Freiwillige Offline-Aufgabe 08-01 (INF & WI): C-String-Funktion `my_strlen()`

(geübte C++ Konstrukte: C-Strings, Funktionen mit Pointern als Parametern)

Programmieren Sie eine Funktion …

```
unsigned int my_strlen (const char * ptr)
```

… welche die Anzahl der Zeichen im C-String zurückgibt. Das Null-Terminierungszeichen soll dabei nicht mitgezählt werden.
Mittels des `const` bei der Parameterdeklaration „verspricht" die Funktion, den Wert des Parameters innerhalb des Funktionsrumpfes nicht zu ändern.

Beispiel:

```
cout << my_strlen("Hallo") << endl;
```

… gibt `5` auf dem Bildschirm aus.

Die Benutzung jeglicher Funktionen aus der `cstring` Library ist nicht gestattet. Benutzen Sie nur den Zugriff auf die einzelnen Zeichen des C-Strings über den Array-Indexoperator `[ ]`.

Schreiben Sie ferner ein Hauptprogramm, welches eine einzeilige Zeichenkette (ggfs. inklusive Leerzeichen) einliest und die ermittelte Anzahl der Zeichen ausgibt (siehe Testläufe). Die eingegebene Zeichenkette sei maximal `20` für den Benutzer sichtbare Zeichen lang.

Testläufe (Benutzereingaben sind unterstrichen):

```
Bitte Text eingeben (ggfs. mit Leerzeichen): ? Hallo
Ergebnis my_strlen(): 5
Drücken Sie eine beliebige Taste . . .
```

```
Bitte Text eingeben (ggfs. mit Leerzeichen): ? aa aa aa
Ergebnis my_strlen(): 8
Drücken Sie eine beliebige Taste . . .
```

```
Bitte Text eingeben (ggfs. mit Leerzeichen): ?  (leere Eingabe)
Ergebnis my_strlen(): 0
Drücken Sie eine beliebige Taste . . .
```

Prof. Dr. Andreas Claßen, Prof. Dr. Thomas Dey

Freiwillige Offline-Aufgabe 08-02 (INF & WI): C-String-Funktion `my_strcmp()`

(geübte C++ Konstrukte: C-Strings, Funktionen mit Pointern als Parametern)

Programmieren Sie eine Funktion …

```
int my_strcmp (const char * ptr1, const char * ptr2)
```

… welche zwei C-Strings vergleicht.

Sind die beiden C-String gleich, so gibt die Funktion den Rückgabewert `0` zurück. Dies gilt insbesondere auch dann, wenn beide C-Strings leer sind. Ansonsten werden die Zeichen an der ersten Position verglichen, an der die beiden C-Strings unterschiedlich sind: Sollte das Zeichen im ersten C-String kleiner sein, so wird der Wert `-1` zurückgegeben. Sollte das Zeichen im ersten C-String größer sein, so wird der Wert `1` zurückgegeben. Dabei können Sie die Zeichen in ihrem Programm mittels `<` bzw. `>` vergleichen.
(Dabei verwendet das System intern wieder die ASCII Tabelle, wodurch die Großbuchstaben kleiner sind als die Kleinbuchstaben!)

Mittels des `const` bei der Parameterdeklaration „verspricht" die Funktion, die Werte der Parameter innerhalb des Funktionsrumpfes nicht zu ändern.

Die Benutzung jeglicher Funktionen aus der `cstring` Library ist nicht gestattet. Benutzen Sie nur den Zugriff auf die einzelnen Zeichen des C-Strings über den Array-Indexoperator `[ ]`. Sollten Sie die Länge eines C-Strings in ihrem Programm benötigen, so müssen Sie auch dies als Hilfsfunktion selbst programmieren.

Schreiben Sie ferner ein Hauptprogramm, welches zwei einzeilige Zeichenketten (ggfs. leer, ggfs. inklusive Leerzeichen) einliest und das Ergebnis des Vergleichs ausgibt (siehe Testläufe). Die eingegebenen Zeichenketten seien jeweils maximal `20` für den Benutzer sichtbare Zeichen lang.

