2 Datentypen - Übungen

Literale Nummern

Warum geben beide Ausgaben unterschiedliche Werte aus?

```
long long bigNumber

bigNumber = (0x10000000 * 10);
printf("%lld\n", bigNumber);        // lld = long long dezimal

bigNumber = (0x10000000LL * 10);
printf("%lld\n", bigNumber);
```

L: Das numerische Literal 0x10000000 ist klein genug um in einem Integer gespeichert zu werden. Es wird daher vom Kompiler als einfaches Integer interpretiert. Anschließend mit dem Integer 10 multipliziert. Das Integer-Ergebnis dieser Operation läuft über und wird daher kleiner. Anschließend wird diese (falsche) Integerzahl in *bigNumber* zugewießen.

Mit der zweiten Folge wird bewusst die "kleine" Zahl 0x10000000 als LL, also Long-Long definiert. Mit der Multiplikation wird das Ergebnis wiederum als Long-Long interpretiert und somit genügend Platz für die Speicherung vorgehalten. Dann wird diese richtige Long-Long-Zahl in *bigNumber* zugewiesen.

Literale Character

- Was ist der Unterschied zwischen den folgenden vier Zuweisungen?

```
char c;
c = 65;
c = 'A';
 = 0x41;        // (0x41 == 65)
c = '\x41';
```

L: Keiner, es sind alle gleichwertig, die Form 'A' ist für diesen Fall sicher die am besten lesbare.

- Was gibt folgender Code mit den Escape-Sequenzen aus und warum?

```
printf("\"ich hab einen ein \bfall \\ und das ist ein\x20fall\a\"");
```

TypeSize (003)

- In C haben Datentypen nicht immer gleichen Speicherbedarf. Bestimmen Sie für die folgenden Datentypen den Speicherbedarf.
- Geben Sie den Typ, Speicherbedarf und Von-Bis-Bereiche tabellarisch in der Konsole aus:

```
Ganzzahlig:

      Type Bytes                  Min                  Max
 ---------------------------------------------------------------
      Char  1                 -128                  127
     UChar  1                    0                  255
     Short  2                -32768                32767
    UShort  2                    0                65535
       Int  4           -2147483648           2147483647
      UInt  4                    0           4294967295
      Long  4           -2147483648           2147483647
     ULong  4                    0           4294967295
  LongLong  8     -9223372036854775808      9223372036854775807
 ULongLong  8                    0     18446744073709551615


Flieszkomma:
```

```
      Type Bytes                  Min                      Max
 ------------------------------------------------------------------
     Float  4           1.175494E-038           3.402823E+038
    Double  8           2.225074E-308           1.797693E+308
```

Hilfreiche Formatspezifizierer: d, u, lld, llu, e

- Aus *limits.h* können die Maximal- und Minimal-Werte der ganzzahligen Typen ermittelt werden. Aus *float.h* können die Maximal- und Minimal-Werte der Fließkommazahlen ermittelt werden.
- Für vorzeichenlose Typen (unsigned) kann der Maximalwert auch so ermittelt werden:

```
unsigned int num = 0;
num--;      // wird um 1 verkleinert, wird der groeste Wert
```

Wochentag-Enumerator

- Was ist der Nachteil des folgenden Code-Fragments?

```
enum {MO, DI, MI, DO, FR, SA, SO} wochentag;
wochentag = MO;

if (wochentag == SO) wochentag = MO;
else wochentag++;
```

L: Grundsätzlich funktioniert dieses Code-Fragment, wenn im enum die Werte oder die Reihenfolge für die Elemente verändert wird, dann stimmt das enum-Inkrement (++) nicht mehr (In C++ kann dafür der ++ -Operator überladen werden).

Typedef

- Erstellen Sie den Typen TINT damit folgendes (schlechtes Beispiel funktioniert):

```
// Defintion TINT

TINT a = 255;
a++;        // a -> 0
```

In diesem Beispiel ist der Name des Typs (TINT) sehr schlecht gewählt. Welcher Name würde besser zutreffen?