Отчёт по лабораторной работе № 5

Волгина А. Д., ИУ5-33Б

Для произвольно выбранного типа данных (например, Maybe) реализуйте функции функтора, аппликативного функтора, монады.

Проверьте для Вашей реализации справедливость соответствующих законов для функтора, монады и аппликативного функтора (тех законов, которые можно проверить с использованием F#). Некоторые законы могут не выполняться. Это означает что данный тип не является в полной мере функтором, аппликативным функтором, монадой.

```
open System



[<EntryPoint>]
let main argv =
    let fmapMaybe f p =
        match p with
        | Some a -> Some (f a)
        | None -> None
    let x1 = Some 3;
    let x2 = fmapMaybe (fun x -> x*x) x1;
    let rec mapList f list =
        match list with
        | [] ->
            []
        | head::tail ->
            // new head + new tail
            (f head) :: (mapList f tail)
```

```
let id x = x;
let list1 = [1; 2; 3];
let list2 = mapList id list1;
let func_f x = x * 2;
let func_g x = x + 2;
let list3_1 = mapList func_f list1;
let list3_2 = mapList func_g list3_1;
let list4 = mapList (func_f >> func_g) list1;
let applyMaybe fOpt xOpt =
    match fOpt,xOpt with
    | Some f, Some x -> Some (f x)
    | _ -> None
let am1 = applyMaybe (Some (fun x -> x * 2)) (Some 2)
let applyList (fList: ('a->'b) list) (xList: 'a list)  =
    [ for f in fList do
          for x in xList do
              yield f x ]
let applyList2 (xList: 'a list)  (fList: ('a->'b) list) =
    [ for f in fList do
        for x in xList do
            yield f x ]
let al1 = applyList [(fun x -> x + 2);(fun x -> x * 2)] list1
let at1 = applyList [id] list1
let app_x = 3
let y1 = [func_f app_x]
let y2 = applyList [func_f] [app_x]
```

```
let at21 = applyList [func_f; func_g] list1
let at22 = applyList2 list1 [func_f; func_g]
let bindMaybe f xOpt =
    match xOpt with
    | Some x -> f x
    | _ -> None
let bindList (f: 'a->'b list) (xList: 'a list)  =
    [ for x in xList do
      for y in f x do
          yield y ]
let b1 = bindMaybe id (Some x1);
let func_h x = Some (x + 1)
let efunc_h = bindMaybe func_h
let x2 = 3
let y31 = efunc_h (Some x2)
let y32 = func_h x2
let func_k x = Some (x * 3)
let group1 = (x2 |> func_f) |> func_g
let group2 = bindMaybe func_k (bindMaybe func_h (Some x2))
0 // return an integer exit code
```