Dokumentasjon for eksamen

Oppgave 2 Dokumentasjon

Denne oppgaven gikk egentlig ganske greit, jeg hadde jo laget bubble sort fra før av, så jeg bare gjenbrukte litt av den koden, og skrev om det jeg trengte.

Startet vel egentlig med å lage den structen som man skulle ha, som skulle inneholde en string, en int, og en peker til arrayet den er en del av. La også pragma pack() over og under for å få fjernet padding I structen (det har jo ikke så veldig mye å si I så små oppgaver. Så jeg kommer nok kanskje ikke til å gjøre det på de neste oppgavene). Etterfulgt kommer prototypene til funksjonene, og disse ligger da I header filen.

Så var bare bare å leke seg litt med å sette informasjon I structen, og bruke argv[] sine argumenter som input-string, og strlen() av den stringen, og gi hver node en peker til arrayet, som blir opprettet I main.

Så var det bare å kjøre BubbleSort funksjonen, som jeg hadde gjort mye av grunnarbeidet fra før av, og lage noen printf() som viser hvordan de ligger iforhold til hverandre usortert og sortert. Denne funksjonen blar jo da igjennom arrayet og “Swapper” plass om det trengs og printer ut riktig informasjon til slutt.

Siste funksjonen GetIndex() tar imot en node (en struct) og begynner å loope igjennom starten av arrayet til den finner riktig node, og printer da ut index plasseringen til oppgitt node. Index’en får jeg da ved hjelp av en int counter som inkrementerer for hver gang den ikke finner riktig element.

Til slutt kjørte jeg en “valgrind ./main c b a” for å sjekke om det var noen minnelekasjer noen sted, og så det manglet å frigjøre 3 stykker, så derfor la jeg til free() I loopen etter hver node ble laget.

Har også kommentert ut 2 GetIndex() tester som du kan teste om du vil.

Oppgave 3 Dokumentasjon

I denne oppgaven var jeg heldig å hadde gjort mye av grunnarbeidet på forhånd, så det jeg startet med var å jobbe ut ifra den koden jeg allerede hadde. Lagde noen nye filer og gjorde om litt på makefilen selvfølgelig.

Det var noe av kodesnutten min som ikke fungerte optimalt fra før av, så gjorde litt forbedringer der til å starte med.
I den oppgaven jeg hadde gjort tidligere så var input hardkodet i main filen (ikke fra bruker input) så til å starte med så prøvde jeg meg frem med hardkodet input, og startet å lage de nye funksjonene som eksamensoppgaven krevde. Da brukte jeg forsovet også den gamle structen som var litt annerledes enn hva oppgaven sa.

Jeg startet med å lage funksjonen for å "Legge til et element i listen" noe som var ganske enkelt.
så lenge man har en struct med pNext,pPrev, og en som holdt styr på hva som var head og tail. Så var det bare å flytte pekerne til det riktige elementet.
(om det ikke var noe fra før av, så ble head og tail = nyNode) og om man skulle legge til noe etter der så var det bare å sette pekeren fra forrige node til nyNode og sette nyNode som en tail, osv..

"Hente ut element N fra listen" hvor første element var "index 1", så var det bare å loope igjennom listen (til den ble null) med en int som holdt styr på hvilken index man var på (som startet på 1 selvfølgelig) og printet bare ut all informasjon om det elementet når man traff den indexen

"Finne element som matcher navn i listen", mye av det samme som det over, bare at man leter etter node->name.

