Probeklausur Programmieren in C
Sommersemester 2007
Dipl. Biol. Franz Schenk
12. April 2007, 13.00-14.45 Uhr
Bearbeitungszeit: 105 Minuten

- Schalten Sie ihr Mobiltelefon aus.
- Bei der Klausur ist **als einziges Hilfsmittel** eines unserer Skripte erlaubt.

Zum **Bestehen** der Klausur sind **45** Punkte hinreichend.

Hinweise zu den Aufgaben:

- Gültiger Sprachstandard ist wie in der Vorlesung ANSI-C (89).
- Eine Aufgabe wird gewertet, wenn sie prinzipiell das Richtige tut. Auch richtige Teilabschnitte können Punkte geben. Systematische Syntaxfehler, nicht korrekte Umsetzung der Aufgabe oder Fehler im Algorithmus führen zu Punktabzügen.
- Falls nicht verlangt, so schreiben Sie keine vollständigen Programme, sondern nur C-Funktionen, wie angegeben! Wenn Sie Funktionen der Standardbibliothek verwenden, schreiben Sie die entsprechende `#include`-Zeile vor die Funktionsdefinition. Wenn nicht verlangt ist, dass eine Funktion Ausgaben produziert, dann dürfen Sie auch keine Ausgabefunktion benutzen!
- Mit den Aufgaben 1 und 2 sollen Sie Funktionen implementieren. Überlegen Sie erst, wie die Funktion auf eine klare Strukturierung
- Aufgaben 3, 4, und 5 sind keine Programmieraufgaben. Hier geht es um Ihr Verständnis von C.

Gutes Gelingen!

	Max. Punkte	Erwartet für “4”
Aufgabe 1 (Listensuche)	20	10
Aufgabe 2 (Stringersetzung)	30	15
Aufgabe 3 (Rekursion)	20	5
Aufgabe 4 (Programmfehler)	10	5
Aufgabe 5 (Ausdrücke)	25	10
Summe	105	45

Aufgabe 1 (Listensuche [20 Punkte])
Gegeben ist als Datenstruktur eine verkettete Liste.

```
struct Knoten
{
        int Wert;
        struct Knoten * Next;
} ;
```

Schreiben Sie eine Funktion, welcher die Adresse des Anfangselements der Liste übergeben wird sowie ein Suchwert:

```
int SucheWert(struct Knoten * Anfang, int SuchWert)
```

Die Funktion soll folgendes realisieren:

- Die Funktion soll 1 zurückgeben, falls der Wert gefunden wurde, 0 zurückgeben, falls er nicht gefunden wurde.
- Falls ein Knoten gefunden wird, welcher den gesuchten Wert trägt, so soll er aus der Liste entfernt werden.
- Sie müssen sich nicht darum kümmern, dass allozierter Speicher wieder freigegeben wird!
- Wir gehen der Einfachheit halber davon aus, dass das erste Element der Liste ein leeres Element ist, also bei Vergleich und Löschen nicht beachtet werden muss.

Lösung

```
int SucheWert(struct Knoten * Anfang, int SuchWert)
{
        struct Knoten * p = Anfang;

        while (p != NULL)
        {
                if (p->Next != NULL && p->Next->Wert == SuchWert)
                {
                    p->Next = p->Next->Next;
                    /* Suche Fortsetzen, falls ein Element
                       gefunden wurde */
                    SucheWert(p->Next,SuchWert);
                    return 1;
                }
                p = p->Next;
        }
        return 0;
}
```

Aufgabe 2 (Stringersetzung [30 Punkte])
Schreiben Sie eine Funktion, die drei Strings als Argumente empfängt und im ersten String jedes Vorkommen des zweiten Strings durch den (gleichlangen!) dritten String inplace ersetzt und einen Zeiger auf den ersten String zurückgibt.
Ein Aufruf mit `("abcdefghijlkm", "def", "123")` soll also `"abc123ghijklm"` als Rückgabe zur Folge haben.

```
const char* dieFunktion(char* s, const char* t, const char* u);
```

- Implementieren Sie die Funktion gemäss der Beschreibung und dem gegebenen Funktionsprototypen.
- Aus der Standardbibliothek dürfen Sie nur die Funktion strlen() verwenden. Sind die Strings t und u nicht gleichlang, gibt die Funktion nur einen Zeiger auf s zurück.

Lösung

```
#include <string.h> /* f\"ur das strlen() */
const char* dieFunktion(char* s, const char* t, const char* u)
{
        int i, j, k;

        if (strlen(t) != strlen(u))
                return s;

        for (i=0; i<strlen(s); i++)
                if (s[i] == t[0])
                {
                        int tunichts = 0;
                        for (j=0; j<strlen(t); j++)
                                if (s[i+j] != t[j])
                                        tunichts = 1;
                        if (tunichts == 0)
                                for (k=0; k<strlen(u); k++)
                                        s[i+k] = u[k];
                }

        return s;
}
```

Aufgabe 3 (Rekursion [20 Punkte])
Gegeben ist folgendes Programmfragment:

```
float mystery_function(float *a, unsigned int k)
{
    if(k == 1)
         return *a;
    else
         return *a + mystery_function(a+1, k-1);
}
```

Die Funktion aufgerufen, indem ihr ein Feld von Float-Werten sowie eine ganzzahlige Zahl übergeben wird.

- Welche einfache mathematische Funktion wird mit der Funktion *mystery_function* implementiert?

  Lösung Die Summe des Feldes wird berechnet.

- Schreiben Sie eine nicht-rekursive Version dieser Funktion.

