Midterm, 2.12.2019 Meno+priezvisko:

Pravidlá:

- **čas 100 minút**
- **môžete používať akékoľvek ale len Vami prinesené materiály,**
- **nie je možné zdieľať akékoľvek pomôcky a materiály,**
- **kolektívne riešenia sa netolerujú,**
- **každý príklad riešte na papieri obsahujúcom jeho zadanie,**
- **midterm obsahuje 5 príkladov spolu za 6+6+4+7+3=26 bodov.**

1. [6 bodov – Fibonacciho slová]

Fibonacciho čísla vám predstavovať určite netreba. V tejto úlohe sú namiesto čísel reťazce, definícia je:

```
fibStr :: Int -> String -> String -> String
fibStr n a b = fibStr' n
               where
                    fibStr' 1 = a
                    fibStr' 2 = b
                    fibStr' n = (fibStr' (n-2)) ++ (fibStr' (n-1))
```

Takto vyzerajú počiatočné hodnoty pre "a", "b" (dĺžky reťazcov sú klasické Fibonacciho čísla):

```
[fibStr i "a" "b" | i <-[1..10]] = ["a","b","ab","bab","abbab","bababbab","abbabbababbab"..]
```

a) [1 bod] Definujte funkciu **fibLinearne::Int->String->String->String**, ktorá vygeneruje n-té Fibonacciho slovo, ale akceptované budú len efektívne riešenia zložitosti O(n). Príklad:

```
fibLinearne 1 "ab" "cde"  == "ab"
fibLinearne 2 "ab" "cde"  == "cde"
fibLinearne 3 "ab" "cde"  == "abcde"
fibLinearne 4 "ab" "cde"  == "cdeabcde"
```

b) [1 bod] Definujte funkciu **jeFib::String->String->String->Bool**, ktorá pre **jeFib kandidat a b** zistí, či **kandidat** môže byť prvkom postupnosti začínajúcej neprázdnymi reťazcami **a, b**. Príklad:

```
jeFib "abcdeabcdecdeabcdeabcdecdeabcde" "ab" "cde" == True
jeFib "abcdeabcdecdeabcdea" "ab" "cde" == False
```

c) [2 body] Definujte polymorfnú verziu funkcie **fibPoly::(t->t->t)->Int->t->t->t**, z ktorej **fibStr** vypadne ako inštancia **fibStr n a b = fibPoly (++) n a b**, a klasické Fibonacciho čísla vypadnú ako **fib n = fibPoly (+) n 1 1.**

d) [2 body] Definujte funkciu **prveDva :String->(String, String)**, ktorá pre **prveDva kandidat** nájde ľubovoľnú dvojicu prvých dvoch členov postupnosti tak, že táto Fibonacciho postupnosť neskôr obsahuje reťazec kandidat, ktorý **je aspoň tretím členom postupnosti**, teda **rôzny od prvých dvoch**. Všetky reťazce v postupnosti musia byť neprázdne. Príklad:

```
prveDva "prvedruheprvedruhedruheprvedruhe"  == ("prve", "druhe")
prveDva "prvedruheprvedruhe"  == ("p", "rvedruhe")
```

-------- v prípade potreby riešte túto úlohu na zadnú stranu **TOHOTO listu**----------

2. [6 bodov – foldový]

V každej časti úlohy si stačí vybrať jednu z verzií `foldr` alebo `foldl`, podľa vašich preferencií. Máte naprogramovať ekvivalenty štandardných funkcií, ktoré dobre poznáte z Haskellu, ale použitím foldl/foldr. Môžete si definovať vlastné pomocné funkcie. Zo štandardných funkcií môžete použiť len elementárne, ako `head`, `tail`, `length`, prípadne nejasností sa spýtajte. **Nepoužívajte rekurziu ani list-comprehension**, veď foldl/foldr schémy sú o tom, že rekurziu viete nahradiť.

1. **[1bod]** Do definície funkcie **`elem'::Eq(t)=>t->[t]->Bool`** doplňte výrazy za symboly **`?`**, aby funkcia fungovala rovnako ako **`elem`**, teda vrátila **`True`**, ak sa prvok nachádza v zozname, inak **`False`**. Môžete porovnávať len hodnoty typu **`t`**.

```
elem' x xs = ? foldr ? ? xs,        elem' x xs = ? foldl ? ? xs.
```

2. **[2body]** Do definície funkcie **`nub'::Eq(t)=>[t]->[t]`** doplňte výrazy za symboly **`?`**, aby funkcia fungovala rovnako jako nub, teda vrátila množinu prvkov pôvodného zoznamu, na poradí prvkov nezáleží.

```
nub' xs = ? foldr ? ? xs,           nub' xs ys = ? foldl ? ? xs.
```

