Allgemeine Hinweise:

Auf diesem Übungsblatt operieren Sie viel mit Zeigern. Bei der Verwendung von Zeigern treten sehr schnell subtile Programmierfehler auf. Prüfen Sie stets, ob ein Zeiger Null ist, bevor Sie ihn verwenden. Denken Sie an Sonderfälle wie leere Listen oder Listen mir nur einem Eintrag und geben Sie zur Not die Speicheradressen mit der `print()`-Anweisung aus, falls Sie nicht gleich wissen, wo sich der Nullzeiger versteckt, den Sie aus Versehen dereferenziert haben.

Übung 1 *LIFO vs. FIFO*

Bei abstrakten Datentypen unterscheidet man zwischen zwei prinzipiellen Arten:

- *Last in First out (LIFO):* Elemente werden in entgegengesetzter Reihenfolge ihrer Ablage wieder aus der Datenstruktur entnommen. Die prototypische Datenstruktur hierfür ist der Stack.
- *First in First out (FIFO):* Elemente werden in der Reihenfolge ihrer Ablage entnommen. Der prototypische Vertreter hier für ist die Queue.

Entscheiden Sie für die folgenden Phänomene aus Alltag und Informatik, welche Art von Datentyp Sie für eine formale Beschreibung des Phänomens verwenden würden und begründen Sie jeweils kurz:

1. Kunden in einer Bäckerei
2. Menschen in einem Aufzug
3. Lebensmittel in Ihrem Kühlschrank
4. Autos auf einer Autofähre
5. Druckanträge auf einem Drucker
6. Besuchen der Knoten eines Baumes nach dem Depth-First Prinzip (siehe beispielsweise den Ableitungsbaum auf Blatt 1)
7. Rekursive Funktionsaufrufe

(4 Punkte)

Übung 2 *Hase und Igel*

Sie haben in der Vorlesung die Datenstruktur der einfach verketteten Liste kennen gelernt. Diese beginnt mit einem Element, auf das keines der Elemente zeigt, und endet mit einem Element, dessen Nachfolger auf die spezielle Adresse `0` zeigt. Wenn man die Definition etwas erweitert, kann es jedoch sein, dass einer der Zeiger zurück auf eines der vorherigen Elemente zeigt. Die Kette ist dann nicht linear, sondern ähnelt dem griechischen Buchstaben ρ. Durchläuft man eine solche Liste, so gerät man in eine Endlosschleife, und die besuchten Elemente wiederholen sich zyklisch.

Ziel dieser Aufgabe ist es, einen Algorithmus zu implementieren, der für eine gegebene Liste feststellen kann, ob diese einen Zyklus enthält oder wie in der Vorlesung vorgestellt linear ist. Dazu werden zwei Zeiger gleichzeitig durch die Liste bewegt, wobei einer von ihnen (der *Hase*) stets zum übernächsten Element springt, während der zweite (der *Igel*) wie üblich alle Elemente der Reihe nach besucht.

a) Begründen Sie theoretisch, wie man mit dem Konzept von *Hase* und *Igel* erkennt, ob eine Liste einen Zyklus enthält. Betrachten Sie dabei den allgemeinen Fall einer Liste mit n-elementigem Zyklus, welchem ein k-elementiger linearer Teil vorausgeht. Beschreiben Sie außerdem, wie man n algorithmisch bestimmen kann. **[(3 Teilpunkte)]**

b) Schreiben Sie eine Funktion, die für gegebenes $k \geq 0$ und $n \geq 0$ eine Liste der Länge n erzeugt, den `next`-Zeiger des letzten Elements von `0` auf die Adresse des ersten Elements ändert und somit einen Zyklus der Länge n konstruiert, und schließlich eine Liste der Länge k erzeugt, die auf ein beliebiges Element dieses Zyklus zeigt. Verwenden Sie die Implementierung der einfach verketteten Liste aus der Vorlesung.

Schreiben Sie außerdem eine Funktion, die den Hase-Igel-Algorithmus implementiert und für eine gegebene Liste die Länge n des Zyklus ausgibt. Testen Sie ihre Funktion für die Fälle

- $k > 0$ und $n > 0$ (Normalfall)
- $k = 0$ und $n > 0$ (reiner Zyklus)
- $k > 0$ und $n = 0$ (lineare Liste)
- $k = 0$ und $n = 0$ (leere Liste)

[(3 Teilpunkte)]

(6 Punkte)

Übung 3 *Warteschlangen (Queues)*

a) Programmieren Sie eine Warteschlange für Integer-Zahlen in Form einer einfach verketteten Liste. Diese soll Funktionen bereitstellen, die ein neues Element in die Schlange einreihen bzw. am anderen Ende entfernen und zurück geben. Benutzen Sie dabei *call by reference*, damit die Liste in der `main()`-Funktion durch Ihre Funktionen geändert wird. Sie können die Listenimplementierung aus der Vorlesung als Basis verwenden. **[(2 Teilpunkte)]**

b) Erweitern Sie nun Ihr Programm, indem Sie die Liste durch eine doppelt verkettete Liste ersetzen. In einer solchen Liste hat jedes Element zusätzlich einen Zeiger auf seinen Vorgänger. Zudem enthält die Liste selbst einen Zeiger auf ihr letztes Element. Die doppelt verkettete Liste soll ebenfalls das Einreihen eines Elements am Anfang und das Entfernen am Ende unterstützen. **[(5 Teilpunkte)]**

c) Füllen Sie Ihre beiden Warteschlangen mit einer großen Zahl (bspw. 100000) zufälliger Integer-Zahlen[†] und entnehmen Sie diese danach wieder [‡]. Benutzen Sie dynamische Speicherverwaltung für die Integerzahlen. Was stellen Sie für Laufzeitunterschiede für die Operationen Hinzufügen und Entfernen bei einfach und doppelt verketteter Liste fest? Erklären Sie! Geben Sie die Komplexität dieser Operationen in der Anzahl n der Listeneinträge an. **[(3 Teilpunkte)]**

(10 Punkte)

[†]Sie können dafür mittels `#include<cstdlib>` die Funktion `rand()` benutzten, die passende Zahlen generiert.

[‡]Hiermit ist keine Ausgabe auf dem Bildschirm gemeint. Sie werden feststellen, dass die Konsolenausgabe für eine Datenmenge dieser Größe nicht geeignet ist.