Oblig – IN2140 – Vår 2019

I denne oppgaven skal du bruke det du har lært til nå og skrive et større program. Vi skal jobbe med filer, structer, strenger, pekere og systemkall.

Oppgavene skal løses selvstendig, se reglene for obligatoriske oppgaver på Obligreglementet. Dersom du har spørsmål underveis kan du oppsøke en orakeltime/gruppetime eller stille spørsmål på Piazza.

Husk å lese hele oppgaveteksten, så du får et helhetsinntrykk av omfanget og hvilke utfordringer du kan møte. *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. Programmene deres **MÅ** kunne kompileres og kjøres på ifi sine login-maskiner (login.ifi.uio.no).

Innlevering i Devilry innen 1. mars 15:00 Merk innleveringstidspunktet.

1 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 ta imot to filer. En fil som beskriver ruterne og kobliger mellom dem. Og en annen fil med kommandoer som skal utføres. Hvordan disse filene ser ut kommer vi til i seksjon 2.

Programmet skal kompileres med make (**dere må altså lage en makefile**), og startes med to filnavn som argumenter. Det første filnavnet skal være navnet på en fil som inneholder data om rutere som kan leses og behandles av programmet. Den andre filen inneholder en eller flere linjer med kommandoer som skal utføres. For eksempel kan man starte på denne måten:

```
$> ./ruterdrift 50_routers_150_edges kommandoer_50_rutere.txt
```

Du kan anta at programmet kjøres med rett antall argumenter og gyldige filer.

Som en del av oppgaven utleveres følgende filer:

oblig.pdf	denne oppgaveteksten
5_routers_fully_connected	en informasjonsfil for 5 rutere
5_routers_fully_connected.png	en figur som illustrerer topologien i 5_routers_fully_connected
10_routers_10_edges	en informasjonsfil for 10 rutere
10_routers_10_edges.png	en figur som illustrerer topologien i 10_routers_10_edges
50_routers_150_edges	en informasjonsfil for 50 rutere
50_routers_150_edges.png	en figur som illustrerer topologien i 50_routers_150_edges
kommandoer_10_routers.txt	en kommandofil tilpasset 10_routers_10_edges
kommandoer_50_rutere.txt	en kommandofil tilpasset 50_routers_150_edges

2 Filstrukturer

I denne seksjonen spesifiserer vi hvordan de to filene som programmet skal lese ser ut.

2.1 Informasjon om rutere (fil 1)

Denne filen inneholder all data som programmet trenger. Filen har ikke (eksklusivt) tekst-data og kan dermed *ikke* inspiseres i standard tekstediterer 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. Kall dette tallet N. 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. Vi kommer kun til å jobbe med filer som har ruterdata. *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.
De neste N linjene inneholder representasjoner av hver ruter på følgende form:

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

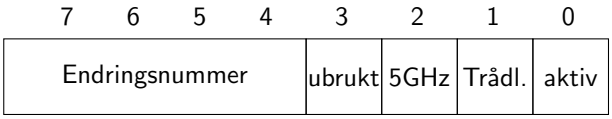
Hver linje i filen inneholder altså 3 bytes med info om ruteren, etterfulgt av produsent/modell. Hver linje skal lagres i en `struct`, som naturlig nok må ha feltene id, flag og modell (dere kan selvfølgelig kalle dem noe annet). Dere burde utvide `structen` etter behov for å løse oppgaven.

I hver struct skal det være en unsigned 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
0	Aktiv	Er ruteren i bruk?
1	Trådløs	Er ruteren trådløs?
2	5GHz	Støtter ruteren 5GHz?
3	*Ubrukt*	Ikke i bruk
4:7	Endringssnummer	Se lenger nede for info

Tabell 1: De ulike bitsene i flagg-feltet

Obs: merk at feltene ruter-ID, flag og lengde 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

Etter de N linjene med informasjon on ruterne følger et ukjent antall linjer med koblinger mellom ruterne. Hver linje inneholder to router-IDer, og linjene er skilt med linjeskift. ID-ene er lagret i binært-format, og er av typen unsigned char. En koblig mellom router R1 og R2 betyr at R1 skal ha en peker til R2. Disse koblingene er en-veis. Så det følger ikke at R2 $\rightarrow$ R1 når R1 $\rightarrow$ R2. Merk at R2 $\rightarrow$ R1 også kan forekomme i filen slik at man får en to-veis kobling.

Dere kan anta at hver node har maksimalt 10 koblinger, og det kan allokeres minne til 10 pekere for hver ruter uansett faktisk antall. Dere kan ha en array av pekere i structen med informasjon om en ruter.

Alle disse koblingene definerer en rettet graf som dere skal traversere med et graf-søk. Detaljer rund graf-søket kommer i seksjon 2.2.

