2 ukers HJEMMEEKSAMEN
PG3401 C Programmering
Tillatte hjelpemidler: Alle
Varighet: 14 dager
Karakterskala/vurderingsform: Nasjonal karakterskala A - F
Dato: 26. april - 10. mai 2024

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 9 sider. Det er totalt 6 oppgaver i oppgavesettet.

Det er 2 ukers frist på denne hjemmeeksamen. Perioden kan overlappe med andre eksamener dere har, det er derfor viktig at dere bruker tiden effektivt og planlegger tiden godt, så dere ikke rett før innlevering blir sittende og skulle levere flere eksamener samtidig. Vær obs på at eksamen MÅ leveres innen fristen som er satt i Wiseflow, og oppgaven kan kun leveres via WISEFLOW. Det vil ikke være mulig å få levert oppgaven etter fristen – det betyr at du bør levere i god tid slik at du kan ta kontakt med eksamenskontoret eller brukerstøtte hvis du har tekniske problemer.

Det presiseres at studenten skal besvare eksamen selvstendig og individuelt, samarbeid mellom studenter og plagiat er ikke tillatt. Det er ikke tillatt å presentere andres arbeid som ditt eget – dette inkluderer arbeid utført av kunstig intelligens (som tekst- eller kode-genereringsmodeller). Eksamen skal løses på Linux.

Merk at oppgavene er laget med stigende vanskelighetsgrad, og spesielt de tre siste oppgavene er vanskeligere enn de første oppgavene. Det oppfordres derfor til å gjøre de første oppgavene (helt) ferdige slik at studenten ikke bruker opp all tid på å gjøre de siste oppgavene først.

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 leverer inn det du har gjort (selv om det ikke virker), og 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 PDF filen, kode som studenten vet ikke virker skal også være kommentert i koden. 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 være i 1 ZIP fil, navnet på filen skal være PG3401_V24_[kandidatnummer].zip. Denne filen skal ha følgende struktur:
\ oppgave_2 \ makefile
\ oppgave_2 \ […]
[…]

I Wiseflow skal du laste opp en tekstbesvarelse med navn «PG3401_V24_[kandidatnummer].pdf», og ZIP filen skal lastes opp som vedlegg til tekstbesvarelsen.

Studenter som leverer inn eksamen som avviker fra disse instruksene vil få trekk i poeng. 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 – sensor vil i shell på Debian Linux 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, men svaret skal ha akademisk kvalitet og påstander skal underbygges av akademiske kilder). Etter den rene tekstbesvarelsen skal det være 1 sides begrunnelse/dokumentasjon for hver oppgave, hver av disse begrunnelsene skal være på en ny side for å gjøre tekstbesvarelsen oversiktlig for sensor. Besvarelsen skal være i PDF format og ha korrekt file-extension til å kunne åpnes på både en Linux og en Windows maskin (.pdf). Besvarelser i andre formater vil ikke bli lest.

Det presiseres at oppgavene må programmeres på Linux (tekstbesvarelsen kan skrives på Mac eller Windows, eller andre operativsystem studenten måtte bruke).

Oppgave 1. Teori (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. Filhåndtering og funksjoner (15 %)

Last ned følgende input-datafil, den inneholder 10 ord der hvert ord er på en linje (det er linjeskift mellom hvert ord):

http://www.eastwill.no/pg3401/eksamen_v24_oppgave2.txt

Last deretter ned følgende kildefiler:

http://www.eastwill.no/pg3401/eksamen_v24_oppgave2_palindrom.c

http://www.eastwill.no/pg3401/eksamen_v24_oppgave2_heterogram.c
http://www.eastwill.no/pg3401/eksamen_v24_oppgave2_all_lower.c
http://www.eastwill.no/pg3401/eksamen_v24_oppgave2_all_upper.c
http://www.eastwill.no/pg3401/eksamen_v24_oppgave2_anagram.c
http://www.eastwill.no/pg3401/eksamen_v24_oppgave2_disjoint.c

Du skal skrive et program som leser ordene i tekstfilen, for hvert av ordene i tekstfilen skal du kalle funksjonene i kildefilene ovenfor for å finne ut om ordene er a) et palindrom, b) et heterogram, c) alle tegn i ordet er store, eller d) alle tegn i ordet er små bokstaver.

