Don Bosco Institute of Technology

DEPARTMENT OF INFORMATION TECHNOLOGY

ITC305 Computer Programming Paradigms Lab

SEMESTER III (Academic Year 2022 – 23)

Programs by Prof. Udaychandra Nayak

List of programs in Haskell & PROLOG

---

1. Show how the library function `last` that selects the last element of a nonempty list could be defined in terms of the library functions such as `head`, `reverse` and `!!` For example, `last [1,2,3,4,5] = 5`.

   ```
   last xs = head (reverse xs)
   ```

   (OR)

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

2. The library function init removes the last element from a non-empty list; for example, init [1,2,3,4,5] = [1,2,3,4]. Show how init could similarly be defined in two different ways.

   ```
   init xs = take (length xs - 1) xs
   ```

   (OR)

   ```
   init xs = reverse (tail (reverse xs))
   ```

3. Decide if an integer is even:

   ```
   even_ n = n `mod` 2 == 0
   ```

4. Split a list at the nth element:

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

5. Returns the absolute value of an integer:

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

6. Returns the sign of an integer:

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

(OR Using Guarded Equations)

```
signum_ :: Int -> Int
signum_ n | n < 0     = -1
          | n == 0    = 0
          | otherwise = 1
```

7. Add two numbers

```
add_ :: Int -> Int -> Int
add_ x y = x + y
```

8. Using library functions, define a function `halve :: [a] -> ([a], [a])` that splits an even-lengthed list into two halves. For example:

```
>halve [1, 2, 3, 4, 5, 6]
([1, 2, 3], [4, 5, 6])
```

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

(OR)

```
halve :: [a] -> ([a], [a])
halve xs = splitAt (length xs `div` 2) xs
```

9. Define a function `third :: [a] -> a` that returns the third element in a list using
   a. Head and tail
   b. List indexing !!

Using Head and tail:

```
third :: [a] -> a
third xs = head (tail (tail xs))
```

Using List indexing:

```
third :: [a] -> a
third xs = xs !! 2
```

10. Consider a function `safetail :: [a] → [a]` that behaves as the library function *tail* , except that *safetail* maps the empty list to itself, whereas *tail* produces an error in this case. Define *safetail* using:
    a. a conditional expression;
    b. guarded equations;
    c. pattern matching.

    Hint: make use of the library function *null*.

Using conditional expression:

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

Using guarded equations:

```
safetail :: [a] → [a]
safetail xs | null xs    = []
            | otherwise = tail xs
```

Using pattern matching:

```
safetail :: [a] → [a]
safetail [] = []
safetail (_:xs) = xs
```

11. Write a program to print prime numbers up to n.

```
factors :: Int -> [Int]
factors n = [x | x <- [1..n], n `mod` x == 0]

prime :: Int -> Bool
prime n = factors n == [1,n]

primes :: Int -> [Int]
primes n = [x | x <- [2..n], prime x]
```

12. Using a list comprehension, give an expression that calculates the sum $1^2 + 2^2 + \ldots 100^2$ of the first one hundred integer squares.

```
total = sum [x^2 | x <- [1..100]]
```

13. In a similar way to the function *length*, show how the library function `replicate :: Int → a → [a]` that produces a list of identical elements can be defined using a list comprehension. For example:

```
> replicate 3 True
[True,True,True]
```

```
replicate :: Int → a → [a]
replicate n x = [x | _ <- [1..n]]
```

14. A triple (*x, y, z*) of positive integers is *Pythagorean* if $x^2 + y^2 = z^2$. Using a list comprehension, define a function `pyths :: Int → [(Int, Int, Int)]` that returns the list of all Pythagorean triples whose components are at most a given limit. For example:

```
> pyths 10
[(3, 4, 5), (4, 3, 5), (6, 8, 10), (8, 6, 10)]
```

```
pyths :: Int → [(Int, Int, Int)]
pyths n = [(x,y,z) | x <- [1..n],
                     y <- [1..n],
                     z <- [1..n],
                     x^2 + y^2 == z^2]
```

