

# W1 - IPS

Nikolaj Gade (qhp695)

May 2022

## Task 1

1. I am a DIKU Computer Science BSc, general profile
2. I have little to no proficiency with F#
3. I have minimal knowledge of assembly programming
4. I'm very interested in the topic of compilers

## Task 2

### a) Completeness and Correctness

All the requested features have been implemented fully. All 10 possible expressions are accounted for, which can be seen by testing each individually, or together.

The program has shown to give the correct answers in all test cases.

Both completeness and correctness can be shown in these tests, all of which return correct results:

```
1 Welcome to the calculator! Type "exit" to stop.
2 Input an expression : 1+2+3+4-(5+6)
3 Evaluation result : INT -1
4 Input an expression : let x=7 in (x*8+9)
5 Evaluation result : INT 65
6 Input an expression : sum x=1 to 10 of (prod y=x to 10 of y)
7 Evaluation result : INT 9864100
8 Input an expression : max x = 0 to 10 of 5*x-x*x
9 Evaluation result : INT 6
10 Input an expression : argmax x = 0 to 10 of 5*x-x*x
11 Evaluation result : INT 2
```

### b) Efficiency

The program runs efficiently and with no obvious uses of excessive memory.

Any singular expression, apart from the **OVER** expressions runs in constant time, ignoring any expressions it contains. This means that any expression that contains  $n$  non-**OVER** expressions runs in linear time,  $O(n)$ .

The **OVER** expressions contain a loop, meaning they run in linear time by themselves.

### c) Code Sharing/Elegance

The code is made readable and with minimal code duplication. The **OPERATE** and **OVER** expressions have code run before the specific expression is determined, which reduced duplication.

## The full eval code

```
1 let rec eval (vtab : SymTab) (e : EXP) : VALUE =
2   match e with
3   | CONSTANT n -> n
4   | VARIABLE v ->
5     lookup v vtab
6   | OPERATE (op, e1, e2) ->
7     let (INT exp1) = eval vtab e1
8     let (INT exp2) = eval vtab e2
9     match op with
10    | BPLUS -> INT (exp1 + exp2)
11    | BMINUS -> INT (exp1 - exp2)
12    | BTIMES -> INT (exp1 * exp2)
13  | LET_IN (var, e1, e2) ->
14    let exp1 = eval vtab e1
15    let vtab = bind var exp1 vtab
16    eval vtab e2
17  | OVER (rop, var, e1, e2, e3) ->
18    let (INT exp1) = eval vtab e1
19    let (INT exp2) = eval vtab e2
20    let results = [
21      for i in exp1..exp2 ->
22        let vtab = bind var (INT i) vtab
23        eval vtab e3
24    ]
25    match rop with
26    | RSUM -> List.fold (fun (INT acc) (INT elem) -> INT (acc + elem)) (INT 0) results
27    | RPROD -> List.fold (fun (INT acc) (INT elem) -> INT (acc * elem)) (INT 1) results
28    | RMAX -> List.fold (fun (INT acc) (INT elem) -> INT (max acc elem)) results[0] results
29    | RARGMAX ->
30      let max_elem = List.fold (fun (INT acc) (INT elem) -> INT (max acc elem)) results[0] results
31      INT ((List.findIndex (|=) max_elem) results) + exp1
```

## Task 3

### a) The mul function

```
1 fun int mul(int x, int y) =
2   if y == 0 then 0
3   else if y < 0 then mul(x, y+1) - x
4   else mul(x, y-1) + x
```

The `mul` function is recursive. The base level, when `y` is 0, 0 is returned. In all other cases, `y` is moved 1 step closer to 0, and `x` is either added or subtracted from the final result.

### b) The dif array

```
16 let dif = map(fn int (int x) => if x == 0 then arr[x] else arr[x] - arr[x-1], iota(n)) in
```

The `dif` array is constructed using an anonymous function in a `map` function call.

## The full Fasto code

```
1 fun int mul(int x, int y) =
2   if y == 0 then 0
3   else if y < 0 then mul(x, y+1) - x
4   else mul(x, y-1) + x
5
6 fun int readInt(int i) = read(int)
7
8 fun int squareSum(int x, int y) = x + mul(y, y)
9
10 fun int main() =
11   let n = read(int) in
12   if n == 0 then let a = write("Incorrect Input!") in 0
13   else if n < 0 then let a = write("Incorrect Input!") in 0
14   else
15     let arr = map(readInt, iota(n)) in
16     let dif = map(fn int (int x) => if x == 0 then arr[x] else arr[x] - arr[x-1], iota(n)) in
17     write(reduce(squareSum, 0, dif))
```