Hinweise zur Bearbeitung und Abgabe

- Bitte nutzen Sie MARS zum Simulieren Ihrer Lösung. Stellen Sie sicher, dass Ihre Abgabe in MARS ausgeführt werden kann.
- Sie erhalten für jede Aufgabe eine separate Datei, die aus einem Vorgabe- und Lösungsabschnitt besteht. Ergänzen Sie bitte Ihren Namen und Ihre Matrikelnummer an der vorgegebenen Stelle. Bearbeiten Sie zur Lösung der Aufgabe nur den Lösungsteil unterhalb der Markierung:

```
#+ Loesungsabschnitt
#+ -----------------
```

- Ihre Lösung muss auch mit anderen Eingabewerten als den vorgegebenen funktionieren. Um Ihren Code mit anderen Eingaben zu testen, können Sie die Beispieldaten im Lösungsteil verändern.
- Bitte nehmen Sie keine Modifikationen am Vorgabeabschnitt vor und lassen Sie die vorgegebenen Markierungen (Zeilen beginnend mit #+) unverändert.
- Auch wenn die Simulation für Ihre Lösung korrekte Ergebnisse liefert, kann die Lösung Fehler enthalten. Eine korrekte Lösung muss auch die bekannten **Registerkonventionen** einhalten!
- Bitte gestalten Sie Ihren Assemblercode nachvollziehbar und verwenden Sie detaillierte Kommentare, um die Funktionsweise Ihres Assemblercodes darzulegen.
- Wir untersuchen alle Abgaben auf Plagiate. **Plagiate sind Täuschungsversuche und führen zur Bewertung „nicht bestanden“ für die gesamte Modulprüfung. Die aktuell bestehende Freiversuchsregelung an der TU Berlin greift in diesem Fall nicht.**
- Die Abgabe erfolgt über ISIS. Laden Sie die zwei Abgabedateien separat hoch.

Aufgabe 1: Ligaturen (6 Punkte)

Es wurde beschlossen, in MIPS-Assembler eine neue Software zur Textverarbeitung zu entwickeln, da es immer wieder zu Abstürzen mit den bestehenden Programmen kommt. Um die Dokumente besonders schön zu gestalten, sollen dabei bestimmte Buchstabenfolgen gesucht und optisch verbessert dargestellt werden. Diese besonderen Darstellungen von kurzen Buchstabenfolgen werden *Ligaturen* genannt. Abbildung 1 veranschaulicht dies mit zwei Beispielen.

Ihre Aufgabe ist es nun, die Funktion `lig` zu implementieren, welche in einer gegebenen Zeichenkette die Buchstabenfolgen `ff` und `fi` findet, durch `FF` und `FI` ersetzt und schließlich die Anzahl der vorgenommenen Ersetzungen zurückgibt. Die Ersetzung soll direkt in der übergebenen Zeichenkette vorgenommen werden (*in-place*) und soll nur bei den kleingeschriebenen Varianten `ff` und `fi` durchgeführt werden.

ff → ff fi → fi

Abbildung 1: Ligaturen

Beispiele:

- `"Waffeleisen"` → `"WaFFeleisen"`, Rückgabewert 1
- `"Goldfische empfinden Hoffnung."`
  → `"GoldFIsche empFInden HoFFnung."`, Rückgabewert 3
- `"Film"` → `"Film"`, Rückgabewert 0

Im folgenden sehen Sie die C-Signatur der gewünschten Funktion und die MIPS-Register für Parameter und Rückgabewert, welche sich aus der MIPS-Registerkonvention ergeben:

`int lig( char *str_in);`
`$v0` `$a0`

Aufgabe 2: Tiere im binären Suchbaum

In dieser Aufgabe wird mit einem Binärbaum gearbeitet. Die Knoten des Binärbaums sind in folgender Form im Speicher abgelegt:

```
struct node {
    char *animal_name;
    int weight;
    struct node *left;
    struct node *right;
}
```

Das Feld `animal_name` gibt einen Tiernamen an. Im Feld `weight` ist das typische Gewicht dieses Tiers in Gramm gespeichert. Die Zeiger `left` und `right` zeigen auf den linken und rechten Kindknoten. Wenn diese Zeiger den Wert 0 (`NULL`) annehmen, zeigt dies an, dass keine Kindknoten vorhanden sind.

