Dr. habil. Bernd Schürmann
Dr. Annette Bieniusa
pinc-support@cs.uni-kl.de

TU Kaiserslautern
Fachbereich Informatik

Lösungshinweise zum Übungsblatt 6: Programmieren in C (WS 2018/19)

1. Zur Beantwortung von Fragen und Hilfe bei Problemen stehen wir Ihnen immer in der Praktische Übung, Mittwoch 15:30, Raum 32-410-PC zur Verfügung sowie per Email an `pinc-support@cs.uni-kl.de`!
2. Sie können sich im Exclaim System unter `https://softech.cs.uni-kl.de/exclaim` mit Ihrem persönlichen STATS-Account einloggen und Dateien zu den einzelnen Übungen hochladen.
3. Die Aufgaben auf diesem Übungsblatt sind alle freiwillig zu bearbeiten.

Aufgabe 1: Operatoren

Abgabe: –

Berechnen Sie die Werte in folgendem Programmfragment der Reihe nach:

```
int a = 0, b = 4, c = 8, d = 3;

a = b + c;          // a=
a = b - c;          // a=
a = c / b;          // a=
a = c * b;          // a=
a = c % b;          // a=
a = c % 3;          // a=
a = 1 + b * d;      // a=
a = (1 + b) * d;    // a=

a = 8; c = 2;
a = a + 1;          // a=
a = a + b * 4;      // a=
a = a / 2;          // a=
a = a % 2;          // a=

b = 5;
a = b / 2;          // a=
c = b % 2;          // c=
d = b << a;         // d=
b = b * -3;         // b=
d = d % 3;          // d=
c = c + (b * d + 4);  // c=
a = --b + d++;     // b=    , a=    , d=

b = 3; c = 2;  d = 6;
a = (b + 2) * 2 * c + 1;     // a=
a = ++b * d++ * ++c * (-1); // a=
b = 1 - --b;
```

```
int a = 0, b = 4, c = 8, d = 3;

a = b + c;          // a = 12
a = b - c;          // a = -4
a = c / b;          // a = 2
a = c * b;          // a = 32

a = c % b;          // a = 0
a = c % 3;          // a = 2

a = 1 + b * d;      // a = 13
a = (1 + b) * d;    // a = 15
```

```
a = 8; c = 2;
a = a + 1;          // a = 9
a = a + b * 4;      // a = 25
a = a / 2;          // a = 12
a = a % 2;          // a = 1

b = 5;
a = b / 2;          // a = 2
c = b % 2;          // c = 1
d = b << a;         // d = 20
b = b * -3;         // b = -15
d = d % 3;          // d = 2
c = c + (b * d + 4);  // c = -25
a = --b + d++;      // a = -14, b = -16, d = 3

b = 3; c = 2;  d = 6;
a = (b + 2) * 2 * c + 1;     // a = 21
a = ++b * d++ * ++c * (-1); // a = -72
b = 1 - --b;                 // Undefiniertes Verhalten!!
```

Die Semantik der letzten Anweisung ist undefiniert. Die Verwendung der Increment-/Decrement-Operatoren wie in den letzten beiden Zeilen führt zu schwer verständlichen Code und hat oft auch undefiniertes Verhalten; bitte vermeiden Sie diese Art von Ausdrücken!

Aufgabe 2: Zeiger

Abgabe: –

1. Ergänzen Sie die folgenden Lücken:

   (a) Ein Zeiger repräsentiert die Adresse und den _____ eines Objekts.

   Ein Zeiger repräsentiert die Adresse und den Typ eines Objekts.

   (b) Angenommen, die Zeiger `p1` und `p2` sind vom selben Typ. Wie lautet die Zuweisung, um `p2` auf dasselbe Objekt wie `p1` zeigen zu lassen?

   ```
   p2 = p1;
   ```

   (c) Wenn `k` eine int-Variable ist, hat der Ausdruck `&k` den Typ _____.

   ```
   int *
   ```

   (d) Die Adresse einer Zeigervariablen `ptr` liefert der Ausdruck _____. Skizzieren Sie die Situation zum Verständnis!

   ```
   &ptr
   ```

   (e) Gegeben sei eine double-Variable `y`. Die Definition eines Zeigers `dptr`, der so initialisiert wird, dass es auf `y` zeigt, lautet _____.

   ```
   double *dptr = &y;
   ```

   (f) Gegeben sei der Zeiger `iptr`, der auf die int-Variable `i` mit Wert 5 zeigt. Was bewirkt der Ausdruck `2**iptr` ?

