## Opdracht 2: Sorteren van een hash table (soort van dubbele gelinkte lijst).

Deze opdracht gaat over gelinkte lijsten, specifieker: het sorteren van een hash table. 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:

a

b

d

ananas

banaan

dadel

appel

braam

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 gemakkelijker 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 geen fruitnamen maar willekeurige combinaties van woorden uit willekeurige talen,
2. de lijst hierboven heeft alleen unieke woorden, in mijn lijst zitten dubbele woordcombinaties,
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.

### Deel 1

Download het startproject van Canvas: startPoint4Students.tgz (of de 64 bit variant als je op een 64 bits systeem werkt!), 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
* objects
  + FileStructure.o
  + Key.o
  + Value.o
* test
  + *is leeg*
* Makefile

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.

Value

Ofwel:

Key

a

b

d

ananas

banaan

dadel

appel

braam

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: bouw het project en voer het uit. 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.

### Deel 2

Schrijf een programma dat de data in beide richtingen sorteert. Dus alle Keys én alle Values in alle Keys moeten gesorteerd zijn. Hieraan zitten een paar eisen:

* Voor jouw eerste oplevering mag je alleen een bubblesort of quicksort algoritme implementeren (je gaat dit programma vaker inleveren, met als doel om een betere performance te halen)  
  *(even expliciet: als je deze eerste deadline mist dan is dit geen ramp, maar dan is de tweede keer jouw eerste oplevering!)*
* Het is niet toegestaan om de data in een map, vector, etc. te zetten en die het sorteren te laten doen.
* Je mag nieuwe klassen aanmaken (en dus nieuwe .cpp en .h files). Deze kun je zonder problemen in Peach inleveren.
* Je mag in Peach alleen jouw code en header files inleveren, dus geen makefiles en zo. Ook testcode kan niet ingeleverd worden.
* De eerste deadline is zondagavond 29 november om 21u.
* Er komen in totaal 3 wekelijkse inlevermomenten, wat je 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 -Werror -pedantic -O3

Tip: 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.