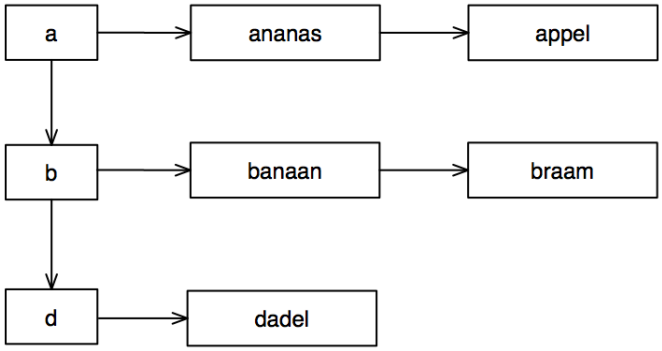
Practicum opdracht

Opdracht 2: Sorteren van iets wat lijkt op een hash table (soort van dubbele gelinkte lijst).

Deze opdracht gaat over gelinkte lijsten, specifieker: het sorteren van iets wat op een hash table lijkt. Een hash table is een data structuur waarin sleutels (keys) gemapt zijn aan data (values). Hoe ziet dat eruit? Bijvoorbeeld een lijst met de volgende fruit soorten:

- ananas
- appel
- banaan
- braam
- dadel

En we kiezen als sleutel de eerste letter:



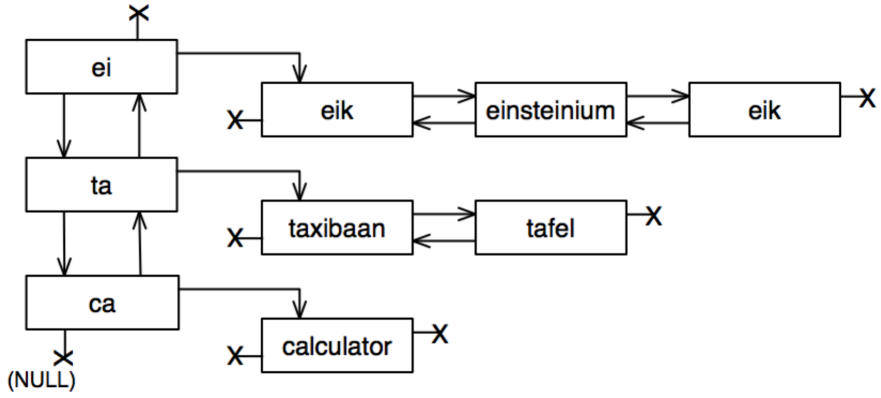
Dan hebben we een gelinkte lijst met keys (a naar b naar d naar NULL) waarbij elke key een pointer heeft naar een eerste data element (ananas naar appel naar NULL, etc).
De structuur is iets complexer dan een normale gelinkte lijst, daar tegenover staat dat als je veel data hebt, je met zo'n structuur véél sneller zaken terug kunt vinden.

Het voorbeeld hierboven is geen *echte* hash table, in een echte hash table gebruik je een hash van je data, ofwel een soort van rekenkundige bewerking over je data heen. Meer informatie over hash tables: http://en.wikipedia.org/wiki/Hash_table.

Om het debuggen van de opdracht minder lastig te maken heb ik er in deze opdracht voor gekozen om geen echte hash te gebruiken, maar een tabel zoals die van het fruit hierboven. Er zijn wel wat verschillen:

1. ik heb niet alleen fruitnamen maar willekeurige woorden uit willekeurige talen,
2. de lijst hierboven heeft alleen unieke woorden, in mijn lijst zitten dubbele woorden,
3. de lijst hierboven is gesorteerd, mijn lijst staat in beide richtingen random door elkaar,
4. als sleutel gebruik ik de eerste 2 karakters van het woord,
5. ik heb véél meer woorden gebruikt.
6. alle elementen hebben zowel een next- als een prev-pointer, het is dus een bidirectionele lijst. Prev in het eerste element is NULL en next in het laatste element is NULL.

De lijst ziet er dus zo uit (maar groter en met meer data):



Practicum opdracht

Een stagiair heeft al wat code geschreven zodat je een snelle start met het project kunt maken:

- Value klasse: deze klasse heeft een woord als data en een 'next' en 'prev' pointer
- Key klasse: deze klasse beheert de lijst met Values en heeft tevens een 'next' en 'prev' pointer naar de volgende en vorige Key
- FileStructure klasse: deze kan data uit een file inlezen en deze in de juiste structuur terug geven en kan de structuur uitlezen en deze in een file opslaan

Er zijn alleen 2 problemen met de genoemde code:

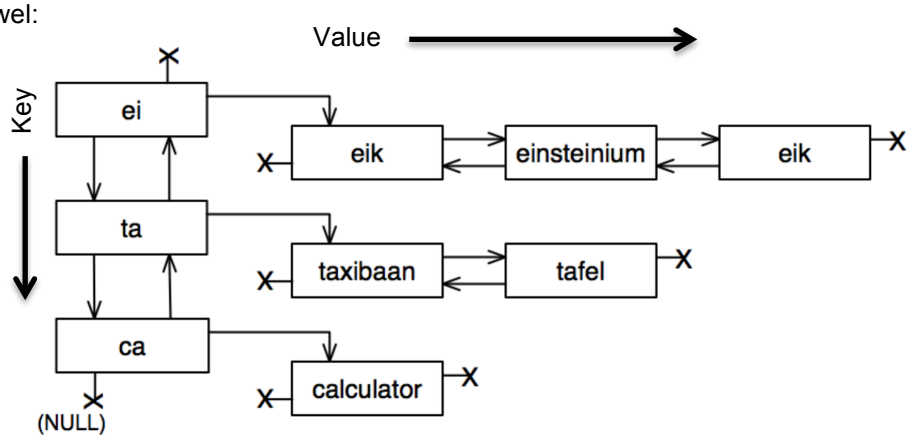
1. Door een ongelukje is de source code verdwenen, er is alleen nog een library file waarin de gecompileerde code staat.
2. De genoemde stagiair is er niet aan toegekomen om de code te optimaliseren, de code is dus niet perse snel.

