

## Eksamen 2022 PG3401 C Programmering

### Oppgave 1 – Språket C

A)

Språket C er et imperativt programmeringsspråk, utviklet i 1972 av Dennis Ritchie, som kombinerer trekk fra både lavnivå og høynivå språk.

C defineres som et høynivåspråk, men er fortsatt maskinnært. Kombinasjonen av hastighet, håndtering av dynamisk minne, og at det er et relativt språk å lære seg, er C et svært populært språk blant utviklere i dag. Språket har store bruksområder innen flere ulike områder. Operativsystemer blir ofte skrevet i C. C er svært nyttig dersom man utvikler en applikasjon der hastighet er essensielt, som for eksempel grafikk programmering, og databasehåndteringssystemer.

C er også mye brukt ved utvikling av innebygde systemer (embedded systems. I de siste årene, med eksplosjonen av IOT, vokser antall utviklere som velger å lære C.

På grunn av dets popularitet, er det også rikelig med biblioteker, noe som igjen gjør språket enda mer attraktivt.

B)

Linus Thorvald, født 28. Desember 1969, er en finsk informatiker. Linus gikk på Helsinki universitet, og fikk sin master grad i informatikk i 1996. Den 25. august 1991, skrev Linus en melding i comp.os.minix, der han erklærte at han arbeidet med et, som han kalte det, «lite hobbyprosjekt».

Dette hobbyprosjektet var et operativsystem, selve basen for det vi i dag kaller Linux operativsystem.

C)

Som en kodebase vokser, og hvis språket denne kodebasen baseres på er et språk uten «garbage collection», skjer det ofte at utviklere allokerer minne, for så å glemme å frigjør det etter bruk. Dette fører til minne lekkasjer, som over tid gjør systemet tregere, og kan føre til ustabil oppførsel.

Valgrind er et instrument som brukes for debugging av Linux programmer. Det er svært nyttig for å finne frem til minne lekkasjer, som ellers er vanskelige å detektere for det menneskelige øye.

Instrumentet brukes også for å debugge multitråd applikasjoner.

## Oppgave 2 - Filhåndtering

Som oppgave 2 for denne eksamenen bes vi å lese verdier fra vil, for deretter å skrive ut hver karakter som sin heksadesimale representasjon til en annen fil.

Det første steget for å løse denne oppgaven var å sette tekst strengen vi skal behandle i en .txt fil.

Da jeg inspiserste tekst strengen la jeg merke til den norske bokstaven «å», som befinner seg på linje 1, kolonne 19. Vi fikk beskjed om å behandle teksten som 7 bit ASCII, noe jeg stusset over da «å» enkodes desimalt til 229 (UTF-8).

Siden oppgaven eksplisitt sier at tekst filen skal lagres i tekstfil bestående av 7 bit ASCII, gjorde dette meg usikker, var dette en feil i oppgaveteksten, eller en godt gjennomtenkt test? Så vidt jeg vet, (etter muligens litt for lite research), representeres alltid utf-8 karakterer med 2 bytes. Dette betyr at output ikke vil bli dobbelt så stort som input, slik oppgaven sier, men heller  $\text{input} * 2 + n$  (hvor n er antall karakterer kodet med UTF-8, her 1). Jeg valgte å ikke hyperfokusere på dette. Da resten av oppgavene virket krevende, er det ikke særlig ønskelig å få oppheng allerede på eksamens andre oppgave.

Etter å få oppsettet i orden, implementeres en metode for å lese tekst strengen fra fil. Metoden heter `readFileContentToCharArray`, og har to parametere: 1 => navn på filen som skal leses, 2 => pointer til bufferet det skrives til. Her leses filens innhold inn i et array som deklarerer i main (arrayet pekeren gitt som parameter 2 peker mot). Metoden er tatt fra forelesning «INSERT HERE». Metoden i fasiten hadde bra feil håndtering, og jeg ønsket derfor å kopiere denne rett inn i eksamens besvarelsen. I denne metoden finner vi lengden på filen (`ulSize`), allokerer minne tilsvarende `ulSize + 1` Ved å ta pluss en, allokerer vi nok minne til å legge til null terminator.

Etter input filens innhold er lest inn i buffer, sendes pointeren til samme buffer, sammen med navn på output fil, som parametere til metoden `writeHexRepresentationToTextFile`. Her åpner vi output filen i «write mode» (hvis fil med filnavn gitt som parameter ikke eksisteres, opprettes den her). Deretter går vi inn i en «while» løkke, hvor hver enkelt karakter fra bufferet leses til en unsigned char variabel, og printes til den designerte output filen som sin heksadesimale representasjon. Løkken avbrytes i møte med bufferets null terminator. Alle karakterene representeres ved en byte, utenom «å», som representeres av to. Output filens lengde blir derfor  $\text{input} * 2 + 1$ .

