integral Development Evolution

1.4 Wenn eine Funktion **nicht** von außerhalb der c-Datei aufgerufen werden soll, dann verwendet man das Keyword … bei der Deklaration.
a) invisible
b) register
c) **static**
d) volatile
e) private
f) uncallable

1.5 Was haben das Einerkomplement und die Vorzeichendarstellung (MSB=1 kennzeichnet negative Zahlen) gemeinsam
a) Beide wurden von Dennis Ritchie erfunden
b) Die negativen Zahlen werden exakt gleich dargestellt
c) Beide sind nur bis 32 Bit anwendbar
d) **Die Null kommt zweimal vor**
e) Bilden die Grundlage für die unsigned-Datentypen (nicht vorzeichenbehaftete Datentypen)

1.6 Wie lässt sich eine Subtraktion im Binärsystem auch abbilden?
a) Invertierung der Bits
b) Differenzbildung mit dem Einerkomplement
c) **Addition des Zweierkomplements**
d) Addition des Einerkomplements
e) Differenzbildung mit dem Zweierkomplement

1.7 Was ist im IEEE 754 Standard beschrieben?
a) **Darstellung von Gleitpunktzahlen**
b) Verwendung des Schlüsselwortes volatile
c) Funktionsweise der String-Funktionen (string.h)
d) Warning Levels
e) Übergabe von Parametern in C
f) Bedienung von Visual Studio

1.8 Die Umrechnung vom Binärsystem in das Dezimalsystem kann nach dem … geschehen.
a) Fourier-Schema
b) Fast-Transfer-Schema
c) Bernoulli-Schema
d) Feigenbaum-Schema
e) **Horner-Schema**

1.9 In welchem Speicherbereich befindet sich der Speicher, der mit malloc allokiert wurde?
a) Stack Segment (Stack)
b) **Heap Segment (Heap)**
c) Data Segment (Globale Variablen)
d) Code Segment (Code)
e) Alloc Segment
f) Dort wo sich der zu speichernde Zeiger befindet

1.10 Wie viel Bytes sind 1 Gigabyte?
a) ca. 10^3
b) ca. 10^6
c) **ca. 10^9**
d) ca. 10^{12}
4) ca. 10^{15}

Aufgabe 2: (10 Punkte)

Schreiben Sie C-Code (keine Funktionen, keine includes etc.) der Zahlen von 1 bis 100 wie in der Abbildung 1 ausgibt. Bei jeder Zahl, die das Vielfache von 3 ist, soll statt der Zahl fizz" ausgegeben werden und bei jeder Zahl, die das Vielfache von 5 ist, soll "buzz" ausgegeben werden. Wenn die Zahl das Vielfache von 3 und 5 ist, soll „fizzbuzz“ ausgegeben werden.

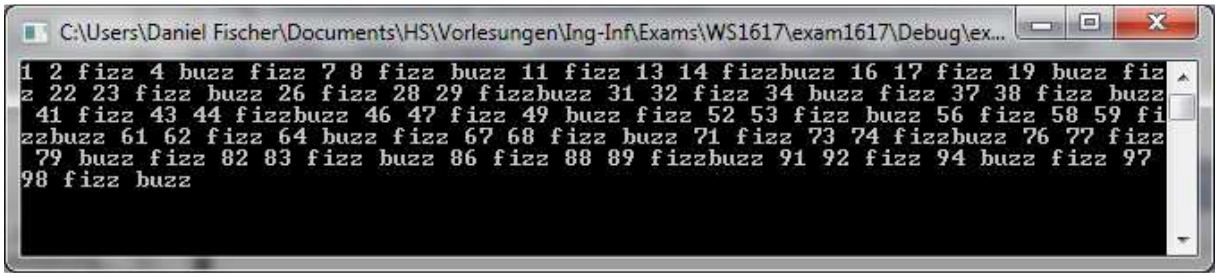

Abbildung 1

```
int i;

for (i = 1; i <= 100; i++)
{
  if (i % 3 == 0)
    printf("fizz");

  if (i % 5 == 0)
    printf("buzz");

  if (i % 5 != 0 && i % 3 != 0)
    printf("%d", i);

  printf(" ");
}
```

Weitere Alternativen sind natürlich auch richtig!

