Oblig 2 - INF1060 - høst 2016

I denne oppgaven skal du bruke det du har lært til nå og gjort i oblig 1, og kombinere det med filbehandling og kall til Linux-systembiblioteket. Vi skal med andre ord jobbe med filer, structer, strenger, pekere og systemkall.

Oppgavene skal løses selvstendig, se ellers Obligreglementet. Du vil finne nyttige ukesoppgaver med tilhørende forklaringer av viktige begreper på gruppelærersiden, og som i de fleste fag er semestersiden stedet for forelesningsfoiler og annen informasjon du kanskje er på utkikk etter.

Dersom du har spørsmål underveis kan du oppsøke en orakeltime eller stille spørsmål på Piazza. Det vil (selvfølgelig) være mye relevant informasjon i plenumstime.

Husk å lese hele oppgaveteksten, så du vet hvor mye arbeid du har igjen totalt sett. *Du vil til slutt i dokumentet finne en liste som viser hva du bør prioritere når du jobber med oppgaven.*

Oppgavene blir testet på Linux på Ifi sine maskiner eller tilsvarende.

Innlevering i Devilry innen 6. oktober 23:59

Oppgaven

Vi tenker oss et scenario der IFI-Drift ønsker et "bedre"system for å administrere ruterne de har i drift ved Ole Johan Dahls og Kristen Nygaards hus. Drift vil ha et program som skal bli brukt til å lagre og behandle grunnleggende informasjon om ruterne. Programmet skal kjøres fra en terminal på et Linux-system, under kjøring skal bruker kunne se på og endre data om ruterne, og mellom kjøringer skal dataene være lagret i en fil.

Programmet skal kompileres med make (**dere må altså lage en makefile**), og startes med et filnavn som eneste argument. Filnavnet skal da være navnet på en fil som inneholder data om rutere som kan leses og behandles av programmet. For eksempel kan man starte på denne måten:

```
$> ./ruterdrift 1router.dat
```

Du finner flere data-filer du kan teste programmet ditt med i Oblig2-repositoriet på github

Filstruktur

Filen inneholder all data som programmet trenger. Filen har ikke (eksklusivt) tekst-data og kan dermed *ikke* inspiseres i standard teksteditorer som Atom/Notepad etc. Dette er ment for å gjøre det enklere å skrive kode for å lese inn talldata, da man ikke trenger å tenke på å gjøre om mellom tall i tekstform og faktiske tall i minnet.

Det første som ligger i filen er en `int`, et 4-byte tall, som forteller oss hvor mange rutere filen har informasjon om. Informasjon om hver ruter er avgrenset med linjeskift, altså er hver ruter en "linje"med informasjon. En gyldig fil uten ruterdata inneholder derfor én linje med kun tallet 0. *Det er viktig å merke seg at denne første linjen er en 4-byte int-verdi, og ikke et tall på leselig form*! Resten av linjene er maksimalt 256 bytes lange (denne restriksjonen skal opprettholdes når man skriver ny data til fil). Se slutten av dokumentet for hint angående håndtering av binærdata i filer.

Hver av de etterfølgende linjene i filen representerer én ruter på følgende form:

- Ruter-ID (unik) - `unsigned char`
- FLAGG - `char`
- Lengde på produsent/modell-strengen - `unsigned char`
- Produsent/modell - `char[]` (maks lengde 253)

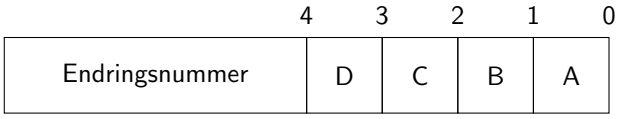
Hver linje i filen inneholder altså 3 bytes med info om ruteren, etterfulgt av produsent/modell. Innad i programmet kan dere bruke en `struct` med disse akkurat lik definisjon som over.
Merk at siste felt i structen er et char-array, og dette kan åpne opp for noen spennende muligheter for spesielt interesserte, se seksjonen "Flexible array member".

I hver struct skal det være en char, som i listen over heter FLAGG, og den skal representere diverse egenskaper en ruter kan ha. Flagg-feltet er forsøkt forklart i tabell 1 og figur 1.

Bit-posisjon	Egenskap	Forklaring	Bokstav (for figur 1)
0	Aktiv	Er ruteren i bruk?	A
1	Trådløs	Er ruteren trådløs?	B
2	5GHz	Støtter ruteren 5GHz?	C
3	*Ubrukt*	Ikke i bruk	D
4:7	Endringssnummer	Se lenger nede for info	

Tabell 1: De ulike bitsene i flagg-feltet

Obs: merk at flagg-feltet potensielt kan ha verdien $'\backslash n'$, som vil terminere et kall på `fgets()` dersom dette er brukt for å lese linjen fra filen.

