KØBENHAVNS UNIVERSITET

IPS - Assignment 1

Mikkel Willén, bmq419

2. maj 2023

Indhold

1 Task

1.1

Jeg er på den generelle linje på datalogi.

1.2

Jeg vil mene jeg kan huske det meste fra PoP, og også den smule vi havde i LinAlg. Jeg kunne dog nok godt trænge til en lille smule brush-up i sproget.

1.3

Jeg vil også mene jeg kan huske det meste fra CompSys i forhold til assembly sprog, dog ser det ud til vi skal bruge et andet sprog, end det vi brugte i CompSys, så kunne nok også godt bruge en smule øvelse i det nye sprog.

1.4

Jeg synes det er meget spændende at lære om, hvordan et programmeringssprog fungerer, og hvordan man laver det underliggende i et programmeringssprog.

2 Task

Jeg har implementeret VARIABLE, OPERATE, LET_IN og OVER.

VARIABLE er implementeret med lookup funktionen fra den udleverede kode.

OPERATE er implementeret med et match case, som tjekker hvilken operant der er blevet brugt, of derefter laver den tilsvarende operation.

LET_IN er implementeret ved, først at evaluere den første expression, og derefter binde den til den variable, som er blevet givet. Den bliver bundet i en nu variable table, da vi gerne vil sikre, at værdien ikke bliver gemt, når vi kommer ud af dette statement i koden. Til sidste bliver den anden expression evalueret i det nye variable tabel, da det er der, vores variable er gemt.

OVER er implementeret med hjælpefunktioner til hvert case.

SUM funktionen starter med at tjekke, om det er en gyldig sumfølge vi arbejder med. Hvis ikke returnere den INT(0). Ellers evaluerer den e3, og lægger det sammen med det rekursive kald, hvor variablen er blevet opdateret til det næste tal i sumfølgen. Det bliver den ved med, til den kommer til en ugyldig sum, hvorefter den lægger 0 til, og returnere den endelige værdi.

PROD fungerer på samme måde som SUM. Den returnerer dog INT(1), når det er en ugyldig produktfølge. Den evalurer også e3, men ganger det sammen med det rekursive kald for den nye e1, i stedet for at lægge det til.

MAX itererer igennem alle værdier for MAXfølgen, og gemmer en værdi, for den nuværende største værdi. Hvis den finder en værdi, som er større, laver den et rekusivt kald, med en ny største værdi, eller laver den et rekusivt kald, med dem gamle største værdi. Til sidst returnerer den, den største værdi.

ARGMAX fungere på samme måde som MAX. Den gemmer dog også en værdi for e1, da det er indexet, som har den største værdi. Den returnerer til sidst indexet for den største værdi.

Herunder er en testkørsel af de forskellige implmentationer:

```
$ dotnet fsi calculator.fsx
Welcome to the calculator! Type "exit" to stop.
Input an expression : 4
Evaluation result   : INT 4
Input an expression : 1 - 2 - 3
Evaluation result   : INT -4
Input an expression : let x = 4 in x + 3
Evaluation result   : INT 7
Input an expression : let x0 = 2 in let x1 = x0 * x0 in let x2 = x1 * x1 in x2 * x2
Evaluation result   : INT 256
Input an expression : sum x = 1 to 5 of x * x
Evaluation result   : INT 55
Input an expression : prod x = 1 to 5 of x
Evaluation result   : INT 120
Input an expression : max x = 0 to 10 of 5 * x - x * x
Evaluation result   : INT 6
Input an expression : argmax x = 0 to 10 of 5 * x - x * x
Evaluation result   : INT 2
Input an expression : exit
```

Som man kan se, returnerer de forskellige kald de værdi, som man kunne antage de ville.
Alle funktioner tjekker kun hvert index en gang, om kører derfor i $O(n)$ tid. Pladsforbruget kunne være mindre i SUM og PROD, hvis man ikke lagde alle tingene sammen på en gang, men gjorde det løbende, for hvert iteration af de rekursive kald. Plads forbruget for de nuværende implementationer af SUM og PROD, er derfor $O(n)$ og kunne have været $O(1)$.

3 Task

Main funktionen starter med at mappe alle talene ind i et array, hvis længden på arrayet er en eller mere, eller giver den en fejl. Derefter kører den en map funktion på arrayet, hvor den tjekker om det er det første element, hvor den bare vil returnere tallet, og ellers vil retunere den i'te værdi minus den forrige. Herefter kører den mul funktionen med map på alle værdierne i arrayet, hvorefter den til sidst lægger alle tallene sammen med reduce funktionen med op +, som operant, hvorefter den retunere den endelige svar.
Vedlagt er nogle testfiler.