Diese Aufgabe wird häufig bei Bewerbungsgesprächen gestellt!

Aufgabe 3: (10 Punkte)

3.1 Wandeln Sie die folgenden positiven Zahlen (unsigned) in das andere Zahlensystem um: (**4 x 1 Punkt**)

1011000101_2 = **23011_4** ->von rechts immer zwei Binärziffern umwandeln
120_3 = **F_{16}** $->1*3^2+2*3^1+0*3^0=15_{10}=F_{16}$
143_5 = **48_{10}** $->1*5^2+4*5^1+3*5^0=48_{10}$
65535_{10} = **$FFFF_{16}$** ->MAX unsigned short int -> sizeof ist 2

3.2 Verständnisfragen **(3 x 2 Punkte nur bei korrekter Begründung):**

Die Funktion malloc liefert einen Rückgabewert vom Typ int*.
□ Stimmt
□ **Stimmt nicht**

Begründung:
malloc bekommt nur die **Anzahl der zu allokierenden Bytes übergeben** (size_t -> Datentyp ist nach 1999 C ISO Standard ein unsigned integer type mit mindestens 16 Bit – in MSVS 2015 ist dies ein unsigned int bei 32-Bit Compilierung) **und kann daher nicht wissen, was für einZeiger-Datentyp zurückgegeben wird**. Malloc liefert daher einen generischen, d.h. datentypunabhängigen, void*-Zeiger zurück.

scanf gilt als unsichere Funktion.
□ **Stimmt**
□ Stimmt nicht

Begründung:
Beim Einlesen von Strings (%s) wird einfach über die Größe der Zeigerkette hinaus überschrieben. Der Compiler liefert auch eine Warnung, wenn die **SDL-Checks** deaktiviert sind. Sind die SDL-Checks eingeschaltet, dann wird sogar ein Fehler vom Compiler generiert. Beim Anlegen eines Projektes wurden im Kurs die SDL-Checks deaktiviert.

Die Elemente (Nodes) einer verketteten Liste liegen **immer** hintereinander im Speicher (wie bei einem Array).
□ Stimmt
□ **Stimmt nicht**

Begründung:
In einer verketteten Liste (einfach verkettete Liste) enthält ein Element (node) wiederum einen Zeiger auf das nachfolgende Element (node). Dieser Zeiger hat oft den Namen **pnext**. **Das nachfolgende Element kann sich somit irgendwie im Speicher befinden.**

Matrikelnummer

Aufgabe 4: (10 Punkte)

Schreiben Sie hinter die Kommentare, was ***exakt*** in der Konsole auf einem 32-Bit System ausgegeben wird**.**
Im Kommentar sind die möglichen Punkte aufgeführt.

MyVars.h

```
struct flags
{
  unsigned char val1 : 3;
  unsigned char val2 : 4;
};

struct elements
{
  char ch[3];
};

union unall
{
  double dv;
  struct flags sf;
  struct elements se;
};

typedef struct flags * PF;
```

```
#include "MyVars.h"



void tpfs(void)
{
  unsigned char uc1;
  unsigned char uc2;
  union unall ua;
  int ia[5] = { 0,1,2,3,4 };
  int * p = &ia[4];
  struct elements data;
  data.ch[0] = 65;
  data.ch[1] = 0;
  data.ch[2] = 0;


  printf("%d\n", sizeof(struct flags));        // 1__________ (1P)
  printf("%d\n", sizeof(PF));                  // 4__________ (1P)
  printf("%d\n", sizeof(ua));                  // 8__________ (1P)
  printf("%s\n", data.ch);                     // A__________ (1P)
  printf("%d\n", sizeof(long long int));       // 8__________ (1P)
  printf("%d\n", *(--p));                      // 3__________ (1P)

  uc1 = 0x96;
  uc2 = 0xA5;
  printf("%x\n", (unsigned char)(uc1^uc2^uc2));      // 96_____ (2P)

  uc1 = 0xCE;
  uc2 = 0xAF;
  printf("%x\n", (unsigned char)((~uc1) & (uc2>>7))); // 1_____ (2P)
}
```

Vorletztes printf: Keinerlei Umrechnung notwendig: zweimaliges bitweise EXOR mit dem gleichen Wert (uc2) hebt sich auf.

