Algorithmique et programmation
Mise en œuvre en Go

Consolidation et structures composées

Structures de contrôle

Retour sur des algorithmes vus au semestre 1

Objectif :
rappel sur les structures de contrôle vues au semestre 1

Retour sur des algorithmes vus au semestre 1

Structures de contrôle : répétition avec compteur

```
Fonction sommeNPremiers(limite)
début
   variable somme entier
   variable nombre entier
   somme = 0
   nombre = 1
   tant que (nombre <= limite) faire
      somme = somme + nombre
      nombre = nombre + 1
   fin  tant que
   retourner somme
fin

Algorithme principal
   variable resultat
   resultat = sommeNPremiers(5)
   afficher(resultat)
```

```
package main

import "fmt"

func sommeNPremiers(limite int) int {
  var (
    somme = 0
    nombre = 1
   )
  for (nombre <= limite) {
    somme = somme + nombre
    nombre = nombre + 1
   }
  return somme
}

func main() {
  var resultat int
  resultat  = sommeNPremiers(5)
  fmt.Println(resultat)
}
```

Calculer la somme des n premiers entiers positifs

Retour sur des algorithmes vus au semestre 1

Structures de contrôle : répétition avec compteur

Déroulement

```
package main

import "fmt"

func sommeNPremiers(limite int) int {
  var (
    somme = 0
    nombre = 1
   )
  for (nombre <= limite) {
    somme = somme + nombre
    nombre = nombre + 1
   }
  return somme
}

func main() {
  var resultat int
  resultat  = sommeNPremiers(5)
  fmt.Println(resultat)
}
```

Calculer la somme des n premiers entiers positifs

Retour sur des algorithmes vus au semestre 1

Structures de contrôle : répétition sur condition

```
Fonction demanderNote(message)
début
    variable note réel
    afficher(message)
    lire(note)
    tant que (note <0 ou note > 20) faire
        afficher("erreur, la note doit être comprise entre 0 et 20")
        afficher(message)
        lire(note)
    fin  tant que
    retourner note
fin

Algorithme principal
variable resultat entier
resultat = demanderNote("quelle note avez-vous eue à l'UE MIA0105T ?")
si resultat >= 10 alors
    afficher("bravo !")
fin si
```

```
package main

import "fmt"

func demanderNote(message string) float64 {
    var note float64
    fmt.Print(message)
    fmt.Scan(&note)
    for (note < 0 || note > 20) {
        fmt.Println("erreur, la note doit être comprise entre 0 et 20")
        fmt.Print(message)
        fmt.Scan(&note)
     }
    return note
}

func main() {
    var resultat float64
    resultat = demanderNote("quelle note avez-vous eue à l'UE MIA0105T ?")
    if resultat >= 10 {
        fmt.Println("bravo !")
    }
}
```

Saisie avec vérification

Retour sur des algorithmes vus au semestre 1

Structures de contrôle : répétition sur condition

Déroulement avec le jeu d'essais suivant :

- essai 1 : 16
- essai 2 : 9
- essai 3 : 20
- essai 4 : -5, 90, 0

à vous

```
package main

import "fmt"

func demanderNote(message string) float64 {
    var note float64
    fmt.Println(message)
    fmt.Scan(&note)
    for (note < 0 || note > 20) {
        fmt.Println("erreur, la note doit être comprise entre 0 et 20")
        fmt.Println(message)
        fmt.Scan(&note)
     }
    return note
}

func main() {
    var resultat float64
    resultat = demanderNote("quelle note avez-vous eue à l'UE MIA0105T ?")
    if resultat >= 10 {
        fmt.Println("bravo !")
    }
}
```

