Question 1

To implement a lexer for the WHILE language, you first need to design the appropriate regular expressions for the following eleven syntactic entities:

A1:
See the code file <<cfl_cw02.sc>>, I use CFUN to substitude CHAR and RANGE.

Question 2

Implement the Sulzmann & Lu lexer from the lectures. For this you need to implement the
H4 functions *nullable* and *der* (you can use your code from CW 1), as well as *mkeps* and *inj*.
H4 These functions need to be appropriately extended for the extended regular expressions
from Q1. Write down the clauses for

$$\mathit{mkeps}([c_1, c_2, \ldots, c_n]) \overset{def}{=} ?$$

$$\mathit{mkeps}(r^+) \overset{def}{=} ?$$

$$\mathit{mkeps}(r^?) \overset{def}{=} ?$$

$$\mathit{mkeps}(r^{\{n\}}) \overset{def}{=} ?$$

$$\mathit{inj}([c_1, c_2, \ldots, c_n])\ c \overset{def}{=} ?$$

$$\mathit{inj}(r^+)\ c \overset{def}{=} ?$$

$$\mathit{inj}(r^?)\ c \overset{def}{=} ?$$

$$\mathit{inj}(r^{\{n\}})\ c \overset{def}{=} ?$$

A2:
Do not have mkeps([c1,c2,...cn]) because the RANGE have only characters which cannot be EMPTY, also do not have mkeps(CFUN).

$$\mathit{mkeps}(r^+) \overset{def}{=} \mathit{Seq}(\mathit{mkeps}(r), \mathit{Stars}[])$$

H4

H4

$$\mathit{mkeps}(r^?) \overset{def}{=} \mathit{if}(\mathit{nullable}(r))\ \mathit{Opt}(\mathit{mkeps}(r))\ \mathit{else}\ \mathit{Empty}$$

$$\mathit{mkeps}(r^{\{n\}}) \overset{def}{=} \mathit{if}(n == 0)\ \mathit{Empty}\ \mathit{else}\ \mathit{Ntime}(\mathit{List}(\mathit{mkeps}(r)) * n)$$

//List(mkeps(r)*n) means there are n copies of mkeps(r) in the List, in code I use List.fill() to create.

$$\mathit{inj}([c_1, c_2, \ldots, c_n])\ c\ \mathit{Empty} \overset{def}{=} c$$

H4

CFUN is the same, inj(CFUN) c Empty = c.

$$inj(r^{+})\ c\ Seq(v, Stars(vs)) \overset{def}{=} Pls((inj\ (r)\ c\ v_1)\ ::\ vs)$$

$$inj(r^{?})\ c\ Opt(v) \overset{def}{=} Opt(inj\ (r)\ c\ v)$$

$$inj(r^{?})\ c\ Empty \overset{def}{=} Opt(inj\ (r)\ c\ v)$$

$$inj(r^{\{n\}})\ c\ Sequ(v, Empty) \overset{def}{=} Ntime(List(inj((r)\ c\ v)))$$

$$inj(r^{\{n\}})\ c\ Sequ(v, Ntime(v_1)) \overset{def}{=} Ntime(List(inj((r)\ c\ v)) + +v_1)$$

```
val WHILE_REGS = (("k" $ KEYWORD) |
("i" $ ID) |
("o" $ OP) |
("n" $ NUM) |
("s" $ SEMI) |
("str" $ STRINGS) |
("p" $ PARA) |
("c" $ COMMENT) |
("w" $ WHITESPACE)).%
```

Token of "read n;":

```
List((k,read), (w, ), (i,n), (s,;))
```

test: NTIMES(CFUN(Set(a)),3) with "aaa"

```
Ntime(List(Cf(Set(a)), Cf(Set(a)), Cf(Set(a))))
```

test: NTIMES(ALT(CFUN(Set(a)),ONE),3) with "aa"

```
Ntime(List(Left(Cf(Set(a))), Left(Cf(Set(a))), Right(Empty)))
```

Question 3

Extend your lexer from Q2 to also simplify regular expressions after each deriva- tion step and rectify the computed values after each injection. Use this lexer to tokenize the programs in Figures 1 – 4. You can find the programms also on KEATS. Give the tokens of these programs where whitespaces are filtered out. Make sure you can tokenise **exactly** these programs.

