Übung 3: Verkettete Listen

Hinweise zu den Praktikumsaufgaben

1. Bitte bearbeiten Sie die Praktikumsaufgaben zu zweit. Die Zweierteams finden Sie in unserem moodle-Raum im Abschnitt „Praktikum 1“.
2. Bitte halten Sie sich an die Programmierrichtlinie, so dass Sie beispielsweise Ihre Programme ausführlich kommentieren und den Variablen selbsterklärende Namen geben.
3. Denken Sie auch an die Behandlung von Fehlern in den Programmen, zum Beispiel ob der Benutzer eine falsche Eingabe tätigt.
4. Achten Sie auf eine gute Benutzerführung, so dass der Benutzer zu jedem Zeitpunkt weiß, was von ihm erwartet wird.
5. Überlegen Sie sich Testfälle, mit denen Sie die korrekte Funktionsweise Ihres Programms überprüfen.
6. Sie können die Aufgaben in demselben Visual Studio Projekt programmieren.
7. Bitte bereiten Sie sich auf die Übung vor, in dem Sie Ihre Lösungen rechtzeitig in moodle abgeben (3 Tage vor Ihrem Übungstermin), Ihre Lösung in einer integrierten Entwicklungsumgebung (IDE) vorbereiten und jedes Teammitglied die Lösungen aller Aufgaben erklären kann. Die abgegebenen Programme müssen kompilierbar und getestet sein.
8. Den erstellten Programmcode führen Sie uns bitte in der Übung vor.
9. Bei Fragen zum Übungsblatt nutzen Sie gerne das Forum in unserem moodle-Raum.

Ziele dieser Übung:

- Die Studierenden beherrschen den Umgang mit speziellen Datentypen.
- Sie können dynamisch Speicherbereiche anlegen und freigeben.
- Sie implementieren und verwenden verkettete Listen.

Aufgabe zur Vorbereitung zuhause: Prüfungsliste

Zur Erfassung aller Prüfungsteilnehmer einer Klausur muss eine Prüfungsliste geführt werden. In der Liste sollen alle relevanten Informationen der Prüfungsteilnehmer gespeichert werden:

- Vor- und Nachnamen des Studierenden
- E-Mail-Adresse
- Matrikelnummer
- Note

Die Liste soll als *verkettete Liste* implementiert werden.

Hinweise:

- In der Implementierung der Aufgabe dürfen Sie keine globalen Variablen verwenden.
- Die Liste darf nicht als statischer Vektor umgesetzt werden.
- Die Studierenden werden in der verketteten Liste aufsteigend nach ihren Matrikelnummern gespeichert.
- Möchte der Prüfer einen neuen Studierenden in die Liste aufnehmen, dessen Matrikelnummer mit der Matrikelnummer eines bereits gespeicherten Studierenden übereinstimmt, erhält er aufgrund einer doppelten Eingabe eine Fehlerausgabe.

Aufgabe 1 Definition der Datenstruktur `sStudent`

Implementieren Sie die notwendigen strukturierten Datentypen und Funktionen zum Umgang mit einer verketteten Liste in einem eigenen Modul, das aus einer `.h`-Datei für die strukturierten Datentypen und Funktionsprototypen sowie einer `.c`-Datei für die Implementierung der Funktionen besteht.

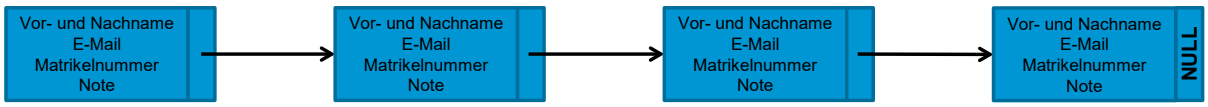
Definieren Sie einen strukturierten Datentyp zur Speicherung der Informationen eines Prüfungsteilnehmers mit dem Namen `sStudent`. Der Name in dem strukturierten Datentyp `sStudent` soll wiederum als strukturierter Datentyp bestehend aus Vor- und Nachnamen als Zeichenketten definiert werden. Die E-Mail-Adresse ist als Zeichenkette und die Matrikelnummer sowie die Note als `int` zu realisieren.

