Oblig 2 – IN2140 – Vår 2020

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.

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 11. mars 23:59.

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 1.1.

Vi må kunne kompilere programmet ved å kalle `make` (**dere må altså lage en makefile**) og programmet må hete `ruterdrift`. Programmet 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 må det være mulig å utføre følgende kommando:

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

Du kan anta at programmet kjøres med rett antall argumenter, men dere bør håndtere filer som ikke eksisterer. Dere må derfor sjekke om åpning av filene er vellykket.

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

Dere bør bruke disse filene i testingen av programmet deres. Det er viktig at systemet ikke har minnelekasjer og andre minnefeil. Dette kan dere sjekke ved å bruke valgrind.

1.1 Filstrukturer

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

1.1.1 Informasjon om rutere (fil 1)

Denne filen inneholder all data som programmet trenger. Filen inneholder binær data (i tillegg til tekst-data) og kan dermed *ikke* inspiseres i standard teksteditorer som Atom/Notepad etc. Til gjengjeld er det enklere å skrive kode for å lese inn or skrive ut 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. Dette er altså ikke noen tall på leselig form. Kall dette tallet N.

Etter disse 4 bytene følger det N informasjonsblokker, da hver informasjonsblokk gir informasjon om en ruter. En gyldig fil uten ruterdata (altså med 0 informasjonsblokker) inneholder kun 4 bytes med verdi 0 (men alle utleverte filer og testfiler har informasjon om minst en ruter).

En informasjonsblokk er maksimalt 256 bytes lang (denne restriksjonen skal opprettholdes når du skriver ny data til fil).
Hvor hver informasjonsblokk inneholder informasjon om en ruter på følgende form:

- RuterID – `unsigned int` (4 bytes)
  Denne tallverdien er unik.
- Flagg – `unsigned char`
- Lengde på produsent/modell-strengen – `unsigned char`
  Denne lengden teller bare de menneskelesbare bokstavene i strengen som følger, altså ikke noen 0-verdi, Tab eller Return.
- Produsent/modell – `char[]` (maks lengde 248)
  Lengden i bytes er nøyaktig den som er gitt i lengde-feltet. Den kan ikke inneholde bokstaver som 0, Tab eller Return.
- Informasjonsblokken avsluttes med en byte med verdi `0`.

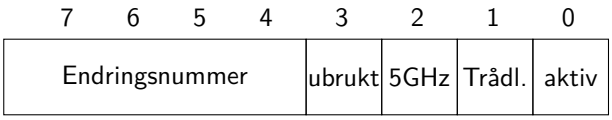
Hver informasjonsblokk i filen begynner altså 6 bytes binær informasjon om ruteren, etterfulgt av lesbare bokstaver for produsent/modell og en avsluttende 0-byte.

I deres programm skal hver informasjonsblokk 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



Figur 1: Flagg-feltet

Etter de N informasjonsblokker om ruterne følger et ukjent antall informasjonsblokker om koblinger mellom ruterne. Hver informasjonsblokk består av 9 bytes og inneholder to ruter-IDer og en avsluttende 0-byte.

ID-ene er lagret i binært-format, og er av typen `unsigned int` (4 bytes). Her kaller vi den første `inten` R1, den andre `inten` R2. En koblig fra ruter R1 til ruter R2 betyr at structen for ruteren R1 skal ha en peker til R2. 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 det er maksimalt 10 koblinger fra enhver ruter R1 til andre rutere. Det kan derfor allokeres minne til 10 pekere for hver kobling uansett faktisk antall. Dere kan ha en array av pekere i structen med informasjon om en ruter, men koden må kunne skille mellom pekere som er i bruk og slike som ikke er det.

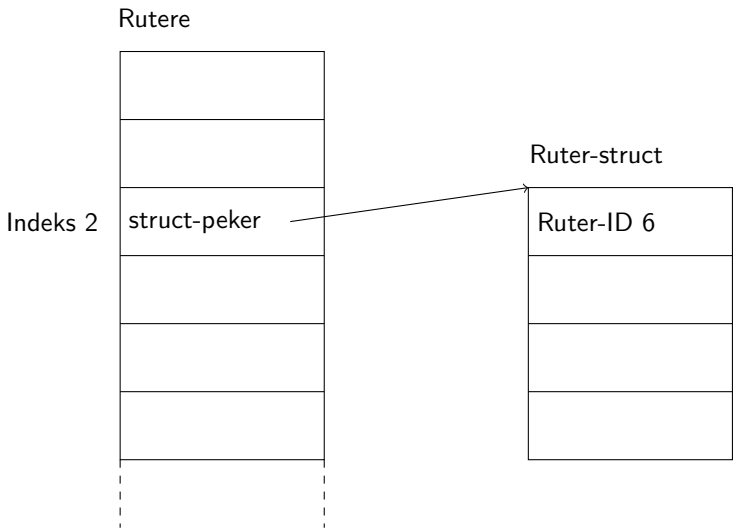
Alle disse koblingene definerer en rettet graf som dere skal traversere med et graf-søk. Detaljer rundt graf-søket kommer i seksjon 1.1.2.

