Friedrich-Alexander-Universität Erlangen-Nürnberg

Schriftliche Prüfung zu der Lehrveranstaltung
„Grundlagen der Informatik II — Systemnahe Programmierung in C (SPiC)“
im Studiengang Elektrotechnik, Elektronik und Informationstechnik

	erreichbare Punkte	erhaltene Punkte	
Aufgabe 1	15		

Beispiele für Klausurfragen

die Fragen sind insgesamt nicht auf eine Bearbeitungszeit von 60 Minuten ausgerichtet, sondern sollen einen Eindruck vermitteln, wie mögliche Fragen aussehen könnten!
Die Prüfungsklausur wird aus 3 Teilen bestehen. Multiple Choice (ca. 15 Minuten), Programmieraufgabe(n) (ca. 30 Minuten) und weitere Fragen (textuell stichwortartig zu beantworten oder programmanalysen, ca. 15 Minuten)

(Name) (Vorname) (Matrikel-Nr.)

(Studiengang) (Semester)

Durch meine Unterschrift bestätige ich
- den Empfang der vollständigen Klausurunterlagen (xx Seiten inkl. Deckblatt),
- die Kenntnisnahme der Hinweise auf Seite 2.

Ich bin damit einverstanden, dass mein Prüfungsergebnis unter Angabe der Matrikel-Nr. veröffentlicht wird. ☐ ja ☐ nein

Erlangen, xx.xx.2006

(Unterschrift)

Hinweise

Die folgenden Informationen bitte aufmerksam lesen und die Erklärung am Ende dieser Hinweise unterschreiben.

- Die Bearbeitungszeit beträgt 60 Minuten.
- Es sind **keine** eigenen Hilfsmittel zugelassen.
- Die Lösung einer Aufgabe soll auf das Aufgabenblatt in den dafür vorgesehenen Raum geschrieben werden. Beachten Sie bitte, dass der freigelassene Platz großzügig bemessen ist und nicht unbedingt der erwarteten Antwortlänge entspricht. Sollte der Platz nicht ausreichen, können Sie die Rückseiten der Aufgabenblätter mitverwenden. Kennzeichnen Sie dabei die Zugehörigkeit Ihrer Lösung zu einer Aufgabe. Bei Bedarf können zusätzliche Lösungsblätter (weiß) ausgeteilt werden. Vermerken Sie vor deren Verwendung unbedingt Ihren Namen und Ihre Matrikelnummer darauf!
- Schmierpapier und Manualseiten dürfen **nicht** abgegeben werden. Bei Bedarf ist von der Aufsicht weiteres Schmierpapier (farbig) erhältlich.
- Das mitgebrachte handgeschriebene Notizblatt muss mit der Klausur am Ende abgegeben werden.
- Fragen zu den Prüfungsaufgaben können grundsätzlich nicht beantwortet werden.
- Tragen Sie Ihren Namen und Vornamen, Ihre Matrikelnummer, Studiengang und Fachsemesterzahl auf dem Deckblatt der Klausur ein.
- Bitte legen Sie Ihren Studenten- und einen Lichtbildausweis zur Kontrolle bereit.
- Sie dürfen den Raum nicht verlassen bevor Ihre Personalien überprüft wurden und Sie die Klausurunterlagen der Aufsicht zurückgegeben haben.
- In den letzten 15 Minuten der Bearbeitungszeit können Sie den Raum nicht mehr verlassen. Bleiben Sie an Ihrem Platz sitzen, bis schließlich alle Klausurunterlagen eingesammelt sind und die Aufsicht das Zeichen zum Gehen gibt.

Die **Klausurergebnisse** werden in ca. einer Wochen am schwarzen Brett vor dem Sekretariat des Lehrstuhls für Verteilte Systeme und Betriebssysteme (vor Raum 0.047 RRZE Erdgeschoss) und im WWW unter der Seite der Vorlesung im WS 2005/06, Unterpunkt "Ergebnisse" veröffentlicht (Benutzername: i4gdi, Passwort: s0s2k4).

Aufgabe 1:

Bei den Multiple-Choice-Fragen ist jeweils nur **eine** richtige Antwort eindeutig anzukreuzen. Auf die richtige Antwort gibt es die angegebene Punktzahl.

Wollen Sie eine Multiple-Choice-Antwort korrigieren, kreisen sie bitte die falsche Antwort ein und kreuzen die richtige an.

