INF-1100 - Lenket liste

I denne oppgaven skal du skrive en lenket liste for å fullføre et program som gjør en biologisk simulering. En lenket liste er en dynamisk datastruktur som kan holde vilkårlig mange elementer i rekkefølge. Mange av dere er allerede kjent med lister fra andre programmeringsspråk som Python.

I tillegg må du fullføre en nesten-ferdig garbage collector ved hjelp av en AI. En garbage collector er en mekanisme som frir opp minne når det ikke lenger brukes.

Den vedlagte prekoden er en simulering for bakteriell vekst. Detaljene i dette er ikke så viktige, men både simuleringen og garbage collector bruker en lenket liste og dermed vil ikke programmet fungere uten at lista er implementert.

Tips: start tidlig.

Oppgavebeskrivelse

Din oppgave er å implementere en *lenket liste* — https://en.wikipedia.org/wiki/Linked_list — dette er en helt grunnleggende datastruktur, og det er en standard øvelse å implementere en. For å hjelpe deg på vei er alle funksjoners signatur allerede gitt i `list.h`. Du må implementere disse i `list.c`.

I tillegg skal du bruke en AI — https://chat.uit.no — til å fullføre fila `gc.c`. Her er det en funksjon som ikke er fyllt inn; din oppgave er å fylle den inn med fungerende kode. Når du har gjort dette skal du på korrekt vis sitere bruk av AI som ekstern kilde. Det er detaljer om hvordan dette bør gjøres nedenfor. Filene `gc.c` og `gc.h` implementerer en *reference counting garbage collector* — https://en.wikipedia.org/wiki/Reference_counting — som brukes i simuleringen i `siumlation.c`.

Merk at den lenka lista skal du implementere selv uten bruk av AI.

Funksjoner

Simuleringskoden og garbage collector bruker en håndfull uimplementerte funksjoner beskrevet i `list.h`. Du må implementere dem i `list.c`. Et sett med funksjoner handler om å håndtere lister i seg selv:

- `list_create` lager en ny liste
- `list_destroy` frigir minnet som lista bruker (liste + noder), men *ikke elementene i lista.*
- `list_addfirst` legger til et element først i lista
- `list_addlast` legger til et element i slutten av lista
- `list_remove` fjerner et element fra lista, men frir kun opp noden i lista, ikke elementet selv.
- `list_size` returnerer antall noder i lista

Det andre settet med funksjoner handler om å iterere over lista element for element (se f.eks. `simulation.c` for bruk av iterator):

- `list_createiterator` returnerer en ny iterator over gitt liste
- `list_destroyiterator` frigir minnet til en iterator
- `list_next` flytt iterator til neste element i lista, returner nåværende element
- `list_resetiterator` returner iterator til første element i lista

Funksjonen som skal fullføres vha en AI er er `gc_free` i `gc.c`. Et tegn på at denne fungerer som den skal er at simuleringen går fra å bruke flere gigabyte med minne til å kun bruke noen få megabyte.

Prekode

Prekoden inneholder følgende filer:

- `Makefile`: brukes for å bygge simuleringen
- `gc.h`: deklarerer funksjonene som garbage collector tilbyr
- `gc.c`: implementerer funksjonene fra `gc.h`. Denne fila må fullføres av deg.
- `list.h`: deklarerer funksjoner for en lenket liste.
- `list.c`: implementerer funksjonene fra `list.h`. Fila er tom, du må fylle den med kode.
- `simulation.h` deklarerer hjelpefunksjoner brukt i simuleringen
- `simulation.c` implementerer simuleringen.

Det skal ikke være nødvendig å endre noe annet enn filene `list.c` og `gc.c`, men du har lov til å endre hva som helst hvis du synes det er nødvendig. Merk at hvis du endrer signaturene i en `.h` fil må du også endre alle steder hvor funksjonene brukes.

Rapport

Du skal levere en rapport om løsningen din. Rapporten har en begrensning på 3500 ord. Du kan skrive på norsk, nynorsk, eller engelsk.

Rapporten skal ha følgende struktur:

- Navn
  - Hva heter du?
- Introduksjon
  - Fortell kort hva som er i denne rapporten og hvorfor
- Teknisk bakgrunn
  - Kort bakgrunnsinformasjon om valgt datastruktur og liknende.
- Design og implementering
  - Hvordan henger systemet ditt sammen helt overordnet?
  - Hva er viktige detaljer ved implementeringen av systemet?
- Eksperimenter (valgfri, skal være med hvis du har valgt å gjøre eksperimenter)

    - Har du gjort noen eksperimenter for å teste hvordan løsningen din fungerer? Beskriv dem her.
- Resultater (valgfri, skal være med hvis du har valgt å gjøre eksperimenter)
    - Gjorde du noen målinger som del av eksperimentene? Vis dem her.
- Diskusjon
    - Hva er fordeler og ulemper med hvordan du valgte å løse oppgaven?
    - Hvordan tolker du resultatene fra eksperimenter? (hvis du gjorde noen)
- Konklusjon
    - Gi et kort sammendrag av hva du gjorde og hva du har lært.