Bei den letzten printfs wird der Wert zwar hexadezimal ausgegeben, allerdings erscheint kein 0x davor!

Aufgabe 5: (15 Punkte)

Adidas Sportschuhe C-Buch iPad 10 Kinokarten Smartphone Schachspiel Rucksack

Abbildung 2

In einer Datei „Geschenke.txt“ (Beispiel siehe links) sind alle Geschenke aufgelistet, die Sie zu Weihnachten bekommen. Jedes Geschenk steht dabei in einer Zeile (abgeschlossen mit CRLF). Der folgende Programmcode soll die Anzahl der Geschenke ausgeben. Der Programmcode enthält **einen Fehler** und müsste **noch sicherer** gemacht werden. Markieren Sie die beiden Stellen (+ Erklärung: 1-2 Sätze). (6+9 P)

```
#include <stdio.h>

#define BUFLEN 300

int count(const char *filename );

int main(void)
{

   printf("Ich bekomme %d Geschenke\n",count("Geschenke.txt"));

   return 0;
}


int count( const char * filename )
{

   FILE *file = 0;
   char buf[BUFLEN];
   int cnt = 0;

   file = fopen(filename,"r");

   if (file != NULL)
   {
      while(fgets(buf,BUFLEN-1,file))
      {
         cnt++;
      }

      fclose (file);
   }


   return cnt;
}
```

Aufgabe 6: (25 Punkte)

Schreiben Sie ein vollständiges C-Programm (Projektname sei check), welches zwei Zeichenketten (ASCII-Code 0-127) übergeben bekommt. Enthalten beide übergebenen Zeichenketten jeweils die gleiche Anzahl jedes Zeichens (ASCII-Code 0-127), so ist die Zeichenkette „Jedes Zeichen kommt gleich oft vor“ auszugeben. Ansonsten ist die Zeichenkette „Ungleich“ auszugeben. Stellen Sie sicher, dass das Programm nicht abstürzt. Funktionsweise von check siehe Abbildung 3.

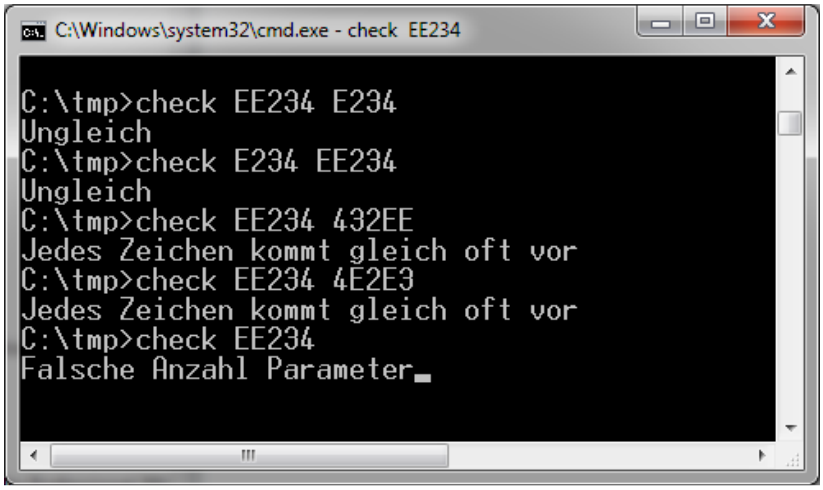

Abbildung 3

```
#include <stdio.h>
#include <string.h>
int main(int argc, char * argv[])
{
   unsigned char str1[128] = { 0 };
   unsigned char str2[128] = { 0 };
   char * plauf;

   if (argc == 3)
   {
      plauf = argv[1];
      while (*plauf)
      {
         str1[*plauf]++;
         plauf++;
      }

      plauf = argv[2];
      while (*plauf)
     {
         str2[*plauf]++;
         plauf++;
      }

      if (memcmp(str1, str2, 128) == 0)
         printf("Jedes Zeichen kommt gleich oft vor");
      else
         printf("Ungleich");
   }
   else
   {
      printf("Falsche Anzahl Parameter");
   }
   return 0;
}
```

