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.