"Slette alle elementer i listen som har et gitt navn" litt av det samme.
- loope igjennom listen
- sjekke om node->name = inputNavn
- flytte litt på pekere (node->pPrev->pNext = node->pNext) og motsatt, for å "fjerne" noden
- free()

"Sletter alle elementer som har en alder lavere(som jeg valgte) enn input", mye av det samme her også
- loope igjennom listen
- sjekke om (node->age < N) og fjerne noden på samme måte som over.

og hvis man fant noden så huske å bruke free()

Etter jeg hadde laget disse funksjonene så begynte jeg å fikse på structen sånn at det ble riktig (var vel nesten bare å bytte litt navn + legge til en til variabel)
Så fant jeg frem en av switch-case oppgavene jeg hadde gjort tidligere og fjernet den hardkodede "input" jeg hadde og spurte heller bruker om de kunne taste inn et tall mellom 1-6, med forksjellige alternativer for hvert tall.

og så flyttet jeg inn de riktige funksjonene til å gjøre hva de skulle om bruker tastet inn "riktig" tall.

Siden jeg brukte getchar() så valgte jeg å escape unna '\n' ved å sjekke om det i øverst av while loopen, for så å "skippe" switch-casen med en case for "\n" som bare breaker.

Jeg valgte å spørre bruker om de vil fortsette for ryddighetens skyld (hva jeg syntes passet best ihvertfall), while-loopen min vil avslutte om input = 6, eller om da bruker svarer noe annent enn “ja” når det spørres om bruker vil

fortsette, hvis bruker ikke da svarer ja, så setter jeg bare input = 6, sånn at while loopen breaker, og jeg kommer ut av loopen, sånn at jeg får bladd igjennom linked listen og får frigjort minne til hver node.
Jeg har også valgt å ha 2 .c filer, en som inneholder main() og printf-delen og en fil som inneholdt funksjonene, siden det ble så mange funksjoner. Begge har jo da #include “Oppgave3.h”

Helt til slutt fikset jeg litt opp i noen bugs som oppsto etter bruker hadde gjort litt forskjellige alternativer :)

Vil få noen warnings når man compiler koden, fordi jeg ikke "sjekker" return valuen til scanf (Kunne "fikset" de ved å lage noen if-setninger til scanf(), men det ble uoversiktlig og unødvendig) derfor har det ikke noe å si i dette tilfelle.(mulig din kompiler ikke klager på dette)

Oppgave 4 Dokumentasjon

Først så tok jeg å skrev koden i en .c fil, og kjørte den for å se om det kom noen feil. Det var noen få ting som måtte til for at koden skulle kompiles. sånn som noen includes stdio stlib math osv. koden ville fortsatt ikke kjøre fordi den ikke fant pow(), så jeg man-paget pow i shell og fant ut at det trengtes -lm når man kompilet filene så jeg la det inn i makefilen min. Jeg tviler på at dette var en av "feilene" i koden, men er verdt å nevne.
Etter dette kompilet koden helt fint, og det kom ingen feilmeldinger og når man kjørte executablen så skjedde ingenting :)))))

Så det jeg startet med var jo å prøve å skjønne hva som skjedde, så jeg laget printf() på nesten alle ting som jeg ikke skjønte hva gjorde, og fant etterhvert ut av noen forskjellige ting. Jeg regnet jo med at det hadde noe med å gjøre med det siste tallet i andre kodeblokk av kredittkortet (som nevnt i oppgaven) så jeg prøvde å teste ut litt forskjellige funksjoner for å se om jeg kunne spotte noen feil. (For en kjip kode å lese..) Men når jeg printet ut den rare for-loopen for å "Calculate the cardnumber as a 64 bit integer" så fikk man et ganske stort minus tall, og så at for-loopen kjørte 5 ganger istedenfor 4 som den skulle. Husker ikke rekkefølgen jeg løste de forskjellige feilene på, men nevner 2 her og nå fordi det er de jeg har funnet så langt.