Testläufe (Benutzereingaben sind unterstrichen):

```
Bitte ersten Text eingeben (ggfs. mit Leerzeichen): ? Hallo
Bitte zweiten Text eingeben (ggfs. mit Leerzeichen): ? Hallo
Die Texte sind identisch. Ergebnis my_strcmp(): 0
Drücken Sie eine beliebige Taste . . .
```

```
Bitte ersten Text eingeben (ggfs. mit Leerzeichen): ? (leere Eingabe)
Bitte zweiten Text eingeben (ggfs. mit Leerzeichen): ? (leere Eingabe)
Die Texte sind identisch. Ergebnis my_strcmp(): 0
Drücken Sie eine beliebige Taste . . .
```

```
Bitte ersten Text eingeben (ggfs. mit Leerzeichen): ? Hallo
Bitte zweiten Text eingeben (ggfs. mit Leerzeichen): ? (leere Eingabe)
Ergebnis my_strcmp(): 1
Drücken Sie eine beliebige Taste . . .
```

```
Bitte ersten Text eingeben (ggfs. mit Leerzeichen): ? (leere Eingabe)
Bitte zweiten Text eingeben (ggfs. mit Leerzeichen): ? Hallo
Ergebnis my_strcmp(): -1
Drücken Sie eine beliebige Taste . . .
```

```
Bitte ersten Text eingeben (ggfs. mit Leerzeichen): ? abcde
Bitte zweiten Text eingeben (ggfs. mit Leerzeichen): ? xyz
Ergebnis my_strcmp(): -1
Drücken Sie eine beliebige Taste . . .
```

```
Bitte ersten Text eingeben (ggfs. mit Leerzeichen): ? ABC
Bitte zweiten Text eingeben (ggfs. mit Leerzeichen): ? abc
Ergebnis my_strcmp(): -1
Drücken Sie eine beliebige Taste . . .
```

```
Bitte ersten Text eingeben (ggfs. mit Leerzeichen): ? ab cd
Bitte zweiten Text eingeben (ggfs. mit Leerzeichen): ? abxcd
Ergebnis my_strcmp(): -1
Drücken Sie eine beliebige Taste . . .
```

Pflicht-Offline-Aufgabe 08-03 (INF & WI): C-String-Funktion `my_strconcat()`

(geübte C++ Konstrukte: C-Strings, `new []`, Funktionen mit Pointern als Parametern)

Programmieren Sie eine Funktion …

```
char* my_strconcat(const char * ptr1,
                   const char * ptr2)
```

… welche einen neuen C-String durch das Aneinanderhängen der zwei Parameter-C-Strings erzeugt.

Für den Ergebnis-C-String soll neuer Speicher auf dem Heap alloziert werden. Dann werden die Zeichen des ersten sowie des zweiten C-Strings in den Ergebnis-C-String kopiert. Der resultierende C-String soll dann einen Null-terminator am Ende der aneinander-gehängten Zeichen haben.

Mittels des `const` bei der Parameterdeklaration „verspricht" die Funktion, die Werte der Parameter innerhalb des Funktionsrumpfes nicht zu ändern.

Die Benutzung jeglicher Funktionen aus der `cstring` Library ist nicht gestattet. Benutzen Sie nur den Zugriff auf die einzelnen Zeichen des C-Strings über den Array-Indexoperator `[ ]`. Sollten Sie die Länge eines C-Strings in ihrem Programm benötigen, so müssen Sie auch dies als Hilfsfunktion selbst programmieren.

Schreiben Sie ferner ein Hauptprogramm, welches zwei einzeilige Zeichenketten (ggfs. leer, ggfs. inklusive Leerzeichen) einliest und das Ergebnis des Aneinanderhängens ausgibt (siehe Testläufe). Die eingegebenen Zeichenketten seien jeweils maximal `20` für den Benutzer sichtbare Zeichen lang. Die Ergebnis-Zeichenkette hat somit eine maximale Länge von `40` für den Benutzer sichtbaren Zeichen. Der Benutzer mache nur korrekte Eingaben und halte sich auch an die vorgegebene maximale Eingabe-Länge.

Testläufe (Benutzereingaben sind unterstrichen):

```
Bitte ersten Text eingeben (ggfs. mit Leerzeichen): ? Hallo
Bitte zweiten Text eingeben (ggfs. mit Leerzeichen): ? Welt
Ergebnis my_strconcat(): HalloWelt
Drücken Sie eine beliebige Taste . . .
```

```
Bitte ersten Text eingeben (ggfs. mit Leerzeichen): ? (leere Eingabe)
Bitte zweiten Text eingeben (ggfs. mit Leerzeichen): ? (leere Eingabe)
Ergebnis my_strconcat():
Drücken Sie eine beliebige Taste . . .
```

```
Bitte ersten Text eingeben (ggfs. mit Leerzeichen): ? Hallo
Bitte zweiten Text eingeben (ggfs. mit Leerzeichen): ? (leere Eingabe)
Ergebnis my_strconcat(): Hallo
Drücken Sie eine beliebige Taste . . .
```

```
Bitte ersten Text eingeben (ggfs. mit Leerzeichen): ? (leere Eingabe)
Bitte zweiten Text eingeben (ggfs. mit Leerzeichen): ? Hallo
Ergebnis my_strconcat(): Hallo
Drücken Sie eine beliebige Taste . . .
```

```
Bitte ersten Text eingeben (ggfs. mit Leerzeichen): ? ab cd
Bitte zweiten Text eingeben (ggfs. mit Leerzeichen): ? 12 34
Ergebnis my_strconcat(): ab cd12 34
Drücken Sie eine beliebige Taste . . .
```

