Universität Mannheim
Fakultät für Mathematik und Informatik
Lehrstuhl für Praktische Informatik IV
Prof. Dr. Wolfgang Effelsberg

Teilprüfung
Software- und Internettechnologie
Programmierkurs 2
Sommersemester 2005

Name: .....
Vorname: .....
Matrikel-Nr.: .....
Studienfach: .....

Hinweise:

1. Überprüfen Sie die Klausur auf Vollständigkeit (**10** einseitig bedruckte Seiten).
2. Tragen Sie die Lösungen direkt in die Klausur ein. Benutzen Sie ggf. auch die Rückseiten der Aufgabenblätter.
3. Lösungen auf farbigem Konzeptpapier werden **nicht** bewertet.
4. Unterschreiben Sie die Klausur auf dem letzten Blatt.
5. Zugelassene Hilfsmittel: nicht programmierbarer Taschenrechner
6. Die Bearbeitungszeit beträgt 66 Minuten.

Aufgabe	max. Punktzahl	erreichte Punktzahl
1	13	
2	14	
3	16	
4	8	
5	15	
Summe	66	

Aufgabe 1: Verständnisfragen [13 Punkte]

a) [3 Punkte] Betrachten Sie folgende fehlerhafte C-Funktion, die alle Einträge eines `int`-Felds a der Länge `n` ausgeben soll:

```
void printFeld(int* a, int n){
  int i=0;
  while (i < n-1)
    printf("%i ",a[++i]);
}
```

Welche Ausgabe bewirken die folgenden Anweisungen?

```
int a[] = {1,2,5,7};
printFeld(a,4);
```

Wie muss die Funktion verändert werden, damit sie das Gewünschte leistet?

b) [4 Punkte] Welchen Wert haben die Variablen `a`, `b`, `c` und `d` nach Ausführung der folgenden C-Anweisungen?

```
unsigned int a = (15 && 12) + 013;
unsigned int b = 12 & 0x11;
unsigned int c = (a ^ 013) | 15;
unsigned int d = c >> (b+1);
```

c) [4 Punkte] Gegeben seien folgende C-Deklarationen:

```
typedef struct Angestellter{
  char* name;
  int gehalt;
} Angest;

Angest klaus;
```

Schreiben Sie eine C-Funktion `erhoeheGehalt`, die das Gehalt eines Angestellten um einen ganzzahligen Wert erhöht.

Geben Sie einen beispielhaften Funktionsaufruf für den Angestellten `klaus` an.

d) [2 Punkte] Geben Sie ein zur MSP430-Assembleranweisung `mov @r10+,r15` äquivalentes Codefragment an, das nicht die Adressierungsart *indirekt mit Postinkrement* verwendet.

Aufgabe 2: C-Programmierung [14 Punkte]

Schreiben Sie ein **vollständiges** C-Programm, das die Eulersche Zahl e mit Hilfe der Formel

$$e = \sum_{m=0}^{n-1} \frac{1}{m!} = 1 + 1 + \frac{1}{2} + \frac{1}{6} + \frac{1}{24} + \ldots + \frac{1}{(n-1)!}$$

näherungsweise berechnet:

- Lesen Sie zunächst die Anzahl der Iterationen n von der Standardeingabe ein. Bei unzulässigen Eingaben soll das Programm eine Fehlermeldung ausgeben und abbrechen. Gehen Sie davon aus, dass der Benutzer nur ganze Zahlen eingibt.
- Berechnen Sie eine Näherung für e mit Hilfe der oben angegebenen Formel. Durch *overflow* oder *underflow* entstehende Fehler können hierbei ignoriert werden.

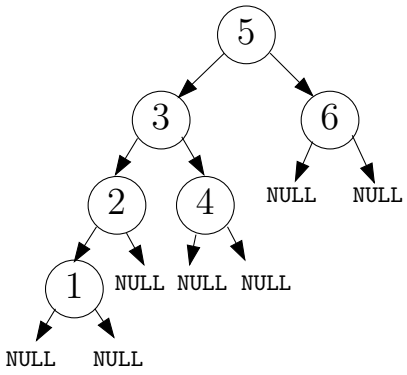
  Zur Erinnerung: Für $m \geq 0$ kann man die Fakultät von m berechnen als

  $$m! = \begin{cases} 1 & \text{für} \quad m = 0 \\ (m-1)! \cdot m & \text{für} \quad m > 0 \end{cases}$$

- Geben Sie das Ergebnis auf der Standardausgabe aus.