2.2 Kommandoer (fil 2)

Denne filen inneholder en rekke kommandoer. Hver linje inneholder en kommando. De forskjellige kommandoene har forskjellig format. Her følger en beskrivelse av kommandoene.

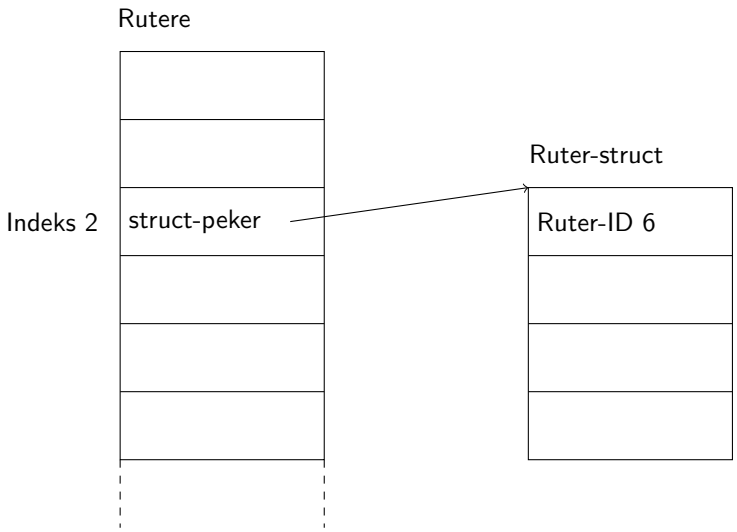
- **print ruter-ID** : Skriver ut til stdout informasjon on ruteren og ID-ene til dens naboer, nærmere bestemt bare de naboene som ruteren peker til.
- **sett_flag ruter-ID flag verdi** : Setter flag $\in 0, 1, 2, 4$ til value $\in [0, 15]$ in ruter med if ruter-ID. Flag 0 setter aktiv-biten til value. Flag 1 setter trådløs-biten til value. Flag 2 setter 5GHz til value. Flag 3 er ugyldig, så skriv gjerne ut en feilmelding. Flag 4 setter endringsnummeret til verdi. For flag 0 til og med 2 kan verdi kun være 0 eller 1.
- **sett_model ruter-ID navn** : Setter navnet til ruteren med ruter-ID.
- **legg_til_kobling ruter-ID ruter-ID** : Legg til en kobling mellom to rutere.
- **slett_ruter ruter-ID** : Slett ruter-informasjon. Merk at alle pekere til denne ruteren også må slettes.
- **finnes_rute ruter-ID ruter-ID** : Sjekker om det finnes en rute mellom to rutere. Du skal foreta et graf-søk fra den første ruteren og se om du kan komme deg til den andre ruteren. En mulighet, som er overraskende enkel med rekursjon, er å gjøre et dybde-først søk. For enkelthet trenger du ikke å skrive ruten du finner, men gjør det gjerne hvis du har tid og er helt sikker på at alt annet fungere som det skal.

ALT i denne filen er lesbart med en tekst-editor. ruter-ID, flag, verdi er alle tall på leselig form. De må derfor konverteres til rette datatyper ved bruk av f.eks *strtol*.

3 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. Utføre kommandoer gitt i en kommandofil.
3. Avslutning og skriving til fil.



Figur 2: Global array for rutere

Innlesing og datastrukturer

Dette skal skje med en gang programmet starter. Den første 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, dynamisk allokert array. Dette minne skal allokere med plass til N (antall rutere) rutere. Når alle linjene har blitt lest inn, fått sin egen struct og sin egen entry i den globale arrayen går programemt videre til kommandofilen.

Avslutning

Når programmet har utført alle kommandoene i kommando-filen 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. Det kan lønne seg å skrive til en annen fil mens dere utvikler programmet.

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. Oppretting av array-strukturen
2. Innlesing av data fra fil
3. Riktig innsetting av data i arrayen
4. God bruk av minne (`malloc` og `free`).
5. Print.
6. Legg til kobling.
7. Slett en ruter.
8. Skriving av data til fil
9. Endring av et navn.
10. Endring av FLAGG-charen i en ruter-struct.
11. Sjekke om det finnes en vei mellom to rutere.

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 kommando-håndtering.

Kommentarer

Vi krever at dere kommenterer følgende i programmet deres:

- kall til malloc: Ca hvor mye minne som allokeres, i bytes.
- kall til free: Hvor minnet ble allokert.
- funksjonsparametere: typer og kort hva de er.
- funksjonsflyt: Kort hva funksjonen gjør med mindre det er åpenbart.

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 IN2140 Oblig

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`
- `strtok`
- `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.

Kjør Valgrind!