Freiwillige Offline-Aufgabe 08-04 (INF & WI): C-String-Funktion `my_strconcat2()`: Aneinanderhängen mit Längenbegrenzung

(geübte C++ Konstrukte: C-Strings, `new []`, Funktionen mit Pointern als Parametern)

Programmieren Sie eine Funktion …

```
char * my_strconcat2(const char * sptr1,
                     const char * sptr2,
                     unsigned int count);
```

… welche einen neuen C-String erzeugt, indem ein C-String mit maximal `count` Zeichen durch Anhängen des C-Strings `sptr2` an den C-String `sptr1` erzeugt wird.
Der resultierende C-String muss in jedem Fall nullterminiert sein und liegt komplett in einem (von der Funktion) neu allozierten Speicherbereich auf dem Heap. D.h. weder der von `sptr1` noch der von `sptr2` belegte Speicher sind Teil des vom Resultatstring belegten Speichers.

Die Benutzung jeglicher Funktionen aus der `cstring` Library ist nicht gestattet. Benutzen Sie nur den Zugriff auf die einzelnen Zeichen des C-Strings über den Array-Indexoperator `[ ]`. Sollten Sie die Länge eines C-Strings in ihrem Programm benötigen, so müssen Sie auch dies als Hilfsfunktion selbst programmieren.

Schreiben Sie ferner ein Hauptprogramm, welches zwei einzeilige Zeichenketten (ggfs. leer, ggfs. inklusive Leerzeichen) einliest und das Ergebnis des Aneinanderhängens ausgibt (siehe Testläufe). Die eingegebenen Zeichenketten seien jeweils maximal `20` für den Benutzer sichtbare Zeichen lang. Die Ergebnis-Zeichenkette hat somit eine maximale Länge von `40` für den Benutzer sichtbaren Zeichen. Der Benutzer mache nur korrekte Eingaben und halte sich auch an die vorgegebene maximale Eingabe-Länge.

Testläufe (Benutzereingaben sind unterstrichen):

```
Bitte ersten Text eingeben (ggfs. mit Leerzeichen): ? Hallo
Bitte zweiten Text eingeben (ggfs. mit Leerzeichen): ? Welt
Anzahl Zeichen: ? 7
Ergebnis my_strconcat2(): HalloWe
Drücken Sie eine beliebige Taste . . .
```

```
Bitte ersten Text eingeben (ggfs. mit Leerzeichen): ? Hallo
Bitte zweiten Text eingeben (ggfs. mit Leerzeichen): ? Welt
Anzahl Zeichen: ? 3
Ergebnis my_strconcat2(): Hal
Drücken Sie eine beliebige Taste . . .
```

```
Bitte ersten Text eingeben (ggfs. mit Leerzeichen): ? (leere Eingabe)
Bitte zweiten Text eingeben (ggfs. mit Leerzeichen): ? (leere Eingabe)
Anzahl Zeichen: ? 99
Ergebnis my_strconcat2():
Drücken Sie eine beliebige Taste . . .
```

```
Bitte ersten Text eingeben (ggfs. mit Leerzeichen): ? Hallo
Bitte zweiten Text eingeben (ggfs. mit Leerzeichen): ? (leere Eingabe)
Anzahl Zeichen: ? 99
Ergebnis my_strconcat2(): Hallo
Drücken Sie eine beliebige Taste . . .
```

```
Bitte ersten Text eingeben (ggfs. mit Leerzeichen): ? (leere Eingabe)
Bitte zweiten Text eingeben (ggfs. mit Leerzeichen): ? Hallo
Anzahl Zeichen: ? 99
Ergebnis my_strconcat2(): Hallo
Drücken Sie eine beliebige Taste . . .
```

```
Bitte ersten Text eingeben (ggfs. mit Leerzeichen): ? ab cd
Bitte zweiten Text eingeben (ggfs. mit Leerzeichen): ? 12 34
Anzahl Zeichen: ? 9
Ergebnis my_strconcat2(): ab cd12 3
Drücken Sie eine beliebige Taste . . .
```

Freiwillige Offline-Aufgabe 08-05 (INF & WI): Wortliste eines Texts, mit Häufigkeiten, mittels C-Strings

(geübte C++ Konstrukte: C-Strings, Pointer, Arrays, Schleifen, struct)

Schreiben Sie ein C++ Programm, welches einen Text in Form von mehreren Eingabezeilen einliest. Das Programm soll die in dem Text auftretenden Worte in der Reihenfolge ihres Auftretens mitsamt der Häufigkeit ihres Vorkommens im Text ausgeben.

