Friedrich-Alexander-Universität Erlangen-Nürnberg

Schriftliche Prüfung zur Vorlesung
“Einführung in die Programmierung für Ingenieure (C)”

Aufgabe 1	Aufgabe 2	Aufgabe 3	Aufgabe 4	Aufgabe 5	Gesamt
40	**10**	**13**	**7**	**10**	**80**

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

(Fachrichtung) (Semester)

Erlangen, 30.07.1998

Organisatorische Hinweise

Die folgenden Informationen bitte aufmerksam lesen und die Erklärung am Blattende unterschreiben.

- Die Bearbeitungszeit beträgt 120 Minuten.
- Es sind **keine** Hilfsmittel zugelassen.
- Schmierpapier darf **nicht** mit abgegeben werden. Die Lösung einer Aufgabe soll auf das Aufgabenblatt in den dafür vorgesehenen freien Raum geschrieben werden, der jeweils der Aufgabe folgt. Sollte der Platz nicht ausreichen, so können Sie die Rückseiten der Aufgabenblätter mitverwenden.
- Bei Bedarf sind von der Aufsicht Schmierpapier (farbig) und zusätzliche Lösungsblätter (weiß) zu erhalten. Vermerken Sie vor dessen Verwendung unbedingt Ihren Namen und Ihre Matrikelnummer darauf! **Schmierpapier darf nicht mit abgegeben werden!**
- Fragen zu den Prüfungsaufgaben können grundsätzlich nicht beantwortet werden.
- Tragen Sie Ihren Namen und Vornamen, Ihre Matrikelnummer, Studienfachrichtung 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.
- Bleiben Sie bitte am Ende der Bearbeitungszeit solange an Ihrem Platz sitzen, bis alle Klausurunterlagen eingesammelt sind.

Die Klausurergebnisse werden am Ende (=Freitag!) der ersten Augustwoche am Abgabekasten ausgehängt!

Erklärung

Durch meine Unterschrift bestätige ich

- den Empfang der vollständigen Klausurunterlagen (11 Seiten inkl. Deckblatt).
- die Kenntnisnahme der obigen Informationen.

Ich bin damit einverstanden, daß mein Prüfungsergebnis unter Angabe der Matrikel-Nr. veröffentlicht wird.

☐ ja ☐ nein

Der Schein soll direkt an das Prüfungsamt geschickt werden:

☐ ja ☐ nein

Erlangen, 30.07.1998

(Unterschrift)

Aufgabe 1: (40 Punkte)

Schreiben Sie ein Programm **bsort** das mit Hilfe des "Bubble-Sort"-Algorithmus ein Array (= Feld) von ganzen Zahlen sortiert. Beim Aufruf soll man dem Programm auf der Kommandozeile mitteilen können, wieviele Zahlen einzulesen sind und ob sie in aufsteigender (Voreinstellung) oder absteigender Reihenfolge (Option **-u** angegeben) sortiert werden sollen. Ein Programmaufruf könnte z.B. wie folgt aussehen:

bsort -u 15

D.h. der Benutzer soll 15 Zahlen eingeben, die anschließend in absteigender Reihenfolge sortiert werden.

Ihnen sind die untenstehende **main**-Funktion sowie die Deklarationen der benötigten Funktionen vorgegeben. Sie sollen nun die deklarierten Funktionen implementieren, sowie zu den Funktionen **init_array** und **sort_array** jeweils ein **Struktogramm** erstellen. (**Achtung**: Verwenden Sie keine globalen Variablen!)

Beschreibung der einzelnen Funktionen:

a) **parse_ops**:
Bearbeitet die Kommandozeilenoptionen. Ist die Option **-u** vorhanden, soll **flag_u** den Wert 1 erhalten, andernfalls 0. Wenn eine Zahl vorhanden ist wird **num** auf den entsprechenden Wert gesetzt, ansonsten wird 10 eingestellt. Bei Fehlaufrufen (z.B. wenn die Zahl<=0 ist, etc.) soll das Programm eine Fehlermeldung ausgeben und terminieren. Zur Wandlung einer Ziffernfolge in einen Integerwert kann die Funktion **int atoi(char *str)** verwendet werden (wobei **str** einen Zeiger auf den zu konvertierenden String darstellt). Hinweis: Falls **str** keine Zahl repräsentiert liefert **atoi** den Wert 0.