Aufgabe 2 Definition der Datenstruktur `sElement`

Definieren Sie zusätzlich einen strukturierten Datentyp `sElement`, der als Element in der verketteten Liste genutzt wird und die Informationen eines Prüfungsteilnehmers enthält. Es gibt verschiedene Möglichkeiten, diese Liste und entsprechend die Listenelemente zu implementieren. Hiervon hängt auch die Definition des strukturierten Datentyp `sElement` ab. Im Folgenden werden 4 Möglichkeiten zur Umsetzung als Beispiel beschrieben:

Möglichkeit 1: Einfach verkette Liste mit Studierendendaten

Bei dieser Möglichkeit werden die Informationen des Studierenden direkt im Listenelement gespeichert. Die einzelnen Listenelemente werden durch einen Zeiger `next*` verkettet. Schematisch würde diese Implementierung wie folgt aussehen:

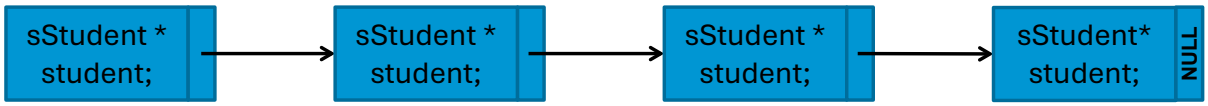


Der Datentyp `sElement` könnte dann zum Beispiel wie folgt aussehen:

```
typedef struct sElement {
	sStudent student;
	struct sElement* next;
} sElement;
```

Möglichkeit 2: Einfach verkettete Liste mit Zeiger auf Studierendendaten

Bei dieser Möglichkeit speichert jedes Listenelement einen Zeiger auf die Informationen des Studierenden. Die einzelnen Listenelemente werden durch einen Zeiger `next*` verkettet. Schematisch würde diese Implementierung wie folgt aussehen:

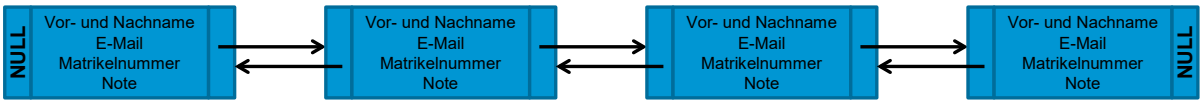


Der Datentyp `sElement` könnte dann zum Beispiel wie folgt aussehen:

```
typedef struct sElement {
	sStudent* student;
	struct sElement* next;
} sElement;
```

Möglichkeit 3: Doppelt verkettete Liste mit Studierendendaten

Bei dieser Möglichkeit werden die Informationen des Studierenden direkt im Listenelement gespeichert. Die einzelnen Listenelemente werden durch einen Zeiger `prev*` auf das vorherige Listenelement und einen Zeiger `next*` auf das nachfolgende Listenelement verkettet. Schematisch würde diese Implementierung wie folgt aussehen:

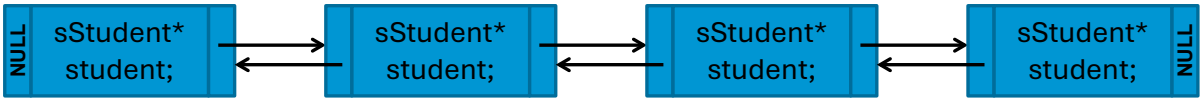


Der Datentyp `sElement` könnte dann zum Beispiel wie folgt aussehen:

```
typedef struct sElement {
	sStudent student;
	struct sElement* prev;
	struct sElement* next;
} sElement;
```

Möglichkeit 4: Doppelt verkettete Liste mit Zeiger auf Studierendendaten

Bei dieser Möglichkeit speichert jedes Listenelement einen Zeiger auf die Informationen des Studierenden. Die einzelnen Listenelemente werden durch einen Zeiger `prev*` auf das vorherige Listenelement und einen Zeiger `next*` auf das nachfolgende Listenelement verkettet. Schematisch würde diese Implementierung wie folgt aussehen:



Der Datentyp `sElement` könnte dann zum Beispiel wie folgt aussehen:

```
typedef struct sElement {
	sStudent* student;
	struct sElement* prev;
	struct sElement* next;
} sElement;
```

Überlegen und diskutieren Sie, wie sich diese 4 Möglichkeiten voneinander unterscheiden und welche Vor- und Nachteile diese jeweils besitzen. Entscheiden Sie sich für EINE der beschriebenen Möglichkeiten zur Umsetzung der verketteten Liste. Hiervon hängt auch die Implementierung der nachfolgenden Funktionen ab. Natürlich sind auch weitere Varianten als die 4 oben beschriebenen Möglichkeiten möglich.

Schreiben Sie Ihre Entscheidung und die Gründe hierfür als Kommentar in Ihren Quelltext.

