

## **72 timers HJEMMEEKSAMEN - KONTE**

### **PG3401 C Programmering**

**Tillatte hjelpemidler:** Alle

**Varighet:** 72 timer

**Karakterskala/vurderingsform:** Nasjonal karakterskala A - F

**Dato:** 21. februar - 24. februar 2025

---

*The exam text is in Norwegian only. You can hand in your exam in either English or Norwegian (Swedish and Danish is also accepted). Programming language used should though be the language C-89, other languages than C will not earn points.*

Oppgavesettet har 7 sider. Det er totalt 5 oppgaver i oppgavesettet.

Det presiseres at studenten skal besvare eksamen selvstendig og individuelt, samarbeid mellom studenter og plagiat er ikke tillatt, besvarelsen skal representere studentens eget arbeid. Det presiseres at dette også betyr at det ikke er lov til å benytte kode- eller tekstgenererende verktøy basert på kunstig intelligens. Vær obs på at forsøk på «bløffing» på praktiske oppgaver i form av skjermbilder funnet på internett, skjermbilder fått av andre studenter eller redigerte skjermbilder gir MINUSPOENG på total scoren på eksamen (som kan resultere i at endelig karakter settes en eller flere karakterer ned), og i grove tilfeller vil det resultere i fusk/plagiat sak. Du skal kun dokumentere det du selv har fått til, ikke forsøke å få det til å se ut som at du har fått til noe du i virkeligheten ikke har klart.

## **Format på innlevering**

Dette er en praktisk programmeringseksamen (bortsett fra oppgave 1), fokus bør derfor være på å forklare hvordan du har gått frem, begrunne valg og legge frem eventuelle antagelser du har gjort i din løsning.

Hvis du ikke klarer å løse en oppgave er det bedre om du forklarer hvordan du har gått frem og hva du ikke fikk til – enn å ikke besvare oppgaven i det hele tatt. Det forventes at alt virker hvis ikke annet er beskrevet i tekstbesvarelsen, hvis du vet at programmet krasjer, ikke kompilerer eller ikke virket slik den var tenkt er det viktig å forklare dette sammen med hvilke skritt du har tatt for å forsøke å løse problemet.

Besvarelsen skal leveres inn i PDF format i en fil med navn

PG3401\_konte25\_[kandidatnummer].pdf. Kildekoden må være vedlagt i 1 ZIP fil, navnet på filen skal være PG3401\_konte25\_[kandidatnummer].zip. Denne filen skal ha følgende struktur:

```
\ oppgave_2 \ makefile
\ oppgave_2 \ [...]
\ oppgave_3 \ makefile
\ oppgave_3 \ [...]
\ oppgave_4 \ makefile
```

\ oppgave\_4 \ [...]

Det gis trekk i poeng for besvarelser som ikke har navngivning som oppgitt over. Vær sikker på at alle filer er med i ZIP filen. Hver mappe skal ha en makefile fil, og det skal ikke være nødvendig med noen endringer, tredjeparts komponenter eller parametere – det skal være mulig for sensor å kompilere ved å, i shell på Debian 10, gå inn i mappen og skrive «make» og dette skal bygge programmet med GCC.

Tekstbesvarelsen skal inneholde besvarelse på oppgave 1 (skriv det kort og konsist, trenger ikke noe stor avhandling). Etter den rene tekstbesvarelsen skal hver oppgave inkludere begrunnelse/dokumentasjon til kildekode, hver av disse begrunnelsene skal være på en ny side for å gjøre tekstbesvarelsen oversiktlig for sensor. Besvarelser i andre formater (enn PDF og ZIP) vil ikke bli lest.

Det spesifiseres at oppgavene må programmeres på Linux (tekstbesvarelsen kan skrives på Mac eller Windows, eller andre operativsystem studenten måtte bruke). På alle oppgaver er det et absolutt krav å bruke makefile filen som er brukt i undervisningen i emnet – som hentet fra forelesning 4, sleide 65 med tittel «Use this makefile», du kan gjøre mindre endringer i makefile filen hvis det er nødvendig for å løse oppgaven, men oppgaver som bruker en annen makefile uavhengig om den er auto-generert eller laget manuelt vil ikke få poeng på oppgaven. Forelesningen er tilgjengelig på følgende URL: [http://www.eastwill.no/pg3401/pg3401\\_lecture04.pdf](http://www.eastwill.no/pg3401/pg3401_lecture04.pdf)