Einzelne Eingabezeilen und Worte sollen als C-Strings gespeichert werden. Die `cstring` Bibliothek mit den darin enthaltenen Funktionen darf im Rahmen dieser Aufgabe aber ***nicht*** verwendet werden, sondern entsprechende Funktionen müssen selbst programmiert werden. (Hinweis: C-Strings als `char*` sind auch ohne die `cstring` Bibliothek verwendbar).

Die „Sammlung aller nicht-leeren Eingabezeilen" sowie die „Sammlung der Worthäufigkeiten" sollen als statische Arrays gespeichert werden. Es kommen maximal `const unsigned int eingabezeilen_max = 5;` Eingabezeilen und maximal `const unsigned int wort_max = 1000;` Worte vor. Jede Eingabezeile enthält maximal `const unsigned int max_line_length = 80;` Zeichen (inklusive des Nullterminators).

Eine leere Eingabezeile beende die Eingabe. Eingabezeilen können auch Leerzeichen enthalten. Leerzeichen, Punkt, Komma sowie Anfang und Ende jeder Eingabezeile fungieren als Wortbegrenzungen. Im Eingabetext sollen auch keine anderen Satzzeichen als Punkt und Komma (und Leerzeichen) vorkommen, ihr Programm kann davon ausgehen, dass der Benutzer nur korrekte Eingaben macht.

Die Worthäufigkeit jedes Worts soll jeweils in einer Datenstruktur …

```
struct w_haeufigkeit {
    char* wort;
    unsigned int haeufigkeit;
};
```

… gespeichert werden (und davon ein Array mit `wort_max` Einträgen).

Es sollen für das Ermitteln der Worthäufigkeiten folgende Funktionen definiert werden (diese Funktionen werden von Jenkins ggfs. auch einzeln getestet, sie müssen also vorhanden sein und wie vorgeschrieben definiert werden):

```
unsigned int my_strlen(const char* const str);
```

… berechnet die Länge eines C-Strings, indem sie die Zeichen bis zum ersten Nullterminator zählt (d.h. ein C-String `"abc"` hat die Länge `3`),

```
int my_strcmp(const char* str1, const char* str2);
```

… vergleicht zwei C-Strings und gibt den Wert `0` zurück, wenn diese zeichenweise identisch sind, ansonsten einen Wert ungleich `0`,

```
char* naechstes_wort(const char* const zeile,
                     unsigned int& pos);
```

… suche ausgehend von der Position `pos` in der Zeile das nächste Wort und gebe es kopiert in einen eigenen Speicherbereich auf dem Heap (für dieses Wort) zurück. `pos` wird dann auf die erste Position (in der Zeile) hinter dem Wort geändert. Sollte es ausgehend von `pos` kein nächstes Wort in der Zeile geben, so werde der `nullptr` als Wert zurückgegeben.

```
void zaehle_wort(char* wort,
                 struct w_haeufigkeit w_haeufigkeiten[],
                 unsigned int& w_count);
```

… suche das Wort `wort` in der Wortsammlung `w_haeufigkeiten[]`, die aktuell bis zur Position `w_count-1` belegt sei. Kommt das Wort schon vor, so werde seine Häufigkeit um `1` erhöht. Kommt das Wort bisher noch nicht vor, so werde es hinzugefügt (der Zeiger auf den Heap-Speicher des Wortes werde in die Datenstruktur umgehängt, d.h. die Datenstruktur wird zum Besitzer des Wortes) und seine Häufigkeit auf `1` gesetzt.

Die Arrays für den Eingabetext und die gesammelten Worthäufigkeiten seien lokale Variablen des Hauptprogramms.

Testläufe (Benutzereingaben sind unterstrichen):

```
Eingabezeile = ? Dies ist die erste Zeile des Texts.
Eingabezeile = ? Und dies, ist, noch          eine Zeile.
Eingabezeile = ? .Und noch eine weitere Zeile, im Text.
Eingabezeile = ?
Dies: 1
ist: 2
die: 1
```

```
erste: 1
Zeile: 3
des: 1
Texts: 1
Und: 2
dies: 1
noch: 2
eine: 2
weitere: 1
im: 1
Text: 1
Drücken Sie eine beliebige Taste . . .
```