  Lösung

```
int nichtrekursiv(float *a, int n)
{
  int i;
  float summe;
  for (i = 0; i < n ; i++)
    summe += a[i];
  return summe;
}
```

Aufgabe 4 (Programmfehler [10 Punkte])
Das folgende Programm berechnet die Summe aller Zahlen eines Feldes. Die Adresse des Feldes sowie seine Länge werden der Funktion übergeben.
Die Funktion gibt aus, dass die Summe aller Zahlen im Feld durch 2 teilbar ist, wenn die Summe aller Zahlen im Feld durch 2 teilbar ist. Die Funktion gibt aus, dass die Summe aller Zahlen im Feld nicht durch 2 teilbar ist, wenn die Summe aller Zahlen im Feld nicht durch 2 teilbar ist.
Im Programm stecken zahlreiche Fehler.

- Acht dieser Fehler bewirken, dass das Programm nicht korrekt compiliert. Markieren und berichtigen Sie diese Fehler. [8 Punkte]

- Zwei Fehler betreffen die Richtigkeit des Algorithmus (es wird also nicht immer oder gar nicht oben beschriebenes Verhalten auftreten. [2 Punkte]

  Zeigen Sie diese Fehler und schreiben Sie Korrekturen, so dass ein lauffähiges Programm ohne Warnungen nach dem ANSI-C 89 Standard compiliert wird (mit den üblichen Parametern -Wall -pedantic -ansi). Zeigen Sie auch, welcher Fehler verhindert, dass der Algorithmus korrekt funktioniert.

```
01 #include <studio.h>
02
03 void myFunction(int* intField; int length)
04 {
05   int c;
06   int j;
07   int k = 0;
08   j = 1;
09   while(j < length)
10   {
11     c += *intField;
12     intField++;
13     j = j+1;
14   }
15
16   if(c % 2 = 0)
17   {
18     printf(Summe durch 2 teilbar!\n);
19     return 1
20   }
21   else
22
23     printf("Summe nicht durch 2 teilbar!\n");
24     return 0;
25   }
26 }
27
```

Lösung

algorithmische Fehler:

Zeile 05: c ist nicht initialisiert, kann zu falscher berechnung
fuehren, verbesserung: int c = 0;
Zeile 08: j startet bei 1, sollte bei 0 starten, verbesserung: j = 0;

syntaktische Fehler:

Zeile 01: falsches include
Zeile 03: ; statt ,
Zeile 07: k wird nicht benutzt, erzeugt Warning
Zeile 16: c%2 ist kein lvalue, verbesserung: ==
Zeile 18: " " um den String f\"ur printf fehlen
Zeile 19: ; fehlt
Zeile 19 und Zeile 25: kein void return, erzeugt Warning
Zeile 22: einleitende { fehlt f\"ur den else-zweig

Aufgabe 5 (Ausdrücke [25 Punkte])

- Gegeben ist ein Programm mit einigen Ausdrücke. Markieren Sie, welche der folgenden Ausdrücke compilieren oder die Compilation des Programms mit einer Fehlermeldung verhindern!

  [Jede richtige Antwort gibt 2 Punkte, jede Falsche Antwort -1 Punkt.]

Code	compiliert	compiliert nicht

int main(void)
{
int i=0, j[2] = {1,0}, k;
int *p = &i;		

int printf, * const v = j + 1;	☐	☐
`putchar("tf"[j[i % 7]]);`	☐	☐
&*&*&*p = &k;	☐	☐
i > 0 ? k = 1 : k = -1;	☐	☐
for(; ;) break;	☐	☐
return 0; }		

Lösung

Code	compiliert	compiliert nicht
int printf, * const v = j + 1;	☒	☐
`putchar("tf"[j[i % 7]]);`	☒	☐
&*&*&*p = &k;	☐	☒
i > 0 ? k = 1 : k = -1;	☐	☒
for(; ;) break;	☒	☐

- Lesen Sie die folgenden Ausdrücke. Welchen Wert nehmen die Variablen nach den

Zuweisungen an?

[1 Punkt pro richtigem Variablenwert, 2 Punkte pro richtigem Variablenwert bei Teilaufgabe c]

(a)
```
int a = 10;
int b = 20;
int c = 30;
a *= 3;
c -= --b + a++;
```

a. . . . . b. . . . . c. . . . . .

Lösung a = 31 b = 19 c = -19

(b) Felder

```
int feld[2][5] = {{1,2,3,4,5},{6,7,8,9,10}};
int *p;
p = feld[0];
```

feld[1][1]..... *(p+3) .....

Lösung feld[1][1] = 7 *(p+3) = 4

(c) Lokale und Gloable Variablen

```
1  int i,j;
2  int main(void)
3  {
4     i = 2;
5     i = testFunktion1();
6     return i;                          i = .........   j = ........
7  }
8  int testFunktion1(void)
9  {
10    int i=0;
11    for (j=0; j<10; j++) i=i+17;
12    testFunktion2(i);
13    return j;                          i = .........   j = ........
14 }
15 int testFunktion2(int input)
16 {
17    i *= input;
18    return i;                          i = .........   j = ........
19 }
```

Lösung main() i=10 j=10 testFunktion1() i=170 j=10
testFunktion2() i=340 j=10

(d) Zeiger

```
int i = 4, j = 2; int *zi, **zzi;
  zi = &i;
 *zi = 3;
  zzi = &zi;
*zzi = &j;
**zzi = 5;
```

i..... j.....

Lösung i = 3 j = 5