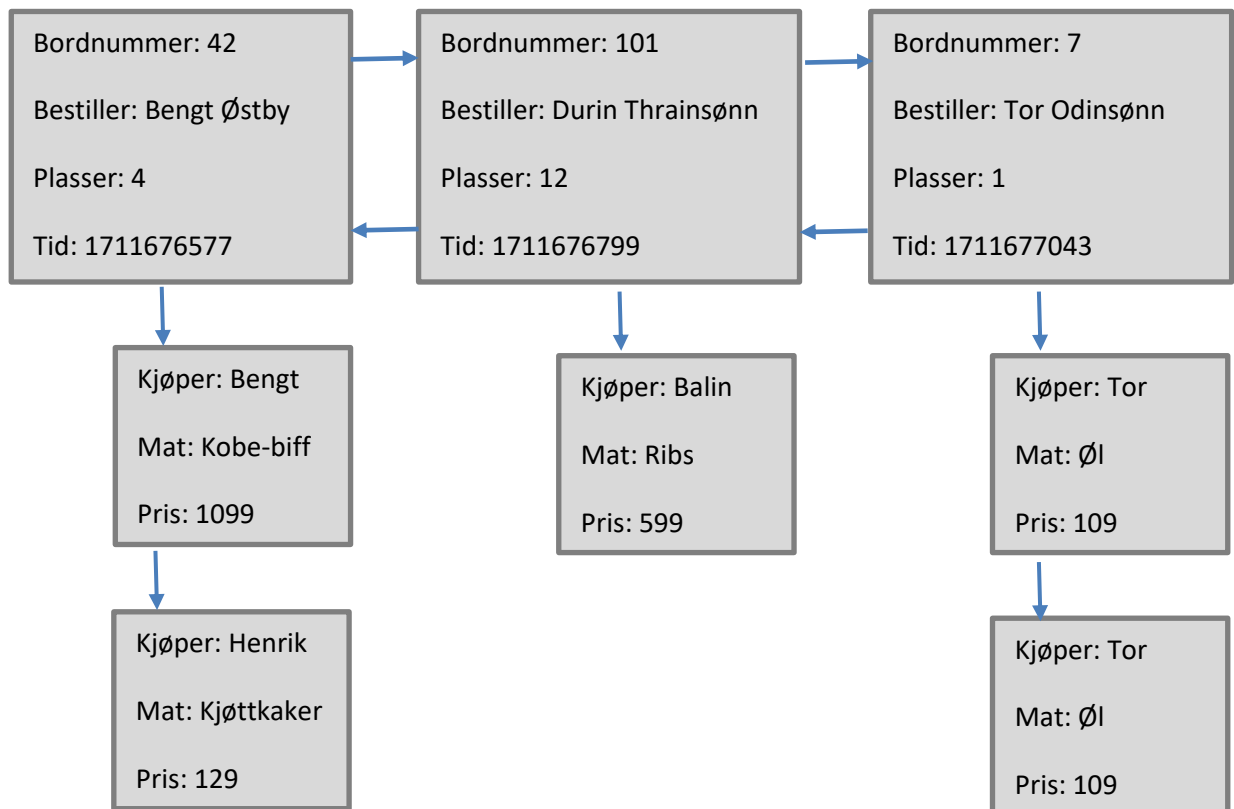
## Oppgave 1. Generelt (5 %)

- a) Forklar hva C programmeringsspråket kan brukes til.
- b) Hvem er Linus Torvalds og hva er han kjent for innen Informasjonsteknologi?
- c) Forklar stegende i kompilering av et C program, fra kildekode til eksekverbar fil.

## Oppgave 2. Liste håndtering (25 %)

Du skal lage en enkel data-struktur for å håndtere bord -reservasjon og -bestillinger på en restaurant. Listens «stamme» skal være en dobbeltlenket liste over bord-reservasjoner, hvert element (struct) i listen skal inneholde pekere til både forrige element og neste element. Elementet skal også inneholde et tall som er BORDNUMMER, en tekststreng som inneholder NAVN på kunden som har reservert bordet, en integer for antall SITTEPLASSER og en integer som skal fungere som TID for reservasjonen. Hvert element skal i tillegg inneholde en enkeltlenket liste over BESTILLINGER av mat gjort på bordet.