## Oppgave 3 – Liste Håndtering

I oppgave 3 skal vi lage en dobbelt lenket liste, som skal representere reservasjoner i et hotell. Vårt hotell har det stilige navnet «CSTARS TONIGHT».

Løsningen baseres på tre strukturer:

`_RESERVATION_INFO`: inneholder relevant informasjon om reservasjonen: navn, antall netter gjesten blir, rommet reservert, og pris per natt.

`_RESERVATION`: inneholder peker til `RES_INFO`, peker til neste reservasjon, og forrige reservasjon.

`_RESERVATION_LIST`: Inneholder første reservasjon, og siste reservasjon (head and tail). Denne strukturen brukes for å iterere gjennom alle opprettede reservasjoner.

Når applikasjonen starter opp, får bruker presentert en meny. Her er det 7 valg, 1 – 6 representerer punktene fra eksamensteksten, mens punkt 7 rydder opp (frigir minne), for så å avslutte applikasjonen.

Det mest krevende med denne oppgaven, var å sette dato. For brukervennlighet, syns jeg det var fint å gi bruker mulighet til å skrive hele dato i en linje, i stedet for en linje for dag, en for måned og en for år. Når bruker skriver inn dato får han beskjed om å skrive det i formatet «ddmmyyyy». Hvis input ikke kan konverteres til tall (`atoi()`), printes en beskjed til konsoll som sier at dato ikke er valid.

Eksamens oppgaven spesifiserer også at det skal være mulig å slette reservasjoner som er utgått. For å gjøre dette må man sammenligne to datoer. Dette fungerer ikke særlig bra med formatet jeg har valgt.

```
int getCurrentDateAsInteger(){
    time_t t = time(NULL);
    char buff[20];
    struct tm tm = *localtime(&t);
    sprintf(buff, "%d%d%d", tm.tm_mday, tm.tm_mon + 1, tm.tm_year + 1900);
    int date = atoi(buff);

    return date;
}
```

Besvarelsen har implementert funksjonalitet for alle punktene i listen gitt i denne oppgaven. Det tredje punktet er ikke helt fullført. Med formatet brukt for å implementere dato systemet, ble det tungvint å sammenligne datoer. Det fungerer på visse tall, men siden det ikke er nok feilhåndtering, kan input av svært lave / høye datoer føre til problemer.

Som nevnt ovenfor brukes `_RESERVATION_LIST` strukturen, sammen med `_RESEVATION` strukturens pekere til «next» og «previous» for å iterere gjennom reservasjonene.

## Oppgave 4 – Finn tre feil

Som oppgave 4, blir vi presentert for en bit kode som skal prosessere http responser fra en server. Denne koden fungerer ikke som tiltenkt, oppgaven er å finne tre feil, og dermed få lest av korrekt verdi fra headeren «Content-Length». Jeg har holdt på en del med denne koden, da den også blir brukt i oppgave 6. I besvarelsen er det derfor gjort endringer som kommer foruten de tre feilene, eks: å øke bufferet til szContentType. Denne verdien var opprinnelig 16, noe som passer headeren dårlig, da den vanlige verdien «application/json», er 16 karakterer lang. Jeg går ikke over disse endringene i detalj, men under nevnes de tre feilene som gjør at den opprinnelige koden ikke fungerer.

### Fremgangsmåte

Først satt jeg koden inn i en .c fil i mappen «oppgave\_4», og lager en main metode. I main metoden deklarerer et array med dummy data som simulerer en http-server respons.

#### Feil 1:

I den originale koden brukes typedef til å sette navn på strukten \_MYHTTP. Det settes to navn, MYHTTP og \*MYHTTP. \*MYHTTP er problematisk. Stjerne symbolet tilsier at det er en pointer. Når minne allokeres (malloc(sizeof(\*MYHTTP)), vil kun allokere nok minne til en enkel pointer (4 bytes på et 32 bit system, 8 bytes på 64 bit system). Dette resulterer i en feil. Ved å ta vekk stjerne symbolet løser vi dette.

```
char szContentType[16]
}MYHTTP, PMYHTTP;
//MISTAKE #1 *MYHTTP
```

