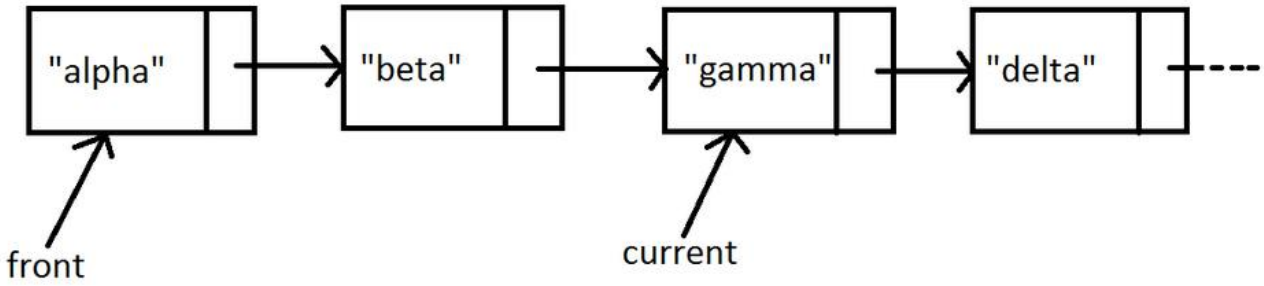
Datenstrukturen: die Liste

Die Datenstrukturen „Stack“ und „Queue“ waren im Prinzip Spezialfälle des etwas allgemeineren Typs „List“. Wie bei Stack und Queue liegt der Struktur auch hier eine Verkettung von Nodes (die ihrerseits die eigentlichen Daten speichern) zugrunde:



In der Abbildung sind vier Nodes zu sehen, die jeweils einen String als Dateninhalt besitzen. Die Datenstruktur „List“ besitzt immer eine Referenz auf das erste Element („front“) und eine veränderliche Arbeitsreferenz („current“), die z.B. hilfreich ist, um sich an den Listenelementen entlangzuhangeln (z.B. um alle Elemente nacheinander auszugeben). Die Nodes selbst besitzen (wie bei Queue und Stack auch) jeweils eine „next“-Referenz, die auf das nächste Listenelemente zeigt. (Hinweis: es gibt unterschiedliche Möglichkeiten, das Listenende zu markieren, welches hier nicht abgebildet ist. Die Abiturklassen benutzen ein besonderes Dummy-Node-Element. Es geht aber auch anders.)

Aufgabe 1: Benutze/Teste die Abitur-List-Klasse mit einem Miniprogramm, welches obige Datenstruktur („alpha“ „beta“ „gamma“ „delta“) anlegt und danach mit Hilfe der current-Referenz, welche alle Elemente durchläuft, wieder ausgibt.

Aufgabe 2: Beschreibe Vor- und Nachteile von Listen gegenüber Arrays:

Aufgabe 3: Untersuche die Methode „insert“ aus der Abitur-List-Klasse. Beschreibe möglichst exakt, was passiert, wenn man in der oben abgebildeten Liste mittels „insert(„omega“) ein weiteres Objekt an der „current“-Position einfügt.

Aufgabe 4: Bei Videospielen gibt es oft eine „High-Score-Liste“. Schreibe eine Klasse „Highscore“, welche nur zwei Methoden kennt:
void insertNewScore(String name, int score);
String toString();
Ein Objekt der Klasse Highscore soll die Scores als Liste verwalten und neue Scores an der richtigen Stelle einfügen. „to String“ versteht sich von selbst.