Saisie avec vérification

Retour sur des algorithmes vus au semestre 1

Structures de contrôle : conditionnelle

```
Fonction demanderNote(message)
début
    variable note réel
    afficher(message)
    lire(note)
    tant que (note <0 ou note > 20) faire
        afficher("erreur, la note doit être comprise entre 0 et 20")
        afficher(message)
        lire(note)
    fin  tant que
    retourner note
fin

Algorithme principal
variable resultat entier
resultat = demanderNote("quelle note avez-vous eue à l'UE MIA0105T ?")
si resultat >= 10 alors
    afficher("bravo !")
fin si
```

```
package main

import "fmt"

func demanderNote(message string) float64 {
    var note float64
    fmt.Println(message)
    fmt.Scan(&note)
    for (note < 0 || note > 20) {
        fmt.Println("erreur, la note doit être comprise entre 0 et 20")
        fmt.Println(message)
        fmt.Scan(&note)
     }
    return note
}

func main() {
    var resultat float64
    resultat = demanderNote("quelle note avez-vous eue à l'UE MIA0105T ?")
    if resultat >= 10 {
        fmt.Println("bravo !")
    }
}
```

Saisie avec vérification

Retour sur des algorithmes vus au semestre 1

Structures de contrôle : conditionnelle

Déroulement avec le jeu d'essais suivant :

- essai 1 : 16
- essai 2 : 9
- essai 3 : 10

à vous

```
package main

import "fmt"

func demanderNote(message string) float64 {
    var note float64
    fmt.Println(message)
    fmt.Scan(&note)
    for (note < 0 || note > 20) {
        fmt.Println("erreur, la note doit être comprise entre 0 et 20")
        fmt.Println(message)
        fmt.Scan(&note)
     }
    return note
}

func main() {
    var resultat float64
    resultat = demanderNote("quelle note avez-vous eue à l'UE MIA0105T ?")
    if resultat >= 10 {
        fmt.Println("bravo !")
    }
}
```

Saisie avec vérification

Retour sur des algorithmes vus au semestre 1

Structures de contrôle : conditionnelle

```
Fonction minimum(x, y)
début
   variable resultat réel
   si x < y alors
      resultat = x
   sinon
      resultat = y
   fin si
   retourner resultat
fin

Algorithme principal
   variable a, b, mini réel
   afficher("donner 2 nombres :")
   lire(a)
   lire(b)
   mini = minimum(a, b)
   afficher(mini)
```

```
package main

import "fmt"

func minimum(x float64, y float64) float64 {
  var resultat float64
  if x < y {
    resultat = x
  } else {
    resultat = y
  }
  return resultat
}

func main() {
  var a, b, mini float64
  fmt.Println("donner 2 nombres :")
  fmt.Scan(&a)
  fmt.Scan(&b)
  mini  = minimum(a, b)
  fmt.Println(mini)
}
```

Calculer le minimum de 2 nombres

Retour sur des algorithmes vus au semestre 1

Structures de contrôle : conditionnelle

Que se passe-t-il à l'exécution ?

à vous

```
package main

import "fmt"

func minimum(x float64, y float64) float64 {
  var resultat float64
  if x < y {
    resultat = x
  } else {
    resultat = y
  }
  return resultat
}

func main() {
  var a, b, mini float64
  fmt.Println("donner 2 nombres :")
  fmt.Scan(&a)
  fmt.Scan(&b)
  mini  = minimum(a, b)
  fmt.Println(mini)
}
```

Calculer le minimum de 2 nombres

Deux nouvelles structures de contrôle

Structure conditionnelle "switch"

- L'instruction **switch** du langage Go (qui existe également dans d'autres langages) facilite l'écriture des instructions conditionnelles comportant de nombreuses branches...
  - attention : dans l'algorithme, on conserve la forme "si … alors … sinon" ; on utilise switch pour la traduction en Go
- Elle a **deux** formes :
  - La **première** permet de tester simplement si une variable contient une certaine valeur.
  - La **seconde** permet de tester plusieurs conditions de façon simple.

