PG3400 Hjemmeksamen 2017

En “key-value” database er i utgangspunktet en enkel liste av unike nøkler med tilhørende dataverdier. Til forskjell fra relasjons-databaser, gir KV-databaser ingen mulighet for å lage relasjoner mellom tabeller siden den faktisk bare er en tabell. Det finnes heller ikke noe generellt spørrespråk som SQL, men det kan finnes språk som brukes til å beskrive hva slags data basen kan inneholde. Blant de meste kjente implementasjonene kan nevnes Dynamo, NoSQL og Apache Cassandra. Sistnevnte ble opprinnelig laget av Facebook for å inneholde søkedata for meldings-innboksene og er et system med automatisk utveksling av data (replikering) mellom et utall av databasetjenere. Det er derfor Facebook er såpass responsiv uansett hvor du er i verden. Du kan lese mer her:

> http://database.guide/what-is-a-key-value-database/
> https://en.wikipedia.org/wiki/Key-value_database
> https://en.wikipedia.org/wiki/Apache_Cassandra

Vi har ikke tenkt å ta dette så langt som Facebook, men implementere en enkel KV-database som oppbevarer dataene i en tre-struktur i minnet.

DEL 1:

Ta en titt på følgende nøkkelverdier med tilhørende data:

```
strings.no.header = "Oppdatering"
strings.no.text = "Oppdater programvaren"
strings.no.button_cancel = "Avbryt"
strings.en.header = "Updating"
strings.en.text = "Update your software"
strings.en.button_ok = "Ok"
strings.en.button_cancel = "Cancel"
config.loglevel = 1
config.update.interval = 32
config.update.server1 = "http://www.aspenberg.no/"
config.update.timeout = 20
```

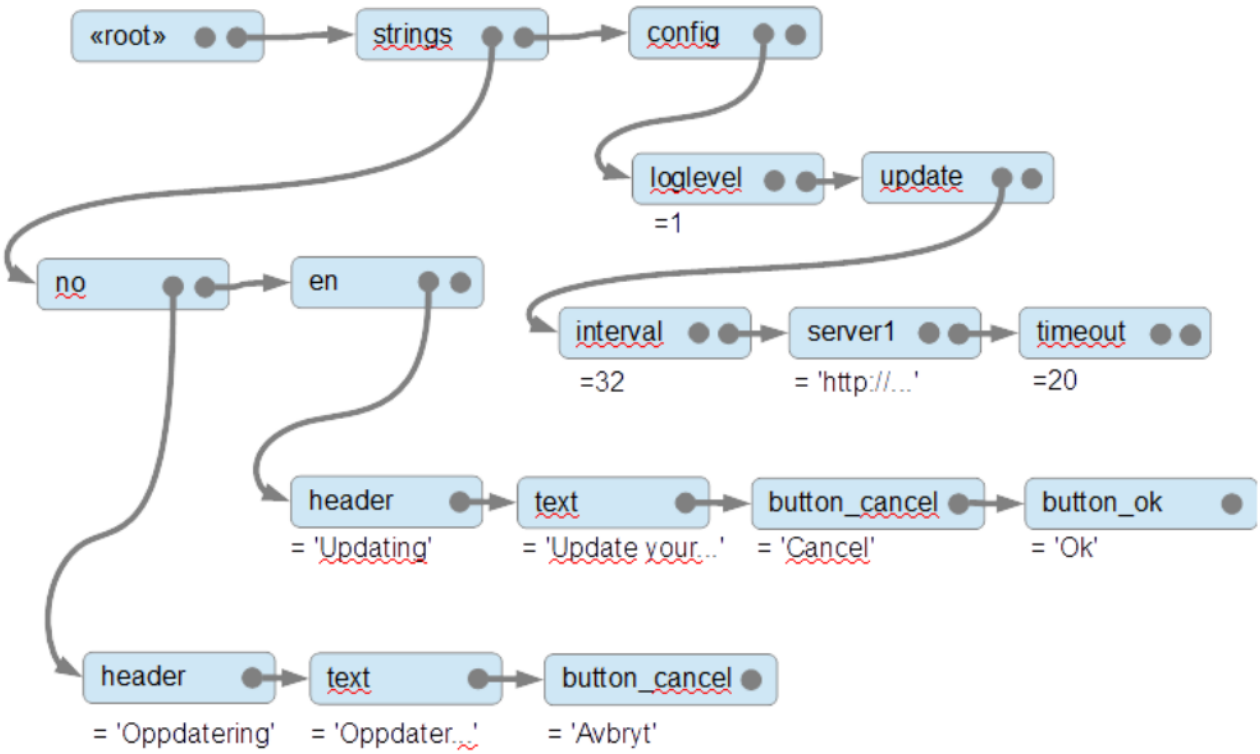
Legg merke til at nøkkelnavnene har en slags tre-struktur hvor punktum «.» markerer nivåer i treet. Det finnes kun to typer data-verdier, enten heltall eller tekst (streng). Legg også merke til at en node kan inneholde en verdi *og* et underliggende nivå. For eksempel er *config.loglevel* en *verdi* mens *config.update* er et undeliggende nivå (vi kaller det en *mappe*). En node vil alltid være *enten* en verdi *eller* en mappe.