Hvert element i bestillings-listen skal inneholde en tekststreng som inneholder navn på KJØPER (av maten), en tekststreng med fritekstbeskrivelse av KJØPT MAT, og en integer som inneholder PRIS på maten.



Du skal lage funksjoner som utfører følgende operasjoner på listen:

- Legge til en bord-reservasjon i listen, listen skal alltid være SORTERT etter navn på den som har reservert bordet (så det må støttes både å legge inn på starten, slutten og midt i listen)
- Henter ut en bord-reservasjon N fra listen (telt fra starten av listen, hvor første element er element nummer 1) og skrive ut på skjermen alle dataene tilhørende bordet inklusive liste over alt som er kjøpt/bestilt
- Tar navn som input og søker etter reservasjoner under det navnet, og skriver ut navn, bordnummer og dato på skjermen
- Sletter en kundes bord-reservasjon
- Legge til mat eller drikke som bestilling til et bord
- Printer ut på skjerm alt som er bestilt på et bord, slik at kunden kan få sin regning – dataene skal avsluttes med en sum for bordet
- Printer ut på skjerm liste over det som er bestilt på et gitt bord og et gitt navn, slik at bordet kan be om «separate regninger» - dataene skal avsluttes med en sum for personen på det gitte bordet

Du skal lage en main funksjon som mottar instruksjoner fra bruker basert på input fra tastaturet, main må altså kunne kalle alle de syv funksjonene over (for eksempel i en form for meny) og be brukeren om data som er nødvendig for å kalle de nevnte

funksjoner. Main skal rydde opp alle data før den returnerer (et valg i menyen må være å avslutte).

Du må lage en makefile fil, og en eller flere header filer for programmet til å kompilere riktig, gjør eventuelle endringer påkrevet i makefile filen for å få programmet til å bygge og kjøre korrekt.

### **Dokumentasjonskrav:**

Du må ta et skjermbilde av programmet når det kjører, skjermbildet skal legges ved besvarelsen i en bildefil med navn oppgave2\_screenshot.png (i samme mappe som kildefilene) OG skal i tillegg settes inn på egnet sted i tekstbesvarelsen (PDF filen).

## **Oppgave 3. Tråder (20 %)**

I praktisk programmering er det ofte effektivt å legge tidkrevende operasjoner ut i arbeidstråder, eksempler på dette er filoperasjoner, nettverksoperasjoner og kommunikasjon med eksterne enheter. I denne oppgaven skal du simulere slike operasjoner med et mindre datasett – for å spare dere for tid under eksamen er det i denne oppgaven allerede laget en ferdig applikasjon som dere skal endre.

*Applikasjon består av 3 tråder; hovedtråden (som kjører main funksjonen) og 2 arbeidstråder. Hovedtråden starter arbeidstrådene med mekanismer for tråd-kommunikasjon og synkronisering (i denne applikasjonen har ikke hovedtråden noen annen funksjon, og starter kun de to trådene som gjør selve jobben). Trådene har et minneområde med plass til 4096 byte for kommunikasjon mellom trådene.*

*Den ene arbeidstråden (tråd A) skal lese en tekstfil, arbeidstråd A skal så sende filen over til den andre arbeidstråden (tråd B) gjennom flere sykluser ved hjelp av minneområdet beskrevet over, og signalere tråd B om at det er data tilgjengelig i bufferet. Tråd B teller så opp antall forekomster av bytes med verdi 00, 01, 02, og så videre til; FF i filen den får sendt over. Tråd A og tråd B går i loop for å prosessere filen helt til den er ferdig. Når filen er sendt over i sin helhet avslutter arbeidstråd A. Arbeidstråd B fullfører sin opptelling av bytes, og så skriver den ut til terminal vinduet antall forekomster av hver av de 256 mulige byte-verdiene, før den også avslutter. Hovedtråden (main) venter på at begge trådene avslutter, rydder opp korrekt og så avslutter applikasjonen.*

Last ned følgende kildefil, dette er en løsning på applikasjonen som beskrevet:

[http://www.eastwill.no/pg3401/eksamen\\_v25\\_oppgave3.c](http://www.eastwill.no/pg3401/eksamen_v25_oppgave3.c)

Last ned en test-fil som kan brukes til denne oppgaven (Hamlet av Shakespeare, hentet fra Project Gutenberg - <https://www.gutenberg.org/ebooks/2265>):

[http://www.eastwill.no/pg3401/eksamen\\_v25\\_oppgave3\\_pg2265.txt](http://www.eastwill.no/pg3401/eksamen_v25_oppgave3_pg2265.txt)