A3:

See the code file, use the cfl_cw02_token.sc file to check the tokenize, I have written the test case in the code. I only filter the whitespaces as required, although I think the comment should also be filtered. It will print the information below:

```
Fib program
List(T_KWD(write), T_STR("Fib"), T_SEMI, T_KWD(read), T_ID(n), T_SEMI, T_ID(minus1),
T_OP(:=), T_NUM(0), T_SEMI, T_ID(minus2), T_OP(:=), T_NUM(1), T_SEMI, T_KWD(while),
T_ID(n), T_OP(>), T_NUM(0), T_KWD(do), T_PAREN, T_ID(temp), T_OP(:=), T_ID(minus2),
T_SEMI, T_ID(minus2), T_OP(:=), T_ID(minus1), T_OP(+), T_ID(minus2), T_SEMI,
T_ID(minus1), T_OP(:=), T_ID(temp), T_SEMI, T_ID(n), T_OP(:=), T_ID(n), T_OP(-),
T_NUM(1), T_PAREN, T_SEMI, T_KWD(write), T_STR("Result"), T_SEMI, T_KWD(write),
T_ID(minus2))
```

Loops program

```
List(T_ID(start), T_OP(:=), T_NUM(1000), T_SEMI, T_ID(x), T_OP(:=), T_ID(start), T_SEMI,
T_ID(y), T_OP(:=), T_ID(start), T_SEMI, T_ID(z), T_OP(:=), T_ID(start), T_SEMI,
T_KWD(while), T_NUM(0), T_OP(<), T_ID(x), T_KWD(do), T_PAREN, T_KWD(while),
T_NUM(0), T_OP(<), T_ID(y), T_KWD(do), T_PAREN, T_KWD(while), T_NUM(0), T_OP(<),
T_ID(z), T_KWD(do), T_PAREN, T_ID(z), T_OP(:=), T_ID(z), T_OP(-), T_NUM(1), T_PAREN,
T_SEMI, T_ID(z), T_OP(:=), T_ID(start), T_SEMI, T_ID(y), T_OP(:=), T_ID(y), T_OP(-),
T_NUM(1), T_PAREN, T_SEMI, T_ID(y), T_OP(:=), T_ID(start), T_SEMI, T_ID(x), T_OP(:=),
T_ID(x), T_OP(-), T_NUM(1), T_PAREN)
```

factors program

```
List(T_COM(// Find all factors of a given input number
), T_KWD(write), T_STR("Input n please"), T_SEMI, T_KWD(read), T_ID(n), T_SEMI,
T_KWD(write), T_STR("The factors of n are"), T_SEMI, T_ID(f), T_OP(:=), T_NUM(2),
T_SEMI, T_KWD(while), T_PAREN, T_ID(f), T_OP(<), T_ID(n), T_OP(/), T_NUM(2), T_OP(+),
T_NUM(1), T_PAREN, T_KWD(do), T_PAREN, T_KWD(if), T_PAREN, T_PAREN, T_ID(n),
T_OP(/), T_ID(f), T_PAREN, T_OP(), T_ID(f), T_OP(==), T_ID(n), T_PAREN, T_KWD(then),
T_PAREN, T_KWD(write), T_PAREN, T_ID(f), T_PAREN, T_PAREN, T_KWD(else),
T_PAREN, T_KWD(skip), T_PAREN, T_SEMI, T_ID(f), T_OP(:=), T_ID(f), T_OP(+), T_NUM(1),
T_PAREN)
```

collatz program

```
List(T_COM(// Collatz series
), T_COM(//
), T_COM(// needs writing of strings and numbers; comments
), T_ID(bnd), T_OP(:=), T_NUM(1), T_SEMI, T_KWD(while), T_ID(bnd), T_OP(<),
T_NUM(101), T_KWD(do), T_PAREN, T_KWD(write), T_ID(bnd), T_SEMI, T_KWD(write),
T_STR(": "), T_SEMI, T_ID(n), T_OP(:=), T_ID(bnd), T_SEMI, T_ID(cnt), T_OP(:=), T_NUM(0),
T_SEMI, T_KWD(while), T_ID(n), T_OP(>), T_NUM(1), T_KWD(do), T_PAREN,
T_KWD(write), T_ID(n), T_SEMI, T_KWD(write), T_STR(","), T_SEMI, T_KWD(if), T_ID(n),
T_OP(%), T_NUM(2), T_OP(==), T_NUM(0), T_KWD(then), T_ID(n), T_OP(:=), T_ID(n),
T_OP(/), T_NUM(2), T_KWD(else), T_ID(n), T_OP(:=), T_NUM(3), T_OP(), T_ID(n), T_OP(+),
T_NUM(1), T_SEMI, T_ID(cnt), T_OP(:=), T_ID(cnt), T_OP(+), T_NUM(1), T_PAREN, T_SEMI,
T_KWD(write), T_STR(" => "), T_SEMI, T_KWD(write), T_ID(cnt), T_SEMI, T_KWD(write),
T_STR("\n"), T_SEMI, T_ID(bnd), T_OP(:=), T_ID(bnd), T_OP(+), T_NUM(1), T_PAREN)
```