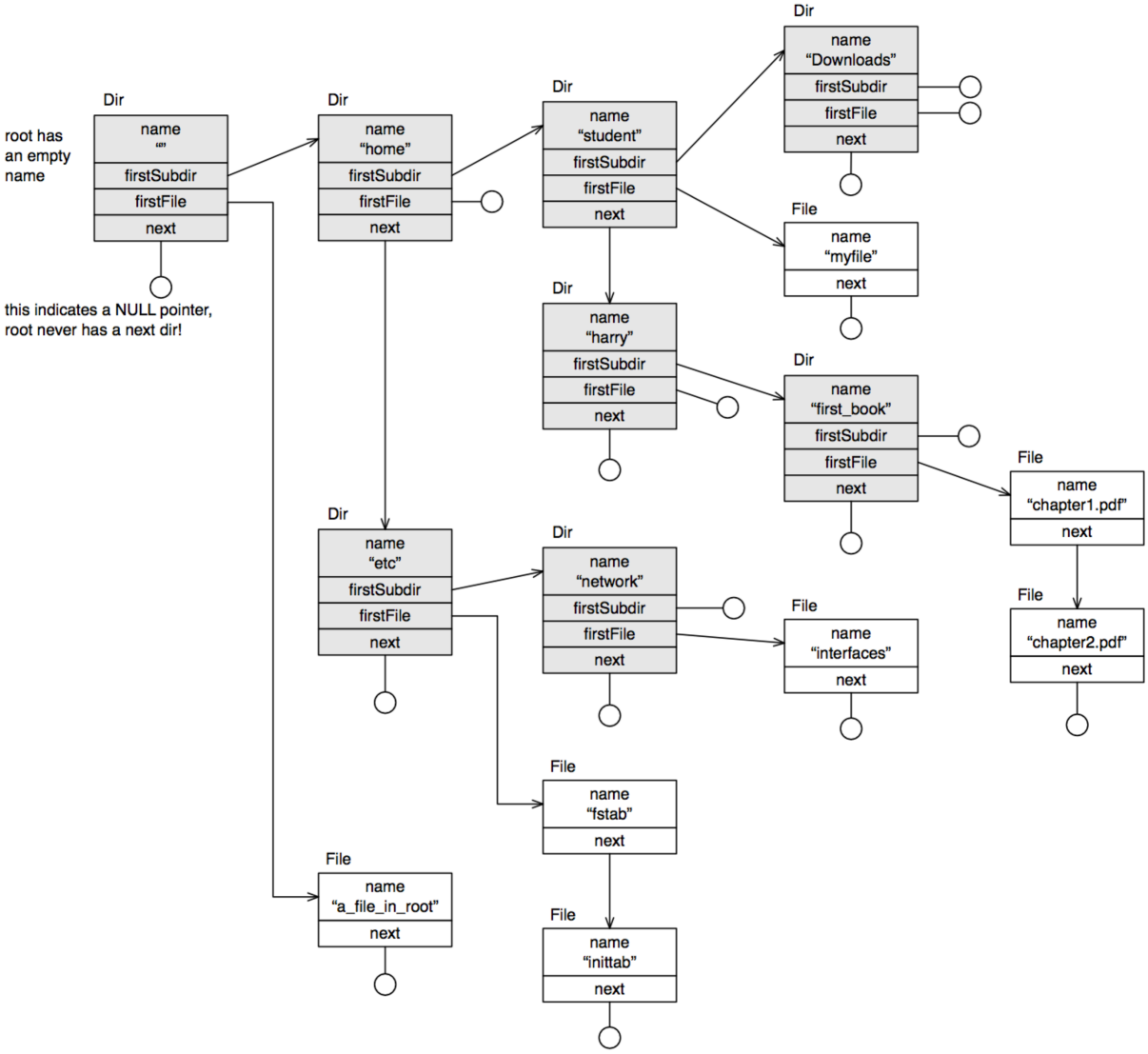
Practicum opdracht

Opdracht 2: Sorteren van een directory structuur (driedubbel gelinkte lijst).

Deze opdracht gaat over gelinkte lijsten, specifieker: het sorteren daarvan. De gekozen structuur is dit keer een directory structuur, deze kun je als volgt maken:



In deze structuur is Dir steeds de root van een driedubbel gelinkte lijst:

- firstSubdir: de eerste directory in deze directory, welke dus 1 niveau dieper ligt: directory "student" ligt 1 niveau dieper dan directory "home".
- firstFile: de eerste file in deze directory
- next: de volgende directory op het zelfde niveau: home ligt dus op het zelfde niveau als etc.

File is een enkel gelinkte lijst: deze heeft alleen een pointer naar de volgende file, welke altijd in dezelfde directory ligt.

In deze structuur hebben we de volgende directories en files:

```
/a_file_in_root
/home/student/Downloads/
/home/student/myfile
/home/harry/first_book/chapter1.pdf
/home/harry/first_book/chapter2.pdf
/etc/network/interfaces
/etc/fstab
/etc/inittab
```

Hierin is "`/home/student/Downloads/`" een lege directory.

Practicum opdracht

Deel 1

Download het startproject van Canvas: startPoint4Students32.tgz (of de 64 bit variant als je op een 64 bits systeem werkt!), met: "tar xzf startPoint4Students32.tgz" kun je deze file uitpakken. Je krijgt nu een directory structuur die er als volgt uitziet:

- code
  - Dir.h
  - File.h
  - FileStructure.h
  - main.cpp
- data
  - gibberish.bin
- objects
  - Dir.o
  - File.o
  - FileStructure.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. Dir bevat functionaliteit om de directory structuur te beheren, Dir maakt vervolgens automatisch een lijst met 'next' dirs (type Dir), 'subdirs' (type Dir) en 'files' (type File) aan.

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 referentie naar een lege Dir. Alle data wordt vervolgens correct in de genoemde Dir structuur gestopt. Met Dir::print kun je de structuur naar stdout printen, je ziet dan de data die in de lijst staat. Uiteindelijk kun je met FileStructure::saveFile 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 Dir en File klassen en maak een plan om de data te sorteren.

Deel 2

Schrijf een programma dat de gehele structuur sorteert. Dus alle dirs, subdirs en files moeten gesorteerd zijn (maar niet de inhoud van de file). Hieraan zitten een paar eisen:

- Het is niet toegestaan om de data in een map, vector, etc. te zetten en die het sorteren te laten doen: je moet zelf sorteren.
- 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 staat in Peach ingesteld.

- Er komen in totaal 2 wekelijkse inlevermomenten, wat je minimaal moet halen bij de laatste inlevering is (daar heb je dus 2 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 (staat default zo in de meegeleverde Makefile)

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.