PR3 - Übungsblatt A.3

Prof. Dr. Holger Peine
Hochschule Hannover
Fakultät IV – Abteilung Informatik
Raum 1H.2.60, Tel. 0511-9296-1830
Holger.Peine@hs-hannover.de

Thema

Datenorganisation und Zeiger

Termin

Ihre Arbeitsergebnisse zu diesem Übungsblatt führen Sie bitte bis zum 30.10.2025 vor. Versuchen Sie aber bitte, die Aufgabe 2 ***schon bis zum 23.10.*** *vorzustellen, sonst dürfte es in den Übungen am 30.10. ziemlich hektisch zugehen.*

1 Speichergrößen skalarer Datentypen (0 Punkte)

basiert auf Vorlesung bis einschl. Abschnitt 4.a

Verwenden Sie die `sizeof`-Funktion, um die Speichergrößen der skalaren Datentypen (einschließlich ihrer `long`- und `short`-Varianten) und des Datentyps `char*` (Zeiger auf `char`, s. Abschnitt 5.a) auf dem benutzten Rechner heraus zu finden. Kompilieren und testen Sie Ihr Programm wenn möglich auf verschiedenen Rechnern, z.B. auf einem 32-Bit-Rechner und einem 64-Bit-Rechner.

2 *n*-dimensionale Matrix (1 Punkt)

basiert auf Vorlesung bis einschl. Abschnitt 4.c

Schreiben Sie eine Funktion `matrix_elem_sum`, die die Summe aller Elemente einer *n*-dimensionalen Matrix aus `double`-Zahlen zurückgibt. Dabei sollen neben der Matrix auch *n* (also die Anzahl der Dimensionen) und auch die *n* Größen der einzelnen Dimensionen als Parameter übergeben werden (letztere in Form eines `int`-Arrays der Länge *n*).

Benutzen Sie das folgende Programm zum Testen, das Sie auch im Moodle-Kurs in `PR3_U_A.3_Anlage/matrix_elem_sum.c` finden. Ignorieren Sie bitte vorerst den Typ der übergebenen Matrix (also `double matrix[]` in der Parameterdeklaration, aber `double*` beim Aufruf); dieser Zusammenhang wird in Kapitel 5 der Vorlesung klar werden.
Lösungshinweis: Es sind keine verschachtelten Schleifen nötig – beachten Sie das Speicherlayout von mehrdimensionalen Arrays in C! Sie müssen dazu verstehen, warum es genügt, dass `matrix_elem_sum()` die Matrix wie einen eindimensionalen Array benutzt. Wie sähe derselbe Fall in Java aus?

```
#include <stdio.h>

double matrix_elem_sum(double matrix[], int n, int lengths[]) {
    /* Ihre Aufgabe */
}

int main(void) {
  double matrix2d[3][2] =
    { {111, 112}, {121, 122}, {131, 132} };
  int lengths2d[2] = {3, 2};

  double matrix3d[2][3][4] = {
    { {111, 112, 113, 114}, {121, 122, 123, 124}, {131, 132, 133, 134} },
```

```
    { {211, 212, 213, 214}, {221, 222, 223, 224}, {231, 232, 233, 234} }
  };
  int lengths3d[3] = {2, 3, 4};

  printf("Sum of matrix2d = %f\n",
        matrix_elem_sum((double*)matrix2d, 2, lengths2d)); /* gibt 729 aus /
  printf("Sum of matrix3d = %f\n",
        matrix_elem_sum((double*)matrix3d, 3, lengths3d)); /* gibt 4140 */

  return 0;
}
```

3 Gemischte Arrays in Java und C (0 Punkte)

basiert auf Vorlesung bis einschl. Abschnitt 4.b

a) Betrachten Sie die Implementierung von Arrays in Java und C (Folien 4-19 und 4-20). Wäre es möglich, auf dieser Grundlage auch "gemischte" Arrays, also Arrays, bei denen die einzelnen Elemente unterschiedliche Typen haben dürfen, zu implementieren? Mit "möglich" ist dabei gemeint, dass jedes Elements im Speicher eindeutig gefunden werden könnte, und nicht, ob die Sprache überhaupt eine Syntax dafür bietet. Beantworten Sie die Frage für Java und für C.
b) Ist es möglich, einen zweidimensionalen Array von int-Elementen anzulegen, dessen Elemente (also die eindimensionalen Sub-Arrays) unterschiedlich lang sind? Beantworten Sie die Frage für Java und für C.

4 Speicherbedarf von Arrays in Java und C (0 Punkte)

basiert auf Vorlesung bis einschl. Abschnitt 4.b

Wie viele Bytes im Speicher belegt ein zweidimensionaler Array von 10x5 `int`'s in Java und in C, wenn ein `int` 4 Bytes und eine Objektreferenz 8 Bytes belegt?