Den ene feilen var jo at når andre segment startet på 123x så skulle noe skje, og hvis tallet 'x' var 9 så kunne det bli feil. Det var fordi (cc->p->digit[cc->p->digit[3]/((cc->p->digit[3]-'0')%9)]) blander characters og tall, så man ville fått ascii-verdien av hvert tall, og noen operasjoner ville blitt utført, ihvertfall så er det en (% 9) på slutten som gjorde at (cc->p->digit[3]-'0') ville blitt '9' - '0' = tallet (9) også ville man gjort 9 % 9 som blir 0, også ville man tatt første del av den koden (cc->p->digit[cc->p->digit[3]) som er '9', så hvis man tar 9 / 0 så blir det feil, dermed var dette feilen, det med 9 % 9 = 0 også dele et tall med 0 går ikke i c. Dette var selvfølgelig hvis char *c = "42421239xxxxxxxx"
Jeg lar også mine printf() og kommentarer bli igjen i filen, så du kan se litt hvordan jeg har tenkt. (kanskje kjipt for deg å lese, men da ser du litt hva jeg har gjort/tenkt) måten jeg løste det på var å sette (j = cc->p->digit[3];) fordi alt man ville var jo å få tak i det 4.tallet i segmentet. Denne if-setningen triggres jo bare hvis "node" nr2 = "123x", Hvis man ikke hadde fikset det problemet her og prøvd å printe ut hva J er, så hadde man fått "Floating point exception(core dumped)".

Feil nr 2, fant jeg ut når jeg printet ut for-loopen og så at den ble gjort 5 ganger, FORDI man mallocer minne til pc->p altså neste "blokk" før man gjør noe med den, i while-loopen tidligere i koden. har også kommentert litt rundt det. Det gjorde at kreditkortnummere så ganske riktig ut, men den siste for-loopen la til en 0.0001 value til kortnummeret, noe som blir oversett pga liten verdi, (fjern den if-setningen jeg la til i while-loopen for å se hvordan det ser ut), det vil også si at jeg løste det problemet ved å lage en if-setning rett etter at i-pointern har inkrementert. Sånn at den ikke mallocer neste del. Det er jo ikke en optimal løsning, men en midlertidig fiks på problemet :D

Til slutt så kom jeg på at jeg kunne bruke valgrind, for å sjekke etter minnelekasjer, så jeg skrev i shell "valgrind ./main" og fikk vite at "total heap usage: 5 allocs, 2 frees, 1,120 bytes allocated", noe som fikk meg til å tenke på at man allokerer minne til pc->p uten å free() de, så hvis man husker å frigjøre alle de 3 "p'ene" på slutten så er problemet fikset.
free(cc->p->p->p);
free(cc->p->p);
free(cc->p);
også i denne rekkefølgen sånn at man klarer å få tak i de siste "blokkene" i linked listen

Til slutt så lagde jeg den utløpsdato sjekk-delen av oppgaven.

Hvor jeg da henter inn lokal tid, med include <time.h> og time_t variabel og struct tm, også fant jeg ut hvordan jeg fant måneder og år ved hjelp av den innebygde structen.
Så valgte jeg å gå for strtok(), men det funket ikke på char *e derfor la jeg over hver bokstav fra e over i en char temp[]-array og fikk så gjort en strtok() på den, hvor jeg da hadde "/" som delimeter, så gjorde at første del av stringen ble måned, så lagret jeg den verdien i en int variabel, også tok jeg en strtok() til for å få neste del av stringen, altså år, og lagde en variabel for år.
Etter det var gjort så var det bare å lage noen if-setninger om utløpsår > inputår, osv osv. sjekk selve koden..

Det er selvfølgelig mange ting her som er helt unødvendig, sånn som å lage en char *i = (char*) e, int a og int n blir jo ikke brukt osv, men disse småfeilene har jeg ikke gjort noe med. Jeg legger ved 2 filer, hovedfilen som vil bli kompilet når du kjører make (som inneholder mange av de printf() jeg brukte og litt kommentarer, for å vise hvordan jeg tenkte, var litt usikker hvordan jeg ellers skulle vise deg det) og en fil som heter Oppgave4Clean.c, den inneholder for det meste det samme, bare at jeg har fjernet mye rot, med fremprovoserte feil også. for å kompile denne kan du da skrive noe som "gcc -o main2 Oppgave4Clean.c -lm", husk å få med -lm for pow :)

Måten jeg tenkte å fremprovosere feilene på var å printe j, i den for-loopen som sammenligner node nr 2 og "123x", og printe ut litt hva som skjer i den for-loopen som legger til tall for llCreditCard
fordi den kjørte jo 5 ganger (nå bare 4). Minnelekasjen var jeg ikke helt sikker på om jeg fikk fremprovosert, på denne måten.