#### Feil 2:

```
strchr(pszPtr, '\n')[0] = 0;
```

I C-programmering er det svært viktig å være vandt med at null terminatorer er viktige å legge til når man splitter en tekst streng, men her fører det til feil. Hvis vi setter null terminator på pszPtr, vil verdiene som kommer etter terminatoren ikke kunne leses av. Denne feilen merke skjer to ganger i koden, i blokken der man leter etter «server», og i blokken «Content-Type». I server delen merkes er det vanskeligere å merke, da Content-Type headeren (etter http protokollen), kommer før «Server». Man kan derfor lese Content-Type av, selv om null terminatoren er satt etter «Server». Når man setter null terminator i server blokken vil man ikke kunne lese av Content-Length, da denne headeren kommer sist. I min kode har jeg fikset dette ved å fjerne denne delen av koden, og setter heller en verdi n til å inkrementere i en while loop slik at den settes til lengden til headerens verdi. Hvis n er større enn buffer størrelsen, vil den settes til bufferstørrelse - 1. Research sa at http

headere skal avslutte «\r\n», men koden har feilhåndtering slik at man, hvis \r ikke forekommer, splitter på \n.

```
n = 0;
while(pszPtr[n] != '\r' && pszPtr[n] != '\n')
    n++;

if(n >= MAX_LENGTH_SERVER){
    n = MAX_LENGTH_SERVER - 1;
}
strncpy(pHttp->szServer, pszPtr, n);
```

Feil 3:

I delen av koden hvor Content-Length settes, finner man den tredje feilen:

```
pHttp->iContentLength = '0' + atoi(pszPtr);
```

Karakteren '0', har ASCII verdi 48. Content-Length verdien satt her vil bli (correctContentLength + 48). Denne feilen rettes ved å fjerne '0', slik at http-iContentLength = atoi(pszPtr);

## Oppgave 5 - Tråder

Oppgave 5 går ut på å lage en applikasjon hvor det brukes tråder. Man skal starte to tråder i main funksjonen. Tråd A skal lese verdier fra en PDF-fil til en buffer, når det er data i bufferet skal dette kommunisere til tråd B. Tråd B skal deretter lese verdier fra buffer, og telle forekomster av de 256 mulige byte verdiene.

Koden her er inspirert av denne videoen:

[https://www.youtube.com/watch?v=l6zkaJFjUbM&ab\\_channel=CodeVault](https://www.youtube.com/watch?v=l6zkaJFjUbM&ab_channel=CodeVault)

I denne besvarelsen brukes en strukt for å lagre relevante verdier. Strukten inneholder to semaphorer, `semFull` og `semEmpty`, og en mutex. I tillegg til disse har vi en integer count, peker til fil som leses, størrelse på fil som leses, og et buffer med plass til 4096 bytes. Når applikasjonen startes, lages to tråder av hovedtråd «main». For den ene tråden, tråd A, settes parameteret «routine», til å være en funksjon som leser bytes fra fil til buffer. For tråd B settes dette parameteret til en funksjon som leser bytes fra buffer, og teller antall forekomster av bytes fra verdi 0-255. Semaforen `semEmpty` holder rede på hvor mange ledige «slots» vi har i buffer, mens semaforen `semFull` holder rede for hvor mange «slots» i bufferet som er tatt av en annen verdi. Verdien `struct->count` inkrementeres av tråd A når byte er lest fra fil, og dekrementet av tråd B når byte leses fra buffer. Når trådene gjør operasjoner mot delte ressurser (buffer / count), låses en mutex, slik gjør vi applikasjonen trådsikker. Når antall elementer lest fra fil = filstørrelse, lukker tråd A gjeldene fil, og «joiner» hovedtråden. Tråd B fortsetter sin opptelling av bytes.

Dersom antall bytes lest fra buffer != filstørrelse, printes en feilmelding til konsoll. Hvis antall bytes == filstørrelse, printes antall forekomster til konsoll, før tråd B også «joiner» hovedtråden.

Når det kommer til implementasjon av mutex i løsningen, er jeg usikker på om den strengt tatt er nødvendig. Jeg prøvde å kjøre applikasjonen uten, og fikk samme resultat. Etter å søke rundt, fikk jeg inntrykket av at det er god praksis å bruke mutex når man deler ressurser mellom tråder, så valgte å ha det værende i koden.

Det jeg var mest usikker på under denne oppgaven, var om det var ønskelig å ha `semEmpty` som en binær semafor, i stedet for å sette start verdien til `BUFFER_SIZE` (4096). I etterkant virker det nemlig tungvindt å inkrementere count variabel, låse opp mutex og gjøre en post til `semFull`, for hver enkel karakter lest fra fil. Kanskje det ville vært mindre tidkrevende (og ressurs krevende), å lese av 4096 bytes av gangen.

Besvarelsens løsning fungerer, men som sagt over, er jeg temmelig sikker på at den kan optimiseres ved å lese mer enn en karakter av gangen. Med mer tid hadde jeg definitivt satt meg ned og analysert de ulike alternativenes bruk av tid/ressurser grundigere.

## Oppgave 6 -Nettverk

I oppgave 6 fikk vi i oppgave å lage en klient/server applikasjon, hvor det skal være mulig for klient å laste ned filer fra server.