Der Binärbaum ist alphabetisch (lexikographisch) nach Tiernamen sortiert. Das bedeutet: Der linke Unterbaum eines Knotens enthält nur Tiere mit Namen, welche alphabetisch vor dem Tiernamen des aktuellen Knotens liegen. Der rechte Unterbaum enthält nur Tiere mit Namen, welche alphabetisch hinter dem Tiernamen des aktuellen Knotens liegen. Die vorliegende Datenstruktur heißt *binärer Suchbaum* (*binary search tree, BST*). Erklärungen und Algorithmen dazu finden Sie beispielsweise im dazugehörigen Wikipedia-Artikel[1].

Um Zeichenketten zu vergleichen, stellt Ihnen die Vorgabe die Funktion `strcmp` zur Verfügung.

`int strcmp( char *a, char *b);`		
`$v0`	`$a0`	`$a1`

Der Rückgabewert von `strcmp` gibt Auskunft über das Verhältnis zwischen `a` und `b`:

- Rückgabewert -1: `a` $<$ `b` (`a` liegt im Alphabet vor `b`.)
- Rückgabewert 0: `a` $=$ `b` (`a` ist identisch zu `b`.)
- Rückgabewert $+1$: `a` $>$ `b` (`a` liegt im Alphabet hinter `b`.)

[1] `https://de.wikipedia.org/wiki/Binärer_Suchbaum`

Teil 2.1: Alphabetische Suche (7 Punkte)

In dieser Aufgabe soll die Funktion `bst_get_weight` implementiert werden. Diese soll ein durch den Tiernamen `animal_name` vorgegebenes Tier im Binärbaum finden und das im Baum vermerkte Gewicht zurückgeben. Der Wurzelknoten des Baums wird im Argument `root` übergeben.

Nutzen Sie bei dieser Aufgabe die Ordnungseigenschaft des Baums, um gezielt nach dem Tier zu suchen.

Falls das gesuchte Tier nicht im Baum gefunden wird, soll der Wert 0 zurückgegeben werden.

Signatur der zu implementierenden Funktion:

`int bst_get_weight( struct node *root, char *animal_name);`		
`$v0`	`$a0`	`$a1`

Teil 2.2: Schwerstes Tier (7 Punkte)

Implementieren Sie die Funktion `bst_heaviest`, welche den Namen des schwersten Tiers im Binärbaum zurückgibt. Der Wurzelknoten des Baums wird im Argument `root` übergeben.

Obwohl wir hier nur nach einem Element im Baum suchen, muss der gesamte Baum *traversiert* werden, da der Baum nicht nach Gewicht, sondern nach Tiername geordnet ist.

Signatur der zu implementierenden Funktion:

`char * bst_heaviest( struct node *root);`	
`$v0`	`$a0`

Hilfestellungen

- Unserer Einschätzung nach ist bei der Teilaufgabe 2.1 eine iterative Lösung am einfachsten umzusetzen, bei der Teilaufgabe 2.2 eine rekursive Lösung.
- Überprüfen Sie die Ausgabe Ihrer Lösung auf Plausibilität. Als schwerstes Tier sollte der Blauwal ausgegeben werden.
- Der in der Vorgabedatei enthaltene Binärbaum ist in Abbildung 2 abgebildet. Diese Abbildung kann bei der Fehlersuche in Ihrem Programmcode helfen und zum Verständnis beitragen.