Du skal utvide koden med følgende endringer:

- Du må lage en makefile fil (bruk makefile fil som beskrevet over – med overskriften «Use this makefile») for å få programmet til å bygge, gjør eventuelle endringer påkrevet i makefile filen og kildefilen for å få programmet til å bygge og kjøre korrekt
- Programmet bruker globale variable, endre dette til at alle variable er lokale variable i main funksjonen og sendes inn som parameter til de to trådene
- Tråd A har hardkodet navnet på filen som skal leses, endre programmet til å ta filnavnet som parameter på kommandolinjen (inn-parameter til main funksjonen) og sender dette videre inn til Tråd A som et argument
- Skriv om koden fra å bruke conditions til å bruke semaforer
- Legg til kode for eksplisitt initialisering av mutex og semaforer (med \*\_init funksjonene)
- Legg til kommentarer i koden for å dokumentere hva koden gjør

I stedet for å telle «byte verdier» (fra 0 til 255) skal du utvide funksjonaliteten til noe mer nyttig, å regne ut en SHA1 hash av filen

- Last ned de to filene:  
<https://raw.githubusercontent.com/clibs/sha1/master/sha1.c>  
<https://raw.githubusercontent.com/clibs/sha1/master/sha1.h>
- Du skal legge til disse filene i makefile filen, og må inkludere sha1.h i eksamen\_v25\_oppgave3.c kildefilen
- For hver iterasjon i tråd B skal du kalle funksjonen i kildefilen over for å regne ut SHA1 hash av filen (eller «legge til i» hashen) – du må lese koden i kildefilen for å forstå hvordan denne skal brukes, merk at koden må initialiseres ved å kalle init funksjonen først
- Når trådene har kjørt ferdig skal Tråd B printe ut SHA1 hashen på skjermen, hashen skal printes som lesbar hex kode på skjermen

### Dokumentasjonskrav:

Du må ta et skjermbilde av programmet når det kjører, skjermbildet skal legges ved besvarelsen i en bildefil med navn oppgave3\_screenshot.png (i samme mappe som kildefilene) OG skal i tillegg settes inn på egnet sted i tekstbesvarelsen (PDF filen). Skjermbildet må vise at programmet har kjørt ferdig, og må vise SHA1 hashen.

## Oppgave 4. Nettverk (30 %)

Du skal i denne oppgaven lage et verktøy for å sette opp et reverse shell på en maskin. Et reverse shell er en applikasjon hvor en tjener sender kommandoer til en klient som klienten kjører «i terminal» som om et menneske hadde sittet på klienten og manuelt skrevet de samme kommandoene i terminal vinduet. Applikasjonen skal kunne startes både som klient og som server, og du skal lage din egen protokoll for kommunikasjon mellom klient og server. Når startet som server (for eksempel signalert ved et parameter fra terminal, som kan være -listen) skal applikasjonen åpne oppgitt port for LISTEN, for denne oppgaven holder det å binde til loopback på

127.0.0.1 (for å ikke eksponere en åpen port eksternt). Det anbefales å bruke TCP.

*Oppgaven skal ikke løses ved bruk av Curl eller andre tredjepartsbiblioteker, og skal ikke basere seg på å starte andre programmer i operativsystemet – kun bruk av Sockets slik vi har lært på forelesning 10 om Nettverk vil gi poeng på oppgaven.*

Når startet som server skal den BINDE til en port brukeren velger, det anbefales for denne oppgaven å kun lytte på adresse 127.0.0.1 for å ikke eksponere porten utenfor egen maskin. Det anbefales å bruke TCP. Server applikasjonen skal eksekveres med portnummer som parameter fra terminal, for eksempel «oppgave\_4 -listen -port 42».

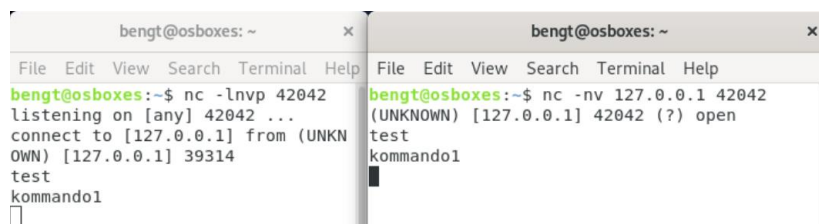
Når startet som klient skal applikasjonen eksekveres med serverens IP adresse og portnummer fra terminal, for eksempel «oppgave\_4 -server 127.0.0.1 -port 42». Når den starter skal applikasjonen CONNECTE til oppgitt port på server-prosessen.