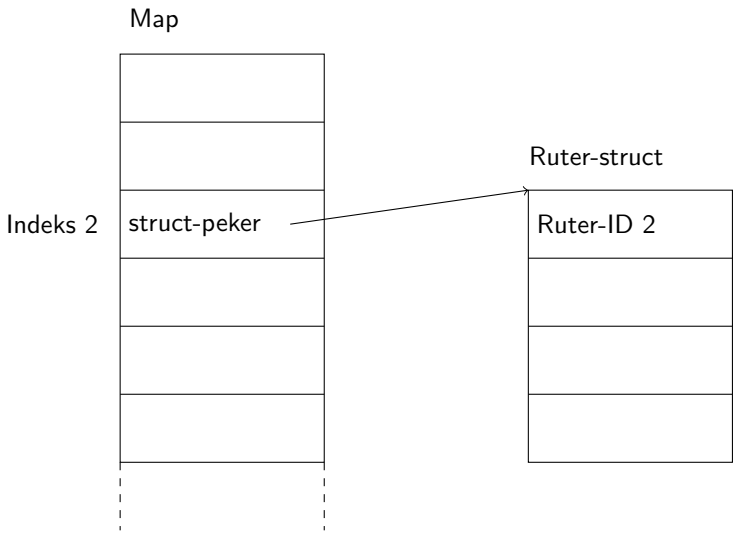


Figur 1: Flagg-feltet

Spesifikasjoner

Her er litt mer spesifikk informasjon om hva en bruker skal kunne bruke programmet til, og hvordan programmet skal fungere. Grovt sett vil programmet være delt i tre:

1. Les inn filen og opprett de datastrukturer du trenger.
2. Ordrestyrt program ("ordreløkke") der brukeren gir kommando og programmet utfører brukerens ønsker.
3. Avslutning og skriving til fil.



Figur 2: Golbalt map for rutere

Innlesing og datastrukturer

Dette skal skje med en gang programmet starter. Filen som er gitt som argument til programmet skal leses inn og dataen skal lagres i minne. For hver linje i filen skal det allokeres plass til en struct (med `malloc`), og structen skal fylles med data fra linjen. En peker til structen skal lagres i et globalt, statisk allokert (ikke `malloc`) array som vi skal bruke som et map. Mappet skal bruke ruter-ID som nøkkel og peker til ruter-strukten for den IDen som verdi, det hele er forsøkt illustrert i figur 2. Når alle linjene har blitt lest inn, fått sin egen struct og sin egen entry i det globale mappet går programemt videre til ordreløkken.

Ordreløkke

Programmet skal kjøre i loop, og gi brukeren mulighet til å gjøre følgende operasjoner:

- Printe info om ruter gitt en ID
- Endre flagg for en ID
- Endre produsent/modell for en ID
- Legge inn en ny ruter (bruker putter inn ID og annen data)
- Slette en ruter fra databasen
- Avslutte programmet

Det er opp til deg hvordan du velger å løse ordreløkke-biten av oppgaven. Du kan selv velge om du vil ha en kommando-basert interaksjon med brukeren (et shell), eller om du vil printe en meny og la brukeren velge menyvalg, f.eks. ved tall. Uansett hvordan du velger å løse oppgaven bør det være intuitivt og enkelt å bruke programmet.

Det bør være intuitivt hvordan dataene i filen kan endres, for eksempel endring av navn og sletting av rutere. Det eneste som kanskje ikke er helt logisk er endringsnummeret i flagg-charen. Dette er et

4-bits tall som skal økes med 1 hver gang det gjøres en endring på ruteren. Dersom tallet er 15 (binær 1111) og brukeren forsøker å gjøre en endring, skal programmet nekte å gjennomføre endringen.

Avlutning

Når brukeren velger å avslutte programmet skal alle allokerte minneområder (allokert med `malloc`) frigis ved kall på `free`, data skal skrives til den samme filen som den ble lest fra (overskrive hele filen), og filen skal lukkes.

Ekstraoppgave: Sett opp en signal-handler som gjør at brukeren også kan trykke `Ctrl-C` for å terminere programmet, uten at dette fører til minnelekasjer eller tap av data.

Viktighet av de forskjellige funksjonalitetene

Her er en liste over viktigheten av de forskjellige delene av oppgaven. Bruk listen som en guide for hva du bør jobbe med (og i hvilken rekkefølge). Dette er med tanke på hva som blir viktig mot hjemmeeksamen og hva som er det sentrale ved oppgaven. Dette vil bli brukt ved retting.