Aufgabe 3 Zählen der Studierenden

Schreiben Sie eine Funktion `zaehleStudierende`, die die Anzahl aller Studierenden in der Prüfungsliste zählt und auf der Konsole ausgibt.

Diese Funktion können Sie in den nachfolgenden Funktionen verwenden.

Aufgabe 4 Einfügen neuer Studierenden

Fragen Sie den Prüfer in der `main`-Funktion, ob er einen weiteren Studierenden in der Prüfungsliste aufnehmen möchte, und nehmen Sie ggf. die Daten in der Liste auf. Die verkettete Prüfungsliste ist hierbei vollständig dynamisch. Der Dialog mit dem Prüfer mittels Aus- und Eingaben könnte zum Beispiel wie folgt aussehen:

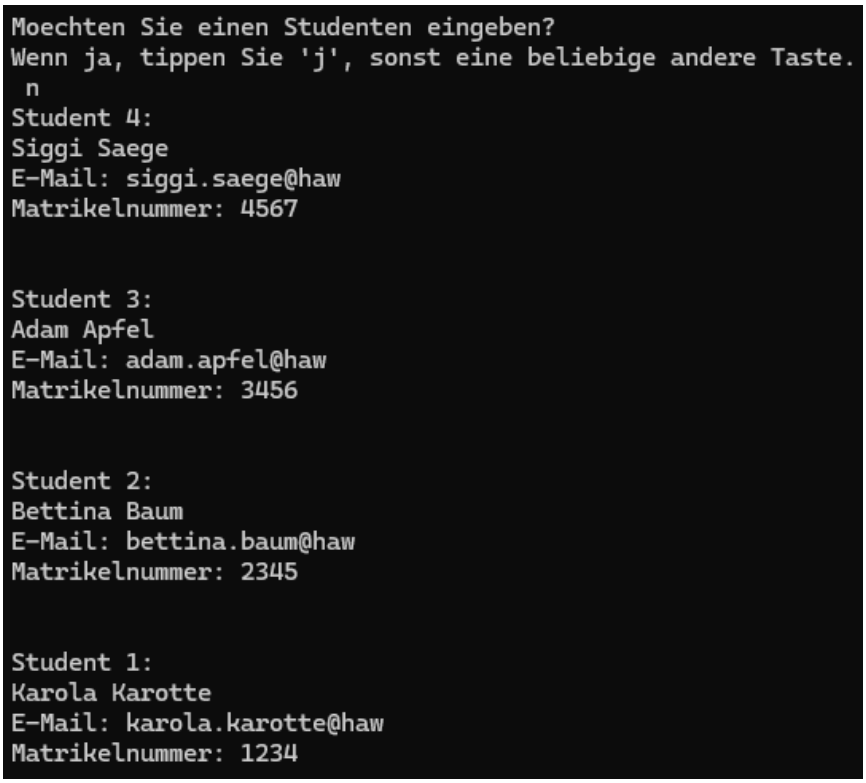
```
Moechten Sie einen Studenten eingeben?
Wenn ja, tippen Sie 'j', sonst eine beliebige andere Taste.
 j
Bitte geben Sie den Vornamen von dem Studenten ein: Bettina
Bitte geben Sie den Nachnamen von dem Studenten ein: Baum
Bitte geben Sie die E-Mail-Adresse ein: bettina.baum@haw
Bitte geben Sie die Matrikelnummer ein: 2345
Bitte geben Sie die Note ein: 14

Moechten Sie einen Studenten eingeben?
Wenn ja, tippen Sie 'j', sonst eine beliebige andere Taste.
 j
Bitte geben Sie den Vornamen von dem Studenten ein: Siggi
Bitte geben Sie den Nachnamen von dem Studenten ein: Saege
Bitte geben Sie die E-Mail-Adresse ein: siggi.saege@haw
Bitte geben Sie die Matrikelnummer ein: 4567
Bitte geben Sie die Note ein: 8

Moechten Sie einen Studenten eingeben?
Wenn ja, tippen Sie 'j', sonst eine beliebige andere Taste.
 j
Bitte geben Sie den Vornamen von dem Studenten ein: Karola
Bitte geben Sie den Nachnamen von dem Studenten ein: Karotte
Bitte geben Sie die E-Mail-Adresse ein: karola.karotte@haw
Bitte geben Sie die Matrikelnummer ein: 1234
Bitte geben Sie die Note ein: 10

Moechten Sie einen Studenten eingeben?
Wenn ja, tippen Sie 'j', sonst eine beliebige andere Taste.
 j
Bitte geben Sie den Vornamen von dem Studenten ein: Adam
Bitte geben Sie den Nachnamen von dem Studenten ein: Apfel
Bitte geben Sie die E-Mail-Adresse ein: adam.apfel@haw
Bitte geben Sie die Matrikelnummer ein: 3456
Bitte geben Sie die Note ein: 5
```