Dokumentasjon 5

Denne oppgaven tok litt tid å gjennomføre med tanke på at vi ikke har gått igjennom base64 converting i timen, så det måtte læres på egenhånd. Så det hele startet med litt research for å forstå hvordan man skal converte fra enkodet b64 tekst til "vanlig", og hva som faktisk skjer. Første del var jo ganske grei, bare lagre den oppgitte teksten fra eksamensoppgaven, i en txt fil, og lage en FILE input og output for å lese av inputfilen og skrive til outputfilen, med "r" og "w". Så lager jeg en buffer for all teksten i inputfilen, og bruker fscanf() for å hente hva som står i inputfilen, printer også ut hva den enkodete teksten er. Etter det så har jeg lagd en funksjon for å håndtere b64 dekodingen.

For å konvertere den enkodete teksten til dekodet, så trenger vi en dekoding table, som inneholder alle tegnene i base 64 index table, så vi referer hvert tegn i dekoding tabellen vår ved hjelp av ascii tabellen, og som vi da ser så representerer vi alle ascii tegnene i rekkefølge etter ascii tabellen og det første tegnet som oppstår i ascii tabellen som er i base64 er "+"-tegnet og derfor starter vi dekoding tabellen vår med 62 ('+' i base64 tabellen er index 62) også tar vi hver ascii karakter helt ned til 'z' men i ascii tabellen mellom '+' og 'z' så finnes bokstaver som ikke er i base64 tabellen, så de gir vi en value som -1 (altså en placeholder for tegn som ikke er i base64 tabellen), på denne måten får vi "koblet" alle ascii tegn og base64 tegn som vi trenger, og derfor vil denne tabellen se rar ut ved første øyekast.

Når vi har dekoding tabellen, så var det bare å starte på funksjonen som skal gjøre om hvert enkelt tegn fra den enkodete teksten til dekodet, og enkodet b64 tekst går utifra 6-bits values med b64 tabellen, så derfor må vi gjøre det om til 8 bits istedenfor, så vi kan konvertere de enkodete bokstavene til dekodet (ascii tegn). I eksempelet under her viser jeg litt hvordan det funker med de 4 første bokstavene fra den enkodete teksten, og viser litt utregning. Minner også om at skaleringen mellom enkodet/dekodet er 4/3, og derfor kan vi sette outputLength = lengden av inputdata/4*3 -1 mener jeg var for '=' tegnet som jeg kommer tilbake til litt senere.

QXJi i dekoding tabellen er da lik:
Q = 16 = 010000 <- 6 bits
X = 23 = 010111
J = 09 = 001001
i = 34 = 100010

Setter de ved siden av hverandre i 8 bits mønster.
aka QXJi = 0100 0001 0111 0010 0110 0010

Første bokstav i ascii = 0100 0001 = 65 = A
Andre bokstav i ascii = 0111 0010 = 114 = r
Tredje bokstav i ascii = 0110 0010 = 98 = b

så "QXJi" enkodet blir da "Arb" dekodet, og vi ser også at det stemmer med output vi får, som er "Arbeidet med dette emne ....".