Deretter skal programmet lese det første ordet (lagre det) og lese resten av ordene, for hvert av disse ordparene skal programmet sjekke om de er e) anagrammer av hverandre, eller er f) ord uten felles bokstaver (orthographically disjoint), dette skal også avgjøres ved å kalle de tilsvarende funksjonene i kildefilene ovenfor. (Disse to verdiene kan settes til «false» for det første ordet når du skriver metadatastrukturen nedenfor, eller en annen virkemåte som du dokumenterer i koden.)

Programmet skal lage en struct og sette disse metadataverdiene til hvert ord til de resulterende verdiene - og skal deretter lage en binær utdatafil der hver STRUCT er skrevet (i binær), etterfulgt av selve ordet. Strukturen for å beskrive hvert ord skal være følgende:

```
struct TASK2_WORD_METADATA {
   int iIndex; // The index of the word, first = 1
   bool bIsPalindrom;
   bool bIsHeterogram;
   bool bIsUppercase;
   bool bIsLowercase;
   bool bIsAnagram;
   bool bIsDisjoint;
   int iSize; // The length word (number of chars)
   char szWord[]; // The word written «after» the struct
}
```

Du må opprette en makefile fil (bruk makefile fil hentet fra forelesning 4, slide 65 - med overskriften "Use this makefile") som kobler sammen kildefilene, og en header fil for hver kildefil for programmet å kompilere riktig, gjør eventuelle endringer som kreves til makefile og kildefiler for å få programmet til å bygge og kjøre riktig.

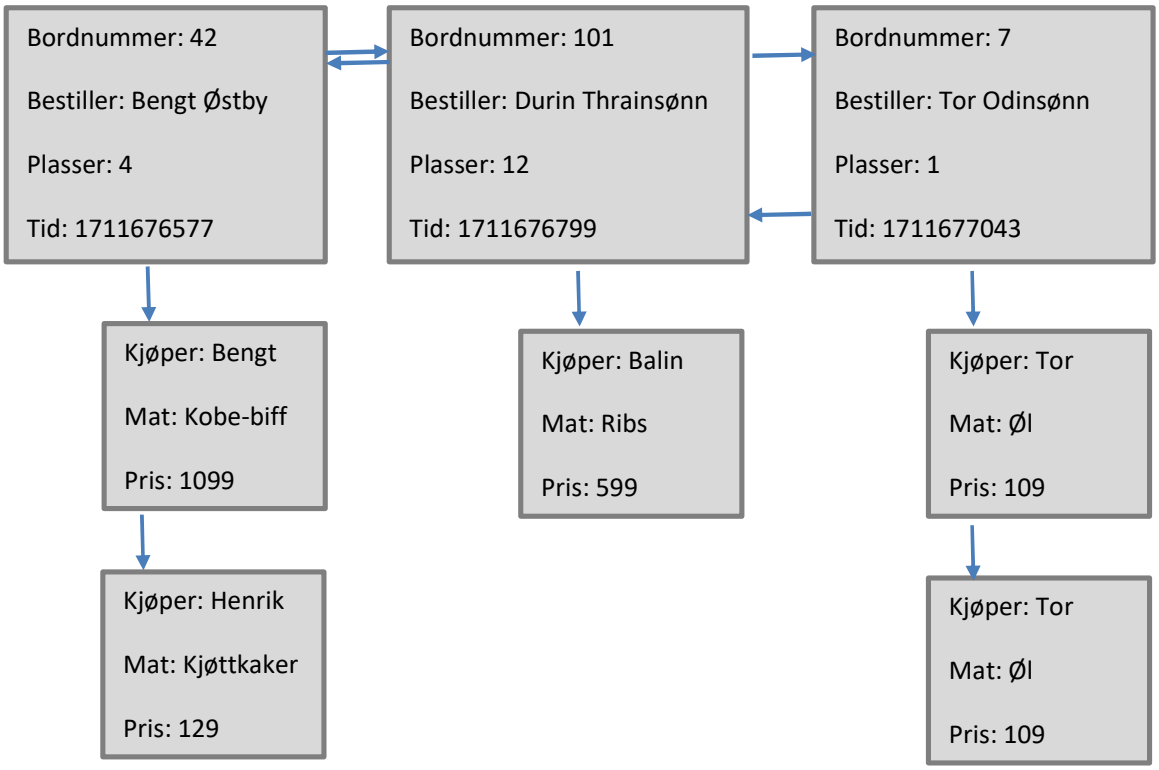
Dokumentasjonskrav:

