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

CSI2520 Paradigmes de programmation

Hiver 2017

Devoir 6 (5%)

Question 1. **[3 points]**

Soit les chiffres $d_k, d_{k-1}, \ldots, d_1, d_0$ constituant un nombre entier positif. Et soit la somme des carrés de ces chiffres: $d_k^2 + d_{k-1}^2 + \cdots, d_1^2 + d_0^2$. Il est possible de créer une suite récursive dans laquelle chaque élément de la suite est la somme des carrés des chiffres de l'élément précédent.

a) Créer la fonction Go `sosd` calculant la somme des carrés des chiffres d'un nombre.

```
func sosd( num int) int {

  sum := 0

  for num > 0 {
        mod := num%10
        sum+= mod*mod
        num /= 10
  }

  return sum
}
```

Il a été démontré que pour tout nombre de départ, la série décrite va toujours atteindre l'un des nombres suivants : 0,1,4,16,20,37,42,58,89,145 (OEIS A039943; Porges 1945). Si la série avec comme départ un nombre `H` atteint le nombre 1, alors le nombre `H` est dit 'nombre heureux'.

b) Créer la fonction Go `stop` vérifiant si un nombre appartient à l'un des nombres ci-haut.

```
func stop(num int) bool {

list := []int{0,1,4,16,20,37,42,58,89,145}
```

```
    for _, j := range list {
        if j == num {
            return true
        }
    }
    return false
}
```

c) Créer la fonction Go `suite_ssod` générant la suite décrite ci-haut en s'arrêtant lorsqu'un nombre d'arrêt est rencontré (la récursivité n'est pas requise ici, vous devriez générer la suite avec une boucle). Le nombre spécifié en entrée ne fait pas partie de la suite.

```
func suite_ssod(num int) []int{

  var suite []int

  for fin := false; !fin; fin= stop(num) {
   num = sosd(num)
   suite = append(suite, num)
  }

  return suite
}
```

d) Créer la fonction Go `heureux` qui retourne vrai si un nombre est heureux. Pour ce faire vous devez générer la liste de nombres avec `suite_ssod` et vérifier la valeur du dernier élément de la liste.

```
func happy(num int) bool {
 var suite []int = suite_ssod(num)
 return suite[len(suite)-1] == 1
}
```

Question 2. **[2 points]**

Le type `Pixel` représente un point alors que `Line` décrit une ligne à l'aide d'un point de départ et d'un point d'arrivée. La méthode `linear` permet d'interpoler les coordonnées des points se trouvant sur cette ligne.

La fonction `main` de ce programme produit des points (c'est un producteur de données).

a) On vous demande de changer la fonction `main` de façon à ce que les points produits soient envoyés dans un *channel*.

b) On vous demande d'ajouter une fonction `consomme` qui va consommer les données envoyées dans le *channel*. Cette fonction doit simplement afficher chaque point reçu et attendre 2 secondes entre chaque point.

```
fmt.Printf("(%f,%f)\n", point.x, point.y)
time.Sleep(2*time.Second)
```

c) On vous demande de modifier la fonction `main` de façon à ce qu'elle lance la fonction `consomme` en go routine et attende que celle-ci termine le traitement de tous les points.

```
import (
	"fmt"
)

type Pixel struct {
	x, y float32
}

type Line struct {
	startPoint, endPoint Pixel
}

// Linear interploation
func (l *Line) linear(t float32) *Pixel {
	return &Pixel{(1.0-t)*l.startPoint.x + t*l.endPoint.x,
		(1.0-t)*l.startPoint.y + t*l.endPoint.y}
}
```

```
func main() {
	l := Line{Pixel{1.0, 3.0}, Pixel{7.0, -2.0}}
	point := make([]Pixel, 10)

	// ICI: créer le channel et lancer le consommateur

	// production de points
	for i, t := 0, float32(0.0); i < 10; i, t = i+1, t+0.1 {
		point[i] = *l.linear(t)
		// **ICI: envoyer le point produit dans un channel
	}

	// **ICI: attendre que tous les points soient consommés

}
```

**solution attendue

```
package main

import (
	"fmt"
	"time"
)

type Pixel struct {
	x, y float32
}

type Line struct {
	startPoint, endPoint Pixel
}


// Linear interploation
func (l *Line) linear(t float32) *Pixel {
	return &Pixel{(1.0-t)*l.startPoint.x +
			t*l.endPoint.x,
		(1.0-t)*l.startPoint.y +
			 t*l.endPoint.y}
}

func consomme(jobs chan Pixel, done chan bool) {
	  for {
```

```
            point, more := <-jobs

            if more {

                 fmt.Printf("(%f,%f)\n", point.x,
                                         point.y)
                  time.Sleep(2*time.Second)

            } else {
                fmt.Println("C'est fini!")
                done <- true
                return
            }
        }
}

func main() {
      l := Line{Pixel{1.0, 3.0},
                Pixel{7.0, -2.0}}
      point := make([]Pixel, 10)

        jobs := make(chan Pixel, 10)
        done := make(chan bool)

        go consomme(jobs,done)

      // production de points
      for i, t := 0, float32(0.0);
                i < 10; i, t = i+1, t+0.1 {
            point[i] = *l.linear(t)
            jobs <- point[i]
      }

        close(jobs)
        <-done
}
```

**une variante intéressante

```
package main

import (
      "fmt"
      "time"
)
```

```
type Pixel struct {
      x, y float32
}

type Line struct {
      startPoint, endPoint Pixel
}


// Linear interploation
func (l *Line) linear(t float32) *Pixel {
      return &Pixel{(1.0-t)*l.startPoint.x +
                        t*l.endPoint.x,
            (1.0-t)*l.startPoint.y +
                         t*l.endPoint.y}
}

func consomme(point *Pixel, done chan bool) {

     time.Sleep(2*time.Second)
     fmt.Printf("(%f,%f)\n", point.x, point.y)
     done <- true
}

func main() {
      l := Line{Pixel{1.0, 3.0},
                Pixel{7.0, -2.0}}
      point := make([]Pixel, 10)

        done := make(chan bool,5)


      // production de points
      for i, t := 0, float32(0.0);
                i < 10; i, t = i+1, t+0.1 {
            point[i] = *l.linear(t)
                go consomme(&point[i],done)
      }

     for i := 0 ; i<10; i++ {
               <-done
        }
}
```