Bei einer **C-Struktur (`struct`)** handelt es sich um eine Reihe von Datenwerten, die aufeinanderfolgend im Speicher abgelegt sind. Um auf diese Datenwerte zuzugreifen, muss ein Offset zum Struktur-Zeiger (`struct node *`) hinzuaddiert werden. Mit Byteadressen berechnen sich die Adressen der Felder eines Beispielknotens `struct node *n` folgendermaßen:

- `&n->animal_name` $:= \mathtt{n} + 0$
- `&n->weight` $:= \mathtt{n} + 4$
- `&n->left` $:= \mathtt{n} + 8$
- `&n->right` $:= \mathtt{n} + 12$

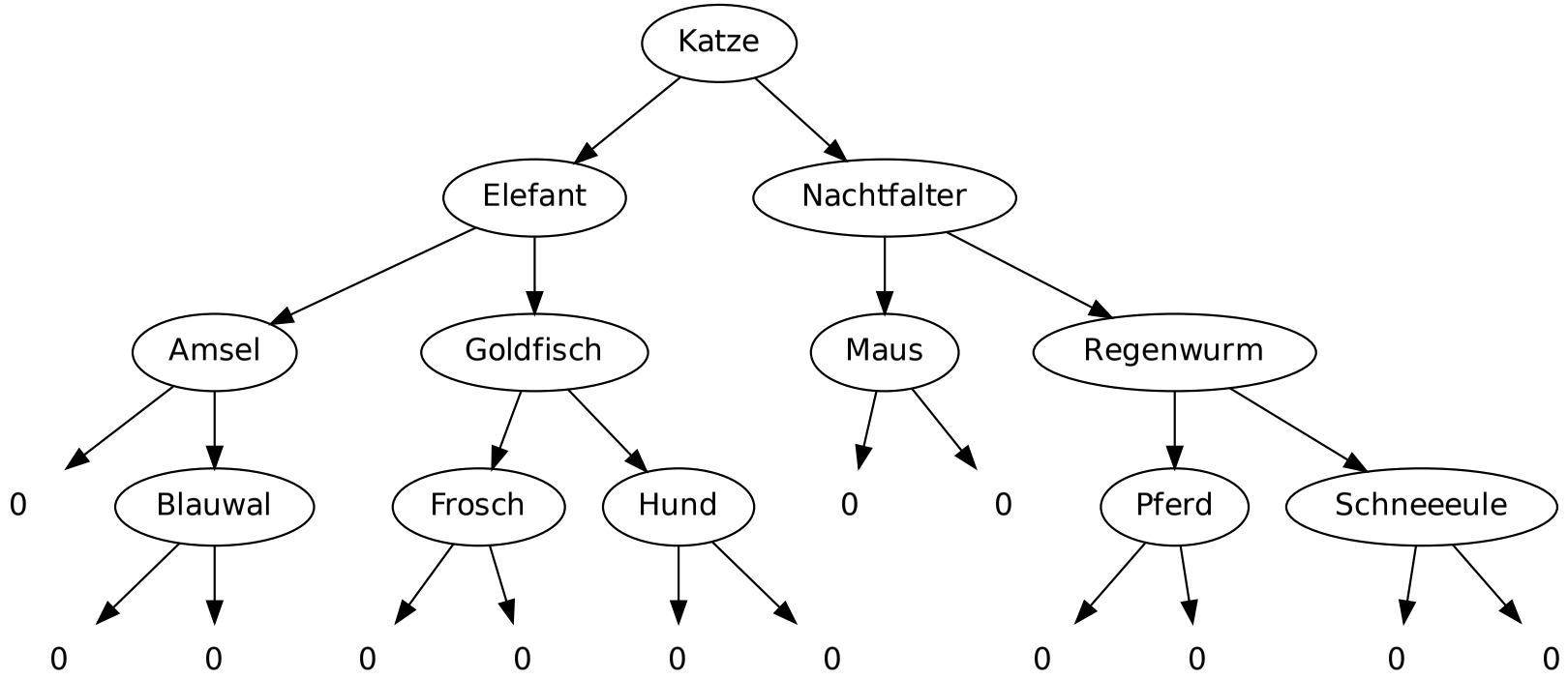

Abbildung 2: Binärer Suchbaum aus der Vorgabedatei