Lesen Sie die Frage genau, bevor Sie antworten.

a) Welche Aussage zur Sichtbarkeit von Variablen ist richtig? 2 Punkte

- ❏ Globale static-Variablen sind nur in einem Modul bekannt. In anderen Modulen sind Variablen mit gleichem Namen nicht erlaubt.
- ❏ Lokale static-Variablen werden innerhalb einr Funktion definiert und sind in damit auch in allen anderen Funktionen des gleichen Moduls bekannt.
- ❏ Auf globale Variablen kann nur außerhalb von Funtionen zugegriffen werden.
- ❏ Um auf globale Variablen eines anderen Moduls zugreifen zu können, müssen sie zunächst mit einer extern-Deklaration bekannt gemacht werden.

b) Welche der folgenden Aussagen zu Lebensdauer und Zugreifbarkeit von Variablen ist richtig? 2 Punkte

- ❏ Der Speicherplatz von globalen static-Variablen eines Moduls wird erst angelegt, wenn zum ersten Mal ein Funktion des Moduls aufgerufen wird.
- ❏ Der Speicherplatz von lokalen automatic-Variablen wird beim Aufruf der Funktion in der sie definiert sind bereitgestellt und beim Rücksprung aus der Funktion wieder freigegeben.
- ❏ Der Speicherplatz von lokalen automatic-Variablen ist zwar nach dem Rücksprung aus der zugehörenden Funktion nicht mehr erreichbar, bei einem erneuten Aufruf der Funktion sind solche Variablen aber wieder mit ihrem früheren Inhalt verfügbar.
- ❏ Es gibt keine andere Möglichkeit auf den Speicherplatz einer globalen Variablen eines anderen Moduls zuzugreifen, als die Variable mit einer extern-Deklaration bekannt zumachen.

c) Eine Funktion gibt einen Zeiger auf eine lokale automatic-Variable zurück. Welche Aussage hierzu ist richtig? 2 Punkte

- ❏ Zeiger auf lokale automatic-Variablen dürfen nicht zurückgegeben werden. Der Compiler wird eine Fehlermeldung erzeugen.
- ❏ Bei einem Zugriff über den zurückgegeben Zeiger wird die andere Funktuin einen Segmentation Fault auslösen.
- ❏ Ein Zugriff über den Zeiger nach dem Rücksprung liefert immer das Resultat 0.
- ❏ Ein Zugriff über den Zeiger kann in manchen Fällen den Wert liefern, den die Funktion dort zuletzt abgelegt hat. Dies ist aber nicht sicher.

d) In Betriebssystemen wie UNIX oder Linux unterscheidet man die Begriffe Programm und Prozess. Welche Aussage ist richtig? 2 Punkte

- ❏ Ein Programm darf immer nur einmal gleichzeitig auf einem Rechner gestartet werden.
- ❏ Ein Programm kann mehrfach gestartet werden. Jede Programmausführung erfolgt in einem Prozess. Tritt bei einer dieser Programmausführungen ein Fehler auf (z. B. Segmentation Fault), so werden alle Prozesse, die das Programm gerade ausführen abgebrochen.
- ❏ Ein Programm kann mehrfach ausgeführt werden. Jede Programmausführung hat dabei ihre eigenen Daten und kann auf die Daten anderer Ausführungen des gleichen Programms nicht zugreifen.
- ❏ Die Begriffe Programm und Prozess bedeuten eigentlich das Gleiche. Der Begriff Programm stammt aus der Windows-Welt, während man in Linux-Systemen statt dessen meisten von Prozessen spricht.

e) Was versteht man unter dynamischem Binden? 2 Punkte

- ❏ Bei dynamischem Binden werden Funktionen aus Bibliotheken nicht zu dem Programm dazugebunden. Erst beim Laden des Programms werden die Bibliotheksfunktionen zusammen mit dem Programm in den Speicher geladen.
- ❏ Bei dynamischem Binden werden Bibliotheksfunktionen nicht beim Compilieren oder Assemblieren zum Programm dazugebunden sondern erst wenn die eigentliche ausführbare Datei im letzten Bindeschritt erzeugt wird.
- ❏ Bei dynamischem Binden werden Funktionen erst zur Laufzeit des Programms zusammengebunden. Sollte eine Funktion dann nicht gefunden werden, bricht die Programmausführung mit Segmentation fault ab.

f) Welcher der folgenden Mechanismen gehört **nicht** zu den Funktionen eines Betriebssystemkerns wie z. B. UNIX oder Windows? 2 Punkte

