Name: _____

Weitere Übung zu Fließkommazahlen

Theorieaufgabe:

Gegeben sei eine `float`-Variable **x** mit dem Wert +∞. Es sei das Bitmuster von **x** darzustellen:

☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

① Die Anweisung **`bits_to_float(str_to_ieee32(myString))`** soll einen gegebenen Dezimalbruch (mit **.** als Kommazeichen) parsen und in eine IEEE-Fließkommazahl (`float`) umwandeln. Der Code der Funktion **`str_to_ieee32`** sei dazu zu vervollständigen:

a) In einer Schleife sollen aufeinanderfolgende Dezimalziffern mit Hilfe von `P2DIGIT(ptr)` eingelesen und zu einem Integer-Wert **`int_part`** aufgebaut werden. Sobald ein Zeichen gefunden wird, bei dem es sich um keine Dezimalziffer handelt, sei die Schleife zu beenden. Kann mindestens eine Dezimalziffer gelesen werden, sei **`has_digits`** auf 1 zu setzen. Der Zeiger **`ptr`** sei in jedem Schleifendurchlauf zu inkrementieren.

b) Falls **`'.'`** als nachfolgendes Zeichen gelesen wird, sei der Nachkommateil analog zu a) als Integer-Wert **`frac_part`** einzulesen. Der Inhalt der Variablen **`frac_divisor`** (durch den **`frac_part`** später in der Codevorlage dividiert wird um den tatsächlichen Wert des Nachkommateils zu erhalten) sei bei jedem Schleifendurchlauf entsprechend anzupassen.

```
#define P2DIGIT(strptr) (*strptr - '0')

static uint32_t str_to_ieee32(char* str) {
    uint32_t res = 0;
    char* ptr = str;
    bool has_digits = false, negative = false;
    long long int frac_part = 0, frac_divisor = 1, int_part = 0;

    if (!str || !*str) return 0x7FC00000; // NaN

    while (*ptr == ' ') { ptr++; }

    if (strstr(ptr,  "NaN") == ptr || strstr(ptr,  "nan") == ptr) { return 0x7FC00000; }
    if (strstr(ptr,  "Inf") == ptr || strstr(ptr,  "inf") == ptr) { return 0x7F800000; }
    if (strstr(ptr, "-Inf") == ptr || strstr(ptr, "-inf") == ptr) { return 0xFF800000; }

    if (*ptr == '-') { negative = true; ptr++; } else if (*ptr == '+') { ptr++; }

    [                                                            ]

    [                                                            ]

    [                                                            ]

    if (!has_digits) { return 0x7FC00000; } // Invalid input

    uint32_t u_int = ll_to_ieee32(int_part);
    uint32_t u_frac = ll_to_ieee32(frac_part);
    uint32_t u_div  = ll_to_ieee32(frac_divisor);
    uint32_t u_inv  = ieee32_inv_u(u_div);
    uint32_t u_frac_scaled = ieee32_mul_u(u_frac, u_inv);

    res = ieee32_add_u(u_int, u_frac_scaled);

    if (negative) { res = res | SIGN_MASK; }
    else res &= ~SIGN_MASK;

    return res;
}
```

② Die Funktion `snprintf` aus `stdio.h` formatiert Text nach einem Formatstring `*format` und schreibt das Ergebnis in einen Puffer `*buffer` mit einer festen Maximalgröße `n`:

```
int snprintf(char* buffer, size_t n, const char* format, ...)
```

‚`...`‘ enthält variadische Argumente passend zu den Platzhaltern im Formatstring.
Die Funktion liefert als Rückgabewert die Anzahl der (theoretisch) geschriebenen Zeichen.
Der Rückgabewert ist hierbei ein 'Längenversprechen' und kein 'Schreibbericht'.

Die Funktion `ieee32_to_str` ist das Gegenstück zu `str_to_ieee32`. Ein Funktionsaufruf der Form `ieee32_to_str(buf, sizeof(buf), float_to_bits(x), 3, 5)` wandelt eine Fließkommazahl `x` in einen String (Zahl als Dezimalbruch mit `.` als Komma) um. Die Argumente `3` und `5` definieren das '**pre**'- bzw. '**post**'-**Padding** der Ausgabe.

```
ieee32_to_str(char* str, size_t n, INT32 val, INT16 pre, INT16 post)
```

Der nachfolgende (gekürzte) Code-Ausschnitt sei derart zu vervollständigen, dass auch der als natürliche Zahl dargestellte Nachkommateil `f_part` mit entsprechendem 'post'-Padding in den Ausgabe-Puffer an die Stelle `*s` geschrieben wird. Wenn `post` > `0` gilt und noch mindestens ein Zeichen geschrieben werden kann (`remain` > `1`), so sei erst einmal an die Stelle von `*s` das Symbol `'.'` zu schreiben, der Zeiger `s` sei zu inkrementieren und der Wert von `remain` zu dekrementieren. Mittels `snprintf` sei dann der Nachkommateil zu schreiben. Man orientiere sich am 'pre'-Padding weiter oben. Der Nachkommateil sei nach spätestens `post` Stellen abzuschneiden. Falls `remain` < `post` gelten sollte, sind die 'fehlenden' Stellen mittels `"%0*lld "` (Formatstring!) aufzufüllen.

Die oben definierten Makros müssen nicht genutzt werden.

```
#include <stdio.h>
#define INT32 uint32_t
#define INT16 uint16_t
#define PRE(pre) (pre > 0 ? pre : 1)

size_t ieee32_to_str(char* str, size_t n, INT32 val, INT16 pre, INT16 post) {

    // [...] Behandlung aller möglicher Sonderfälle

    size_t remain = n;
    char* s = str;

    // Schreiben des Vorzeichens
  if (val & SIGN_MASK) { if (remain > 1) { *s++ = '-'; remain--; }; val &= ~SIGN_MASK; }

    // [...] Zerlegung in Vorkommateil (i_part) und Nachkommateil
    // [...] Skalierung des Nachkommateils auf eine natürliche Zahl (f_part)

    // Schreiben des Vorkommateils
    int written = snprintf(s, remain, "%0*lld", PRE(pre), i_part); // 'pre'-
    Padding if (written > 0 && (size_t)(written) < remain) {
        s += written;
        remain -= written;
    } else {
        return n; // Puffer ist voll!
    }

    // Schreiben des Nachkommateils + 'post'-Padding

    return strlen(str);
}
```