Instruction switch : première forme

```
if variable == val1 {
    instructions si variable == val1
} else if variable == val2 {
    instructions si variable == val2
} ...
} else {
   instructions si variable ne correspond à aucun cas
}
```

```
switch variable {
   case val1:
      instructions si variable == val1
   case val2:
      instructions si variable == val2
   ...
   default:
      instructions si variable ne correspond à aucun cas
}
```

```
if variable == val1 || variable == val2 || variable == val3 {
    instructions si variable == val1 ou val2 ou val3
} else if variable == val4 {
    instructions si variable == val4
} ...
} else {
   instructions si variable ne correspond à aucun cas
}
```

```
switch variable {
   case val1, val2, val3:
      instructions si variable == val1 ou val2 ou val3
   case val4:
      instructions si variable == val4
   ...
   default:
      instructions si variable ne correspond à aucun cas
}
```

Première forme du switch : exemple

Cette forme du switch est particulièrement utile pour gérer les réponses de l'utilisateur et, plus généralement, pour tester si une variable **est égale à** une valeur précise :

```
var (
   choix char
   ok       bool
)

fmt.Println("Choisir une option :")
fmt.Println("Option A : A")
fmt.Println("Option B : B")
fmt.Println("Option C : C")
fmt.Print("Votre choix : ")
fmt.Scan(&choix)

ok = false
for !ok {
   ok = true
   switch choix {
      case 'a', 'A':
       // traiter l'option A
      case 'b', 'B':
      // traiter l'option B
      case 'c', 'C':
      // traiter l'option C
       default:
        ok = false
   }
}
// On continue...
```

Instruction switch : deuxième forme

```
if variable == val1 {
    instructions si variable == val1
} else if variable2 < val2 {
    instructions si variable2 < val2
} else if variable3 > variable {
    instructions si variable3 > variable
} else {
    instructions aucun cas précédent n'a été rencontré
}
```

```
switch {
    case variable == val1:
        instructions si variable == val1
    case variable2 < val2:
        instructions si variable2 < val2
    case variable3 > variable:
        instructions si variable3 > variable
    default:
        instructions si aucun cas précédent n'a été rencontré
}
```

Cette forme du switch permet de simplifier les longues séries de **else if**...

Deuxième forme du switch : exemple

```
func main() {
    const maxCoups = 5
    var (
        secret, nbEssais, nombre int
        trouve                   bool
    )

    rand.Seed(time.Now().UnixNano()) // Pour initialiser le générateur de nbres aléatoires
    secret = rand.Intn(100) + 1
    nbEssais = 0
    trouve = false
    for nbEssais < maxCoups && !trouve {
        fmt.Print("Entrez une valeur comprise entre 1 et 100 : ")
        fmt.Scan(&nombre)
        nbEssais = nbEssais + 1
        if nombre < secret {
            fmt.Println("C'est plus...")
        } else if nombre > secret {
            fmt.Println("C'est moins...")
        } else {
            fmt.Printf("Bravo vous avez trouvé en %v coups\n", nbEssais)
            trouve = true
        }
    }
    if !trouve {
        fmt.Printf("Désolé, il fallait trouver %v\n", secret)
    }
}
```

```
switch {
  case nombre < secret:
    fmt.Println("C'est plus...")
  case nombre > secret:
    fmt.Println("C'est moins...")
  default:
    fmt.Println("Bravo, vous avez trouvé en %v coups\n", nbEssais)
    trouve = true
}
```

Structure conditionnelle switch

exercice

Ecrire le programme Go qui affiche tous les nombres de 1 à 100 avec les contraintes suivantes :

- le nombre est divisible par 3, on affiche "pif"
- le nombre est divisible par 5, on affiche "paf"
- le nombre est divisible par 3 et par 5, on affiche "pifpaf"
- pour les autres cas on affiche le nombre

Structure conditionnelle switch

exercice (corrigé)

Ecrire le programme Go qui affiche tous les nombres de 1 à 100 avec les contraintes suivantes :

