Institut für Informatik
Albert-Ludwigs-Universität Freiburg
Prof. Dr. Peter Thiemann
Dr. Philipp Klaus Krause

Sommersemester 2019

4. Programmieren in C

Abgabe bis 28. Mai, 10:15 GMT+2.

Aufgabe 0:

Schreibe Funktionen

```
unsigned char keilstrtodigit(const char keilstr[])
void keildigittostr(char keilstr[], unsigned char digit)
```

in `keildigit.c`, zur Umwandlung zwischen Strings und Ziffern! Ein String in babylonischer Keilschrift bzw. dessen Anfang soll dabei einer Ziffer im Sexagesimalsystem entsprechen. Im Fehlerfalle soll erstere Funktion `0` zurückgeben, letztere Funktion `""` in `keilstr` schreiben. Beispiele:

```
assert (keilstrtodigit("𒐜𒐐") == 4);
assert (keilstrtodigit("𒐐𒐜") == 44);
assert (keilstrtodigit("X𒐜𒐐") == 0);
char buffer[MAX_MB_LEN * 2 + 1];
keildigittostr(buffer, 44);
assert(!strcmp(buffer, "𒐐𒐜"));
```

Aufgabe 1:

Schreib eine Funktion

```
unsigned long keilstrtoul(const char keilstr[])
```

in `keilnumber.c`, zur Umwandlung zwischen Strings und Zahlen! Ein String in babylonischer Keilschrift soll dabei der kleinstmöglichen Ganzzahl, die er darstellen kann, entsprechen. Im Fehlerfalle soll die Funktion `0` zurückgeben. Beispiele:

```
assert (keilstrtoul("𒐜𒐐") == 280);
assert (keilstrtoul("𒐐𒐜") == 44);
assert (keilstrtoul("X𒐐𒐜") == 0);
```

Aufgabe 2:

Implementiere einen Stapel (auch Stack oder Keller genannt) von Strings in `strs.c`. Das Interface besteht aus dem Typ `strs_t` und den Funktionen

```
const char *push(strs_t *strs, const char str[])
const char *pop(strs_t *strs)
const char *top(const strs_t *strs)
```

`push` legt eine Kopie eines Strings auf dem Stapel ab und gibt einen Zeiger auf diese zurück. `pop` entfernt den obersten String vom Stapel (und gibt dabei soweit sinnvoll Speicher frei), und gibt einen Zeiger auf den danach obersten String zurück (falls es einen solchen gibt, sonst 0). `top` gibt einen Zeiger auf den obersten String zurück. Alle Funktionen geben im Fehlerfall immer 0 zurück. Ein neuer Stapel soll durch Initialisierung mit `{0}` nutzbar sein. Beispiel:

```
sstrs_t s = {0};
assert(top(&s) == 0);
push(&s, "Hallo");
assert(!strcmp(top(&s), "Hallo"));
push(&s, "Welt");
assert(!strcmp(top(&s), "Welt"));
pop(&s);
assert(!strcmp(top(&s), "Hallo"));
pop(&s);
assert(top(&s) == 0);
```

Aufgabe 3:

Erweitere das Spiel wumpus aus Aufgabe 3, Blatt 3 wie folgt!

Die Höhle soll nun 20 Räume haben. Dabei sollen von jedem Raum aus 2 bis 4 andere erreichbar sein. Jeder Raum soll mindestens eine symmetrische (also in beiden Richtungen begehbare) Verbindung zu einem anderen Raum haben. Der Geruch des Wumpus ist nun so stark, dass man ihn in allen Räumen, von denen aus man den Wumpus mit ein oder zwei Schritten erreichen kann, riecht. Fledermäuse hört man weiterhin aus allen Räumen, von denen aus man sie in einem Schritt erreichen kann.

Hinweise zur Auffrischung der Babylonischkenntnisse:

Babylonische Zahlen wurden etwa von 2000 v. Chr. bis 500 v. Chr. verwendet; im Laufe der Zeit gab es leichte Variationen der Schreibung, aber für dieses Blatt reicht das folgende:

Bei babylonischen Zahlen wurde abwechselnd Basis 10 und 6 verwendet, so dass sich schließlich Ziffern mit Basis 60 ergeben. Es gibt im babylonischen Zahlensystem keine 0. Innerhalb einer Zahl (aber nicht am Anfang oder Ende) wird die Ziffer 0 mit einem Leerzeichen dargestellt. Diese Darstellung ist nur bis auf Multiplikation mit Potenzen von 60 eindeutig (z.B. kann 𒁹 also 3600 oder 60, 1, $\frac{1}{60}$, $\frac{1}{3600}$ oder eine andere Potenz von 60 bedeuten. Welche Zahl gemeint ist, muss aus dem Kontext klar sein. Einerziffern werden mit senkrechten Keilen geschrieben, Zehnerziffern mit waagerechten oder leicht schrägen Keilen. Die Zahl 61 wird also als 𒐗𒐗 geschrieben (ohne Leerzeichen zwischen den Ziffern), die Zahl 3601 als 𒁹 𒁹 (mit einem Leerzeichen zwischen den Ziffern), während die Zahl 2 als 𒐖 geschrieben wird (überlappende Keile).

Folgende Tabelle gibt die Unicode-Codepoints der Ziffern an:

Babylonische Ziffer	Codepoint
𒁹	0x12079
𒐕	0x1222b
𒐙	0x12408
𒐚	0x12409
⋮	⋮
𒐞	0x1240e
𒑋	0x1230b
𒐜[1]	0x12399
𒑍	0x1230d
𒐟	0x1240f
𒐠	0x12410

[1] Aufgrund von Schriftartproblemem lässt sich das Zeichen im pdf hier nicht kopieren