1.1.2 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 verdi $\in [0, 15]$ in ruter med if ruter-ID. Flag 0 setter aktiv-biten til verdi. Flag 1 setter trådløs-biten til verdi. Flag 2 setter 5GHz til verdi. 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, f.eks ved bruk av *strtol*.



Figur 2: Global array for rutere

1.2 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.

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 ruter-informasjonsblokk i filen skal det allokeres plass til en struct (med `malloc`), og structen skal fylles med data fra informasjonsblokken.

En peker til structen skal lagres i et globalt, dynamisk allokert array. Dette minne skal allokeres med plass til N (antall rutere) rutere. Når alle ruter-informasjonsblokker har blitt lest inn, fått sin egen struct og sin egen entry i den globale arrayen, går programmet videre til å lese koblings-informasjonsblokker. Koblings-informasjonsblokkene leses i en løkke helt frem til leseoperasjonen oppdager filslutten (end-of-file).

Hver gang en koblings-informasjonsblokk er lest inn oppdateres ruter-structen som er fra-siden i en kobling. Er det feil i en koblings-informasjonsblokk slik at enten fra- eller til-siden ikke finnes så skriver programmet en warning og fortsetter med neste koblings-informasjonsblokk.

Etter at hele informasjonsfil er lest går programmet 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 oppretting 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.

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!

2 Multiple choice

Hodet på en harddisk må bevege seg mellom sporene på en disk for å svare på lese- og skriveforespørsler. Dette er den tregeste delen av diskdriften og krever gode strategier.

Se for deg følgende scenario:

Det innerste sporet på en disk er spor 0, det ytterste er spor 31. Diskhodet er plasert på spor 18 og beveger seg utover.

Disken har mottatt følgende forespørsler i den gitte ordren:

yngste forespørsel	4
	30
	10
	20
	14
	7
eldste forespørsel	24

Etter at disken har fullført flyttingen til det andre sporet den skal lese (I tilfelle FIFO betyr det at den er ferdig med lesing av spor 24 og den har allerede flyttet til spor 7 for lesing. For andre strategier er andre sporer.), mottar den en annen forespørsel som følger:

ny forespørsel	23

2.1 Spørsmål 1

Hvis strategien er C-SCAN med leseretning utover og den ikke-lesende bevegelsen innover, i hvilken retning beveger diskhodet seg umiddelbart etter å ha lest spor 30? Kryss av riktig svar:

innover	
utover	

2.2 Spørsmål 2

Hva er svaret på spørsmål 2.1 hvis strategien er C-LOOK? Kryss av riktig svar:

innover	
utover	

2.3 Spørsmål 3

Hva er summen av avstander som hodet må reise fra sin nåværende stilling (18) til disken har besvart alle forespørsler og hodet befinner seg på sporet til den siste besvarte forespørselen? Fyll inn summene for følgende strategier:

FIFO	
SSTF	
LOOK	

Hint

Et hode som beveger seg direkte fra spor 5 til 22 beveger seg over en distanse på 17, ikke 18. Sporet som hodet forlater teller ikke.

Format

Svaret for oppgave 2 må leveres i en egen PDF-fil som heter oppgave2.pdf.

3 Levering

1. Lag en mappe med ditt brukernavn: `mkdir bnavn`
2. Lag mapper for de to deloppgaver: `mkdir bnavn/oppgave1` og `mkdir bnavn/oppgave2`
3. Kopier alle filer som er en del av innleveringen inn i deloppgavens mappe:
   f.eks. `cp *.c bnavn/oppgave1/` og `cp oppgave2.pdf bnavn/oppgave2/`
4. Komprimer og pakk inn mappen:
   `tar -czvf bnavn.tgz bnavn/`
5. Logg inn på Devilry
6. Lever under IN2140 Oblig