4.9.A Datenstrukturen/In-Place-Substitution – Arbeitsblatt

- Die folgenden Aufgaben dienen der Übung; verwenden Sie deshalb bitte keine Funktionen aus `<string.h>`.
- Legen Sie in den Funktionen keine Hilfs-Arrays an.
- Verwenden Sie in Ihren Funktionen keine (globalen) Konstanten (wie z. B. `BUFLEN`), sondern nur die Werte aus der Parameterliste. Ihre Funktionen sind dann universell einsetzbar (*Prinzip der schmalen Datenkopplung*).

Aufgabe 1: Erstellen einer Zeichenkette durch eine Funktion

Jedes Programm dieser Aufgabe ist gleich aufgebaut:

```
1 #include <stdio.h>
2 #define BUFLEN 41
3 int main(void)
4 {
5     char slcd[BUFLEN]="Irgendein Inhalt";
6     makestring(slcd, BUFLEN);
7     printf("Neuer Inhalt von slcd: >>%s<<\n", slcd);
8     return 0;
9 }
```

Die Zeichenkette `slcd` soll den Inhalt einer 40 Zeichen langen LCD-Zeile aufnehmen können. Zuerst wird `slcd` vereinbart und mit einem Wert initialisiert. Anschließend ruft man die Beispielfunktion `makestring` der entsprechenden Teilaufgabe auf. Dadurch soll `slcd` einen neuen Inhalt bekommen. Zum Schluss wird der neue Wert von `slcd` ausgegeben. Wichtig ist: Nur durch den zweiten Parameter weiß `makestring`, wie viele Zeichen (in diesem Fall mit Terminator) `slcd` maximal aufnehmen kann.

a) Welchen Prototyp muss `makestring` haben?

---

b) `makestring1.c`: Der gesamte Platz von `slcd` soll mit `'\0'` gefüllt werden.

c) `makestring2.c`: Die Variable `slcd` soll an allen sichtbaren Stellen mit Punkten aufgefüllt werden. Anschließend soll `slcd` terminiert werden.

d) `makestring3.c`: Zu Testzwecken soll die Variable `slcd` an allen sichtbaren Stellen mit dem Muster 012345678901234... aufgefüllt werden. Anschließend soll `slcd` terminiert werden.

Aufgabe 2: Substitution von Zeichen einer Zeichenkette durch eine Funktion

Auch in dieser Aufgabe ist wieder jedes Programm im Prinzip gleich aufgebaut: Zuerst wird wieder eine Zeichenkette vereinbart, diesmal mit einer Länge von 71 Zeichen. Zuerst soll ein String von der Tastatur eingelesen und sofort auf den Bildschirm ausgegeben werden. Anschließend soll eine Funktion den String verändern. Dabei wird die Länge des Strings nicht verändert! Zum Schluss soll der veränderte String ausgegeben werden. Hier ein Beispieldialog zur ersten Teilaufgabe:

Terminal

```
schueler@debian964:~$ a.out
Zeichenkette eingeben: Bus+AND+USB+OR+Firewire
Zeichenkette: >>Bus+AND+USB+OR+Firewire<<
Zeichenkette: >>Bus AND USB OR Firewire<<
```

a) `ips_subst1.c`: Zeichenketten, die aus einem Formular in einer Webseite stammen, enthalten oft anstelle der Leerzeichen Pluszeichen (das liegt an der CGI-Erweiterung des HTTP-Protokolls). Erstellen Sie eine Funktion `void plus2space(char *s)`, die eine erhaltene Zeichenkette so modifiziert, dass jedes Pluszeichen durch ein Leerzeichen ersetzt wird.

b) `ips_subst2.c`: Die Funktion `void leichtverschleiern(char *s)` soll in der Zeichenkette `s` jede Ziffer durch die Ziffer `0` ersetzen.

c) `ips_subst3.c`: Die Funktion `void starkverschleiern(char *s)` soll in der Zeichenkette `s` jede Ziffer durch die Ziffer `0` ersetzen, jeden Kleinbuchstaben durch das Zeichen `a` und jeden Großbuchstaben durch das Zeichen `A`.

