Hinweise zur Bearbeitung und Abgabe

Die Lösung der Hausaufgabe muss **eigenständig** erstellt werden. Abgaben, die identisch oder auffällig ähnlich zu anderen Abgaben sind, werden als **Plagiat** gewertet! **Plagiate sind Täuschungsversuche und führen zur Bewertung „nicht bestanden" für die gesamte Modulprüfung.**

- 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.
- Eine korrekte Lösung muss die bekannten **Registerkonventionen** einhalten. Häufig können trotz nicht eingehaltener Registerkonventionen korrekte Ergebnisse geliefert werden. In diesem Fall werden trotzdem Punkte abgezogen.
- Falls Sie in Ihrer Lösung zusätzliche Speicherbereiche für Daten nutzen möchten, verwenden Sie dafür bitte ausschließlich den **Stack** und keine statischen Daten in den Datensektionen (`.data`).
- Die zu implementierenden Funktionen müssen als Eingaben die Werte in den **Argument-Registern** (`$a0`–`$a3`) nutzen.
- Bitte gestalten Sie Ihren Assemblercode **nachvollziehbar** und verwenden Sie detaillierte **Kommentare**, um die Funktionsweise Ihres Assemblercodes darzulegen.
- Die Abgabe erfolgt über ISIS. Laden Sie die zwei Abgabedateien separat hoch.

Aufgabe 1: Fehlstände einer Permutation (10 Punkte)

Hintergrund: Eine Permutation („Vertauschung") ist eine Anweisung, wie Elemente eines Arrays umgeordnet werden sollen. Die Beispielpermutation π wird in Abbildung 1 auf ein Array A angewandt. π ersetzt A[0] durch A[**1**], A[1] durch A[**2**] und A[2] durch A[**0**], A[3] behält das Element A[**3**]. Daraus ergibt sich die Darstellung der Permutation als Tupel $\pi = (\mathbf{1} \quad \mathbf{2} \quad \mathbf{0} \quad \mathbf{3})$.

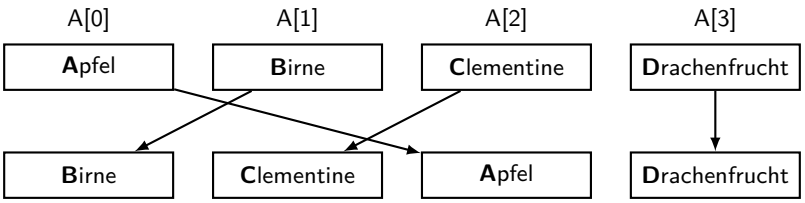


Abbildung 1: Permutation π wird auf Array A angewendet.

Wenn die Reihenfolge zweier Elemente durch die Permutation umgekehrt wird, liegt ein Fehlstand vor. Ein Fehlstand (*inversion*) lässt sich in der Abbildung 1 daran erkennen, dass sich zwei Pfeile kreuzen. Die Fehlstandszahl (Anzahl von Fehlständen, englisch *inversion count*) von π beträgt also 2. Die Fehlstandszahl misst die Unordnung, die eine Permutation erzeugt.

Aufgabe: Implementieren Sie die Funktion `numinv`, welche die Fehlstandszahl einer Permutation `perm` berechnet und zurückgibt. Bei `perm` handelt es sich um ein Wort-Array mit `length` Elementen. Nachfolgend die C-Signatur der zu implementierenden Funktion mit den MIPS-Registern für die Parameter und den Rückgabewert:

`int`	`numinv(`	`unsigned int* perm,`	`int length);`
`$v0`		`$a0`	`$a1`

Die Funktion `numinv` soll mit zwei Zählvariablen i und j die Einträge der Permutation durchlaufen. Die Zählvariablen sollen alle Kombinationen durchlaufen, für die gilt: $i < j$ und $0 \leq i < \texttt{length} - 1$ und $1 \leq j \leq \texttt{length} - 1$. Falls $\pi(i) > \pi(j)$ ist, soll die Fehlstandszahl um 1 erhöht werden.

Beispiel: In der folgenden Tabelle werden die Fehlstände der Beispielpermutation aus Abbildung 1 gezählt. Der korrekte Rückgabewert des Aufrufs `numinv`$\big(\begin{smallmatrix}1 & 2 & 0 & 3\end{smallmatrix}\big)$, 4) ist also 2.

i	**j**	Permutation $\big(\begin{smallmatrix}1 & 2 & 0 & 3\end{smallmatrix}\big)$	Fehlstand? ($\pi(i) > \pi(j)$)
0	1	**1** **2**	nein
0	2	**1** **0**	**ja**
0	3	**1** **3**	nein
1	2	**2** **0**	**ja**
1	3	**2** **3**	nein
2	3	**0** **3**	nein

Aufgabe 2: In-Place Permutation (10 Punkte)