Aufgabe 3: Dynamische Datenstrukturen [16 Punkte]

In einem (geordneten) Binärbaum hat jeder Knoten außer der Wurzel genau eine eingehende Kante und höchstens zwei ausgehende Kanten, die zu seinem linken Sohn bzw. seinem rechten Sohn führen. Jeder Knoten des Binärbaums erfüllt die Ordnungsbedingung, dass sein Inhalt größer oder gleich dem Inhalt seines linken Sohns und kleiner als der Inhalt seines rechten Sohns ist, falls diese Knoten existieren.
In jedem Knoten wird ein `unsigned int` gespeichert. Der Datentyp `knoten` sei daher wie folgt definiert:




```
typedef struct node{
  unsigned int wert;
  struct node *linkerSohn;
  struct node *rechterSohn;
} knoten;
```

Falls ein Knoten keinen linken bzw. rechten Sohn hat, gilt `linkerSohn==NULL` bzw. `rechterSohn==NULL`.

a) [5 Punkte] Schreiben Sie eine C-Funktion

```
knoten* neuerKnoten(unsigned int neuerWert,
                    knoten* links,
                    knoten* rechts)
```

die einen neuen Knoten mit den übergebenen Parametern anlegt und einen Zeiger auf den Knoten zurückliefert. Dabei bezeichnen `neuerWert` den Inhalt des neuen Knotens sowie `links` und `rechts` den linken bzw. rechten Nachfolger-Binärbaum. Falls nicht genügend Speicherplatz für den neuen Knoten zur Verfügung steht oder die Parameter die Ordnungsbedingung verletzen, soll die Funktion `NULL` zurückgeben.

b) [5 Punkte] Schreiben Sie eine rekursive C-Funktion

```
void listeAbsteigend(knoten* b, void (*print)(unsigned int z))
```

die die im Binärbaum `b` gespeicherten Zahlen mit Hilfe der Funktion `print` absteigend sortiert ausgibt.

c) [6 Punkte] Implementieren Sie eine rekursive C-Funktion

```
int zaehle(knoten* b, unsigned int z)
```

die berechnet, in wie vielen Knoten des Binärbaums mit der Wurzel `b` die Zahl `z` gespeichert ist. Nutzen Sie dabei die Struktur des Baums aus, um möglichst wenige seiner Knoten zu besuchen.

Aufgabe 4: Sensorknotensteuerung [8 Punkte]

Schreiben Sie ein **Unterprogramm** in MSP430-Assembler, das in einer Endlosschleife ein abwechselnd rot und grün leuchtendes Hinweislicht mit Hilfe des in der Übung verwendeten Sensorknotens implementiert.

Ihr Unterprogramm soll die folgende Form besitzen:

```
.alert:    ...             ; Ihre Assemblerbefehle
           ...
           ...
           ret
```

Ein Hauptprogramm, das das Unterprogramm aufruft, braucht nicht angegeben zu werden.

Hinweise:

- Die drei Leuchtdioden des Sensorknotens werden über das Byte an der (absoluten) Adresse `0x0029` im Speicher des Prozessors gesteuert. Das niederwertigste Bit (Bit 0) an dieser Adresse steuert die rote, das Bit 1 die grüne und das Bit 2 die gelbe Leuchtdiode. Eine Leuchtdiode ist genau dann eingeschaltet, wenn das entsprechende Bit auf 0 gesetzt ist.
- Um die Leuchtzeiten zu steuern, kann die Unterprozedur `wait` verwendet werden. Die Wartedauer wird über die Register `R14` und `R15` übergeben, wobei die höherwertigen 16 Bit der Wartedauer in `R15` erwartet werden.

(Platz für die Lösung von Aufgabe 4)

Aufgabe 5: MSP430-Assembler [15 Punkte]

In dieser Aufgabe soll ein **Unterprogramm** in MSP430-Assembler implementiert werden, das zwei sortierte Integer-Felder zu einem sortierten Integer-Feld zusammenfügt.

Die Anfangsadresse bzw. die Anzahl der Einträge des ersten Felds werden in Register `R8` bzw. `R9` übergeben, die Adresse bzw. die Länge des zweiten Felds in `R10` bzw. `R11`. Die Adresse des Speicherbereichs, in dem das Ergebnis abgelegt werden soll, wird in Register `R12` übergeben. Alle Feldeinträge bestehen aus zwei Bytes.

Beispiel:

Parameter

```
R8 : 0x0300
R9 : #3
R10: 0x0400
R11: #4
R12: 0x500
```

Eingabefelder

Adresse	Inhalt
0x0300	#1
0x0302	#5
0x0304	#8
0x0400	#2
0x0402	#3
0x0404	#5
0x0406	#6

Ergebnisfeld

Adresse	Inhalt
0x0500	#1
0x0502	#2
0x0504	#3
0x0506	#5
0x0508	#5
0x050a	#6
0x050c	#8

Das Unterprogramm soll die folgende Form besitzen:

```
.merge:    ...              ; Ihre Assemblerbefehle
           ...
           ...
           ret
```

a) [5 Punkte] Formulieren Sie zunächst einen Algorithmus in **Pseudocode**, der die Einträge eines sortierten Felds `a[]` der Länge `lengthA` und eines sortierten Felds `b[]` der Länge `lengthB` in einem sortierten Feld `c[]` zusammenfügt.

b) [10 Punkte] Implementieren Sie nun Ihren Algorithmus in einem MSP430-Assembler Unterprogramm. Ein Hauptprogramm, das das Unterprogramm aufruft, braucht nicht angegeben zu werden.

Kommentieren Sie Ihr Programm sinnvoll!