Aufgabe 7: (20 Punkte)

Die **moderne Fibonacci-Folge f(n)**, wobei $n \in N_0$, ist die unendliche Folge von natürlichen Zahlen, die mit einer 0 und einer 1 beginnt. Im Anschluss ergibt sich aus der Summe zweier vorheriger Zahlen die unmittelbar danach folgende Zahl der Fibonacci-Folge.

0 1 1 2 3 5 8 13 21 34 usw.

f(0)=0 f(1)=1 f(2)=1 f(3)=2 f(4) =3 f(5)=5 f(6)=8 f(7)=13 f(8)=21 f(9)=34 usw.

Das allgemeine Bildungsgesetz für $n \in N_0$ lautet:

$$\boldsymbol{f(n)} = \begin{cases} \boldsymbol{0} & ; \boldsymbol{n = 0} \\ \boldsymbol{1} & ; \boldsymbol{n = 1} \\ \boldsymbol{f(n-1) + f(n-2)} & ; \boldsymbol{n > 1} \end{cases}$$

Beispiel: $f(5) = f(4) + f(3) = 3 + 2 = 5$

Implementieren Sie eine rekursive Funktion *fibonacci*, welche *f(n)* zurückgibt. Überläufe sind **nicht** zu berücksichtigen!

```
unsigned int fibonacci(unsigned int val)
{
  unsigned int uiret;

  if (val < 2 )
  {
    uiret = val;
  }
  else
  {
    uiret = fibonacci(val - 1) + fibonacci(val - 2);
  }

  return uiret;
}
```

Viel Erfolg!

Anhang: ASCII-Tabelle

Dez.	Hex.	Zeichen	Dez.	Hex.	Zeichen	Dez.	Hex.	Zeichen	Dez.	Hex.	Zeichen
0	0	NUL	32	20		64	40	@	96	60	`
1	1	SOH	33	21	!	65	41	A	97	61	a
2	2	STX	34	22	"	66	42	B	98	62	b
3	3	ETX	35	23	#	67	43	C	99	63	c
4	4	EOT	36	24	$	68	44	D	100	64	d
5	5	ENQ	37	25	%	69	45	E	101	65	e
6	6	ACK	38	26	&	70	46	F	102	66	f
7	7	BEL	39	27	'	71	47	G	103	67	g
8	8	BS	40	28	(	72	48	H	104	68	h
9	9	HT	41	29	)	73	49	I	105	69	i
10	0A	LF	42	2A	*	74	4A	J	106	6A	j
11	0B	VT	43	2B	+	75	4B	K	107	6B	k
12	0C	FF	44	2C	,	76	4C	L	108	6C	l
13	0D	CR	45	2D	-	77	4D	M	109	6D	m
14	0E	SO	46	2E	.	78	4E	N	110	6E	n
15	0F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	DEL

Anhang: memcmp

C Language: memcmp function (Compare Memory Blocks)

In the C Programming Language, the **memcmp function** returns a negative, zero, or positive integer depending on whether the first *n* characters of the object pointed to by *s1* are less than, equal to, or greater than the first *n* characters of the object pointed to by *s2*.

Syntax

The syntax for the memcmp function in the C Language is:

```
int memcmp(const void *s1, const void *s2, size_t n);
```

Parameters or Arguments

s1

An array to compare.

s2

An array to compare.

n

The number of characters to compare.

Returns

The memcmp function returns an integer. The return values are as follows:

Return Value	Explanation
0	*s1* and *s2* are equal
Negative integer	The stopping character in *s1* was less than the stopping character in *s2*
Positive integer	The stopping character in *s1* was greater than the stopping character in *s2*

Required Header

In the C Language, the required header for the memcmp function is:

```
#include <string.h>
```