b) **init_array**:
Hier muß Speicher für das Array beschafft werden. Anschließend sollen vom Benutzer die Werte in das Array eingetragen werden. Die Funktion liefert als Ergebnis einen Zeiger auf das Array. (zuerst Struktogramm erstellen!)

c) **sort_array**:
Die Werte im Array werden sortiert, indem das Array wiederholt von vorne durchlaufen wird und je zwei benachbarte Werte ihren Platz tauschen, falls der erste größer (bzw. kleiner bei gesetzter Option **-u**) als der zweite ist. So steigt in einem Durchlauf das größte (kleinste) Element wie eine "Blase" ans Ende des Arrays. Das wird wiederholt, bis in einem Durchlauf keine Elemente mehr getauscht werden mußten. (Struktogramm nicht vergessen!)

d) **print_array**:
Funktion zum Ausgeben aller Zahlen des Feldes.

e) **swap**:
Funktion zum Tauschen zweier Integerwerte.

f) **main**:
Ergänzen Sie die **main**-Funktion so, daß der dynamisch belegte Speicher wieder freigegeben wird.


```
#include <stdio.h>

void parse_ops(int argc, char *argv[], int *num, int *flag_u);
int *init_array(int num);
void sort_array(int *arr, int num, int flag_u);
void print_array(int *arr, int num);
void swap(int *a, int *b);

int main(int argc, char *argv[])
{
    int num, flag_u;
    int *arr;

    parse_ops(argc, argv, &num, &flag_u);
    printf("num = %d, flag_u = %d\n", num, flag_u);

    arr = init_array(num);
    sort_array(arr, num, flag_u);
    print_array(arr, num);

    /* Speicher freigeben (Teilaufgabe f): */


    return 0;
}
```

Aufgabe 2: (10 Punkte)

Markieren und verbessern Sie die Fehler (nicht mehr als 20!) in dem untenstehenden Programm, welches aus Personendaten einen Login-Namen erzeugt. Der Login-Name wird aus dem ersten und letzten Buchstaben des Vornamens und den ersten sechs Buchstaben des Nachnamens gebildet. Der Benutzer Brian Kernighan würde so z.B. BnKernig als Login-Namen erhalten.

Falsch erkannte Fehler werden negativ gewertet!

```
#include <stdio.h>
define NAME_LEN 30;

struct name
{
    vn[NAMELEN],
    nn[NAMELEN],
    int alter
}

int main()
{
    int j = 6;
    char loginname[9];
    struct name name = ('   ', '   ', 0);
    struct *n = &name;

    printf("Vorname: ");
    scanf("%s", n->vn);
    printf("Nachname: ");
    scanf("%s", n->nn);
    printf("Alter: ");
    scanf("%d", n->alter);

    loginname[0] = name.vn[0];
    for(i=0, n.vn[i], i++)
        ;
    loginname[1] = n->vn[i-1];
    i = 0;
    while(name.nn[i] & (i<j)) do
       loginname[i+2] = name.nn[i];
       ++i;
    loginname[i+2] = \0;
    printf("Loginname: %s\m", loginname);

}
```


Aufgabe 3: (13 Punkte)

Was gibt das folgende Programm **lo** in Zeile 9 aus, wenn es wie folgt aufgerufen wird: **lo hal** ? Geben Sie die Inhalte der verwendeten Variablen nach jeder ausgeführten Anweisung an! Verwenden Sie dazu die untenstehende Tabelle!

```
1   #include <stdio.h>
2   int f(char *a, char *b);
3   int main(int argc, char *argv[])
4   {
5     int i, j=0;
6     char a[256];
7     for(i=argc-1; i>=0; i--)
8       j += f(&a[j], argv[i]);
9     printf("count: %d, string: %s\n", j, a);
10    return j;
11  }

12  int f(char *a, char *b)
13  {
14    int c = 0;
15    while(*b)
16      {
17        *a = *b;
18        a++; b++; c++;
19      }
20    *a = '\0';
21    return c;
22  }
```