d) `ips_subst4.c`: Die Funktion `void kleinnachgross(char *s)` soll in der Zeichenkette `s` jeden Kleinbuchstaben durch den entsprechenden Großbuchstaben ersetzen.

e) `ips_subst5.c`: Die Funktion `void grossnachklein(char *s)` soll in der Zeichenkette `s` jeden Großbuchstaben durch den entsprechenden Kleinbuchstaben ersetzen.

f) `ips_subst6.c`: Beim Geocaching und in News-Groups gibt es an manchen Stellen Botschaften, bei denen der Leser vor versehentlichem Lesen geschützt wird. Bei den News-Groups wird er vor umstrittenen oder eventuell anstößigen Passagen geschützt, beim Geocaching vor Hinweisen, die den Spaß an der Suche verderben können. Der Schutz geschieht dadurch, dass man die Botschaften durch ein einfaches Verfahren verschlüsselt, das außerdem jeder kennt: Die ROT13-Verschlüsselung.

   ROT13 entsteht, indem jeder Buchstabe im Alphabet um 13 Stellen nach hinten geschoben wird. Nach dem Ende des Alphabets wird wieder bei A begonnen. Die Buchstaben A bis M werden also wie folgt ersetzt: `c=c+13;` Die am Ende des Alphabets liegenden 13 Buchstaben N bis Z werden so ersetzt: `c=c+13; c=c-26;` Diese Verschlüsselung ist symmetrisch, d.h., wenn man sie auf einen verschlüsselten String noch einmal anwendet, erhält man wieder das Original.

   Erstellen Sie eine Funktion `rot13()`, die mit In-Place-Substitution eine ROT13-Verschlüsselung vornimmt, so dass folgende Ausgabe entsteht:

Terminal
```
schueler@debian964:~$ a.out
Zeichenkette eingeben: Das ist das Haus vom Nikolaus.
Zeichenkette: >>Das ist das Haus vom Nikolaus.<<
Zeichenkette: >>Qnf vfg qnf Unhf ibz Avxbynhf.<<
```

Aufgabe 3: Vertauschung von Zeichen einer Zeichenkette

Wie in der vorherigen Aufgabe soll wieder eine Zeichenkette vereinbart, von der Tastatur eingegeben, durch eine Funktion verändert und danach wieder ausgegeben werden. Die Funktionen sind hier so beschaffen, dass Zeichen vertauscht werden (Permutationen).

a) `ips_permut1.c`: Falls die Zeichenkette `s` mehr als ein Zeichen hat, sollen das Zeichen Nr. 0 und das Zeichen Nr. 1 vertauscht werden. Andernfalls soll `s` nicht verändert werden.

Terminal
```
schueler@debian964:~$ a.out
Zeichenkette eingeben: Das ist das Haus vom Nikolaus.
Zeichenkette: >>Das ist das Haus vom Nikolaus.<<
Zeichenkette: >>aDs ist das Haus vom Nikolaus.<<
```

   Hinweis: Wenn man zwei Variablen vertauscht, braucht man eine Hilfsvariable:

```
1 int a=1, b=2, hilf;
2 hilf = a; // a nach hilf zwischenspeichern
3 a = b;    // b nach a schreiben, a wird ueberschrieben!
4 b = hilf; // alten Wert von a aus hilf holen und nach b schreiben
```

b) `ips_permut2.c`: Falls die Zeichenkette `s` mehr als null Zeichen hat, sollen das Zeichen Nr. 0 und das Zeichen direkt vor dem Terminator vertauscht werden. Andernfalls soll `s` nicht verändert werden.

Terminal

```
schueler@debian964:~$ a.out
Zeichenkette eingeben: Das ist das Haus vom Nikolaus.
Zeichenkette: >>Das ist das Haus vom Nikolaus.<<
Zeichenkette: >>sas ist das Haus vom NikolauD.<<
```

c) `ips_permut3.c`: Falls die Zeichenkette `s` mehr als null Zeichen hat, soll sie umgekehrt werden.

Terminal

```
schueler@debian964:~$ a.out
Zeichenkette eingeben: Das ist das Haus vom Nikolaus.
Zeichenkette: >>Das ist das Haus vom Nikolaus.<<
Zeichenkette: >>.sualokiN mov suaH sad tsi saD<<
```

d) `ips_permut4.c` (Zusatzaufgabe): Falls die Zeichenkette mehr als ein Zeichen hat, soll sie nach dem Zufallsprinzip verwürfelt werden.