- le nombre est divisible par 3, on affiche "pif"
- le nombre est divisible par 5, on affiche "paf"
- le nombre est divisible par 3 et par 5, on affiche "pifpaf"
- pour les autres cas on affiche le nombre

```
// Renvoie "pif", "paf", "pifpaf" ou le nombre sous forme de chaîne
func pifPaf(nbre int) string {
        var res string
        switch {
        case nbre%3 == 0 && nbre%5 == 0: // ou case nbre % 15 == 0:
                res = "pifpaf"
        case nbre%3 == 0:
                res = "pif"
        case nbre%5 == 0:
                res = "paf"
        default:
                res = fmt.Sprint(nbre)
        }
        return res
}

func main() {
    var nombre int

    nombre = 1
    for nombre <= 100 {
        fmt.Printf("%s ", pifPaf(nombre))
        nombre++
    }
    fmt.Println()
}
```

Exercice

à partir du site https://www.tisseo.fr/les-tarifs/la-gamme, écrire un programme qui calcule et affiche le prix à payer pour un abonnement annuel (avec carte pastel) en fonction de l'âge donné par l'utilisateur.

Conception de l'algorithme : utilisez la méthode vue au semestre 1 (quelles données sont nécessaires (variables/constantes) ? quelles actions/calculs ? algorithme principal ? raffinage des sous-programmes ?)

Puis, traduction en Go

Une nouvelle structure répétitive : le "Pour"

- On a souvent vu que, pour répéter une suite d'instructions un certain nombre de fois, on utilisait une boucle "Tant Que" avec un compteur :

```
cpteur = 1
TQ cpteur <= valeurMax FAIRE
    …
    cpteur = cpteur + 1
FTQ
```

```
cpteur = 1
for cpteur <= valeurMax {
    …
    cpteur = cpteur + 1
}
```

- La boucle "Pour" permet de regrouper toutes les instructions nécessaires à la répétition au même endroit :

```
POUR cpteur de 1 à valeurMax FAIRE
        …
FIN POUR
```

```
for cpteur := 1; cpteur <= valeurMax; cpteur++ {
        …
}
```

Variable de boucle d'un "Pour"

Le "Pour" est une boucle (répétition) avec compteur mais avec un compteur **local** (voir transparent suivant)

Algorithmique

```
POUR cpteur DE deb À fin FAIRE
    instructions
FIN POUR
```

Go

```
for cpteur := deb; cpteur <= fin; cpteur++ {
    instructions
}
```

Exemples

```
POUR i DE 1 À 10 FAIRE
  afficher(i)
FIN POUR
```

```
for i :=1; i <= 10; i++ {
   fmt.Println(i)
}
```

Remarques sur la boucle "Pour"

- La syntaxe est plus condensée que celle d'un "Tant Que"
  - Comparez ces deux codes Go : le premier utilise un "Tant que", le second utilise un "Pour".
  - Les deux boucles font la même chose : afficher la valeur de ***cpteur*** de 1 à 5...mais remarquez la forme plus courte du "Pour". …

Remarques sur la boucle "Pour"

- La boucle "Pour" utilise une variable de boucle **locale** : cette variable est *créée par la boucle et* ***cesse d'exister à la fin de la boucle***.

  - Dans la version du "Pour", l'affichage de ***cpteur*** après la sortie de la boucle échoue car ***cpteur*** n'existe plus : ***cpteur*** est *local* à la boucle, ce qui n'est pas le cas de la version "Tant Que", où il continue d'exister bien qu'il ne soit plus utile (cf le cours précédent sur les espaces de noms).