Du skal lage en protokoll for trafikk mellom server og klient, som et minimum må denne inneholde KOMMANDO som en tekststreng som sendes til klienten, og RESULTAT som en tekststreng som sendes tilbake til serveren. Når en klient er tilkoblet skal server applikasjonen akseptere input fra brukeren i form av tekststrenger i terminal, disse skal være Linux (terminal) kommandoer som skal sendes til klient applikasjonen på et format studenten velger. Klient applikasjonen skal EKSEKVERE disse kommandoene, og sende RESULTATET tilbake til serveren som printer dette ut på skjerm. Eksempel på kode som gjør dette (som dere kan bruke i besvarelsen) kan dere finne her:

[http://www.eastwill.no/pg3401/konte\\_v25\\_oppgave4\\_exec.c](http://www.eastwill.no/pg3401/konte_v25_oppgave4_exec.c)

Begge applikasjoner skal kunne avsluttes med Ctrl-C, begge applikasjoner bør håndtere dette (uten å krasje). Når en applikasjon avslutter (enten klient eller server) skal den andre applikasjonen også avslutte.

Et tips for å teste server/klient applikasjoner er å åpne to terminal vinduer i Linux og starte den ene instansen «oppgave\_4 -listen -port 42» i ett vindu og den andre instansen «oppgave\_4 -server 127.0.0.1 -port 42» i det andre vinduet. Eksempel (fra verktøyet netcat som er et eksisterende verktøy som kan gjøre det samme som deres oppgave skal gjøre):



```
benkt@osboxes: ~  
File Edit View Search Terminal Help  
benkt@osboxes:~$ nc -lvp 42042  
listening on [any] 42042 ...  
connect to [127.0.0.1] from (UNKN  
OWN) [127.0.0.1] 39314  
test  
kommando1
```

```
benkt@osboxes: ~  
File Edit View Search Terminal Help  
benkt@osboxes:~$ nc -nv 127.0.0.1 42042  
(UNKNOWN) [127.0.0.1] 42042 (?) open  
test  
kommando1
```

Du må lage en makefile fil, og en eller flere header filer for programmet til å kompilere riktig, gjør eventuelle endringer påkrevet i makefile filen og kildefilen for å få programmet til å bygge og kjøre korrekt.

### Dokumentasjonskrav:

Du må ta et skjermbilde av programmet når det kjører, skjermbildet skal legges ved besvarelsen i en bildefil med navn oppgave4\_scrnshot.png (i samme mappe som kildefilene) OG skal i tillegg settes inn på egnet sted i tekstbesvarelsen (PDF filen).

## Oppgave 5. Filhåndtering og tekst-parsing (20 %)

Du skal lage en applikasjon som fungerer som en «kode beautifier», en applikasjon som endrer kildekode til noe som passer forfatterens kodestil (gjør koden «penere»).

Applikasjonen skal ta et filnavn til en C-source fil som parameter når den startes fra terminal. Applikasjonen skal lese denne filen og gjøre 1 endring i filen før den lagrer filen med samme navn, men lagt til `_beautified` før `.c` i filnavnet.

Applikasjonen ta alle forekomster av for-looper og erstatte disse med while-looper, det betyr at kode som dette:

```
for (a = 0; a < b; a++) {...}
```

skal erstattes med en while-loop og vil se noe slik ut:

```
a = 0;
while (a < b) {... a++; }
```

Hvor ... angir resten av koden i løkken, og naturligvis må beholdes slik den er.

Du må lage en makefile fil, og en eller flere header filer for programmet til å kompilere riktig, gjør eventuelle endringer påkrevet i makefile filen og kildefilen for å få programmet til å bygge og kjøre korrekt.

### Dokumentasjonskrav:

Du må ta et skjermbilde av programmet når det kjører, skjermbildet skal legges ved besvarelsen i en bildefil med navn oppgave5\_scrnshot.png (i samme mappe som kildefilene) OG skal i tillegg settes inn på egnet sted i tekstbesvarelsen (PDF filen).

I tillegg må både filen du har testet med, og den resulterende «beautified» filen legges ved oppgaven.

+

**Slutt på oppgavesettet**