   Er wertet aus zum Wert 10.

   (g) Der float-Zeiger `fptr` enthalte die Adresse 20000. Nach der Zuweisung `fptr = fptr + 1;` enthält `fptr` die Adresse _____.

   20004

   (h) Wieso ist folgende Anweisung fehlerhaft?

   ```
   int a = 2, int b = 4, *p1 = &a, *p2 = &b;
   printf("%d\n", *(p1+p2));
   ```

> Bei der Auswertung des Ausdrucks `*(p1+p2)` werden zwei Addresswerten addiert; dies ist nicht erlaubt.

2. Berechnen Sie von Hand in jeder Zeile die Variablen- und Zeigerwerte:

```
int a = 2, b = 5, *c = &b, *d = &a;

a = *c * *d;     // a =   , b =
*d -= 3;         // a =   , b =
b = a * b;       // a =   , b =
c = d;           // a =   , b =
b = *d - 13;     // a =   , b =
a = *c % *d;     // a =   , b =
```

```
int a = 2, b = 5, *c = &b, *d = &a;

a = *c * *d;     // a = 10 , b = 5
*d -= 3;         // a = 7 , b = 5
b = a * b;       // a = 7 , b = 35
c = d;           // a = 7 , b = 35
b = *d - 13;     // a = 7 , b = -6
a = *c % *d;     // a = 0 , b = -6
```

Aufgabe 3: Speicherrepräsentierung von Daten

Abgabe: speicher.c

In der Vorlesung wurde erläutert, dass der Speicherbedarf für Integer-Zahlen (int, short, long) vom Prozessor abhängt.

- Schreiben Sie ein C-Programm, welches die tatsächlichen Speichergrößen für die einzelnen Integervarianten Ihres Computers anzeigt. Man übergibt dafür dem `sizeof()` Befehl den Datentyp und erhält die Speichergröße in Byte. Wie groß ist der jeweilige Wert in Bit für char, short, int, long, float und double? Das Programm soll auch die Wertebereiche, d.h. die kleinste und größte Zahl, des jeweiligen ganzzahligen Datentyps (z.B. `INT_MAX`, `INT_MIN` und `UINT_MAX`) ausgeben. Die Namen der Konstanten finden Sie in `limits.h`.
- Ergänzen Sie das Programm, so dass es Ihnen die Werte und die Adressen folgender Variablen `a`,`b`,`c` und `d` ausgibt, wie sie im Speicher des Rechners abgelegt sind. Wählen Sie passende Formatelemente für die Ausgabe. Ermitteln Sie zusätzlich die Größe der Adressen in Bit (d.h. den Speicherbedarf für einen Pointer).

  ```
  int a = 46, b = 89;
  double c = 5.3, d=2017.411;
  ```

  Deklarieren Sie geeignete Pointer für die Variablentypen. Weisen Sie Ihnen die Werte der Variablen zu und geben Sie den Wert, die Speicheradresse und die Größe des Pointertyps aus.

  Die Ausgabe kann folgendermaßen aussehen:

  ```
  sizeof(char)   = 1 Byte; ([-128 ... 127])
  sizeof(short)  = 2 Byte; ([-32768 ... 32767])
  sizeof(int)    = 4 Byte; ([-2147483648 ... 2147483647])
  sizeof(long)   = 8 Byte; ([-9223372036854775808 ... 9223372036854775807])
  sizeof(float)  = 4 Byte;
  sizeof(double) = 8 Byte;
  Wert von a: 46, Speicheradresse von a:0x7ffee8c94718
  Wert von b: 89, Speicheradresse von b:0x7ffee8c94714
  Wert von c: 5.300000, Speicheradresse von c:0x7ffee8c94708
  Wert von d: 2017.411000, Speicheradresse von d:0x7ffee8c94700
  ```

  Hinweis: Der Wert des Pointers ist in jedem Programmlauf verschieden!

```c
#include <stdio.h>
#include <limits.h>

int main(void)
{
    printf("sizeof(char)   = %ld Byte; ([%d ... %d])\n", sizeof(char),
    CHAR_MIN, CHAR_MAX);
    printf("sizeof(short)  = %ld Byte; ([%d ... %d])\n", sizeof(short),
    SHRT_MIN, SHRT_MAX);
    printf("sizeof(int)    = %ld Byte; ([%d ... %d])\n", sizeof(int), INT_MIN,
     INT_MAX);
    printf("sizeof(long)   = %ld Byte; ([%ld ... %ld])\n", sizeof(long),
    LONG_MIN, LONG_MAX);
    printf("sizeof(float)  = %ld Byte;\n", sizeof(float));
    printf("sizeof(double) = %ld Byte;\n", sizeof(double));

    int a = 46, b = 89;
    double c = 5.3, d = 2017.411;
    printf("Wert von a: %d, Speicheradresse von a:%p\n", a, &a);
    printf("Wert von b: %d, Speicheradresse von b:%p\n", b, &b);
    printf("Wert von c: %f, Speicheradresse von c:%p\n", c, &c);
    printf("Wert von d: %f, Speicheradresse von d:%p\n", d, &d);

    return 0;
}
```