Stap 1

Download het startproject van Canvas: startPoint4Students.tgz, met: "tar xzf startPoint4Students.tgz" kun je deze file uitpakken. Je krijgt nu een directory structuur die er als volgt uitziet:

- code
  - FileStructure.h
  - Key.h
  - main.cpp
  - Value.h
- data
  - gibberish.bin
- lib
  - 32 en 64 bit libraries met daarin de code van FileStructure, Key en Value
- test
  - *is leeg*
- Makefile
- submit.sh

Zoals je ziet krijg je om te starten 3 klassen, een main file en een binaire blob met gecodeerde data. FileStructure bevat functionaliteit om gibberish.bin uit te lezen en te decoderen. Key bevat functionaliteit om de hash table te beheren, Key maakt vervolgens automatisch een lijst met Values aan.
Ofwel:



Zoals je in main.cpp kunt zien hoef je alleen maar een file in te lezen met FileStructure::loadFile. Hieraan geef je de naam van de gecodeerde, binaire file mee en een pointer naar een lege Key. Alle data wordt vervolgens correct in de genoemde hash table gestopt. Met Key::print kun je de hash table naar stdout printen, je ziet dan de data die in de lijst staat. Uiteindelijk kun je de gesorteerde data weer opslaan in een binaire file.

De eerste stap is: installeer met `make install` de juiste library op jouw systeem, bouw het project met `make sort` en voer het uit (`./sort`). Je ziet nu de lijst op het scherm verschijnen (dit kan

eventjes duren, het zijn er best veel!). Bestudeer de Key en Value klassen en maak een plan om de data te sorteren.

Stap 2

Nu komt het echte werk: schrijf een programma dat de gehele structuur sorteert. Dus alle Keys en Values moeten gesorteerd zijn. Hieraan zitten een paar eisen:

- Het is niet toegestaan om de data in een map, vector, of iets dergelijks te zetten en die het sorteren te laten doen: je moet de data sorteren door in de huidige structuur objecten te verschuiven (effectief: door pointers te veranderen de volgorde aanpassen)
- Je mag nieuwe klassen aanmaken (en dus nieuwe .cpp en .h files). Deze kun je zonder problemen inleveren.
- Zoals al gezegd: de implementatie van Key en Value is niet perse snel. Als je echt een snelle tijd wilt krijgen zul je Key en Value ook moeten implementeren. Hierbij mag je de headerfile aanpassen, zolang je zorgt dat alle methodes die er nu in staan blijven bestaan. In sommige gevallen zal de FileStructure library opnieuw gecompileerd moeten worden, doe dit in overleg met je docent.
- Je kunt FileStructure niet opnieuw maken, hiervan heb je geen specificatie.
- **BELANGRIJK: om in te leveren: voer** `make submit` **uit op de commandline en lever de file in die door dat script gemaakt wordt.**
  Code die niet aan het bovenstaande formaat voldoet, wordt automatisch door mijn controle script afgekeurd! Je doet dan dus niet mee met de wedstrijd en jouw docent gaat er dan vanuit dat jouw code niet aan de eisen voldoet!
- De eerste deadline staat in Canvas ingesteld.

- Er komen in totaal 3 wekelijkse inlevermomenten, wat je voor een voldoende minimaal moet halen bij de laatste inlevering is (daar heb je dus 3 kansen voor!):
  - de code sorteert alle data correct, volgens de eisen die hierboven staan
  - de code heeft geen memory leaks, crasht nergens en heeft geen oneindige lussen
  - de code heeft geen compiler warnings met de volgende compiler instellingen:
    -Wall -Wextra -Werror -pedantic -O3 (staat default zo in de meegeleverde Makefile)

Tip 1: Op je eigen systeem kun je de performance meten met het commando time: `time ./sort`
Dit geeft bijvoorbeeld de volgende output:

```
real  0m0.237s
user  0m0.104s
sys   0m0.128s          (hint: dit zijn geen reële tijden)
```

Tel de onderste twee bij elkaar op voor een zo nauwkeurig mogelijke meting.

Tip 2: Als je met een kleinere structuur wilt testen dan alle data in gibberish.bin, haal dan het Filestructure object tijdelijk weg, maak een Key object aan en voeg daar met de methode addValue een aantal woorden aan toe. Deze woorden moeten een minimale lengte van 2 hebben, anders kan er geen Key gemaakt worden van de eerste 2 letters!
Om de structuur onderaan de eerste pagina te krijgen heb je de volgende code nodig:

```
Key k;
k.addValue("eik");
k.addValue("einsteinium");
k.addValue("eik");
k.addValue("taxibaan");
k.addValue("tafel");
k.addValue("calculator");
```