Hva kan C programmeringsspråket brukes til?

Kort fortalt? Alt.

Den lengre forklaringen er at det spørs. Det er som andre programmeringsspråk. Det kan være lettere å bruke C til visse områder, mens det kan være raskere å bruke andre. Det kan være mer tungvint hente data fra en database dersom en bruker C, da en må skrive all nettverkskommunikasjonen selv i tillegg til tilgangskontroll og enkryptering/dekryptering. Mens dette fortsatt er mulig i C, vil det ta eksepsjonelt mye lengre tid.

Der C skinner derimot er når det gjelder effektivitet og portabilitet. I C har du total kontroll over hva som skjer, og kan derfor brukes på gjenstander med begrenset minne. Dette er et mindre problem i dag enn tidligere, men det er ennå ting (som bankterminaler, automater osv...) som ikke har råd til å kaste bort minne på lite optimaliserte oppgaver. En bankterminal har kanskje ikke plass til å laste ned hele JVM for å kunne kode i Java. Dersom ytelse og størrelse er viktig, er C en champion.

Hvem er Linus Torvalds og hva er han kjent for innen Informajonsteknologi?

Linus Torvalds var en programmerer med mastergrad i computer science. Han er kjent for å ha laget linuxkjernen, og har blitt berømt på grunn av dette. Linus er skaperen av Git versjonskontroll. Han har videre statert at han ikke lenger anser seg selv som en programmerer, og det nærmete han kommer det igjen er ved å skrive pseudokode. Fun fact: Hans kone er flink i karate.

Forklar hva kommandoen sudo gjør i Linux shell, og hvorfor det brukes (ofte).

Sudo kommandoen lar deg kjøre program som rot-brukeren. Dette går over de instillingene hvor du kan sette tillatelser per bruker. Tidligere var kommandoen **su** brukt, men den ga flere problemer som **sudo** har løst. Da man brukte **su** måtte en huske å logge ut av prosessen igjen, eller risikere å kjøre andre kommandoer mens en fortsatt er logget inn. Den ga heller ikke noen log på hva som skjer, noe som gjorde feilsøking vanskelig. **Sudo** løste dette ved å la brukeren kjøre kommandoen som rot-bruker én gang, og logge det.

Oppgave 2 – Dokumentasjon

Startes med ./main TEKST TEKST TEKST …….

Denne oppgaven var kanskje den jeg slet mest med. Startet oppgaven med å lage linked list, og bubble sorte nodene. Etter nærmere undersøkelse av oppgaveteksten måtte jeg gjøre den om til en array av strukter.

DISCLAIMER: Jeg har kodet i Ubuntu, så det vil være scanf og tilvarende som returnerer til en ubrukt variabel. Dette var kun for å fjerne irriterende feilmeldinger i terminalvinduet. Jeg vet at det er mulig å bruke dette som en sjekk for riktig input, men valgte å ikke gjøre dette. Jeg har også prøvd koden på Debian, som kjørte programmet annerledes. Jeg fikset dette, men fikk ikke mulighet til å gjennomføre samme tester på resten av oppgavene.

Oppgaven består som følger:
Main tar inn argc og argv. Sjekker om riktig input er tastet i terminalvinduet. Legger inn teksten som ble skrevet inn, lengden på teksten, en id, størrelsen på arrayet og en strukt. Inne i denne løkken var jeg nødt til å skrive inn en tom printf for å ikke få buffer overflow. Jeg har ikke funnet noen grunn til hvorfor, så jeg lot det bare være sånn.

Jeg sorterer så arrayet i en vanlig bubble sort funksjon, og printer ut arrayet på nytt, og denne gangen oppdaterer id på hvert element, og setter alle elementene til å peke på første element igjen.

Oppdaterer alle elementene med hvilket element brukeren vil finne, og sender kun ett element av strukten.

Jeg oppfattet oppgaven som at denne funksjonen skulle kun skrive ut ETT element, så jeg valgte at brukeren skriver inn elementet den vil at funksjonen skal skrive ut indexen av. Med dette vet brukeren hvilken plass i arrayet elementet er, men ikke funkjonen. Funksjonen skal så finne ut av dette helt selv og printe det ut. Det er mulig å skrive inn en tekst og printe ut alle som har samme navn med informasjon om plassene deres, men det er ikke slik jeg

Oppgave 3

Startes med ./main

Oppgaven starter med at main kjører menu funkjonen. Denne skriver ut en meny som styres av en switch case. Alternativ 1 legger til en person i listen, separert med space. Det er også mulig å skrive inn informasjonen hver for seg, ved å trykke enter imellom hvert steg. Elementene blir lagt til foran i listen, uten å ha noen spesiell grunn til dette.

Alternativ 2 finner et element ved å sammenlikne navn. Her itererer den bare igjennom elementene og sammenlikner om navnene er like. Denne blir brukt med en liten justering når man skal slette.

Alternativ 3 skriver ut en ny meny med nye valg rundt å slette elementer.

Alternativ 4 er å printe ut hele listen, veldig lik metode som find()

Alternativ 5 avslutter programmet.

Menyen gir en feilmelding dersom en verdi er ugyldig.

Den nye menyen har alternativ fra 1-4 med forskjellige måter å slette på, etterfulgt av alternativ 5 som returnerer til hovedmenyen. Dersom bruker taster inn noe ugyldig vil beskjed om dette komme.

Alle slette funskjonene fungerer på samme måte, hvor den eneste forskjellen er if-statementen. Etter hvert som den går igjennom elementene vil den sende aktuelle elementer inn til slettefunksjonen, som tar imot elementet, og sletter det. Jeg var veldig fornøyd med deleteNode, da den ser veldig fin og lesbar ut, samtidig som at den har ønsket funksjonalitet.

