4.11.A Datenstrukturen/Rückgabe von Arrays durch Funktionen – Arbeitsblatt

Aufgabe 1: Funktion `static_upcase`

Die Funktion `static_upcase` bekommt einen String als Parameter. Er soll in einen zweiten String kopiert werden, dabei sollen in der Kopie alle Kleinbuchstaben in Großbuchstaben umgewandelt werden. Die Startadresse der Kopie soll zurückgegeben werden:

```
1  char *p;
2  p = static_upcase("abc");
3  printf("Ergebnis: %s\n", p);
```

Als Ergebnis dieses Programmstücks sollte `ABC` auf dem Bildschirm erscheinen.

Dazu muss die Funktion im Innern eine lokale Variable enthalten. Die enthaltene lokale Variable muss außerdem mit dem Schüsselwort `static` eine Lebensdauer bekommen, die der Programmdauer entspricht.

a) Wie lautet die Vereinbarung der Variablen?

---

b) Wie lautet der Prototyp der Funktion?

---

c) Schreiben Sie die Funktion!

d) Betten Sie die Funktion in ein Programm `static_upcase.c` ein, welches die Funktion einmal aufruft und das Ergebnis ausgibt.

Aufgabe 2: Funktion `static_plus2space`

Die Funktion `static_plus2space` bekommt einen String als Parameter. Sie kopiert den String in einen zweiten String, wandelt dabei aber jedes Pluszeichen in ein Leerzeichen um. Wieder soll zum Schluss die Startadresse der Kopie an die aufrufende Funktion zurückgegeben werden:

```
1  char *p;
2  p = static_plus2space("So+ein+schoenes+Wetter");
3  printf("Ergebnis: %s\n", p);
```

Die Funktion muss im Innern eine mit `static` gekennzeichnete lokale Variable enthalten.

a) Wie lautet die Vereinbarung der Variablen?

---

b) Wie lautet der Prototyp der Funktion?

---

c) Schreiben Sie die Funktion!

d) Betten Sie die Funktion in ein Programm `static_plus2space.c` ein, welches die Funktion einmal aufruft und das Ergebnis ausgibt.

e) Die Funktion soll ergänzt werden: Falls die Länge des Strings, der als Parameter übergeben wurde, größer ist als die interne lokale Variable, soll der `NULL`-Zeiger zurückgegeben werden (`static_plus2space2.c`).

Aufgabe 3: Aufruf-Zähler

Mit einer `static`-Variablen kann man eine Funktion zählen lassen, wie oft sie aufgerufen wurde.

a) `aufrufzaehler.c`
Schreiben Sie eine Funktion `void az(void)`, die bei jedem Aufruf eine lokale `static`-Variable um eins hochzählt und auf dem Bildschirm ausgibt! Rufen Sie die Funktion dreimal in Ihrem Programm auf. Hinweis: Vergessen Sie nicht, die Variable zu initialisieren!

b) `erstmals.c`
Schreiben Sie die Funktion `void erstsodannso(void)`, die beim ersten Aufruf `Das ist neu.` ausgibt und bei jedem weiteren Aufruf `Das kenne ich schon.`. Das soll mit Hilfe einer `static`-Variablen verwirklicht werden. Rufen Sie die Funktion dreimal in Ihrem Programm auf. Hinweis: Vergessen Sie nicht, die Variable zu initialisieren!

Aufgabe 4: Funktion `itoa`

In manchen C-Umgebungen ist sie vorhanden, aber im Standard fehlt sie: Die Funktion `itoa()`. Sie wandelt eine `int`-Zahl in eine Zeichenkette um. Aus der `int`-Zahl 2014 (im RAM im Dualsystem als 0000011111011110 gespeichert) wird das `char`-Array `{'2', '0', '1', '4', '\0'}` bzw. in Kurzform `"2014"`. Im Innern der Funktion muss eine Dual- nach Dezimal-Wandlung stattfinden.