Både inndatafilen og utdatafilen skal ligge i samme mappe som programmet, og begge filene må være en del av den innleverte besvarelsen.

Oppgave 3. Liste håndtering (20 %)

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 instrukser 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 (hentet fra forelesning 4, sleide 65), 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 oppgave3_scrnshot.png (i samme mappe som kildefilene) OG skal i tillegg settes inn på egnet sted i tekstbesvarelsen (PDF filen).

Oppgave 4. Tråder (15 %)

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_v24_oppgave4.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_v24_oppgave4_pg2265.txt

Du skal utvide koden med følgende endringer:

- Du må lage en makefile fil (bruk makefile fil hentet fra forelesning 4, sleide 65 – 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 stedefor 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_v24_oppgave4.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 oppgave4_scrnshot.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 5. Nettverk (25 %)

I denne oppgaven skal du opprette en server og klient for en "enkel meldingstjeneste", serveren og klientkoden kan være en enkelt kjørbar fil eller to kjørbare filer (ditt valg).

Du bør lage din egen protokoll for kommunikasjon mellom klient og server. Når det startes som en server (for eksempel signalisert av en parameter fra terminal, som kan være -listen), skal programmet åpne den angitte porten for LISTEN, for denne oppgaven bør du binde til loopback på 127.0.0.1 når du tester programmet (for ikke å eksponere en åpen port eksternt). Det anbefales å bruke TCP.

Oppgaven skal ikke løses ved hjelp av Curl eller andre tredjepartsbiblioteker og skal ikke være basert på å starte andre applikasjoner i operativsystemet - bare bruk av Sockets som vi har lært i forelesning 10 om Network vil gi poeng på oppgaven.

Når det startes som en server, skal programmet BINDE til en port brukeren velger, når du tester denne applikasjonen anbefales det for denne oppgaven å lytte bare på adressen 127.0.0.1 for ikke å eksponere porten utenfor din egen maskin. Det anbefales å bruke TCP. Serverapplikasjon skal startes i terminalen og kontrollere at inngangsparametere er riktige for en server, for eksempel «oppgave_5 -listen».

Når den startes som klient, skal applikasjonen kjøres med serverens IP-adresse og portnummer fra terminal, i tillegg skal en identifikator oppgis av brukeren, som vil simulere et "telefonnummer", for eksempel «oppgave_5 -server 127.0.0.1 -port 42 -telefon 1234». Når den starter, skal applikasjonen CONNECTE til den gitte porten på serverprosessen.