Slettingen i meny nr. 2 sjekker ikke om det er skrevet inn noe annet enn tall. Jeg har prøvd å sjekke dette, ved å sjekke om den returnerer 1 eller ikke, men den vil ikke høre på det. Jeg er klar over HVORFOR det skjer, men jeg har ikke funnet en god løsning på det enda. Om bruker skriver inn bokstaver her vil det resultere i en uendelig løkke.

Oppgave 4

Startes med ./main

Denne var en seig en! Jeg laget en main metode som kaller på en versjon av ProcessCreditcardPayment med alle feilene fikset, og en versjon av funksjonen slik den var originalt.

Jeg tror ikke jeg skal gå igjennom hvordan og hvorfor hele funksjonen er satt opp slik den er, men heller ha som var feil, hvorfor det var feil, hvordan og hvorfor jeg fikset det slik jeg gjorde det.

Den første feilen jeg fant var på linje 130, hvor cortnummeret omgjøres til en 64 bit integer. Her var det et problem som gjorde at løkken kjørte én gang for mye, og J endte opp med å bli -4, istedenfor 4. Dette løste jeg med å endre for condition til å være fra pc; til pc->p; slik at løkken ender på 0. Etter å ha tatt printf på denne verdien ser det nå ut til at den ikke printer ut for mye.

Den andre og tredje feilen var derimot litt uklare. Grunnen til at applikasjonen kræsjer er på grunn av at tallet 9 i 123x resulterer i '9' / ('9' – '0' % 9) = 57 / 0 = udefinert. Fiksen for denne var enkel, så jeg satt modulus til å være 10, så kompilerer den. Dette var den andre feilen.

Tredje feilen var vanskelig, ettersom at å ta printf på alle verdier syntes å være korrekt. Det eneste som skurret var igjen på samme linje som feil nr. 2, hvor x er bonuskort. Det er forklart veldig lite om hvor mange typer det finnes, og dersom flere tall gir samme type cash-back. Jeg kan bare anta at det gir mening at det skal være unik bonuskort til hvert tall, som da tilsvarer at dette er feil nr.3. Hele ligningen fra feil nr.2 er indeksen til cc->p->digit (cc->p->digit['x' / ('x' – '0') % 10]), som sier at indeks til cc->p->digit skal være 0,xxx. Dette gir ikke så mye mening, så jeg forenklet ganske enkelt denne ligningen til (cc->p->digit[3] – '0') % 10, som resulterer i x = 0-9. Dette gir backend et godt svar på hvilken type det er.

Det som virker rart er at disse to feilene skal være på samme linje. Det er uvisst om du mener at dette er to separate feil, eller regnes som én. Jeg kunne i hvert fall ikke finne flere feil (utenom kompileringsfeil).

Dato verifikasjon er en enkel splitting og omgjøring til int. Jeg henter datoen med time_t og struct tm og sjekker med if statement om kort dato er før eller etter dagens dato.

Oppgave 5

Startes med ./main

Oppgave 5 starter i main, hvor jeg åpner én fil, sjekker at den er ok.

Åpner en annen fil, sjekker at den er ok.

Den kjører så en while løkke som henter ut hver karakter frem til End of File, og lagrer det i en buffer. Jeg sender så bufferen til dekoder metden, og sender det til den nye filen.

Decoder funksjonen tar imot arrayet og går igjennom det. Den sjekker hver bokstav og sjekker om det er stor bokstav, liten bokstav, tall eller tegn. Den omgjør så verdien ved å trekke fra det laveste innenfor thresholdet.

Oppgave 6

Startes med ./main textfile.txt

I oppgave 6 lager en while løkke tråder til å hente ut 10 karakterer av en tekst fil. Den legger så til de tallene som er lest og skriver ut dette etter trådene er ferdig med å skrive ut fra filen.

Tråden låser seg så fort som mulig, slik at ikke de tukler med samme data samtidig. Den åpner filen, og kopierer 10 + 10 bokstaver for hver gang tråden kalles. Alle utenom de 10 siste bokstavene blir overskrevet i bufferen, og den legger det til i strukten.

Main printer dette så ut, og resetter bufferen i strukten.

Oppgaver 7

Denne var gøy!

KLIENT:

Startes med ./client PORT

Oppretter enkelt ent socket og definerer socket adresse. Sjekker om den får koblet til serveren. Den mottar enda en melding fra server i form av handshake, og gir bruker input til å bestemme om en vil fortsette eller ikke. Hvis bruker fortsetter kan han skrive inn meldinger til server, og gå ut ved å skrive goodbye.

SERVER:

Startes med ./server ID PORT

Server oppretter en socket med en adresse. Den har også en if som tvinger maskinen til å åpne porten om den allerede er i bruk (bra for testing). Serveren hører så etter opp til 5 klienter for tilkobling, og starter en funksjon dersom forbindelsen skjer feilfritt. Den lukker så forbindelsen med serveren og lukker socketen til seg selv.

dostuff() funksjonen sender en velkomstmelding og resetter bufferen etter. Den sjekker så om brukeren sender noe mer. Dersom den mottar noe annet enn y eller Y, vil den avslutte forbindelsen. Dersom brukeren vil fortsette, iverksettes en while loop der serveren bekrefter at den ennå hører på, og henter inn verdien brukeren skriver inn. Dette skrives ut i terminalen. Den sjekker også om brukeren disconnecter igjennom prosessen.