Så det vi gjør videre i funksjonen vår da er å lage en for-loop som blar igjennom hvert bokstav av inputdata og lager 'i' og 'j' pga 4/3 skaleringen og plusser da i += 4 og j += 3, vi lagrer hver value[i]-43(pga vi går utifra asciitabellen hvor ‘+’ starter på 43 og gjør en left shift 6(0000 0001 << 6 == 0100 0000) og gjør en OR operasjon på den valuen, med den orginale verdien. Når vi skal lagre hver enkelt value i outputten vår så right shifter vi 16 og gjør en AND operasjon med 0xFF(1111 1111) På denne måten får vi endret de binære verdiene fra b64 til ascii binære tall, som i eksempelet mitt over. Legg også merke til at vi sjekker etter '=' som egentlig ikke er et tegn i tabellen, men det betyr padding, altså om det blir leftovers fra konverteringen, og derfor må vi flytte bitsene på en litt annen måte. Poenget er at vi vil mellomlagre 4 b64

bokstaver på 6bits til og gjøre de om til 3 8bits asciibokstaver, sånn som eksempelet.

Etter jeg har kjørt denne funksjonen så legger jeg til en 0-terminering på output-stringen sånn at alt skal bli riktig. Så er det bare å bruke fputs() med output tekst til outputfil, for at filen skal inneholde den dekodete teksten. Har printet det i konsollen også.

Jeg har ikke med så mye errorhandling her, men siden jeg vet at det bare er 1 tekst vi skal konvertere og jeg vet at den skal la seg konverteres, så har jeg latt mange av de sjekkene forbli.

Oppgave 6 Dokumentasjon

Det startet med at jeg lagde en struct som inneholdt en char array med 11 plasser, bruker array[11] til 0-termineringen
Så lagde jeg en tekstfil.txt fil og skrev noe random der, for å teste om ting funket.
Så lagde jeg en FILE pointer for å lese av filen, som jeg bruker fread() for å lese 10 og 10 bokstaver om gangen for så å plassere de i structen sin char[], Jeg satte også [11] = '\0' med mindre lengden av det som ble lest var mindre enn 10 bokstaver, da satte jeg den på slutten selv med buff[strlen(buff)].

For å veksle mellom main-tråden og arbeids-tråden så valgte jeg å bruke semaphorer, fordi jeg hadde lyst til å prøve det, er ikke sikkert det er den beste løsningen, men det funker helt fint.
Så i main så initialiserte jeg 2 navngitte semaphores med sem_open(), O_CREAT er et flag som jeg bruker I sem_open() og lager semaphoren om den ikke finnes fra før av (include <fcnt1.h> for O_constants). MODE er noe jeg definet(sammen med navnet) og satte til 0644, 0 = prefix til octal, 6 = read/write permisison til user, 4 = read(group), 4 = read(others), dvs “rw-r--r--” det viktigste er at (user) aka oss, har read/write, de andre er ikke så farlig I dette tilfelle(kunne hatt andre verdier) dette blir også brukt til å lage semaphoren.
Til slutt satte jeg valuen til 0 på den ene og 1 på den andre, sånn at jeg kunne bruke sem_post() og sem_wait() for å inkrementere/dekrementere valuen, sånn at de "låste" seg for hver gang en av de skjedde.
så main funksjonen starter med en sem_wait() med den første semaphoren som var 0, og siden jeg starter arbeidstråden sin funksjon med sem_wait() på semaphor2 som var = 1, så blir den = 0(aka arbeidstråd starter først) og avslutter i while-loopen med sem_post() som inkrementerer den valuen, sånn at main kan gjøre sem_wait, noe som da gir et klar-signal til main tråden at det er den sin tur, og sånn veksler jeg imellom de 2 trådene.
Jeg legger også til tallet 10 for hver gang arbeidstråden kjører til en int i structen som da holder styr på hvor mange bokstaver det er totalt. Om filePointern er på slutten av filen(feof) så bruker jeg strlen(buffer)-1 itilfelle bufferet ikke har 10 tegn.
Jeg åpnet også en tråd I main funksjonen som jeg da brukte til å kjøre funksjonen jeg ville at den tråden skulle handle. (pthread_create())
Jeg fant også ut av at jeg ville lagre filnavnet i structen min, og sendte da med hele structen til tråd funksjonen. I structen har jeg også en selvlagd boolean som sier ifra hvis arbeidstråden har nådd feof(), sånn at main tråden også vet når den skal stoppe.
Merk også at hvis det er en newline I tekstfilen, så telles det også som et tegn, når jeg leser av filen.