Anta at teksten er lagret i en tekstfil. Skriv kode som leser fila, linje for linje, tolker innholdet og bygger opp treet i minnet. Når du leser inn teksten fra fila må du passe på at det kan være et varierende antall mellomrom mellom nøkkelnavnet, likhetstegnet og selve verdien. Tekstverdier er omsluttet av doble gåseøyne, mens tallverdier ikke er det.

Treet kan enten lages som en todimensjonal linket liste, eller ha en B-tre-liknende struktur. De neste avsnittene diskuterer hvordan dette kan gjøres.

Alternativ A:

En todimensjonal lenket liste som kan visualiseres som:



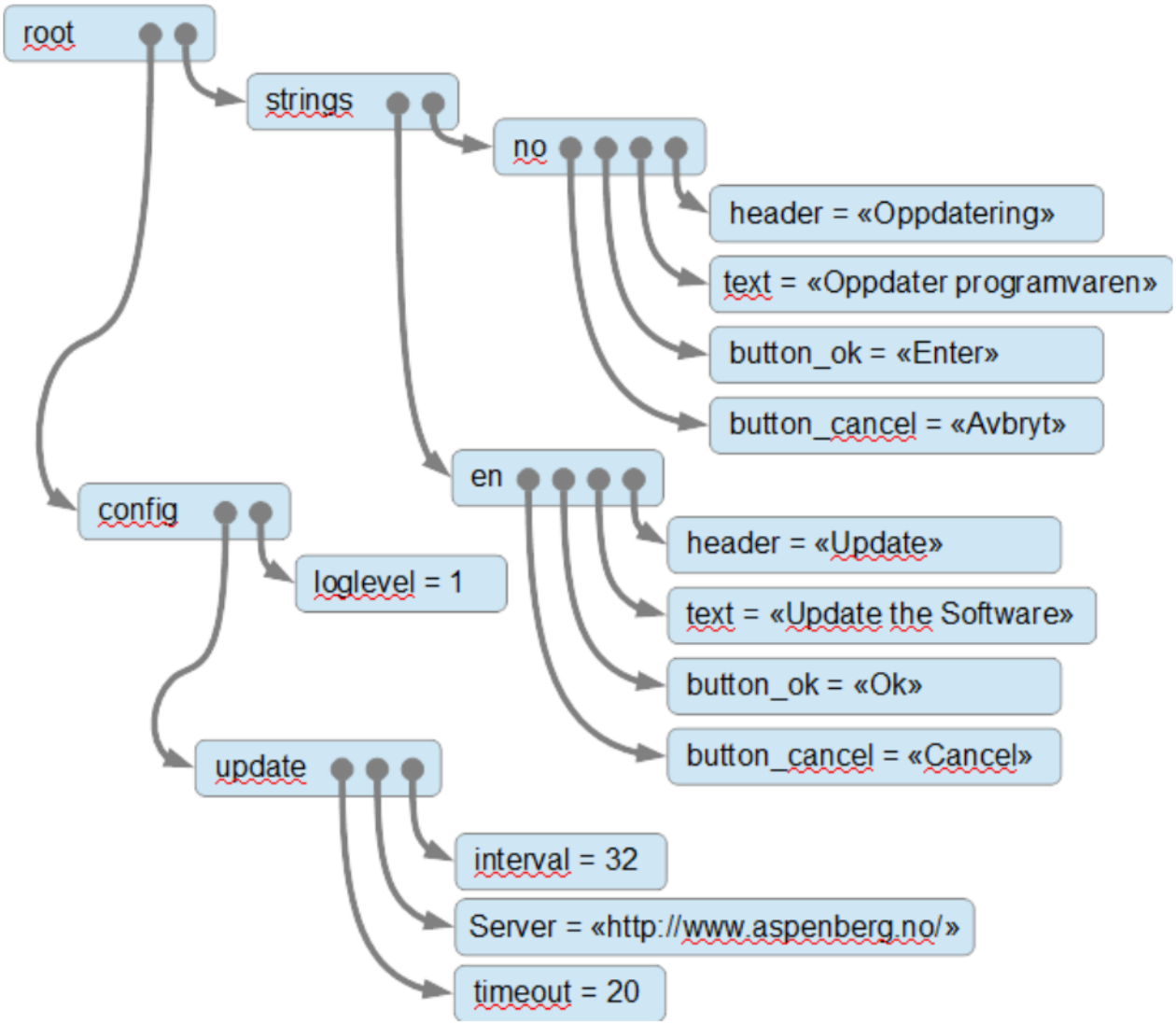
Et eksempel på en struct som kan brukes til å lage noe slikt er:

```
typedef unsigned long ULONG;

typedef struct _NODE {
   char *pszName;    // Navnet på denne noden.
   ULONG ulIntVal;   // Hvis numerisk så er dette verdien.
   char *pszString; // Peker til streng eller NULL hvis numerisk.
   struct _NODE *pNext;  // Neste node på samme nivå.
   struct _NODE *pDown;  // Noden under denne.
} NODE;
```

