Université d'Ottawa
Faculté de génie

École de science
d'informatique
et de génie électrique

uOttawa
L'Université canadienne
Canada's university

University of Ottawa
Faculty of Engineering

School of Electrical
Engineering
and Computer Science

Assignment 3
CSI2120 Programming Paradigms
Winter 2018
Due on April 10th before 11:00 pm in Virtual Campus
6 marks

There are [8 points] in this assignment. The assignment is worth 6% of your final mark.

All code must be submitted in a go file(s). Screenshots, files in a format of a word editor, pdfs, handwritten solutions, etc. will not be marked and receive an automatic 0.

Question 1. **[2.5 points]**

Write a function `AbsDiff` that calculates the absolute difference of the corresponding elements of two slices of floating point numbers. The function also should return a slice of floating point numbers containing the results. If the two slices have different length an error is to be returned that prints as

```
func AbsDiff(sliceA, sliceB []float32)
                              (res []float32, err error)
```

If the two slices have different length an error is to be returned that prints as
`Slices are not the same length.`

Create a main function that repeatedly asks the user for new input of a slice and calculates the result with two most recent slices.
Example (note that the first previous slice is set in the program):

```
Previous slice: [3.2 -6.77 42 -0.9]
Enter another slice of floating point numbers (Anything else to end
slice)
5.4 6 7.8 -10
Result: [2.2 12.77 34.2 9.1]
q to quit (Anything else to continue): c

Previous slice: [5.4 6 7.8 -10]
Enter another slice of floating point numbers (Anything else to end
slice)
17.5 1123.98 0.001
```

```
Slices are not the same length
q to quit (Anything else to continue): q
```

Update the function to provide options for handling varied length input slices. An extra integer should select the strategy.

```
func AbsDiff(sliceA, sliceB []float32, version int)
                            (res []float32, err error)
```

a) If version is 0, the function should work as before and return an error if slices are not the same length.
b) If version is -1, assume that the missing elements in the shorter slice have the value of 0. The result will be of length equal to the longer slice.
c) If version is 1, ignore the extra values of the longer list. The result will be of length equal to the shorter list.

Question 2. **[3.5 points]**

1. Create a `struct Bread` with the following fields :
   - A `string` with the `name` of the bread
   - A map of with key of type `string` and value of type `Item` for the `ingredients` of the bread
   - An `float32` with `weight` of the bread in kilograms
   - A `struct baking` containing baking information.

   The helper `struct baking` should have three fields: the `bakeTime` and `coolTime` in minutes, the `temperature` in Celcius, all as `int`. The helper `struct Item` should have a single field `weight` of type `int`.
2. Create an interface `Baker` with the following two methods
   - `shoppingList` which accepts a list of ingredients as a map of `string`:`Item` and returns two list of ingredients as maps of `string`:`Item`.
   - `printBakeInstructions` with no arguments and no return.
   - `printBreadInfo` with no arguments and no return.
3. Implement the following global functions
   - `NewBread` returning a pointer to `Bread` with the field `name` set to “Whole Wheat”, the field `ingredients` set to a map containing the key value pairs `"whole wheat flour":{500}, "yeast":{25}, "salt":{25}, "sugar":{50}, "butter":{50}, "water":{350}`. The bake time is 2hrs at 180 degrees Celsius and cool time is 1 hr. Calculate the weight of the bread as the sum of the weight of the ingredients.

- `NewBreadVariation` also returning a pointer to `Bread` which accept a new name and two maps of `string`:`Item` possibly nil that specify extra ingredients to be added and ingredients to be removed. All unchanged fields in the structure `Bread` that are not set should be identical as in the bread returned by `NewBread`

4. Implement methods of the interface `Baker` for a pointer to `Bread`
   - The method `shoppingList` that allows one to obtain a list of missing items, i.e., a map of `string`:`Item` containing the difference between the available items in a `string`:`Item` and the needed ingredients for the baked good. The second map should return the items leftover after baking the bread.
   - The method printBakeInstructions that prints temperature and duration in minutes of baking required.
   - The method printBakeInfo that prints the name, ingredients and weight to console.
5. Supply a main routine that constructs two breads in a slice of Baker, one standard whole wheat and one sesame with half whole wheat and half white flour. Then obtain a shopping list assuming you have currently 5kg of whole wheat flour, 500g of salt and 1Kg of sugar available. The shopping list must contain all the ingredients that are needed in addition to what is currently available to bake all breads in the slice (one whole wheat and one sesame in the example). Print the shopping list and then print the baking instructions.

Example Output:

```
Whole Wheat bread
map[whole wheat flour:{500} yeast:{25} salt:{25} sugar:{50}
butter:{50} water:{350}]
Weight 1.000 kg

Sesame bread
map[water:{350} white flour:{200} sesame:{50} whole wheat flour:{250}
yeast:{25} salt:{25} sugar:{50} butter:{50}]
Weight 1.000 kg


Shopping List:
map[yeast:{50} butter:{100} water:{700} white flour:{200}
sesame:{50}]

Baking Instructions:
Bake at 180 Celsius for 120 minutes and let cool for 60 minutes.
Bake at 180 Celsius for 120 minutes and let cool for 60 minutes.
```

Question 3. **[2 points]**

The function `RandomArray` generates an array of random numbers.

```
func RandomArray(len int) []float32 {
    array := make([]float32, len)
    for i := range array {
        array[i] = rand.Float32()
    }
    return array
}
```

Below is a (incomplete) main program that calculates 1000 arrays of varying length concurrently.

```
func main() {
   rand.Seed(100) // use this seed value

   out := make(chan float32)
   defer close(out)

   for i := 0; i<1000 ; i++ {
      a:= RandomArray(2*(50+rand.Intn(50)))
      go Process(a,out)
   }

   // ******
   // read here the results of the processing
   // and sum these results

   fmt.Println(sum)
}
```

The function Process is to implement the following procedure:

- Split each array into two equal-sized sub-arrays. (The first N/2 elements and the remaining N/2 elements).
- Call the function `AbsDiff` with the two sub-arrays
- Sum the elements of the array obtained with `AbsDiff`.

Write the function `Process` and complete the `main` function. Please hand in your complete program.