1. Innlesing av data fra fil
2. Oppretting av map-strukturen
3. Riktig innsetting av data i mappet
4. God bruk av minne (`malloc` og `free`)
5. Endring av FLAGG-charen i en ruter-struct.
6. Skriving av data til fil
7. Fungerende brukerinteraksjon (ordreløkken)
8. De resterende funksjonene (oprette, slette) er omtrent like viktige

Med andre ord: **sørg for at innlesing og opretting av datastrukturen fungerer først, så sørg for å skrive til fil riktig. Pass på minnebruk hele veien.** Først når dette fungerer bør du begynne å se på de andre funksjonene i programmet.

Flexible array member (ekstraoppgave)

De som vil ha litt ekstra moro og læring kan lese seg opp om det som kalles "flexible array member"i en strukt i C. Kort fortalt går det ut på at man kan ha en array uten lengde som siste element i en struct, og så allokere plass til structen + så mange array elementer man trenger, og deretter bruke array-feltet i structen til å indeksere utover den oprinnelige størrelsen til structen. Resultatet av å bruke dette er at vi allokerer ikke mer plass enn vi trenger til hver struct, og vi slipper å allokere to ganger ved å bruke en peker til et nytt allokert område.

Dette er en helt frivillig del av oppgaven.

Levering

1. Lag en mappe med ditt brukernavn: `mkdir bnavn`
2. Kopier alle filene som er en del av innleveringen inn i mappen:
   `cp *.c bnavn/` (f.eks.)
3. Komprimer og pakk inn mappen:
   `tar -czvf bnavn.tgz bnavn/`
4. Logg inn på Devilry
5. Lever under INF1060 Oblig X

Relevante man-sider

Disse man-sidene inneholder informasjon om funksjoner som kan være relevante for løsing av denne oppgaven. Merk at flere av man-sidene inneholder informasjon om flere funksjoner på én side, som `malloc/calloc/realloc`.

- `malloc/calloc/realloc`
- `fgetc`
- `fread/fwrite`
- `fopen/fclose`
- `scanf/fscaf`
- `strcpy`
- `memcpy`
- `memmove`
- `atoi`
- `strtol`
- `isspace/isdigit/alnum` m.fl.
- `strdup`
- `strlen`
- `printf/fprintf`

Andre hint

Da ruterdataen ikke kan leses som vanlig tekst er det lurt å ha en annen måte å inspisere filen(e) på. Til dette finnes blant annet terminalverktøyet `xxd`. Dette viser filen som hexadesimale tall ved siden av en tekstlig representasjon. Man kan da for eksempel enkelt se at de første fire bytene inneholder tallverdien som sier hvor mange rutere det er i filen.

```
00000000: 0500 0000 0a00 910c 4e65 7467 6561 7220  ........Netgear
00000010: 7433 760a 0101 095a 7958 454c 2073 320a  t3v....ZyXEL s2.
00000020: 02b7 0c43 6973 636f 2064 3830 3030 0a03  ...Cisco d8000..
00000030: 900b 5a79 5845 4c20 6d39 3078 0a04 8309  ..ZyXEL m90x....
00000040: 5a79 5845 4c20 6338 0a                   ZyXEL c8.
```

Figur 3: Eksempeloutput fra `xxd`

Figur 3 viser `xxd`-output fra filen `5router.dat`. Hver hexadesimale siffer utgjør fire siffer i binært, altså trengs 2 hex-sifre for å representere én byte. De markerte rutene inneholder følgende data:

- `0500 0000` - antall rutere i denne filen[1].
- `0a` - karakteren `\n`, linjeskift. OBS: denne kan forekomme ved uhell i flagg-feltet; det er ikke trygt å lese filen med `fgets`!
- `00` - ruter-IDen for denne ruteren.
- `91` - ruterens flagg. På binærform er dette `10010001`: Ifølge tabell 1 betyr dette at ruteren er aktiv (bit på posisjon 0 er lik 1; vi sier at biten er *satt*), og at den har blitt endret 9 ganger.
- `0c` - lengden på strengen som inneholder produsent- og modellnavn. `0c` er 12 i desimal. Merk at linjeskiftet er medregnet.
- `4e65 7467 6561 7220 7433 76` - strengen "Netgear t3v".

[1] De spesielt interesserte kan lese om hvorfor den laveste byten er lagret først: Wikipedia: Endianness. Dette er ikke nødvendig kunnskap for å løse oppgaven.