Du bør opprette en protokoll for server-klienttrafikk, når du kobler til serveren, skal klienten sende en "koble" -melding til serveren:

```
struct TASK5_CONNECT_USR {
   int iMagicNumber; // Set to a fixed protocol id no
   int iMessageSize; // Set to sizeof(TASK5_CONNECT_USR)
   int iIpAddress;   // Set to local client IP address
   int iPhoneNumber; // Set to «phone number»
}
```

For å unngå hendelsesdrevet programmering vil vi håndheve en begrensning at kun to klienter kan sende data (i en reell applikasjon som dette vil serveren selvfølgelig videresende hver melding til noen eller alle klienter ved hjelp av et fleksibelt antall tråder, men det vil være mer avansert programmering enn forventet i dette kurset). At

vi begrenser implementeringen til kun å tillate to klienter betyr at studenten skal lage 2 tråder som hver kaller lytt og bind-funksjonene, og deretter håndtere klientene etter at de er koblet til. De to trådene må være bundet til forskjellige porter. (Hvis du vil, kan du bestemme deg for å binde til bare én port og håndtere flere tilkoblede klienter på samme port, men merk at slik flertråd nettverksprogrammering er utenfor det som forventes i dette kurset, og det gis ingen ekstra poeng for å gjøre dette fremfor å lage to tråder og binde dem til to forskjellige porter.)

Når en klient er tilkoblet skal den akseptere input fra brukeren i form av tekststrenger i terminalen, klientprogrammet skal sende disse meldingene til serveren og serveren som skriver ut dette på skjermen. Meldingene som sendes bør bruke følgende protokoll (som kan sendes som en eller to meldinger avhengig av nettverkskode design - szMessage enten en del av meldingen eller sendt som en egen melding):

```
struct TASK5_SENDMESSAGE {
   int iMagicNumber; // Set to a fixed protocol id no
   int iMessageSize; // Set to sizeof(TASK5_SENDMESSAGE)
                     // + strlen(szMessage)
   char szMessage[]; // A variable length message
}
```

Utdataene på serverkonsollen skal se slik ut:

```
Number 1234: This is my message...
Number 5678: This is my reply to that...
Number 5678: What is your point?
Number 1234: Old men has no point, there is no point...
```

Et tips for testing av server/klientapplikasjoner er å åpne tre terminalvinduer i Linux og starte en instans av applikasjonen med "oppgave_5 -listen" i ett vindu og de to andre vinduene med for eksempel "oppgave_5 -server 127.0.0.1 -port 42 -phone 1234" og "oppgave_5 -server 127.0.0.1 -port 43 -phone 5678".

Merk at du bestemmer hvilke portnumre du bruker, 42 og 43 brukes bare her som et eksempel, og du må koble til portnummeret du har valgt å binde serverapplikasjonen til.

Du må lage en makefile fil (hentet fra forelesning 4, sleide 65), 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). Du må enten ta et skjermbilde som viser alle 3 instansene, eller ett av hver instans.

Oppgave 6. 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 3 endringer i filen før den lagrer filen med samme navn, men lagt til _beautified før .c i filnavnet.

A) Først skal applikasjonen ta alle forekomster av while-looper og erstatte disse med for-looper, det betyr at kode som dette:

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

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

```
for (a = 0; a < b; a++) {…}
```

Hvor … angir resten av koden i løkken, og naturligvis må beholdes slik den er. Det kan forutsettes at kun «enkle» løkker (som burde vært en for-loop) hvor en variabel settes til en verdi rett før løkken og løkken har en enkel test med denne verdien (som i eksempelet over) erstattes – og alle andre bruk av while beholdes slik de er.

B) Applikasjonen skal så tvinge frem bruk av Hungarian notation i koden, dette skal løses ved å finne alle variable av typen «char *» og endre navn på alle variable av den typen til å være «pszAbc» hvor Abc er variabelens opprinnelige navn, eksempel vil «char *name» resultere i at alle forekomster av variabelen endres til «pszName». (Du skal ikke gjøre dette for andre typer, da det blir for mye jobb.)

C) I tillegg skal alle forekomster av 3 mellomrom (space) erstattes med en tabulator (ASCII kode 0x09).

Du må lage en makefile fil (hentet fra forelesning 4, sleide 65), 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:

For denne applikasjonen må du også lage en test fil som du kaller oppgave5_test.c (i samme mappe som kildefilene), både denne – og den opprettede beautified-filen må leveres inn sammen med besvarelsen.

+

Slutt på oppgavesettet