Zur Erinnerung: Bei der Dezimal- nach Dual-Wandlung wurde die Ursprungszahl immer wieder durch zwei geteilt, bis das Ergebnis null war. Die zwischendurch entstandenen Reste bildeten dann die gesuchte Dual-Darstellung, nur in umgekehrter Reihenfolge:

```
11(dez.)=1011(dual)
11/2=5 Rest 1
 5/2=2 Rest 1
 2/2=1 Rest 0
 1/2=0 Rest 1
```

Ebenso kann auch der Computer die Wandlung ins Dezimalsystem vornehmen:

```
621(dual)=621
621/10=62 Rest 1
 62/10=6  Rest 2
  6/10=0  Rest 6
```

Diese (hier drei) Ziffern müssen dann noch einzeln in ASCII-Zeichen übersetzt werden und fertig ist die Umwandlung von `int` nach `char*` (nur die Terminierung fehlt noch).

a) Erstellen Sie eine Funktion `char *beispiela(void)`, die die Zeichenkette `"bestanden"` zurückgibt! Denken Sie daran, den Rückgabestring als `static`-Variable zu deklarieren! Testen Sie Ihre Funktion mit dem Aufruf `printf(beispiela());`!

b) Erstellen Sie eine Funktion `char *beispielb(char a, char b, char c)`, die aus den drei Zeichen `a`, `b` und `c` eine terminierte Zeichenkette macht und diese zurückgibt. So soll beim Aufruf `beispielb('T', 'C', 'P')` als Ergebnis die Zeichenkette `"TCP"` geliefert werden.
Testen Sie Ihre Funktion mit dem Aufruf `printf(beispielab('T', 'C', 'P'));`!

c) Erstellen Sie eine Funktion `char *beispielc(int z2, int z1, int z0)`, die aus den Zahlen `z2`, `z1` und `z0` die Zeichenkette macht, die zu $100z_2 + 10z_1 + z_0$ passt. Beispiel: Der Aufruf `beispielc(6,2,1)` soll als Ergebnis die Zeichenkette `"621"` liefern.

Damit dem ASCII-Code Genüge getan wird, muss zu jeder Ziffer die Zahl 48 hinzugefügt werden, damit aus der Ziffer 0 die 48 wird (=`'0'`), aus der Ziffer 1 die 49 (=`'1'`), aus der Ziffer 2 die 50 (=`'2'`) usw.

Denken Sie wieder daran, den Rückgabestring als `static`-Variable zu deklarieren! Denken Sie auch daran, den String zu terminieren!

d) Erstellen Sie nun eine Funktion `void beispield(unsigned int x)`, die — wie oben beschrieben — nacheinander die Zahl `x` zerlegt und die Ziffern auf den Bildschirm ausgibt. Der Aufruf `beispielc(621)` soll als Ergebnis die Ausgabe `1,2,6` haben. Die Funktion ist vom Rückgabetyp `void` und braucht daher nichts zurückgeben.

e) Nun ist die Funktion `char *beispiele(unsigned int x)` dran: Auch sie soll die Zahl `x` zerlegen. Die Ziffern sollen nun aber nacheinander in einem String abgelegt werden. Damit dem ASCII-Code Genüge getan wird, muss wieder zu jeder Ziffer 48 hinzugefügt werden.

Denken Sie wieder daran, den String zu terminieren!
– Test mit: `printf(beispield(621);`

f) Die Funktion aus der letzten Teilaufgabe gibt den String noch in verkehrter Reihenfolge wieder. In `char *utoa(unsigned int x)` soll der String daher zusätzlich (am Schluss der Funktion) umgedreht werden.

g) In `char *itoa(int x)` schließlich soll berücksichtigt werden, dass `x` auch negativ sein kann. In dem Fall muss vorher das Zweierkomplement gebildet werden.
– Test mit: `printf(itoa(-621);`