Projekt 1

Compiler Design Lab
Prof. Dr. Sebastian Hack
Dipl.-Inform. Christoph Mallon

MIPS-Potpourri (10 Punkte)

Im Rahmen dieses Projekts implementieren Sie vier Unterprogramme in MIPS. Hier sind vor den einzelnen Aufgabenbeschreibungen einige allgemeine Hinweise:

- Machen Sie sich zunächst mit Papier und Stift klar, wie sie die einzelnen Aufgaben algorithmisch angehen können. Fertigen sie hierzu Skizzen der Datenstrukturen an und wie sich diese Strukturen und Verweise darauf schrittweise ändern.
- Testen Sie regelmäßig. Zu jeder Teilaufgabe ist eine Testinfrastruktur sowie einige Tests mitgeliefert. Die Tests können Sie jeweils mit `./runtests` im jeweiligen Aufgabenverzeichnis starten. Weitere Tests können Sie im jeweiligen Unterverzeichnis `tests` ablegen. Schauen Sie sich hierzu den Aufbau der gegebenen Tests an.
- Überlegen Sie, welche Randfälle auftreten können. Behandeln Sie diese in Ihrer Implementierung und erstellen Sie entsprechende Testfälle.
- Beachten Sie die Aufrufkonventionen von MIPS.
- Gehen Sie davon aus, dass die Eingaben an Ihr Unterprogramm korrekt sind. So ist z.B. die Reihung für die binäre Suche garantiert sortiert. Verwechseln Sie jedoch nicht Randfälle mit ungültigen Eingaben!
- Achten Sie darauf, dass sich Ihre Lösung ausschließlich in der jeweiligen Datei zu der entsprechenden Aufgabe befindet. Siehe hierzu die Kommentare *TODO* in den Projektdateien. Das Testsystem wird Änderungen an anderen Dateien (zum Beispiel an `main.s`) sowie von Ihnen hinzugefügte Dateien beim Testen ignorieren. Selbstverständlich können Sie andere Dateien für Ihre eigenen Tests verändern.
- Erzeugen Sie in den Abgabedateien auch keine Ausgaben. Andernfalls wird das Testsystem Ihre Abgabe als falsch bewerten.
- Beachten Sie, dass Ihre Unterprogramme etwas *zurückgeben* und *nicht* etwas ausgeben sollen.
- Sie erhalten das Projekt unter `https://prog2scm.cdl.uni-saarland.de/git/project1/$NAME`, wobei für `$NAME` Ihr Benutzername einzusetzen ist.
- Als Ihre Abgabe wird der zum Abgabezeitpunkt aktuelle Stand des Zweigs *master* auf dem Projektserver (*prog2scm*) gewertet.
- Projektabgabe ist am *13.5.2014 23:59 MESZ*.

1 Ist die Reihung sortiert? (1 Punkt)

Schreiben Sie ein Unterprogramm, das prüft, ob eine Reihung bestehend aus vorzeichenbehafteten Wörtern aufsteigend sortiert ist.

Das Programm erhält als erstes Argument die Adresse auf das erste Element der Reihung. Das zweite Argument ist die Adresse direkt hinter der letzten Zahl der Reihung.[1]

Das Unterprogramm soll die Adresse der ersten Stelle in der Reihung zurückgeben, die nicht mindestens so groß ist wie das Element davor. Ist die Reihung sortiert, so soll das Unterprogramm 0 zurückgeben.

[1] Dies ist analog zum Übungsblatt.

2 Binäre Suche (4 Punkte)

Schreiben Sie ein Unterprogramm, das das Vorhandensein eines vorzeichenbehafteten Wortes in einer aufsteigend sortierten Reihung von vorzeichenbehafteten Wörtern prüft.

Das Programm erhält als erstes Argument die Adresse auf das erste Element der Reihung. Das zweite Argument ist die Adresse direkt hinter der letzten Zahl der Reihung.[1] Das dritte Argument ist das gesuchte Wort.

Wird das Wort gefunden, so soll das Unterprogramm die Adresse, an der die Zahl in der Reihung steht, zurückgeben. Andernfalls soll es 0 zurückgeben.

Binäre Suche nutzt die Eigenschaft, dass die Eingabereihung sortiert ist, um schneller das gesuchte Element zu finden. Hierzu wird zunächst das Element genau in der Mitte der beiden Grenzen betrachtet. Hat dies den gesuchten Wert, so ist die Suche beendet. Andernfalls ist das Element entweder größer oder kleiner als die gesuchte Zahl. Ist das Element größer, so wird seine Position nun als neue obere Grenze verwendet. Andernfalls wird es als neue untere Grenze verwendet. Nun wird der Vorgang mit den angepassten Grenzen wiederholt. Dabei wird in jedem Schritt die Größe des Intervalls halbiert. Ist das Intervall leer, so wird die Suche erfolglos abgebrochen.

3 Funktionsanwendung auf eine verkettete Liste (2 Punkte)

Schreiben Sie ein Unterprogramm A, das ein Unterprogramm B auf jedes Element einer einfach verketteten Liste in Listenreihenfolge anwendet.[2]

Die Adresse des ersten Elements der Liste wird als erstes Argument übergeben. Die Adresse des anzuwendenen Unterprogramms B wird als zweites Argument übergeben.

Am Anfang jeden Elements steht die Adresse des nächsten Elements. Das letzte Element der Liste enthält hier den Wert 0. Nach der Adresse folgt jeweils direkt eine unbekannte Menge an Daten.[1]

Das aufzurufende Unterprogramm B erwartet als erstes (und einziges) Argument die Adresse, die auf den Anfang der Daten eines Elements zeigt.

4 Destruktives filtern einer verketteten Liste (3 Punkte)

Schreiben Sie ein Unterprogramm A, das Elemente abhängig von einem Unterprogramm B, das als Prädikat dient, aus einer einfach verketteten Liste entfernt.[3] Es wird hierbei keine neue Liste erstellt, sondern die bestehende modifiziert. Die Reihenfolge der übrigbleibenden Elemente soll erhalten bleiben.

Die Adresse des ersten Elements der Liste wird als erstes Argument übergeben. Die Adresse des Unterprogramms B wird als zweites Argument übergeben.

Am Anfang jeden Elements steht die Adresse des nächsten Elements. Das letzte Element der Liste enthält hier den Wert 0. Nach der Adresse folgt jeweils direkt eine unbekannte Menge an Daten.[1]

Das Unterprogramm B erwartet als erstes (und einziges) Argument die Adresse, die auf den Anfang der Daten eines Elements zeigt. Ist der Rückgabewert des Unterprogramms B 0, so soll das Element aus der Liste entfernt werden. Andernfalls soll es erhalten bleiben.

Das Unterprogramm hat als Rückgabewert die Adresse des ersten Elements der gefilterten Liste.

[2] Dies ist eine Verallgemeinerung der Funktionen foldl und map in Programmierung 1.
[3] Dies ist ähnlich der Funktion filter in Programmierung 1.