Alternativ B:

En annen måte å gjøre dette på er å konstruere et B-tre som kan se slik ut:



Hver node har et antall pekere til nodene under seg. Hver node i en slik struktur kan for eksempel defineres slik:

```
#define MAX_NODES 10

typedef struct _NODE {
   char *pszName;    // Navnet på denne noden.
   ULONG ulIntVal;   // Hvis numerisk så er dette verdien.
   char *pszString; // Peker til streng eller NULL hvis numerisk.
   struct _NODE *pnNodes[MAX_NODES]; // Pekere til under-noder.
} NODE;
```

Slik som det er definert her, vil hver node maksimalt kunne ha MAX_NODES (10) under-noder.

Alternativ C:

En mer fleksibel måte å gjøre dette på er å tillate et variabelt antall under-noder per node. Dette kan for eksempel defineres på denne måten:

```
typedef struct _NODE {
   char *pszName;    // Navnet på denne noden.
   ULONG ulIntVal;   // Hvis numerisk så er dette verdien.
   char *pszString; // Peker til streng eller NULL hvis numerisk.
   int iNodes;       // Antall noder under denne. 0 for ingen.
   struct _NODE *pnNodes[]; // Pekere til under-noder.
} NODE;
```

Denne struct'en kan holde et dynamisk antall under-noder i den åpne peker-tabellen pnNodes[]. iNodes inneholder antall noder akkurat nå. Denne tabellen må da selvfølgelig allokeres dynamisk og utøkes eller krympes etter hvert som antall noder endrer seg.

Velg en av de tre datastukturene når du skal implementere trestrukturen, enten lenket liste (akternativ **A**), eller en av B-tre-strukturene (alternativ **B** eller **C**). Lenket liste er det enkleste å implementere. Et B-tre med et fiksert antall under-noder noe vanskeligere, og et B-tre med et dynamisk antall under-noder regnes som vanskeligst.