Fügen Sie jeden neuen Studierenden in aufsteigender Reihenfolge seiner Matrikelnummer in die Prüfungsliste ein (siehe auch obenstehende Hinweise). Beachten Sie hierbei auch die verschiedenen Sonderfälle, die beim Einfügen in der Prüfungsliste auftreten können (zum Beispiel leere Liste und Listenende).

Aufgabe 5 Ausgabe der Prüfungsliste

Erstellen Sie eine Funktion zur Ausgabe der Prüfungsliste, die alle Studierenden in der Reihenfolge ihrer Matrikelnummern (also aufsteigend) ausgibt. Alternativ können Sie die Studierenden auch in absteigender Reihenfolge ausgeben.

Die Ausgabe des obigen Beispiels in absteigender Reihenfolge der Matrikelnummern würde wie folgt aussehen:



Aufgabe für die Laborzeit

Aufgabe 6 Löschen der Prüfungsliste

Implementieren Sie eine Funktion `loescheListe`, die die gesamte Prüfungsliste mit allen gespeicherten Studierenden löscht.

Aufgabe 7 Löschen eines Studierenden

Implementieren Sie eine Funktion `loescheStudi`. Diese Funktion fragt den Prüfer nach der Matrikelnummer des Studierenden, der aus der Prüfungsliste (zum Beispiel wegen Krankheit) entfernt werden soll. Entfernen Sie den Studierenden aus der verketteten Liste und geben Sie danach die Liste mittels der Funktion aus Aufgabe 5 auf der Konsole aus.

Aufgabe 8 Aufwand

Überlegen Sie sich den Aufwand für jede der implementierten Funktionen im Best Case und im Worst Case, wenn die Funktion zur Laufzeit ausgeführt wird. Wie sieht der Best Case, wie sieht der Worst Case aus? Bei welchen Funktionen ist der Aufwand bei der Ausführungszeit immer derselbe, also sowohl im Best Case als auch im Worst Case? Sie können Ihre Antworten auch mit Hilfe von Ausgaben in den einzelnen Funktionen überprüfen.

Aufgabe 9 Vergleich der Umsetzungsmöglichkeiten

Nachdem Sie nun die Prüfungsliste implementiert haben, schauen Sie sich noch einmal die 4 in Aufgabe 2 beschriebenen Implementierungsmöglichkeiten an. Welche Möglichkeit halten Sie für die beste Implementierungsmöglichkeit für die Prüfungsliste hinsichtlich Einfügens eines Studierenden, Löschens eines Studierenden und Ausdrucken der Prüfungsliste? Würden Sie basierend auf ihren Erfahrungen beim nächsten Mal eine andere Implementierungsmöglichkeit wählen?

Optionale Zusatzaufgaben: Dynamische Speicherverwaltung

- Implementieren Sie auch eine oder mehrere unter Aufgabe 2 beschriebenen alternativen Implementierungsmöglichkeiten für die Prüfungsliste.
- Implementieren Sie die Filmverwaltung aus Übung 2 dynamisch mit einer verketteten Liste (also ohne einen statischen Vektor). Nutzen Sie hierfür die dynamische Speicherverwaltung.
- Erweitern Sie Ihr Programm *Vokale zählen* aus Übung 1, Aufgabe 4.1. unter Nutzung der dynamischen Speicherverwaltung wie folgt:
    - Fragen Sie den Benutzer, wie viele Buchstaben der String maximal umfasst, dessen Vokale der Benutzer zählen lassen möchte.
    - Legen Sie sich anschließend ein Datenobjekt an, das genauso viele Zeichen wie vom Benutzer definiert speichern kann.
    - Lesen Sie dann ein Wort vom Benutzer ein und speichern dieses in dem erstellten Speicherbereich.
    - Zählen Sie anschließend die Vokale in dem Wort und geben diese Anzahl dem Benutzer aus.
- Implementieren Sie die Funktionen zum Zählen von Vokalen aus Übung 1, Aufgabe 4 so, dass der Benutzer dynamisch eingibt, von wie lange eine Zeichenkette sein darf und / oder in wie vielen Zeichenketten er die Vokale zählen möchte.

Viel Spaß und viel Erfolg beim Programmieren!