15. A positive integer is *perfect* if it equals the sum of its factors, excluding the number itself. Using a list comprehension and the function *factors*, define a function `perfects :: Int → [Int]` that returns the list of all perfect numbers up to a given limit. For example:

```
> perfects 500
[6, 28, 496]
```

```
Perfects n = [x | x <- [1..n], sum (init(factors x)) == x]
```

16. The scalar product of two lists of integers xs and ys of length n is given by the sum of the products of corresponding integers:

$$\sum_{i=0}^{n-1}(xs_i * ys_i)$$

Define a function `scalarproduct :: [Int] -> [Int] -> Int` that returns the scalar product of two lists. For example:

```
>  scalarproduct [1,2,3] [4,5,6]
32
```

```
scalarproduct :: [Int] -> [Int] -> Int
scalarproduct xs ys = sum [x*y | (x,y) <- zip xs ys]
```

17. Write factorial function using recursion.

```
fac :: Int -> Int
fac 0 = 1
fac n = n * fac (n – 1)
```

18. Write the product function using recursion.

```
product_ [] = 1
product_ (n:ns) = n * product ns
```

19. Write a function that calculates the nth Fibonacci number.

```
fib :: Int -> Int
fib 0 = 0
fib 1 = 1
fib n = fib (n – 2) + fib (n – 1)
```

20. Write a function that implements quicksort.

```
qsort []     = []
qsort (x:xs) = qsort smaller ++ [x] ++ qsort larger
               where
                  smaller = [a | a <- xs, a <=x]
                  larger  = [b | b <- xs, b > x]
```

21. Write a program to check whether a number is even. (PROLOG)

```
checkeven(N):-M is N//2,N=:=2*M.
```

22. Define and test a predicate which takes two arguments, both numbers, and calculates and outputs the following values: (a) their average, (b) the square root of their product and (c) the larger of (a) and (b). (PROLOG)

```
pred(A,B):-X is (A+B)/2,write('Average is: '),write(X),nl, Y is
sqrt(A*B), write('Square root of product is: '),write(Y),nl,
Z is max(X,Y),write('Larger is: '),write(Z),nl.
```

23. Write a program to print numbers from X to Y, both inclusive. (PROLOG)

```
output_values(Last, Last):- write(Last),nl, write('end of example'),nl.
output_values(First, Last):-First=\=Last, write(First),nl, N is
First+1,output_values(N, Last).
```

24. Define a predicate to find the sum of the integers from 1 to N (say for N = 100). (PROLOG)

```
sumto(1,1).
sumto(N,S):-N>1,N1 is N-1,sumto(N1,S1),S is S1+N.
```

?- sumto(100,N).
N = 5050

25. Define a predicate to output the squares of the first N integers, one per line. (PROLOG)

```
writesquares(1):-write(1),nl.
writesquares(N):-N>1,N1 is N-1,writesquares(N1), Nsq is N*N,
write(Nsq),nl.
```

?- writesquares(6).
1
4
9
16
25
36
yes

26. Define a predicate to output the values of the squares of the integers from N1 to N2 inclusive and test it with N1 = 6 and N2 = 12. (PROLOG)

```
outsquare(N1,N2):-N1>N2.
outsquare(N1,N2):- write(N1),write(' squared is '),Square is N1*N1,
write(Square),nl, M is N1+1,outsquare(M,N2).
```

?- outsquare(6,12).
6 squared is 36
7 squared is 49
8 squared is 64
9 squared is 81
10 squared is 100
11 squared is 121
12 squared is 144
yes

27. Classify a number as zero, or positive or negative. (PROLOG)

```
classify(0,zero).
classify(N, negative):-N<0.
classify(N, positive).

Example:
```

?- classify(0,N).
N = zero

?- classify(-4,X).
X = negative

28. Write a predicate to calculate the factorial. (PROLOG)

```
factorial(1,1).
factorial(N,Nfact):-N1 is N-1,factorial(N1,Nfact1),Nfact is N*Nfact1.
```

?- factorial(6,N).
N = 720

(Note: You will get 2 Haskell programs & 1 PROLOG program in your practical exam).