Et forslag til tilnærming er å skrive en funksjon **SetInt()** og en funksjon **SetString()** som legger noder til treet av den gitte typen og så lese og tolke tekst-fila for å kalle den riktige Set-funksjonen avhenging om du finner en numerisk eller en tekst-verdi. Hvis en av Set-funksjonene finner ut at en node allerede finnes, og er av rett type (numerisk eller tekst), skal eksisterende verdi overskrives med den nye verdien. Med andre ord skal Set-funksjonene enten oppdatere en eksisterende verdi eller legge til en ny. Funksjonene bør returner en feilkode dersom den prøver å oppdatere en eksisterende verdi av feil type.

Ekstra: Hold treet sortert (alfabetisk på nøkkel-navn) for hver node som settes inn. Dette kan også brukes til å optimalisere søking etter verdier senere.

DEL 2:

Definer konstanter som kan brukes for å identifisere de to typene. Skriv deretter en funksjon **GetType()** som tar et nøkkel-navn (f.eks «config.update.interval») som parameter og returnerer hva slags node dette er, numerisk eller tekst, eller en feilkode dersom noden ikke finnes. Lag så **GetInt()** og **GetString()** som tar et nøkkel-navn som parameter og returnere verdien dersom den finnes, eller en av to feilkoder; feil type eller finnes ikke.

Ekstra: Lag funksjone **GetValue()** og **SetValue()** som kombinerer Set- og Get- funksjons-parene til to funksjoner. Hvordan kan dette gjøres? (Hint: Funksjoner med variablel parameterliste).

DEL 3:

Lag en funksjon **Enumerate()** som tar deler av et nøkkelnavn, f.eks. *"config.update.*"* og enumererer alle noder som inneholder *verdier* på det angitte nivået i treet. Funksjonen skal hoppe over foldere. I det gitte eksemplet, vil funksjonen liste ut *interval*, *server1* og *timeout*. For hver node som finnes skal det kalles en call-back-funksjon som har to parametre; nøkkel-navnet og verdien. Husk at det er to typer verdier, og det må håndteres i call-back-funksjonen.

DEL 4:

Lag en funksjon **Delete()** som tar et nøkkelnavn som parameter og sletter noden dersom den finnes. Hvis det er en folder som slettes må alt som ligger i folderen, dvs. alle underliggende noder slettes. Hvis slettingen (av en folder eller en verdi) resulterer i en tom folder på dette nivået, skal folderen slettes. Derfor må du også ta hensyn til at resultatet av slettingen på dette nivået kan påvirke nivået over dette, og over dette, videre oppover helt opp til roten av treet.

DEL 5:

Lag en funksjon **GetText()** som tar et streng-navn som første parameter og ønsket språk som andre parameter. Fuksjonen skal returnere strengen. F.eks. vil **GetText("button_cancel","no")** returnere "Avbryt". Hvis strengen ikke finnes, skal funksjonen se på samme sted i «en»-grenen i treet og returnere det som finnes der. For eksempel vil funksjonen **GetText("button_ok","no")** returnere «Ok» siden teksten ikke finnes i «no»-grenen men i «en» grenen. Funksjonen skal returnere NULL dersom strengen ikke finnes i det hele tatt.

SLUTTORD OG TIPS

Siden datastrukturen i treet kan sees på som en rekursiv struktur vil faktisk noe av koden bli enklere dersom du skriver den på en rekursiv måte. All kode som manipulerer treet skal være i en egen kildefil (c-fil). Hovedprogrammet skal kun inkludere en header-fil (h-fil) og kalle de nødvendige funksjonene for først å lese og tolke tekst-filen, deretter teste de ulike Set, Get, Delete og Enumerate-funksjonene. Legg ved en **makefile** som bygger det hele og en enkel forklaring på hvordan programmet skal kjøres med forventede resultater.

Hvis du ikke får til alt, sørg i hvert fall for at *strukturen* i programmet er fornuftig. Legg heller inn funksjoner uten innhold for det du ikke har blitt ferdig med slik at strukturen i programmet er tydelig. Legg til kommentarer dersom koden ikke forklarer seg selv.

Pakk det hele inn i en ZIP-pakke. Vær nøyaktig og sørg for at programmet bygger og lar seg kjøre. Test ZIP-pakka på en annen maskin, eller i hvertfall i en annen *katalog* slik at du ser at du har fått med deg alle de nødvendig filene og ingenting annet *før* du leverer på ItsLearning.