Terminal

```
schueler@debian964:~$ a.out
Zeichenkette eingeben: Das ist das Haus vom Nikolaus.
Zeichenkette: >>Das ist das Haus vom Nikolaus.<<
Zeichenkette: >>datus m.aDasvo k HlNuiio ssas <<
```

Hinweis: Für die Zufallsfunktion sollte man `srand()` und `rand()` aus `<stdlib.h>` verwenden. Das Verwürfeln kann z. B. so erfolgen, dass man nacheinander das letzte, vorletzte, drittletzte Zeichen festlegt. Und zwar so, dass man es mit einem zufällig ermittelten Zeichen tauscht (wobei man die Zeichen dahinter, die schon festgelegt wurden, nicht mehr einbezieht).

e) `ips_permut5.c` (Zusatzaufgabe): Falls die Zeichenkette mehr als ein Zeichen hat, soll sie nach ASCII-Reihenfolge (also quasi-alphabetisch) sortiert werden.

Terminal

```
schueler@debian964:~$ a.out
Zeichenkette eingeben: Das ist das Haus vom Nikolaus.
Zeichenkette: >>Das ist das Haus vom Nikolaus.<<
Zeichenkette: >>     .DHNaaaadiiklmoosssssstuuv<<
```

Hinweis: Sie können den (allerdings nicht sehr effizienten) Bubblesort-Algorithmus aus dem Arbeitsblatt C4.4 verwenden.

Aufgabe 4: Kürzen von Zeichenketten

Wie in der vorherigen Aufgabe soll wieder eine Zeichenkette vereinbart, von der Tastatur eingegeben, durch eine Funktion verändert und danach wieder ausgegeben werden. In dieser Aufgabe wird die Zeichenkette durch den Funktionsaufruf verkürzt. Es werden also eines oder mehrere Zeichen aus der Zeichenkette entfernt.

a) `ips_kuerz1.c`: Falls die Zeichenkette nicht leer ist, soll das letzte Zeichen der Zeichenkette, also das Zeichen direkt vor dem Terminator, entfernt werden. Das geschieht, indem man es mit dem Terminator überschreibt:

Terminal
```
schueler@debian964:~$ a.out
Zeichenkette eingeben: Das ist das Haus vom Nikolaus.
Zeichenkette: >>Das ist das Haus vom Nikolaus.<<
Zeichenkette: >>Das ist das Haus vom Nikolaus<<
```

b) `ips_kuerz2.c`: Durch den Aufruf `ips_kuerz2(s, 3)` sollen die letzten drei Zeichen vor dem Terminator entfernt werden. Es reicht dabei, das drittletzte Zeichen mit dem Terminator zu überschreiben. Aufpassen muss man bei Zeichenketten mit weniger als drei Zeichen; bei ihnen soll das vorderste Zeichen mit dem Terminator überschrieben werden. Für andere (nicht-negative!) Werte als 3 soll die Funktion entsprechend arbeiten.

Terminal
```
schueler@debian964:~$ a.out
Zeichenkette eingeben: Das ist das Haus vom Nikolaus.
Wie viele Zeichen sollen entfernt werden: 5
Zeichenkette: >>Das ist das Haus vom Nikolaus.<<
Zeichenkette: >>Das ist das Haus vom Niko<<
```

c) `ips_kuerz3.c`: Bei einer nicht-leeren Zeichenkette soll das vorderste Zeichen entfernt werden. Die anderen Zeichen sollen entsprechend ein Zeichen nach vorne rücken:

Terminal
```
schueler@debian964:~$ a.out
Zeichenkette eingeben: Das ist das Haus vom Nikolaus.
Zeichenkette: >>Das ist das Haus vom Nikolaus.<<
Zeichenkette: >>as ist das Haus vom Nikolaus.<<
```

d) `ips_kuerz4.c`: Durch den Aufruf `ips_kuerz2(s, 3)` sollen die vordersten drei Zeichen entfernt werden. Die anderen Zeichen sollen entsprechend drei Zeichen nach vorne rücken. Falls die Ursprungs-Zeichenkette weniger als drei Zeichen hatte, soll ein Leerstring entstehen. Für andere (nicht-negative) Werte als 3 soll die Funktion entsprechend arbeiten.