Zeile	i_{main}	j_{main}	a_{main}	c_f	a_f	b_f

Aufgabe 4: (7 Punkte)

Die Binomialkoeffizienten $\binom{n}{k}$ lassen sich durch folgende rekursive Berechnungsvorschrift ermitteln:

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

$$\binom{0}{0} = 1$$

$$\binom{n}{k} = 0 \qquad \text{für} \begin{cases} k > n \\ k < 0 \\ n < 0 \end{cases}$$

Schreiben Sie eine Funktion, die **rekursiv** – analog der oben angegebenen Formel – den Binomialkoeffizienten $\binom{n}{k}$ berechnet.

Aufgabe 5: (10 Punkte)

a) Nennen Sie zwei Gründe für die Verwendung von Funktionen

b) Erstellen Sie eine Struktur, mit der Personendaten (Vorname, Nachname, Geburtsdatum) in einer verketteten Liste verwaltet werden können. Gehen Sie davon aus, daß ein Name aus maximal 40 und ein Geburtsdatum aus maximal 8 Zeichen besteht.

Die Teilaufgaben c) bis e) beziehen sich auf die folgenden Programmodule, die zu einem lauffähigen Programm zusammengefügt werden (z.B. durch den Compileraufruf cc -Aa -o prog prog.c mod.c).

<u>Datei prog.c:</u>

```
#include "mod.h"

int c = 0;

int main()
{
   a = 0;
   b = 0;
   c = f(a);
   return 0;
}
```

<u>Datei mod.h</u>

```
extern int a;
extern int c;
int f(int i);
```

<u>Datei mod.c</u>

```
#include "mod.h"

int a;
static int b;

int f(int i)
{
   static int a = 0;
   int b = 0;
   a++;
   b++;
   c++;
   return a;
}
```

Kreuzen Sie bei den folgenden Teilaufgaben **alle wahren Aussagen** an. Es können pro Teilaufgabe mehrere Antworten richtig sein! Falsche Antworten zählen negativ.

c)

- ❑ In **prog.c** darf in Zeile 7 nicht auf **a** zugegriffen werden.
- ❑ In **mod.c**, Zeile 8, darf kein neues **a** angelegt werden, da bereits global ein **a** existiert (**mod.c**, Zeile 3).
- ❑ Die Erhöhung von **a** in **mod.c**, Zeile 10 ist nutzlos, da beim nächsten Aufruf von **f()** in Zeile 8 **a** wieder auf 0 gesetzt wird.
- ❑ Die Erhöhung von **a** in **mod.c**, Zeile 10 hat keinen Einfluß auf das in **main** angesprochene **a**.

d)

- ❑ In **prog.c**, Zeile 8, kann nicht auf **b** zugegriffen werden, der Compiler liefert deshalb eine Fehlermeldung.
- ❑ In der Funktion **f()** (**mod.c**, Zeile 9) darf kein neues **b** angelegt werden, da bereits global ein **b** existiert (**mod.c**, Zeile 4).
- ❑ In **mod.c**, Zeile 4, meldet der Compiler einen Fehler, da **static** nur innerhalb von Funktionen verwendet werden darf.
- ❑ Die Erhöhung von **b** in **mod.c**, Zeile 11 ist nutzlos, da einem weiteren Aufruf von **f()** die Variable **b** in Zeile 9 wieder auf 0 gesetzt werden würde.

e)

- ❑ Zeile 3 in **prog.c** ist fehlerhaft, da globale Variablen bei ihrer Definition nicht initialisiert werden können.
- ❑ Durch Zeile 3 in **prog.c** wird eine globale Variable angelegt, auf die auch in **mod.c** zugegriffen werden darf.
- ❑ Zeile 3 in **prog.c** steht im Konflikt mit Zeile 2 in **mod.h**.
- ❑ Der Ausdruck **c++** in mod.c, Zeile 12, erhöht den Wert der in **prog.c** definierten globalen Variable **c**.
- ❑ Der Ausdruck **c++** in mod.c, Zeile 12, erzeugt einen Fehler, da zwar eine globale Variable **c** existiert, diese aber in **mod.c** nicht bekannt ist.