Go Piscine

Go 09

Summary: THIS document is the subject for the Go 09 module of the Go Piscine @ 42Tokyo.

Contents

Chapter I

Instructions

- Only this page will serve as reference; do not trust rumors.
- Watch out! This document could potentially change up to an hour before submission.
- These exercises are carefully laid out by order of difficulty - from easiest to hardest. We `will not` take into account a successfully completed harder exercise if an easier one is not perfectly functional.
- Make sure you have the appropriate permissions on your files and directories.
- You have to follow the submission procedures for every exercise.
- Your exercises will be checked and graded by your fellow classmates.
- You cannot leave any additional file in your directory than those specified in the subject.
- Got a question? Ask your peer on the right. Otherwise, try your peer on the left.
- Your reference guide is called `Google / man / the Internet / ....`
- Examine the examples thoroughly. They could very well call for details that are not explicitly mentioned in the subject...
- If no other explicit information is displayed, you must use the latest versions of Go.
- Your turn-in directory for each exercise should look something like this:

```
ex[XX]
|-- main.go
|-- vendor
    |-- ft
        |-- printrune.go
    |-- piscine
        |-- [excercisename].go
```

Chapter II

Exercise 00 : listpushback

	Exercise 00
listpushback	
Turn-in directory : *ex*00/	
Files to turn in : `*`	
Allowed packages : `fmt`	
Allowed builtin functions : `None`	

Write a function ListPushBack that inserts a new element NodeL at the end of the list l while using the structure List.

- Expected function and structure

```
type NodeL struct {
        Data interface{}
        Next *NodeL
}

type List struct {
        Head *NodeL
        Tail *NodeL
}

func ListPushBack(l *List, data interface{}) {

}
```

- Usage

```
package main

import (
        "fmt"
        "piscine"
)

func main() {

        link := &piscine.List{}

        piscine.ListPushBack(link, "Hello")
        piscine.ListPushBack(link, "there")
        piscine.ListPushBack(link, "how are you")

        for link.Head != nil {
                fmt.Println(link.Head.Data)
                link.Head = link.Head.Next
        }
}
```

- Output of usage

```
$ go mod init ex00
$ go run .
Hello
there
how are you
$
```

Chapter III

Exercise 01 : listpushfront

	Exercise 01
listpushfront	
Turn-in directory : *ex*01/	
Files to turn in : `*`	
Allowed packages : `fmt`	
Allowed builtin functions : `None`	

Write a function ListPushFront that inserts a new element NodeL at the beginning of the list l while using the structure List

- Expected function and structure

```
type NodeL struct {
        Data interface{}
        Next *NodeL
}

type List struct {
        Head *NodeL
        Tail *NodeL
}

func ListPushFront(l *List, data interface{}) {

}
```

- Usage

```go
package main

import (
        "fmt"
        "piscine"
)

func main() {

        link := &piscine.List{}

        piscine.ListPushFront(link, "Hello")
        piscine.ListPushFront(link, "there")
        piscine.ListPushFront(link, "how are you")

        it := link.Head
        for it != nil {
                fmt.Print(it.Data, " ")
                it = it.Next
        }
        fmt.Println()
}
```

- Output of usage

```
$ go mod init ex01
$ go run .
how are you there Hello
$
```

Chapter IV

Exercise 02 : listsize

	Exercise 02
listsize	
Turn-in directory : *ex*02/	
Files to turn in : `*`	
Allowed packages : `fmt`	
Allowed builtin functions : `None`	

Write a function ListSize that returns the number of elements in a linked list l.

- Expected function and structure

```
type NodeL struct {
        Data interface{}
        Next *NodeL
}

type List struct {
        Head *NodeL
        Tail *NodeL
}

func ListSize(l *List) int {

}
```

- Usage

```
package main

import (
        "fmt"
        "piscine"
)

func main() {
        link := &piscine.List{}

        piscine.ListPushFront(link, "Hello")
        piscine.ListPushFront(link, "2")
        piscine.ListPushFront(link, "you")
        piscine.ListPushFront(link, "man")

        fmt.Println(piscine.ListSize(link))
}
```

- Output of usage

```
$ go mod init ex02
$ go run .
4
$
```

Chapter V

Exercise 03 : listlast

Exercise 03
listlast
Turn-in directory : *ex*03/
Files to turn in : `*`
Allowed packages : `fmt`
Allowed builtin functions : `None`

Write a function ListLast that returns the last element of a linked list l.