Terminal
```
schueler@debian964:~$ a.out
Zeichenkette eingeben: Das ist das Haus vom Nikolaus.
Wie viele Zeichen sollen entfernt werden: 12
Zeichenkette: >>Das ist das Haus vom Nikolaus.<<
Zeichenkette: >>Haus vom Nikolaus.<<
```

e) `ips_kuerz5.c`: Aus der Zeichenkette sollen alle Leerzeichen herausgenommen werden. Die anderen Zeichen rücken damit zusammen.

Terminal
```
schueler@debian964:~$ a.out
Zeichenkette eingeben: Das ist das Haus vom Nikolaus.
Zeichenkette: >>Das ist das Haus vom Nikolaus.<<
Zeichenkette: >>DasistdasHausvomNikolaus.<<
```

f) `ips_kuerz6.c`: Um Speicherplatz zu sparen, sollen aus der Zeichenkette alle Vokale herausgenommen werden. Die anderen Zeichen rücken damit zusammen.

Terminal
```
schueler@debian964:~$ a.out
Zeichenkette eingeben: Das ist das Haus vom Nikolaus.
Zeichenkette: >>Das ist das Haus vom Nikolaus.<<
Zeichenkette: >>Ds st ds Hs vm Nkls.<<
```

Aufgabe 5: Verlängern von Zeichenketten

Wie in den vorherigen Aufgaben soll wieder eine Zeichenkette vereinbart, von der Tastatur eingegeben, durch eine Funktion verändert und danach wieder ausgegeben werden. In dieser Aufgabe wird die Zeichenkette durch den Funktionsaufruf verlängert. Es werden also eines oder mehrere Zeichen in der Zeichenkette eingefügt. Dabei ist ein zusätzlicher Parameter nötig, der die Größe des Arrays angibt:

```
1  ips_laeng1(slcd, 71);
```

Mithilfe dieser Angabe muss die Funktion verhindern, über den Rand des Arrays hinaus zu schreiben.

a) `ips_laeng1.c`: Am Schluss soll an die Zeichenkette ein Leerzeichen angehängt werden, wenn die Zeichenkette dadurch nicht zu lang wird.

Terminal
```
schueler@debian964:~$ a.out
Zeichenkette eingeben: Das ist das Haus vom Nikolaus.
Zeichenkette: >>Das ist das Haus vom Nikolaus.<<
Zeichenkette: >>Das ist das Haus vom Nikolaus. <<
```

b) `ips_laeng2.c`: Mit dem Aufruf `ips_laeng2(s, 71, 3)` sollen an die Zeichenkette `s` drei Leerzeichen angehängt werden. Falls mehr als 71 Zeichen gebraucht werden, sollen nur so viele Zeichen angehängt werden wie möglich (unter Berücksichtigung des Platzes für den Terminator). Für andere (nicht-negative) Werte als 3 soll die Funktion entsprechend arbeiten.

Terminal
```
schueler@debian964:~$ a.out
Zeichenkette eingeben: Das ist das Haus vom Nikolaus.
Wie viele Leerzeichen sollen angehaengt werden: 5
Zeichenkette: >>Das ist das Haus vom Nikolaus.<<
Zeichenkette: >>Das ist das Haus vom Nikolaus.     <<
```

c) `ips_laeng3.c`: Am Beginn soll an die Zeichenkette ein Leerzeichen eingefügt werden, wenn die Zeichenkette dadurch nicht zu lang wird.

Terminal
```
schueler@debian964:~$ a.out
Zeichenkette eingeben: Das ist das Haus vom Nikolaus.
Zeichenkette: >>Das ist das Haus vom Nikolaus.<<
Zeichenkette: >> Das ist das Haus vom Nikolaus.<<
```

d) `ips_laeng4.c`: Mit dem Aufruf `ips_laeng4(s, 71, 3)` sollen am Beginn der Zeichenkette `s` drei Leerzeichen angehängt werden. Falls zu wenig Platz zur Verfügung steht, sollen so viele Zeichen eingefügt werden wie möglich (unter Berücksichtigung des Platzes für den Terminator). Für andere (nicht-negative) Werte als 3 soll die Funktion entsprechend arbeiten.