Aufgabe 4: Arrays

Abgabe: `array.c`

Schreiben Sie ein C-Programm, das beliebig viele Integer-Zahlen von der Konsole einliest und in ein Array speichert. Das Array soll die Länge N haben (N soll als Präprozessor-Konstante mit Wert 10 definiert werden). Es sollen solange Zahlen von der Konsole eingelesen werden, bis eine Null gelesen wird oder das Array voll ist.

- Entwerfen Sie das Flussdiagramm.
- Implementieren Sie Ihren Algorithmus.
- Wandeln Sie Ihr Programm so ab, dass bei jeder eingegebenen Zahl die beiden vorangegangenen Zahlen auf die aktuelle Eingabe aufaddiert und die Summe im Array eingetragen wird. (Beachten Sie hierbei den Sonderfall der beiden zuerst eingelesenen Zahlen!)

Beispiel: Bei Eingabe von `1 2 3 0` soll `1 3 6` ausgegeben werden.

```c
#include <stdio.h>
#define N 10

int main(void)
{
    int i =0; // Fuellgrad des Arrays
    int eingabe;
    int a[N];
    int letzte = 0, vorletzte = 0;

    while (i < N)
    {
        scanf("%d", &eingabe);
        if (eingabe == 0)
        {
            break;
        }
        else
        {
            a[i] = eingabe + letzte + vorletzte;
            vorletzte = letzte;
            letzte = eingabe;
            i++;
        }
    }
```

```
    for (int j = 0; j < i; j++)
    {
        printf("%d ", a[j]);
    }

    printf("\n");
    return 0;
}
```

Aufgabe 5: Zugriff auf Komponenten von Strukturen

Abgabe: `zeit.c` in Exclaim

Übernehmen Sie das Beispiel zur Systemzeit in C/Unix von den Vorlesungsfolien (Kap. 03, Seite 96), übersetzen es und führen Sie es aus. Ändern Sie das Programm ab, dass es die Sekundenanzahl von der Konsole einliest und ändern Sie die Ausgabe, so dass bei Eingabe von 1543165768 folgendes ausgegeben wird:

```
Sonntag, den 25. November 2018, 18 Uhr 9
```

Hinweis: Verwenden Sie zum Umwandeln von Eingabe folgende Anweisungen:

```
int eingabe;
scanf("%d", &eingabe);              // Einlesen der Sekundenanzahl
sekunden = (time_t)eingabe;         // Umwandeln in Wert vom Type time_t
ortszeit = localtime(&sekunden);    // Ermitteln der Ortszeit
```

```
#include <time.h>
#include <stdio.h>

int main(void)
{
    time_t sekunden;
    struct tm *ortszeit;
    char *tag[7] = {"Sonntag", "Montag", "Dienstag", "Mittwoch", "Donnerstag",
     "Freitag", "Samstag"};
    char *monat[12] = {"Januar", "Februar", "Maerz", "April", "Mai", "Juni", "
    Juli", "August", "September", "Oktober", "November", "Dezember"};

    // Einlesen
    int eingabe;
    scanf("%d", &eingabe);
    sekunden = (time_t)eingabe;
    ortszeit = localtime(&sekunden);

    printf("%s, den %d. %s %d, %d Uhr %d\n", tag[ortszeit->tm_wday], ortszeit
    ->tm_mday, monat[ortszeit->tm_mon], 1900 + ortszeit->tm_year, ortszeit->
    tm_hour, ortszeit->tm_min);
}
```

Aufgabe 6: Arrays von Strukturen

`Abgabe: score.c`

Ein Spielprogramm benötigt eine Score-Liste (Tabelle) mit den Namen und Punkteständen der Spieler. Die Score-Liste soll maximal 10 Spieler aufnehmen können. Schreiben Sie ein C-Programm, das die folgenden Schritte ausführt:

- Definieren Sie zunächst einen Strukturtyp `player`, der den Namen und den Punktestand eines Spielers speichern kann. Ein Spielername soll nur aus einem Wort bestehen (max. 20 Zeichen); der Punktestand eine natürliche ganze Zahl.
- Lesen Sie zunächst 10 verschiedene Spielerdaten von der Konsole ein und speichern Sie diese in einem Array ab.
- Geben Sie dann die Spielerdaten in der eingegebenen Reihenfolge wieder aus. Definieren Sie dazu einen Pointer auf Strukturen vom Typ `player`. Initialisieren Sie diesen Pointer mit

dem ersten Listen-Element. Iterieren Sie nun über die Einträge zu den Spielern, indem Sie auf die Namen und Punktestände über den Pointer zugreifen und den Pointer auf die nächsten Eintrag in der Score-Liste weitersetzen.

- Geben Sie dann die Spielerdaten in umgekehrter Reihenfolge aus.
- Ermitteln Sie die Summe der Punkte aller Spieler.
- Ermitteln Sie den Spieler mit dem höchsten Punktestand.
- Geben Sie die Spieler sortiert nach Punktestand aus, den Spieler mit den meisten Punkten zuerst. Falls zwei Spieler den gleichen Punktestand haben, sollen sie nach der Reihenfolge in der Eingabeliste geordnet sein.

Beispiel

Eingabe:

```
10, anna
16, bert
19, carl
25, david
11, erwin
78, franz
56, gustav
98, hans
20, immi
91, jan
```

Ausgabe:

```
anna : 10
bert : 16
carl : 19
david : 25
erwin : 11
franz : 78
gustav : 56
hans : 98
immi : 20
jan : 91
jan : 91
immi : 20
hans : 98
gustav : 56
franz : 78
erwin : 11
david : 25
carl : 19
bert : 16
anna : 10
Gesamtpunktzahl: 424
Bester Spieler ist hans mit 98 Punkt(en).
hans : 98
jan : 91
franz : 78
gustav : 56
david : 25
immi : 20
carl : 19
bert : 16
erwin : 11
anna : 10
```

```c
#include <stdio.h>
#include <string.h>
#define LAENGE 10
#define MAX_LAENGE_NAME 20

struct player
{
    char name[MAX_LAENGE_NAME + 1];
    int points;
};

int main(void)
{
    struct player score_liste[LAENGE];
```

```c
int i;

for (i = 0; i < LAENGE; i++)
{
    scanf("%d, ", &score_liste[i].points);
    // Einlesen von max. MAX_LAENGE_NAME vielen Zeichen
    fgets(score_liste[i].name, MAX_LAENGE_NAME, stdin);
    // Entfernen des Zeilenumbruchs am Ende der Eingabe
    int j = strcspn(score_liste[i].name, "\n");
    score_liste[i].name[j] = '\0';

    /* Variante:
        scanf("%d, %s", &score_liste[i].points, score_liste[i].name);
        Achtung: Diese Variante kann zu sicherheitskritischen
Speicherzugriffen führen, falls die Eingabe länger als MAX_LAENGE_NAME ist
.
     */
}

for (i = 0; i < LAENGE; i++)
{
    printf("%s : %d\n", score_liste[i].name, score_liste[i].points);
}

for (i = LAENGE - 1; i >= 0; i--)
{
    printf("%s : %d\n", score_liste[i].name, score_liste[i].points);
}

int summe = 0;
for (i = 0; i < LAENGE; i++)
{
    summe += score_liste[i].points;
}
printf("Gesamtpunktzahl: %d\n", summe);

int max = score_liste[0].points;
struct player *max_player = &score_liste[0];
for (i = 1; i < LAENGE; i++)
{
    if (score_liste[i].points > max)
    {
        max = score_liste[i].points;
        max_player = &score_liste[i];
    }
}
printf("Bester Spieler ist %s mit %d Punkt(en).\n", max_player->name,
max_player->points);

// Sortieren der Eintraege mittels Sortieren durch Auswahl

for (i = 0; i < LAENGE; i++)
{
    int max = score_liste[i].points;
    int max_pos = i;
    for (int j = i; j < LAENGE; j++)
    {
        if (score_liste[j].points > max)
        {
            max = score_liste[j].points;
            max_pos = j;
        }
    }
    struct player helper = score_liste[max_pos];
    score_liste[max_pos] = score_liste[i];
    score_liste[i] = helper;
}

// Ausgabe der sortierten Liste
for (i = 0; i < LAENGE; i++)
{
    printf("%s : %d\n", score_liste[i].name, score_liste[i].points);
}
```

```
    return 0;
}
```