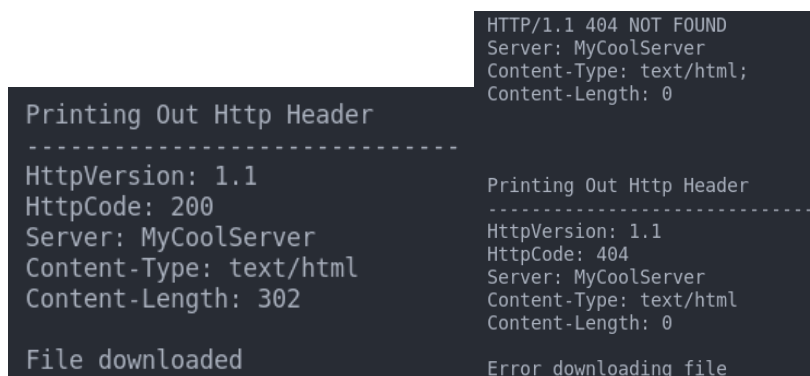
I denne besvarelsen er dette gjort. Applikasjonen støtter kun nedlasting/opplasting av tekst filer. Løsningen kan testes ved: ./oppgave\_6\_client include/httpfunc.h, mens server kjører. Da vil klient laste ned serverens httpfunc.h fil i en tekst fil kalt output.txt.

Denne applikasjonen er langt fra elegant, men har noe feilhåndtering, som for eksempel hvis fil ikke eksisterer hos server, vil ikke data sendes. Jeg har også prøvd å simulere et «TCP handshake», ved å sende «Hello Server» fra klient til server, «hello from server» fra server, og et til «hallo» fra klient. Dette er kanskje unødvendig, men tenkte det var litt morsomt å legge til.

Når filen sendes til klient, settes http headerne «Content-Length» og «status code», sammen med hardkodete verdier for «Server», «Content-Type» (text/html), og «http version» (1.1). Kode for å sette/prosessere http-headerne er tatt fra oppgave\_4.

Når server applikasjonen starter, lytter den til port 30000. Når klient «connecter», lages en socket som brukes for videre kommunikasjon. Når server har mottatt GET requesten, leses filnavnet av requesten. Hvis filen eksisterer hos server, leses den til en buffer. Størrelsen på bufferet (strlen) settes som «Content-Length», videre settes de resten av headerne. Deretter sendes headerne, sammen med innholdet av buffer, til klienten. Hos klienten prosesseres server-responsen. Hvis statuscode == 200, leser klient data til fil i egen mappe.

http responsen printes til konsoll, som sett på eksemplene under.



```

HTTP/1.1 404 NOT FOUND
Server: MyCoolServer
Content-Type: text/html;
Content-Length: 0

Printing Out Http Header
-----
HttpVersion: 1.1
HttpCode: 200
Server: MyCoolServer
Content-Type: text/html
Content-Length: 302
File downloaded

Printing Out Http Header
-----
HttpVersion: 1.1
HttpCode: 404
Server: MyCoolServer
Content-Type: text/html
Content-Length: 0
Error downloading file
  
```

Av og til når jeg starter klienten en andre / tredje gang, får man opp «connection refused». Da kan enten lukke IDE og starte på nytt. Man unngår dette ved å ikke avslutte server mens klienten er koblet til. Man kan avslutte klienten sli man ønsker, men server bør kjøre i bakgrunnen mens man tester.



## Oppgave 7 – Filhåndtering

Som oppgave 7, skulle vi lage en «code beautifier».

Besvarelsen starter ved å lese av fil oppgitt som parameter.

Vi starter med å telle antallet for løkker som forekommer i filen. For hver for loop vil koden gjøre gjentatte steg:

Finne initialization (eks: `int a;`), evaluation (eks: `a < b`), og update (eks: `a++`), og sette hver i sin separate buffer. Deretter brukes flere ulike while loop for å skippe over mellomrom og andre tegn.

Når vi har de nødvendige verdiene våre, brukes memmove funksjonen for å flytte resten av koden til venstre/høyre.

Hvis det er flere for løkker, oppdateres pointer til å peke på ny løkke.

Hvis koden ser sånn her ut:

```
int main(void){
    int b = 10;
    int a;
    for(    a = 0;    a < b;    a++    ) {
        printf("In loop %d\n", a);
    }

    return 0;
}
```

Vil den ende opp slik:

```
3
4  int main(void){
5      int b = 10;
6      int a;
7      a=0;
8      while(a<b){
9          a++;
10         printf("In loop %d\n", a);
11     }
12
13     return 0;
14 }
```

I Visual Studio Code.

For å finne antall tabs brukes funksjonen `strchr()`. Deretter gjør vi det samme som ovenfor: flytter resten av teksten med `memmove`, for deretter å bruke `memcpy` for å kopiere inn tre mellomrom per tab.

Takk for et fint år, og god jul. Du er en utrolig dyktig lærer.