Terminal
```
schueler@debian964:~$ a.out
Zeichenkette eingeben: Das ist das Haus vom Nikolaus.
Wie viele Leerzeichen sollen zu Beginn eingefuegt werden: 5
Zeichenkette: >>Das ist das Haus vom Nikolaus.<<
Zeichenkette: >>     Das ist das Haus vom Nikolaus.<<
```

e) `ips_laeng5.c`: Mit dem Aufruf `ips_laeng5(s, 71, 5, 3)` sollen ab Position fünf der Zeichenkette (also nach fünf Zeichen) `s` drei Leerzeichen angehängt werden. Falls zu wenig Platz zur Verfügung steht, sollen so viele Zeichen eingefügt werden wie möglich (unter

Berücksichtigung des Platzes für den Terminator). Falls die Position nach dem Terminator liegt, soll die Funktion nichts tun.

Terminal
```
schueler@debian964:~$ a.out
Zeichenkette eingeben: Das ist das Haus vom Nikolaus.
Wie viele Leerzeichen sollen eingefuegt werden: 3
Ab welcher Position sollen sie eingefuegt werden: 5
Zeichenkette: >>Das ist das Haus vom Nikolaus.<<
Zeichenkette: >>Das i   st das Haus vom Nikolaus.<<
```

f) `ips_laeng6.c`: Die Funktion aus der vorherigen Teilaufgabe soll erweitert werden: Mit dem Aufruf `ips_laeng5(s, 71, -4, 3)` sollen vor dem viertletzten Zeichen der Zeichenkette `s` drei Leerzeichen angehängt werden. Falls zu wenig Platz zur Verfügung steht, sollen so viele Zeichen eingefügt werden wie möglich (unter Berücksichtigung des Platzes für den Terminator). Falls die errechnete Position vor dem Anfang der Zeichenkette liegt, soll die Funktion nichts tun.

Terminal
```
schueler@debian964:~$ a.out
Zeichenkette eingeben: Das ist das Haus vom Nikolaus.
Wie viele Leerzeichen sollen eingefuegt werden: 3
Ab welcher Position sollen sie eingefuegt werden: -4
Zeichenkette: >>Das ist das Haus vom Nikolaus.<<
Zeichenkette: >>Das ist das Haus vom Nikol   aus.<<
```

g) `ips_laeng7.c`: Nach jedem Leerzeichen in der Zeichenkette soll ein weiteres Leerzeichen eingefügt werden.

Terminal
```
schueler@debian964:~$ a.out
Zeichenkette eingeben: Das ist das Haus vom Nikolaus.
Zeichenkette: >>Das ist das Haus vom Nikolaus.<<
Zeichenkette: >>Das  ist  das  Haus  vom  Nikolaus.<<
```

h) `ips_laeng8.c` (Zusatzaufgabe): Zwischen den einzelnen Worten sollen so viele Leerzeichen eingefügt werden, dass die Speichergröße des Strings (hier: 70 Zeichen plus Terminator) voll ausgenutzt wird. Die Zwischenräume zwischen den Worten sollen dabei möglichst gleich sein.

Aufgabe 6: Beliebige Zeichenersetzung

Die nun zu erstellende Funktion `char_tr()` soll mehrere Zeichen einer Zeichenkette ersetzen können. Neben der zu bearbeitenden Zeichenkette soll der Funktion eine Zeichenkette übergeben werden, die die zu ersetzenden Zeichen enthält und eine Zeichenkette, die die Ersatzzeichen enthält (siehe Beispiel unten).

Diese Funktion soll mit einem kurzen Programm `char_tr.c` getestet werden, das folgenden Dialog ausgibt:

Terminal
```
Zeichenkette eingeben: Fischers Fritze fischt frische Fische.
Zu ersetzende Zeichen eingeben: abcdef
Ersatzzeichen eingeben........: ABCDEF
Zeichenkette: >>Fischers Fritze fischt frische Fische.<<
Zeichenkette: >>FisChErs FritzE FisCht FrisChE FisChE.<<
```