- Noter le := et les ;

```
for i :=1; i <= 10; i++ {
   fmt.Println(i)
}
```

- La variable de boucle ne doit pas être modifiée dans le corps de la boucle, ni l'intervalle des valeurs

Espaces de noms et boucle "Pour"

- Dans le code ci-contre, analysez les espaces de noms et la portée des variables

```
func sommeNPremiers(lim int) int {
   var res = 0
   for i := 1; i <= 10; i++ {
        res = res + i
    }
    return res
}
…
somme = sommeNPremiers(10)
fmt.Println(somme)
```

Remarques (suite) : progression de la variable de boucle

Algorithmique

```
POUR cpt DE deb À fin FAIRE
    instructions
FIN POUR
```

Go

```
for cpt := deb; cpt <= fin; cpt++ {
    instructions
}
```

En algorithmique, la variable de boucle progresse de 1 par défaut. mais on peut choisir un autre pas de progression :

```
POUR cpt DE deb À fin PAR pas FAIRE
    instructions
FIN POUR
```

```
for cpt := deb; cpt <= fin; cpt += pas {
    instructions
}
```

pas indique le pas de progression.

Et voici comment faire progresser la variable à l'envers : compte à rebours

Boucle Pour : exemple

```
func main() {
    const nbPas = 5

    color("purple")
    down()
    for cpteurCotes := 1; cpteurCotes <=  4; cpteurCotes++ {
        forward(nbPas)
        right()
    }
    up()
}
```

la boucle s'exécute depuis cpteurCotes == 1 jusqu'à cpteurCotes == 4 compris, soit 4 fois

cpteurCotes peut être utilisé dans les instructions à l'intérieur du for mais ne doit pas être modifié

Exercice

Ré-écrire le programme principal de pifpaf (ci-contre) en remplaçant la boucle "Tant que" par une boucle "Pour"

```
// Renvoie "pif", "paf", "pifpaf" ou le nombre sous forme de chaîne
func pifPaf(nbre int) string {
        var res string
        switch {
        case nbre%3 == 0 && nbre%5 == 0: // ou case nbre % 15 == 0:
                res = "pifpaf"
        case nbre%3 == 0:
                res = "pif"
        case nbre%5 == 0:
                res = "paf"
        default:
                res = fmt.Sprint(nbre)
        }
        return res
}

func main() {
    var nombre int

    nombre = 1
    for nombre <= 100 {
        fmt.Printf("%s ", pifPaf(nombre))
        nombre++
    }
    fmt.Println()
}
```

Boucle "Pour" ou boucle "Tant Que" ?

- Lorsque vous avez compris que votre algorithme devra répéter une suite d'opérations, vous devez ensuite vous poser la question de savoir quel type de boucle correspond le mieux à la situation. **Cette décision est simple** :
  - Si **vous savez** combien de fois il faudra répéter une suite d'opérations, c'est un "**Pour**", qui ira de 1 au nombre de répétitions voulu.
  - Si **vous devez utiliser un compteur**, c'est un "**Pour**" dont la variable de boucle représentera ce compteur, qui ira donc de sa valeur de début (souvent 0 ou 1) à sa valeur de fin (en utilsant un test **<** ou **<=**). Même principe pour un compte à rebours.
  - Sinon, ce sera un "**Tant Que**" qui bouclera ***tant qu'une certaine condition est vraie***, mais sans savoir le nombre de répétitions qu'il faudra faire…

Boucle "Pour" ou boucle "Tant Que" ?

- En Go, le "Pour" et le "Tant Que" de l'algorithmique se traduisent par une boucle **for** dont la forme est différente selon le cas :
  - Pour un "Pour", on a une boucle de la forme

    for cpt := deb; cpt <= fin; cpt++ {…}, où **deb** et **fin** déterminent le nombre de répétitions (la mise à jour de la variable peut changer et le test **<=** peut être différent…)

  - Pour un "Tant Que", on a une boucle de la forme

    for condition {…}, où **condition** est un test qui doit valoir **true** pour entrer et repéter la boucle. Attention, si on est entré dans cette boucle, il faudra que **condition** repasse à **false** pour en sortir…

- Tous les autres langages utilisent des mots-clés différents pour les deux boucles : **for** pour le "Pour" et **while** pour le "Tant Que"...