- ❏ Dateisystem
- ❏ Prozessverwaltung
- ❏ Compiler
- ❏ Interrupt-Behandlung

g) Wenn man in einer Interrupt-Service-Routine längere Berechungen durchführt, kann dies zu Problemen führen. Welches der folgenden Probleme **gehört nicht dazu**? 2 Punkte

- ❏ Durch die gleichzeitige Sperre des Interrupt können weitere Interrupts verloren gehen.
- ❏ Je mehr Code ausgeführt wird, desto größer ist die Gefahr, dass auf Daten zugegriffen wird, die auch im eigentlichen Programmablauf genutzt werden.
- ❏ Beim Eintreffen anderer Interrupts werden ggf. andere Interrupt-Bearbeitungen ausgelöst. Dies kann zu einem sehr unübersichtlichen Programmablauf führen.
- ❏ Wenn eine Berechnung in einer Interrupt-Routine zu lange dauert, wird sie nach einem festen Timeout von der CPU abgebrochen.

h) In welcher der folgenden Situationen spricht man von Nebenläufigkeit? 2 Punkte

- ❏ Folgende Situation: Funktion f1 ruft Funktion f2 auf und Funktion f2 ruft ihrerseits wieder Funktion f1 auf.
- ❏ Wenn neben der eigentlichen Programmausführung Funktionsaufrufe durch Interrupts angestossen werden.
- ❏ Wenn ein Prozess durch Polling Ereignisse an einer externen Schnittstelle abfragt.
- ❏ Wenn durch Interrupt-Sperren während des Zugriffs auf kritische Datenstrukturen das Eintreffen von Interrupts verzögert wird.

Aufgabe 2:

Ein Mikrocontroller bietet 4 Ports mit jeweils 8 Anschluss-Pins an. Diese Anschluss-Pins können über zugeordnete Register angesteuert werden. Die Register sind in den Speicher des Mikrocontrollers auf folgende Adressen abgebildet:

Port 0: Adresse 0x40 Port 1: Adresse 0x48
Port 2: Adresse 0x50 Port 3: Adresse 0x58

Schreiben Sie eine Funktion

`void setport(int port, char pins[]);`

die Folgendes leistet:
Auf dem Port **`port`** werden die Pins entsprechend dem Muster in dem char-Feld **`pins`** gesetzt. Für jeden Pin gibt es ein Zeichen, **`0`** steht für Pin auf 0V setzen, **`1`** steht für Pin auf V_{cc} setzen, und **`x`** steht für Pin unverändert lassen.

Beispiel: setport(2, "011xxx10"); setzt Pin 0 und 7 auf 0, Pin 1, 2 und 6 auf 1 und lässt Pin 3, 4 und 5 unverändert.

Aufgabe 3:

Programmieren Sie eine Funktion feld_max, die ein Feld von integer-Werten und dessen Länge übergeben bekommt und als Ergebnis das Maximum der Werte zurückliefert.

Aufgabe 4:

Die folgende Funktion wird irgendwann während der Ausführung eines größeren Programms aufgerufen:

```
1   int f1(int a, double b) {
2       int *p;
3       *p = a;
4       while ( b-- != 0 ) {
5           *p++;
6       }
7       return(a+*p);
8   }
```

Welche beiden Anweisungen in der Funktion sind besonders kritisch und welche Fehler könnten bei diesen Anweisungen zur Laufzeit auftreten?

Aufgabe 5:

Beschreiben Sie wie die Interrupt-Bearbeitung bei einem Mikrocontroller (wie z. B. bei der AVR-Serie) abläuft.

Aufgabe 6:

Was gibt das folgende Programm **demo** in Zeile 12 aus, wenn es wie folgt aufgerufen wird:

```
demo out of memory
```

Geben Sie die Inhalte der verwendeten Variablen nach jeder ausgeführten Anweisung an. Verwenden Sie dazu die untenstehende Tabelle!

```
 1  #include <stdio.h>
 2  #define LEN 5
 3
 4  int main(int argc, char *argv[])
 5  {
 6    int i;
 7    char *zk, erg[LEN];
 8    for (i=0; argv[i] != NULL ; i++) {
 9      zk = argv[i]; erg[i] = zk[0];
10    }
11    erg[i] = '\0';
12    printf("%s\n", erg);
13    return 0;
14  }
```

Zeile	i	zk	erg erg[0] erg[1] erg[2] erg[3] erg[4]