Human-Computer Interaction Group

Leibniz Universität Hannover

Programmieren 1 - WS 2022/23

Prof. Dr. Michael Rohs, Tim Dünte, M.Sc., Jan Feuchter, M.Sc.

Präsenzübung 8

Diese Aufgaben sind zur Lösung während der einstündigen Präsenzübung gedacht. Sie können die Aufgaben auf einem mitgebrachten Laptop oder auf Papier lösen.

Aufgabe 1: Arrays und Strings

Schreiben Sie eine Funktion, die alle Leerzeichen aus einem String entfernt. Der Originalstring soll „in-place“ verändert werden. Die Funktion darf keine anderen Funktionen aufrufen. Die Template-Datei ist remove_spaces.c. Die Signatur der Funktion ist `void remove_spaces(char* s)`. Beispielaufruf mit einem char-Array der Länge 16:

Argument: `Hello w o r l d\0`

Ergebnis: `Helloworld\0 l d\0` (neues String-Ende beim ersten `\0` bei Index 10)

`0123456789012345` (Indizes)

Aufgabe 2: Zeiger und Dereferenzierung

Gegeben sei folgende Situation im Speicher:

1000..1003	1004..1007	1008..1015	1016..1023
0	1	1000	1008
a	b	c	d

a) Definieren Sie die Variablen a-d in der Sprache C. Die Variablen a und b sind vorzeichenbehaftete ganzzahlige Variablen, c und d sind Zeigervariablen.
b) Was sind die Werte von: `&a`, `a`, `*a` und `**a`?
c) Was sind die Werte von: `&c`, `c`, `*c` und `**c`?
d) Was sind die Werte von: `&d`, `d`, `*d` und `**d`?
e) Was ist der Effekt von `**d = **d + 2;` ?
f) Was ist der Effekt von `*d = *d + 1;` ?

Aufgabe 3: Zeiger und Typumwandlungen

Was ist die Ausgabe dieses Programfragments (auf einem little-endian-Prozessor)?

```
long a = 0x8877665544332211;
char *c = (char*)&a;
printf("%02x\n", *c);
short *s = (short*)&a;
printf("%04x\n", *s);
int *i = (int*)&a;
printf("%08x\n", *i);
long *l = (long*)&a;
printf("%016lx\n", *l);
```

Aufgabe 4 (optional): Binäre Suche

Erläutern Sie, warum die nachfolgende, fehlerhafte Implementierung der binären Suche nicht funktioniert. Geben Sie ein Beispiel an, für das die Funktion nicht das korrekte Ergebnis liefert.

```
int binary_search(int a[], int n, int x) {
    require("sorted", forall(int i = 0, i < n - 1, i++, a[i] <= a[i+1]));
    if (a == NULL || n <= 0) return -1;
    int i = 0;
    int j = n;
    do {
        int mid = (i + j) / 2;
        if (x < a[mid]) {
            j = mid;
        } else {
            i = mid + 1;
        }
    } while (i < j);
    if (j < n && a[j] == x) return j;
    return -1; // no match
}
```

Beispieldaten:

```
    int a[] = { 3, 5, 6, 11, 12, 14, 16, 17 };
    int n = sizeof(a) / sizeof(int); // n == 8
```

Aufgabe 5 (optional): Standardeingabe und Standardausgabe

Schreiben Sie ein Programm, das die Zeichen in einer Textdatei zählt. Die Zeichen sind im UTF-8-Format codiert. Im UTF-8-Format werden Zeichen durch 1-4 Bytes codiert. Die 7-Bit-ASCII-Codierung ist eine Teilmenge der UTF-8-Codierung: ASCII-Zeichen werden als einzelne Bytes codiert, bei denen das höchstwertige Bit gelöscht ist. In zwei Bytes codierte Zeichen beginnen mit den höchstwertigen Bits 110, in drei Bytes codierte Zeichen mit 1110 und in vier Bytes codierte Zeichen mit 11110. Bei Zeichen, die in mehreren Bytes codiert sind, folgt auf das initiale Byte eines oder mehrere Bytes, die mit den Bits 10 beginnen. Diese Codierung ist in der folgenden Tabelle zusammengefasst. Die x stehen für die einzelnen Bits eines Zeichens. Für die 1-Byte-Codierung stehen also 7 Bits zur Verfügung, für die 2-Byte-Codierung 11 Bits, usw.

Anz. Bytes	Byte 1	Byte 2	Byte 3	Byte 4	Beispiel
1	0xxxxxxx				X
2	110xxxxx	10xxxxxx			ä
3	1110xxxx	10xxxxxx	10xxxxxx		€
4	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx	⊙

Ihr Programm muss nur mit korrekt UTF-8-codierten Textdateien umgehen können. Die Textdatei soll über die Standardeingabe eingelesen werden. Beispiel:

```
./count_utf8 < utf8.txt
```