3. **[3body]** Do definície funkcie **`sort'::Ord(t)=>[t]->[t]`** doplňte výrazy za symboly **`?`**, aby funkcia fungovala rovnako ako `sort`, teda vrátila utriedený vstupný zoznam. Na zložitosti vami vybraného triediaceho algoritmu nezáleží, kľudne aj BubbleSort, ale s foldami, bez rekurzie.

```
sort' xs = ? foldr ? ? ?,           sort' xs ys = ? foldl ? ? ?.
```

-------- v prípade potreby riešte túto úlohu na zadnú stranu **TOHOTO listu**----------

3. [4 body - Kuchársky]
V školskej kuchyni varia 7 druhov polievok a 7 hlavných jedál, každé z nich obsahuje niektoré alergény. V kuchyni rozpoznávajú 7 druhov alergénov, a kódujú ich bitmi, 1,2,4,8,16,32,64.
Bitové operácie v Haskelli sú: **`.|.`** je bitové OR, **`.&.`** je bitové AND, **`xor`** je XOR, **`complement`** je negácia, **`shift`** `x 5` je `x<<5`, **`shift`** `x (-3)` je `x>>3`.
Toto je pohľad do vymyslenej kuchyne, alergény nezodpovedajú skutočnej realite:

```
type Alergeny = Int   -- existuje 7 alergénov, ich bity sú 1,2,4,8,16,32,64, takže 9 = (1001) sú alergény 1 a 8
type Jedlo  = (String, Alergeny)
polievky    :: [Jedlo]
polievky    = [("sosovicova",32+4+1),("pomodoro",8+2+1),("pohankova",0),("hokajdo",16+2),("gulasovka",8+4+2),("drzkova",32+2+1),("brokolicova",64+16+1)]
hlavne      :: [Jedlo]
hlavne      = [("spagety",16+8+4),("pizza",2+1),("gulas",32+8+4+2),("hambac",1+2+4+8+16+32+64),("dukatove",8+4+1),("perkelt",32+1),("ratatoille",16+2)]

type Menu  = (Jedlo, Jedlo)
```

a) [1 bod] Menu je polievka a hlavné jedlo. Definujte funkciu **`pocetAlergenov::Menu->Int`**, ktorá vráti počet rôznych alergénov nachádzajúcich sa v menu, teda v polievke aj hlavnom. Príklad:
`pocetAlergenov (("sosovicova",32+4+1),("spagety",16+8+4)) == 5` lebo `1,4,8,16,32`,
`pocetAlergenov (("pohankova",0),("hambac",1+2+4+8+16+32+64)) == 7` lebo hambáč prebije všetko.

b) [1 bod] Definujte funkciu **`jedla::[Jedlo]->[Jedlo]->Alergeny->[Menu]`**, ktorá zo zoznamu polievok a hlavných chodov vytvorí všetky dvojice typu Menu, ktoré sú **akceptovateľné** pre osobu citlivú na alergény, čo tretí argument funkcie. Príklad:

```
length $ jedla polievky hlavne 0 == 49          lebo 7*7 = 49, bez alergií
length $ jedla polievky hlavne (1+2+4) == 0
length $ jedla polievky hlavne (1+2) == 1       len   (("pohankova",0),("spagety",16+8+4))
length $ jedla polievky hlavne 1 == 9           napr. (("gulasovka",8+4+2),("gulas",32+8+4+2) :)
```

c) [2 body] Vytvorte všetky možné týždenné (5 dní) jedálne lístky **`tyzdenny::[Listok]`**, bez ohľadu na alergény, ale s podmienkou, že sa v týždni neopakuje žiadna polievka ani hlavné jedlo. Príklad:

```
type Listok = [Menu]   -- jedálny lístok
tyzdenny     :: [Listok]
length tyzdenny == 6350400
```

-------- v prípade potreby riešte túto úlohu na zadnú stranu **TOHOTO listu**----------

4. [7 bodov – Go kvíz]

[1bod] Čo vypíše tento program, napíšte jeho výstup v vpravo od neho:

```
func main() {
   fmt.Println("Start")
   go func () {
      fmt.Println("begin")
      time.Sleep(1000)
      fmt.Println("end")
   }()
   fmt.Println("Stop")
}
```

[1bod] Čo vypíše tento program, napíšte v vpravo, miesto UTC času stačí sekunda behu:

```
func main() {
   fmt.Print(time.Now().UTC()); fmt.Println("\tStart")
   go func () {
      fmt.Print(time.Now().UTC()); fmt.Println("\tbegin1")
      time.Sleep(1*time.Second)
      fmt.Print(time.Now().UTC()); fmt.Println("\tend1")
   }()
   go func () {
      fmt.Print(time.Now().UTC()); fmt.Println("\tbegin2")
      time.Sleep(2*time.Second)
      fmt.Print(time.Now().UTC()); fmt.Println("\tend2")
   }()
   time.Sleep(3*time.Second)
   fmt.Print(time.Now().UTC()); fmt.Println("\tStop")
}
```

