Algdat Excercise 4

Tomas Beranek and Callum Gran

September 2022

1 Aritmetikk med lenkede lister

1.1 Programmet

Første del av øvingen ble løst ved hjelp av dobbel-lenkede lister slik etterspurt. Både addering og subtrahering er gjennomført. I begge tilfeller har ikke rekkefølge noe konsekvens på svaret. Lengden er testet og fungerer for mer enn 20 siffer, faktisk fikk vi gyldige resultater med 400 siffer.
Programmet starter med å ta inn tre variabler inn i main. Fra dette ser den om det er addisjon eller subtrahering. Tallene tas inn som integer og placeres inn i den dobbelt lenka listen. Programmet sjekker hvilke liste som er lengst og bytter eventuelt plasser. Den lengste listen sin nåværende node blir da satt til posisjonen av den første noden i den kortere listen før den aritmetiske metoden kalles.

1.2 Addisjon

Addisjon setter det nåværende elementet til seg selv pluss elementet i den andre listen. Deretter sjekkes det om tallet er større enn 9. Hvis så, da sjekker den om det forrige elementet finnes, hvis det ikke finnes legges det til en 0 i starten av listen. Deretter kalles funksjonen av seg selv rekursivt, men nå med 1 og ikke elementet i den andre listen og det trekkes fra 9 på nåværende elementet. Dette blir worst case $O(n^2)$. Det kan løses med $O(n)$ ved å starte på halen, men denne måten gjorde at programmet ble ryddigere for subraherings funksjonen.

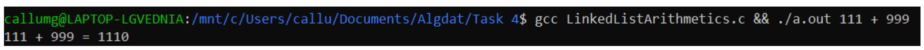

Figure 1: Addisjon med lån.

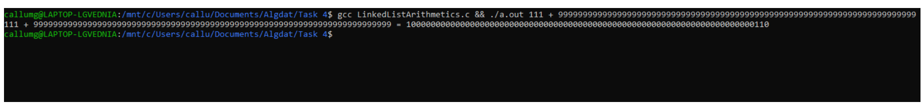

Figure 2: Addisjon med flere siffer i andre liste.

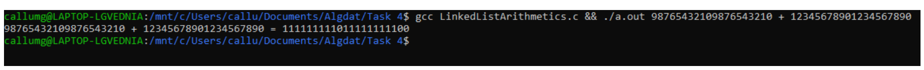

Figure 3: Addisjon med 20 siffer.

1.3 Subtraksjon

Subtraksjon setter det nåværende elementet til seg selv minus elementet i den andre listen. Deretter sjekkes det om tallet er mindre enn 0. Siden den allerede vet at den listen som trekkes fra er større, sjekker den ikke at det forrige elementet finnes og legger 10 på seg selv. Så kaller den seg selv rekursivt, men nå med 1 i stedet for elementet fra den andre listen. Hvis elementet i hodet nå er lik 0, så fjerner den node og setter den nåværende noden til hodet. Siden dette skjer rekursivt, vil det aldri være noen store hopp hvor man hopper over det som skulle vært hodet.

```
callumg@LAPTOP-LGVEDNIA:/mnt/c/Users/callu/Documents/Algdat/Task 4$ gcc LinkedListArithmetics.c && ./a.out 977 - 888
977 - 888 = 89
```

Figure 4: Vanlig subtrahering med lån.

```
callumg@LAPTOP-LGVEDNIA:/mnt/c/Users/callu/Documents/Algdat/Task 4$ gcc LinkedListArithmetics.c && ./a.out 37218939812 - 9837217838123214
37218939812 - 9837217838123214 = -9837180619183402
```

Figure 5: Subtrahering med negativt svar.

```
callumg@LAPTOP-LGVEDNIA:/mnt/c/Users/callu/Documents/Algdat/Task 4$ gcc LinkedListArithmetics.c && ./a.out 564738392101029384756 - 557389298218901289309
564738392101029384756 - 557389298218901289309 = 7349093882128095447
```

Figure 6: Subtrahering med 20 siffer.

2 Binært søketre

2.1 Programmet

I denne løsningen så har vi prøvd å gjøre alt så gjenbrukbart som mulig. Derfor har vi sendt funksjonsdefinisjoner i stedet for å kalle hard kodede funksjoner. Det blir også brukt void pointers der dette er nødvendig og typecaster helle variablene når du brukes i behandlings funksjoner. Denne løsningen bruker heller ikke strings, vi tenkte at siden vi allerede lærer om lister så kunne vi bruke dette i stedet. Dette medførte til en interessant løsning.

2.2 Input i treet

Siden vi bruker lenkede lister, så kan vi ikke bruke string compare. Derfor har vi laget en compare metode som sammenligner listene element for element. Samtidig så fant vi ut at kun vanlige ASCII karakterer har positive verdier i vanlig C. I stedet for å importere noen pakker, gjorde vi noen endringer til compare metoden, derimot blir ikke dette 100% riktig om man skal sammenligne spesialtall fra forskjellige land, men vi sa oss fornøyd med dette.

2.3 Behandling av treet

Løsningen vi har kommet fram til er ved å først initialisere en output med linjebredde, linje-avstander og linjer man vil printe. Så puttes rot elementet i lista og gjennomfører nivåtraversering nærmest slik det er angitt i boka. Behandlings metoden vår tar i mot den nå værende TreNoden og pointer til outputen. Vi tar nytte av statiske lokale variabler for å vite nivå, antall noder og nå værende start-sted for å legge til tegn. Først sjekkes det om vi noden eksisterende. Hvis så, settes en temporær liste til elementet i noden. Om denne listen eksisterer så finner vi plassen vi skal ha før og etter ordet og legger det til. Hvis ikke, legges kun tomrom til. Så sjekker vi om vi er på maks node mengde i nivået og eventuelt retter opp i startstedet til den nye linjen. Deretter legger den til tomme noder med en NULL liste på den nåværende noden hvis den ikke har barne noder.

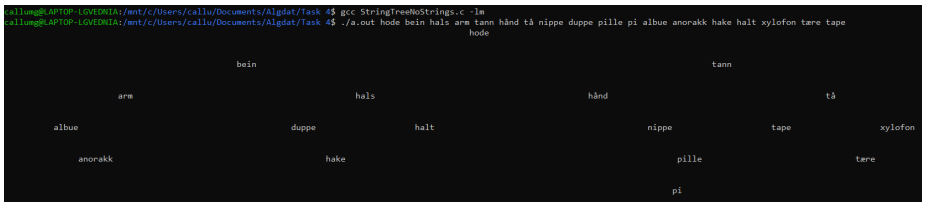

Figure 7: Resultat med 6 linjer.

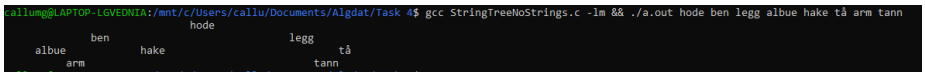

Figure 8: Resultat slik angitt i oppgaven.