Referanser, plagiat, etc.

Det skal vises tydelig hvor man henter kunnskap. Dette gjøres ved sitering av kilder slik som her [1]. Hvis man kopierer noe direkte fra en kilde (direkte sitat), så må dette også tydelig merkes slik som her:

> For a number of years I have been familiar with the observation that the quality of programmers is a decreasing function of the density of **go to** statements in the programs they produce. [2]

Hvis du bruker kode fra andre steder enn prekoden må dette også merkes tydelig:

```
/* denne funksjonen har jeg hentet fra www.eksempel.no/exp-funksjon */
float exp(float x) {
 ...
}
```

Poenget er at det skal være helt tydelig hva som er ditt eget arbeid og hva du ikke har gjort selv. Det er lov å bruke kode fra din egen løsning av obligatorisk eller frivillig oppgave. Dette skal også merkes, f.eks.:

```
/* denne funksjonen skrev jeg selv til løsningen av arbeidskravet i INF1100,
   høst 23. */
int factorial(int n) {
 ...
}
```

Hvis man ikke siterer kilder i rapport eller kode blir dette tolket som plagiat, som er en form for fusk. Fusk har potensielt store akademiske konsekvenser nærmere forklart her: https://uit.no/eksamen

Forsvarlig bruk av AI

AI-verktøyer som copilot og ChatGPT (gratis tilgang på https://chat.uit.no) må også siteres som bruk av eksterne kilder. En tommelfinger-regel er at å bruke kode som disse verktøyene har skrevet er omtrent som å bruke kode funnet i en annen kilde: det er helt OK å gjøre det men du må være tydelig på at det er

det du gjør. Bruk av AI til skriving av kode merkes altså på lignende måte, den store forskjellen er at man skal legge ved chat-loggen:

```
// denne funksjonen ble skrevet med chat.uit.no; loggen er vedlagt som fil
// chatlog_1.txt
void swap(int *a, int *b) {
 ...
}

// Denne funksjonen har jeg debugget med hjelp fra chat.uit.no; loggen er
// vedlagt som fil chatlog_2.txt
void gc_free(...) {
 ...
}
```

Hvis man bruker en AI til noe som ikke direkte har med koden å gjøre, f.eks. språkvask av rapporten og denslags, nevnes det i rapporten, og chat log legges ved.

Gruppearbeid

Dere må gjerne jobbe i grupper på 2. Dette gjøres ved å melde seg inn i en gruppe i Canvas og så kan en levere for dere begge.

Innlevering

Oppgaven leveres i to deler: Rapporten leveres som hoveddokument i `.pdf`. Koden leveres som et vedlegg i `.zip` med følgende struktur (merk at for å lette rettingen vil vi ha rapport i `.zip`-fila også):

- `username000-inf1100.zip`
  - `src/`
    - `Makefile`
    - `gc.c`
    - `gc.h`
    - `list.c`
    - `list.h`
    - `simulation`
    - `simulation.c`
    - `simulation.h`
  - `rapport.pdf`

Har du laget flere filer som en del av løsningen må naturligvis disse også være med.

Vurderingsskjema

I denne oppgaven krever vi at dere bruker en AI (feks ChatGPT) til å fylle inn `gc_free`. Vi krever også at dere har sitert bruken av dette verktøyet riktig, som beskerevet over. Det er automatisk ikke-godkjent hvis disse to punktene mangler.

I tillegg til dette, bruker vi retteskjema nedenfor. Hvis hjelpelærer kan svare "ja" på 6 av 8 spørsmål nedenfor vil du få godkjent denne oppgaven.

- Er listefunksjonene implementert?
- Er iteratorfunksjonene implementert?
- Er det valgt beskrivende variabelnavn i koden?
- Er det skrevet gode/forklarende kommentarer i koden?
- Følger rapporten strukturen som er gitt i oppgavetekst?
- Beskriver rapporten strukturen som er brukt til liste/iterator?
- Beskriver rapporten mekanismene bak liste/iterator?
- Er evt. svakheter/problemer med løsningen beskrevet i rapporten?

Referanser

- [1] Kernighan, B. W., & Ritchie, D. M. (2002). The C programming language.
- [2] Dijkstra, E. W. (1968). Letters to the editor: go to statement considered harmful. Communications of the ACM, 11(3), 147-148.