[1bod] Čo vypíše tento program, napíšte v vpravo, miesto UTC času stačí sekunda behu:

```
func main() {
   fmt.Print(time.Now().UTC()); fmt.Println("\tStart")
   go func () {
      for i := 1; i < 30; i++ {
         time.Sleep(1*time.Second)
         fmt.Print(time.Now().UTC()); fmt.Printf("\ttick%v\n",i)
      }
   }()
   time.Sleep(10*time.Second)
   fmt.Print(time.Now().UTC()); fmt.Println("\tStop")
}
```

[1bod] Bude sa vo výpise tohoto programu **zaručene striedať tick a tack**? Nejako to zdôvodnite.
[1bod] Ak nie, tak ho opravte tak, aby sa zaručene striedali.
Ak áno, netreba nič (ak ste správne odpovedali na predošlú otázku, aj zdôvodnili).

```
func main() {
   rand.Seed(time.Now().UnixNano())
   fmt.Print(time.Now().UTC()); fmt.Println("\tStart")
   ch := make(chan bool)
   go func () {
      for _ = range ch {
         fmt.Print("tick")
         ch <- true
         time.Sleep(time.Duration(rand.Intn(5))*time.Millisecond)
      }
   }()
   go func () {
      for _ = range ch {
         fmt.Print("tack")
         ch <- false
         time.Sleep(time.Duration(rand.Intn(5))*time.Millisecond)
      }

   }()
   ch <- true
   time.Sleep(30*time.Second)
   fmt.Print(time.Now().UTC()); fmt.Println("\tStop")
}
```

[1bod] Bude sa vo výpise tohoto programu **zaručene striedať tick, tack a tuck** ? Nejako to zdôvodnite. Jediný rozdiel oproti programu z minulej strany, že pribudla ďalšia takmer identická go rutina, ktorá robí tuck. Nič viac.
[1bod] Ak nie, tak ho opravte tak, aby sa zaručene striedali v abecednom poradí (tack-tick-tuck). Ak áno, netreba nič.

```
func main() {
   rand.Seed(time.Now().UnixNano())
   fmt.Print(time.Now().UTC()); fmt.Println("\tStart")
   ch := make(chan bool)
   go func () {
      for _ = range ch {
         fmt.Print("tick")
         ch <- true
         time.Sleep(time.Duration(rand.Intn(5))*time.Millisecond)
      }
   }()
   go func () {
      for _ = range ch {
         fmt.Print("tack")
         ch <- false
         time.Sleep(time.Duration(rand.Intn(5))*time.Millisecond)
      }

   }()
   go func () {
      for _ = range ch {
         fmt.Print("tuck")
         ch <- false
         time.Sleep(time.Duration(rand.Intn(5))*time.Millisecond)
      }
   }()
   ch <- true
   time.Sleep(30*time.Second)
   fmt.Print(time.Now().UTC()); fmt.Println("\tStop")
}
```

-------- v prípade potreby riešte túto úlohu na zadnú stranu **TOHTO listu** ----------

5. [3 body – konkurentný]

Toto je rekurzívny sekvenčný MergeSort v jazyku GO, ktorý triedi pole čísel `numbers` s použitím pomocného `temp`. Navrhnite jeho konkurentnú verziu prostriedkami jazyka GO. **Nepíšte celý kód**, **len vyznačte potrebné zmen**y v tomto kóde, prípadne prepíšte, len, čo treba.

```
type Pole []int
var ( numbers Pole // globálna premenná ako triedene pole
      temp    Pole // pomocne pole pre mergesort
)
func mergesort(low, high int) {
          if low < high {
                  middle := low + (high-low)/2
                  mergesort(low, middle)
                  mergesort(middle+1, high)
                  merge(low, middle, high)
          }
}
func merge(low, middle, high int) {
//zlepí dve časti numbers[low:middle] a numbers[(middle+1):high] cez pole temp
          for i := low; i <= high; i++ {
                  temp[i] = numbers[i] // prekopiruje do pomocneho pola
          }
          i := low
          j := middle + 1
          k := low
          for i <= middle && j <= high {
                  if temp[i] <= temp[j] {
                          numbers[k] = temp[i]
                          i++
                  } else {
                          numbers[k] = temp[j]
                          j++
                  }
                  k++
          }
          for i <= middle {
                  numbers[k] = temp[i]
                  k++
                  i++
          }
}
```

-------- v prípade potreby riešte túto úlohu na zadnú stranu **TOHOTO listu**----------