Til slutt, huske pthread_join() for å sikre at tråden blir ferdig før main, fclose(fp) og sem_close() på begge semaphorene.

Husk at jeg bruker ubuntu, sånn at gcc’en din (I makefilen) blir riktig hvis du skal teste ut
-pthread -lrt (bruker jeg)

Dokumentasjon Oppgave 7

Denne oppgaven var ganske grei ettersom jeg hadde gjort deler av oppgaven fra før av.

Men det jeg har gjort var å lage en mappe for server og en for client, og gjøre klart makefiles osv..

Server filen:

I main() så looper jeg først igjennom hele argv[] som man sender med når man skal kjøre koden, og lagrer port og id om man har skrevet -port og -id etterfulgt av da portnr/id-navn.
Så lager jeg en socket, og så setter jeg client settings, så bind() man socket med en adress,etterfulgt av listen() for å lytte på socketen etter connections, så accepter() man til slutt, som da venter til en client har knyttet seg. Etter det så kjører jeg funksjonen som skal "snakke" med klient/server, og huske å close socketen når man er ferdig.

Funksjonen har en evig loop, for å kommunisere med en klient, måten jeg har valgt å løse oppgaven på er å hardkode de første 2 meldingene mellom klient/server, så først mottar server ingenting, bare for at server kan sende tilbake ID noe som blir laget I main, og blir sendt med funksjonen. Slik jeg tolket oppgaven så, hvis da bruker skriver 'Y' så fortsetter kommunikasjonen og man kan da sende så mange meldinger man vil til hverandre annenhver gang(fra server og klient terminalen), hvis man ikke skriver Y, så valgte jeg å avslutte både server og klient (I terminalen, kanskje ikke så realistisk men, det kunne jeg fort endret om jeg ville det). For at kommunikajsonen skal fortsette så bruker man read(), for å sende informasjon fra server så bruker jeg en while-loop som kjører så lenge det ikke har blitt tastet inn enter-knappen(\n) med getchar(); og lagrer så hver bokstav I et buffer som jeg sender med write(), for så å read() igjen, sånn kommuniserer server/klient med hverandre, helt til man trykker ctrl+c eller at klient ikke skriver 'y' på første spørsmål.

Klient filen:

I main() så henter jeg inn hvilken server(port) man skal koble seg opp imot, på samme måte som server-filen. Åpner en socket, å bruker 127.0.0.1(localhost) til loopback. Så connecter() man, og kjører funksjonen sin.

Funksjonen har en evig loop, som server. Og jeg har valgt å "hardkode" de første 2 meldingene imellom server/klient, den første sender en tom melding, som nevnt over. Den andre lagrer ID som man får tilsendt fra server'en og spør om man vil fortsette ved å skrive 'Y' eller noe annet for å avslutte. Hvis man fullfører handshaken så får man en melding om at handshake var successful. Ellers er den ganske lik som server, read() for å vente på svar fra server, write() for å sende meldinger, og huske å bzero()/memset() på bufferet for å fjerne det som var der. Memset skal være en bedre variant av bzero() så bruker heller memset.

Så for å kjøre server må man skrive "./main -port 6666 -id MittNavn" (eksempel portnr) og for å kjøre klient da må man skrive "./main -server 6666"
Hvis man skal bruke en port mindre enn 1024 så må man bruke "sudo" før "./main .."

Til slutt vil jeg bare si at det har vært en bra eksamen, og jeg har lært masse underveis også :)