5 `strcat` nachprogrammieren (0 Punkte)

basiert auf Vorlesung bis einschl. Abschnitt 4.c

Implementieren Sie eine Funktion `mystrcat(char s1[], char s2[])`, die die Funktion `strcat` aus der Standard-Bibliothek (siehe http://www.cplusplus.com/reference/cstring/strcat/ und auch Folie 4-35) zum Anhängen des Strings `s2` an den String `s1` nachahmt. `strcat` überprüft nicht, ob am Ende von `s1` noch genügend Platz für `s2` ist. `strcat` gibt immer `s1` zurück.

Sie können Ihr `mystrcat` mit dem Programm `mystrcat_test.c` aus den Anlagen zu diesem Übungsblatt testen. Implementieren Sie dazu Ihren Code als zwei Dateien `mystrcat.c` und `mystrcat.h`.

6 Strukturtyp und Zeichenketten (2 Punkte)

basiert auf Vorlesung bis einschl. Abschnitt 4.d

Schreiben Sie ein Programm, das positive ganze Zahlen mit bis zu 20 Stellen erzeugen, addieren und ausgeben kann. Solch große Zahlen werden in der Regel von den eingebauten

skalaren Typen nicht unterstützt. Die größte durch `long` darstellbare Zahl ist meist 2^{31}-1 = 2147483647 mit 10 Dezimalstellen.

Programmieren Sie einen Strukturtyp namens `int20`, der die Zahl intern als Folge von Zeichen (`char`) speichert, für jede Dezimalziffer ein Zeichen. Eine Variable dieses Typs belegt dann genau 20 Bytes.[1] Folgender Code soll anschließend übersetzt werden:

```
struct int20 a= create20("12345678901234567890");
struct int20 b= create20("100");
struct int20 sum= add20(a, b);
print20(a); printf("\n");
print20(b); printf("\n");
print20(sum); printf("\n");
```

Die gewünschte Ausgabe ist:

```
12345678901234567890
100
12345678901234567990
```

Ein weiteres Beispiel:

bitte wenden

```
struct int20 a= create20("9700");
struct int20 b= create20("422");
struct int20 sum= add20(a, b);
print20(a); printf("\n");
print20(b); printf("\n");
print20(sum); printf("\n");
```

Die gewünschte Ausgabe ist:

```
9700
422
10122
```

Lösungshinweise:

- Für die Funktion create20 sieht der Prototyp so aus:
  `struct int20 create20(char val[]);`
  d.h. die Funktion bekommt einen `char`-Array übergeben, aus dem sie das struct initialisiert.
- Auch wenn der Parameter von `create20()` ein String ist, ist es trotzdem nicht ratsam, die Zeichen in einem `int20` als String (also mit `'\0'` am Ende) zu speichern (weil sie dort nicht als String, sondern als einzelne Ziffern verarbeitet werden): Verarbeiten Sie die Zeichen wie üblich bei Arrays, also mit Schleifen.
- Die Addition können Sie so implementieren, wie Sie sie einmal in der Schule gelernt haben (ziffernweise bei der letzten Stelle beginnend, jeweils mit Übertrag für die nächste Stelle).
  Ein Zeichen wie '2' können Sie in den entsprechenden int-Wert 2 umwandeln, indem Sie den int-Wert des '0'-Zeichens substrahieren:
  `int cAsInt = c - '0';`
  Daraus können Sie auch ableiten, wie Sie in umgekehrter Richtung umwandeln können.

[1] Man könnte die Speicherung in dem Strukturtyp natürlich noch optimieren. Tatsächlich reichen 8 Bytes aus, um 20-stellige positive ganze Zahlen zu speichern. Da wir dann aber einiges an Bit-Operationen durchzuführen hätten, verzichten wir erst einmal auf diese Speicherplatzersparnis.

7 Zeiger (1 Punkt)

basiert auf Vorlesung bis einschl. Abschnitt 5.a

Betrachten Sie folgenden Programmcode:

```
#include <stdio.h>
int main(void) {
    int i;
    int* ip;
    printf("Eingabe: ");
    scanf("%d", &i);
    ip = /* Hier fehlt Ihr Code */ ;
    printf("Eingabe war %d\n",
           /* Hier einen Ausdruck einsetzen, der nur ip, aber nicht i enthält */ );
    return 0;
}
```

Ergänzen Sie Code nur an den vorgesehenen Stellen, so dass die vom Benutzer eingegebene Zahl ausgegeben wird. Sie dürfen nur Code ergänzen und keinen bestehenden Code ändern oder entfernen.