Computer Programming Paradigms Lab – Lab 1

1. Define a user-defined function `last_` that selects the last element of a non-empty list.
For example, `last_ [1,2,3,4,5] = 5`.

```
last_xs :: [Int] → Int
last_ xs = head (reverse xs)

last_ xs = xs !! (length xs - 1)
```

2. Define a user-defined function `init_` that removes the last element of a non-empty list.
For example, `init_ [1,2,3,4,5] = [1,2,3,4]`.

```
init_ :: [Int] → [Int]
init_ xs = take (length xs - 1) xs

init_ xs = reverse (tail ( reverse xs))
```

3. Define a user-defined function `even_` that decides whether a given number is even.

```
even_ :: Integral a => a → Bool
even_ n = n `mod` 2 == 0
```

4. Define a user-defined function `splitAt_` that splits the list at the nth element.

```
splitAt_ :: Int → [a] → ([a], [a])
splitAt_ n xs = (take n xs, drop n xs)
```

5. Define a user-defined function `abs_` that returns the absolute value of a given number using:

a. If - then – else

```
abs_ :: Int → Int
abs_ n = if n >= 0 then n else -n
```

b. Guarded equations

```
abs_ n | n >= 0    = n
       | otherwise = -n
```

6. Define a user-defined function `signum_` that returns the sign of a given integer.

a. If – then -else

```
signum_ :: Int → Int
signum_ n = if n < 0 then -1 else
               if n == 0 then 0 else -1
```

b. Guarded equations

```
signum_ n    | n < 0 = -1
             | n == 0
             | otherwise = 1
```

7. Define a user-defined function `halve` that splits an even-lengthed list into two halves.

```
halve :: [a] → ([a], [a])
halve xs = (take n xs, drop n xs)
           where n = length xs `div` 2

halve xs = splitAt_ (length xs `div` 2) xs
```

8. Define a user-defined function `third` that returns the third element in a list using:

i) Head and tail

```
third xs = head (tail (tail xs))
```

ii) List indexing !!

```
third xs = xs !! 2
```

iii) Pattern matching

```
third (_:_:x:_) = x
```

9. Consider a function `safetail` that behaves in the same way as tail except that it maps the empty list to itself rather than producing an error. Using `tail` and the function `null` that decides if a list is empty or not, define `safetail` using:

a. A conditional expression

```
safetail :: [a] → [a]
safetail xs = if null xs then [] else tail xs
```

b. Guarded equations

```
safetail xs  | null xs   = []
             | otherwise = tail xs
```

c. Pattern matching

```
safetail (_:xs) = xs
```

10. The `luhn` algorithm is used to check bank card numbers for simple errors, and proceeds as follows:

a. Consider each digit as a separate number.
b. Moving left, double every other number from the second last.
c. Subtract 9 from each number that is now greater than 9.
d. Add all the remaining numbers together.
e. If the total is divisible by 10, the card is valid.

Define a function `luhnDouble` that doubles a digit and subtracts 9 if the result is greater than 9. For example,

- ➢ `luhnDouble 3`
- ➢ `6`

- ➢ `luhnDouble 6`
- ➢ `3`

Using luhnDouble and the function mod, define a function `luhn:: Int → Int → Int→ Int → Bool` that decides if a four digit bank card is valid. For example:

- ➢ `luhn 1 2 3 4`
- ➢ `False`

- ➢ `luhn 5 6 7 8`
- ➢ `True`

```
luhnDouble :: Int → Int
luhnDouble d = if n > 9 then n-9 else n
               where n = d*2

luhn:: Int → Int → Int → Int → Bool
luhn a b c d = (a1 + b + c1 + d) `mod` 10 == 0
               where
                  a1 = luhnDouble a
                  c1 = luhnDouble c
```