Aufgabe 1.1 Schleifen und Zeichen

Schreibe ein Programm, das alle Groß- und Kleinbuchstaben abwechselnd auf die Konsole schreibt:

```
AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpQqRrSsTtUuVvWwXxYyZz
```

Selbstverständlich solltest Du nicht einfach `printf("AaBbCc...")` aufrufen, sondern eine Schleife verwenden. Studiere dazu die relative Lage der Buchstaben im ASCII-Code.

Aufgabe 1.2 Zeichenweise Ein- und Ausgabe

Schreibe ein Programm, das jedes eingelesene Zeichen wieder auf die Konsole schreibt; dabei sollen alle Kleinbuchstaben in Großbuchstaben umgewandelt werden:

```
Eingabe: Infinite recursion may cause a stack overflow.
Ausgabe: INFINITE RECURSION MAY CAUSE A STACK OVERFLOW.
```

Anmerkung: Es darf in jedem Projekt nur eine `main`-Funktion existieren. Wenn Du versuchst, mehrere `main`-Funktionen in ein und demselben Projekt zu schreiben, wird der Compiler sich beschweren. Um dieses Problem zu lösen, gibt es mindestens drei Ansätze:

- Erstelle für jede Aufgabe ein eigenes Visual-Studio-Projekt.
- Kommentiere die alte `main`-Funktion aus oder benenne sie um.
- Löse die Aufgaben nicht innerhalb der `main`-Funktion, sondern in eigenen Funktionen, und rufe diese nacheinander aus der `main`-Funktion heraus auf:

```
int main(void)
{
    aufgabe_1_1();
    aufgabe_1_2();
    // ...
    return 0;
}
```

Aufgabe 1.3 Formatierte Ausgabe

Schreibe ein Programm, das die Quadratzahlen von 1*1 bis 10*10 auf die Konsole schreibt:

```
 1 *  1 =   1
 2 *  2 =   4

...

 9 *  9 =  81
10 * 10 = 100
```

Bekommst Du es hin, dass die Zahlen so schön rechtsbündig angeordnet sind?

Schreibe als nächstes ein Programm, das für alle Pärchen (i, k) aus dem Wertebereich von 1 bis 10 das entsprechende Produkt i*k auf die Konsole schreibt:

```
 1 *  1 =   1
 1 *  2 =   2
 1 *  3 =   3

...
```

```
 2 *  1 =   2
 2 *  2 =   4
 2 *  3 =   6

...

10 *  1 =  10
10 *  2 =  20
10 *  3 =  30

...

10 *  9 =  90
10 * 10 = 100
```

Besonders hübsch bzw. übersichtlich ist das leider nicht. Schreibe zuletzt ein Programm, das diese 100 Produkte in Form einer Multiplikationstabelle auf die Konsole schreibt:

```
    |   1   2   3   4   5   6   7   8   9  10
----+----------------------------------------
  1 |   1   2   3   4   5   6   7   8   9  10
  2 |   2   4   6   8  10  12  14  16  18  20
  3 |   3   6   9  12  15  18  21  24  27  30
  4 |   4   8  12  16  20  24  28  32  36  40
  5 |   5  10  15  20  25  30  35  40  45  50
  6 |   6  12  18  24  30  36  42  48  54  60
  7 |   7  14  21  28  35  42  49  56  63  70
  8 |   8  16  24  32  40  48  56  64  72  80
  9 |   9  18  27  36  45  54  63  72  81  90
 10 |  10  20  30  40  50  60  70  80  90 100
```

Aufgabe 1.4 Arrays auf die Konsole schreiben

Schreibe eine Funktion `void print_numbers(int numbers[], int len)`, die die übergebenen Zahlen in geschweiften Klammern und durch Kommata getrennt ausgibt:

```
int test[] = {2, 3, 5, 7, 11, 13, 17, 19};
print_numbers(test, 8);
```

Ausgabe:

```
{2, 3, 5, 7, 11, 13, 17, 19}
```

Die Kommata sollen nur *zwischen* den Elementen auftauchen; vor dem ersten und nach dem letzten Element steht keins! Leere Arrays sind als Klammerpaar `{}` auszugeben.

Aufgabe 1.5 Zeichenketten

Schreibe und teste folgende Funktionen:

1. `int length(char s[])` liefert die Anzahl Zeichen vor der terminierenden 0.
2. `void print_in_reverse(char s[])` schreibt die Zeichen in umgekehrter Reihenfolge auf die Konsole, verändert das Array selbst aber nicht.
3. `void reverse(char s[])` dreht die Zeichenkette um, ändert also das Array.

Aufgabe 1.6 Zahlen einlesen

Schreibe eine Funktion `int read_number(void)`, die mittels Schleife und `getchar()` einen Strom von Zeichen in eine Zahl umwandelt und zurückliefert. Die Eingabe gilt als beendet, sobald ein Zeichen auftaucht, das keine Ziffer ist. Es kann davon ausgegangen werden, dass die Ziffern `'0'` bis `'9'` im Zeichensatz direkt aufeinander folgen.

Aus den Zeichen `'1'`, `'2'`, `'3'` und `'\n'` soll z.B. die Zahl 123 generiert werden.

Diese Aufgabe beinhaltet zwei interessante Probleme:

1. Wenn der Anwender eine Ziffer eingibt, dann liefert `getchar` einen Wert zwischen 48 und 57 (warum?), wir brauchen aber einen Wert zwischen 0 und 9.
2. Der Datentyp `int` bietet keine Operation zum Anhängen einer Ziffer an eine Zahl. Wenn man beispielsweise an die Zahl 123 die Ziffer 4 anhängen möchte, um daraus die Zahl 1234 zu machen, dann muss man dafür mehrere Rechenschritte ausführen.

Um die Funktion `read_number` auszuprobieren, schreibe ein Programm, das in einer Endlosschleife immer zwei Zahlen einliest und deren Summe ausgibt:

```
x? 11
y? 13

11 + 13 = 24

x? 17
y? 19

17 + 19 = 36 ...
```