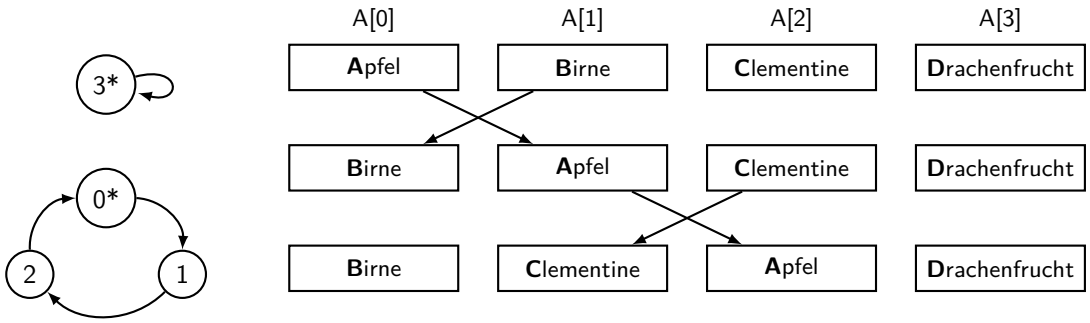
Aufgabe: Implementieren Sie die Funktion `permutate`, welche mittels einer übergebenen Permutation `perm` die Elemente des Arrays `objects` umordnet. Dabei soll das Eingabearray schrittweise durch Tauschen von jeweils zwei Elementen transformiert werden. Es dürfen **keine zusätzlichen Arrays**, z. B. für Kopien des Eingabearrays oder für ein separates Ergebnisarray, genutzt werden (*in-place*). Die Länge der Permutation wird als Parameter `perm_len` übergeben. (`perm_len` ist ebenfalls die Länge des Arrays `objects`.) Signatur der zu implementierenden Funktion:

`void`	`permutate(`	`char *objects,`	`int *perm,`	`int perm_len);`
		`$a0`	`$a1`	`$a2`

Hintergrund: Elemente der Permutation, die untereinander vertauscht werden, bilden einen Zyklus[1]. Für die Lösung dieser Aufgabe nutzen wir, dass Permutationen in solche Zyklen zerlegt werden können. Die Beispielpermutation $\pi = \big(\begin{smallmatrix}1 & 2 & 0 & 3\end{smallmatrix}\big)$ besteht aus zwei Zyklen. Diese sind in Abbildung 2a dargestellt.

Ein Zyklus der Länge n kann durch $n-1$ Tauschoperationen auf dem Array `objects` angewandt werden: Für den Zyklus 0 1 2 (Länge 3) sind Element 0 und 1 zu tauschen, dann Element 1 und 2. Der Zyklus 3

[1] Weiterführend: `https://de.wikibooks.org/wiki/Mathe_für_Nicht-Freaks:_Permutationen`. Interessant zu dem Thema ist auch das „Problem der 100 Gefangenen“: `https://www.youtube.com/watch?v=iSNsgj1OCLA`.



(a) π besteht aus zwei Zyklen. (b) Der Zyklus 0 1 2 wird durch zwei Vertauschungen auf Array A angewandt.

Abbildung 2: Die Permutation π wird in Zyklen zerlegt, aus welchen sich die Tauschoperationen ergeben.

(Länge 1) braucht keine Tauschoperationen. Die Hintereinanderausführung der zwei Tauschoperationen ist in Abbildung 2b dargestellt. Das Ergebnis ist identisch zum Ergebnis in Abbildung 1.

Das Element eines Zyklus mit dem geringsten Index nennen wir Kopfelement. Jeder Zyklus hat genau ein solches Kopfelement. Die vorgegebene Hilfsfunktion `cycle_head` gibt den Wert 1 zurück, wenn es sich bei dem Element `idx` der Permutation `perm` um das Kopfelement eines Zyklus handelt. Andernfalls wird 0 zurückgegeben. Kopfelemente sind in Abbildung 2a mit Sternchen markiert. Über die Kopfelemente kann sichergestellt werden, dass die Tauschoperationen für jeden Zyklus nur einmal ausgeführt werden. Signatur von `cycle_head`:

`int`	`cycle_head(`	`int *perm,`	`int idx)`
`$v0`		`$a0`	`$a1`

Die Elemente des Arrays `objects` sind 16 Byte lange Strings. Um zwei Elemente des Arrays zu tauschen, ist die vorgegebene Hilfsfunktion `swap` zu nutzen, welche `objects` sowie die zu tauschenden Indizes `k` und `l` als Parameter erhält:

`void`	`swap(`	`char *objects,`	`int k,`	`int l);`
		`$a0`	`$a1`	`$a2`

Algorithmus: Basierend auf diesen Hintergründen ist `permutate` wie folgt zu implementieren: Iterieren Sie über die Elemente der Permutation `perm` und prüfen Sie für jedes Element, ob es sich um ein Kopfelement handelt. Für jedes Kopfelement, das gefunden wird, sollen Tauschoperationen entlang des Zyklus ausgeführt werden, bis das Kopfelement wieder erreicht ist. Die Hilfsfunktionen `swap` und `cycle_head` müssen verwendet werden.

Erwartete Ergebnisse

Die folgende Tabelle zeigt die Ergebnisse, die korrekte Lösungen für `numinv` (Aufgabe 1) und `permutate` (Aufgabe 2) für die vorgegebenen Permutationen (`inputs`) liefern sollen.

Permutation	**Ergebnis** `numinv`	**Ergebnis** `permutate` (Früchte zwecks Übersichtlichkeit abgekürzt.)
(1 2 0 3)	2	B C A D
(0 1 3 2)	1	A B D C
(0 1 2 3)	0	A B C D
(0 1 2 3 4 5 6 7 8 9)	0	A B C D E F G H I J
(1 2 0 4 5 3 7 8 9 6)	7	B C A E F D H I J G
(9 0 1 2 3 4 5 6 7 8)	9	J A B C D E F G H I
(2 7 4 6 9 10 5 1 0 8 3 11)	30	C H E G J K F B A I D L
(4 3 2 15 5 14 11 10 1 8 0 12 9 13 7 6)	58	E D C P F O L K B I A M J N H G

A = Apfel, B = Birne, C = Clementine, D = Drachenfrucht, E = Erdbeere, F = Feige, G = Granatapfel, H = Himbeere, I = Ingwer, J = Johannisbeere, K = Kirsche, L = Limette, M = Mango, N = Nektarine, O = Orange, P = Pfirsich