Boucle "Pour" ou boucle "Tant Que" ? Exemples

On veut saisir **10** entiers...

```
var valeur int
for cpteur := 1; cpteur <= 10; cpteur++ {
    fmt.Scan(&valeur)
    ...
}
```

cpteur n'existe plus à la sortie de la boucle...

On veut saisir **des** entiers jusqu'à ce que l'un vaille 42

```
var valeur int = 0
for valeur != 42 {
    fmt.Scan(&valeur)
    ...
}
```

Ici, pour entrer dans la boucle, il faut d'abord que **valeur** soit différent de 42.

Retour sur des algorithmes vus au semestre 1

Objectif : déterminer la structure de contrôle
switch, tant que, pour ?

Retour sur des algorithmes vus au semestre 1

Switch, Tant que, Pour ?

```
Fonction demanderNote(message)
début
    variable note réel
    afficher(message)
    lire(note)
    tant que (note <0 ou note > 20) faire
        afficher("erreur, la note doit être comprise entre 0 et 20")
        afficher(message)
        lire(note)
    fin  tant que
    retourner note
fin

Algorithme principal
variable resultat entier
resultat = demanderNote("quelle note avez-vous eue à l'UE MIA0105T ?")
si resultat >= 10 alors
    afficher("bravo !")
fin si
```

Repérer les structures de contrôle :

- peut-on remplacer par un Switch, un Tant que ou un Pour ?
- si oui, transformer et traduire en Go

à vous

Saisie avec vérification

Retour sur des algorithmes vus au semestre 1

Switch, Tant que, Pour ?

```
Fonction minimum(x, y)
début
    variable resultat réel
    si x < y alors
        resultat = x
    sinon
        resultat = y
    fin si
    retourner resultat
fin

Algorithme principal
    variable a, b, mini réel
    mini = minimum(a, b)
    afficher(mini)
```

Repérer les structures de contrôle :

- peut-on remplacer par un Switch, un Tant que ou un Pour ?
- si oui, transformer et traduire en Go

à vous

Calculer le minimum de 2 nombres

Retour sur des algorithmes vus au semestre 1

Switch, Tant que, Pour ?

```
package main

import "fmt"

func sommeNPremiers(limite int) int {
  var (
    somme = 0
    nombre = 1
   )
  for (nombre <= limite) {
    somme = somme + nombre
    nombre = nombre + 1
   }
  return somme
}

func main() {
  var resultat int
  resultat  = sommeNPremiers(5)
  fmt.Println(resultat)
}
```

Repérer les structures de contrôle :

- peut-on remplacer par un Switch, un Tant que ou un Pour ?
- si oui, transformer

à vous

Calculer la somme des n premiers entiers positifs

Retour sur des algorithmes vus au semestre 1

Switch, Tant que, Pour ? comparer les deux codes Go

```
package main

import "fmt"

func sommeNPremiers(limite int) int {
  var (
    somme = 0
    nombre = 1
   )
  for (nombre <= limite) {
    somme = somme + nombre
    nombre = nombre + 1
   }
  return somme
}

func main() {
  var resultat int
  resultat  = sommeNPremiers(5)
  fmt.Println(resultat)
}
```

```
package main

import "fmt"

func sommeNPremiers(limite int) int {
  var somme = 0
  for nombre := 1 ; nombre <= limite ; nombre ++ {
    somme = somme + nombre
   }
  return somme
}

func main() {
  var resultat int
  resultat  = sommeNPremiers(5)
  fmt.Println(resultat)
}
```

Calculer la somme des n premiers entiers positifs

Exercice

Ecrire un programme qui calcule et affiche le total à payer pour 5 articles ; l'utilisateur donne le prix (> 0) de chacun des 5 articles.

Conception de l'algorithme : utilisez la méthode vue au semestre 1 (quelles données sont nécessaires (variables/constantes) ? quelles actions/calculs ? algorithme principal ? raffinage des sous-programmes ?)

Puis, traduction en Go