- Expected function and structure

```
type NodeL struct {
        Data interface{}
        Next *NodeL
}

type List struct {
        Head *NodeL
        Tail *NodeL
}

func ListLast(l *List) interface{} {

}
```

- Usage

```go
package main

import (
        "fmt"
        "piscine"
)

func main() {
        link := &piscine.List{}
        link2 := &piscine.List{}

        piscine.ListPushBack(link, "three")
        piscine.ListPushBack(link, 3)
        piscine.ListPushBack(link, "1")

        fmt.Println(piscine.ListLast(link))
        fmt.Println(piscine.ListLast(link2))
}
```

- Output of usage

```
$ go mod init ex03
$ go run .
1
<nil>
$
```

Chapter VI

Exercise 04 : listclear

	Exercise 04
listclear	
Turn-in directory : *ex*04/	
Files to turn in : `*`	
Allowed packages : `fmt`	
Allowed builtin functions : `None`	

Write a function ListClear that deletes all nodes from a linked list l.

- Tip: assign the list's pointer to nil.
- Expected function

```
func ListClear(l *List) {

}
```

- Usage

```
package main

import (
        "fmt"
        "piscine"
)

type List = piscine.List
type Node = piscine.NodeL

func PrintList(l *List) {
        link := l.Head
        for link != nil {
                fmt.Print(link.Data, " -> ")
                link = link.Next
        }
        fmt.Println(nil)
}

func main() {
        link := &List{}

        piscine.ListPushBack(link, "I")
        piscine.ListPushBack(link, 1)
        piscine.ListPushBack(link, "something")
        piscine.ListPushBack(link, 2)

        fmt.Println("------list------")
        PrintList(link)
        piscine.ListClear(link)
        fmt.Println("------updated list------")
        PrintList(link)
}
```

- Output of usage

```
\$ go mod init ex04
$ go run .
------list------
I -> 1 -> something -> 2 -> <nil>
------updated list------
<nil>
\$
```

Chapter VII

Exercice 05 : listat

	Exercise 05
listat	
Turn-in directory : *ex*05/	
Files to turn in : `*`	
Allowed packages : `fmt`	
Allowed builtin functions : `None`	

Write a function ListAt that takes a pointer to the list l and an int pos as parameters. This function should return the NodeL in the position pos of the linked list l.

- In case of error the function should return nil.
- Expected function and structure

```
type NodeL struct {
        Data interface{}
        Next *NodeL
}


func ListAt(l *NodeL, pos int) *NodeL{

}
```

- Usage

```
package main

import (
        "fmt"
        "piscine"
)

func main() {
        link := &piscine.List{}

        piscine.ListPushBack(link, "hello")
        piscine.ListPushBack(link, "how are")
        piscine.ListPushBack(link, "you")
        piscine.ListPushBack(link, 1)

        fmt.Println(piscine.ListAt(link.Head, 3).Data)
        fmt.Println(piscine.ListAt(link.Head, 1).Data)
        fmt.Println(piscine.ListAt(link.Head, 7))
}
```

- Output of usage

```
$ go mod init ex05
$ go run .
1
how are
<nil>
$
```

Chapter VIII

Exercise 06 : listreverse

	Exercise 06
listreverse	
Turn-in directory : *ex*06/	
Files to turn in : `*`	
Allowed packages : `fmt`	
Allowed builtin functions : `None`	

Write a function ListReverse that reverses the order of the elements of a given linked list l.

- Expected function and structure

```
type NodeL struct {
        Data interface{}
        Next *NodeL
}

type List struct {
        Head *NodeL
        Tail *NodeL
}

func ListReverse(l *List) {

}
```

- Usage

```go
package main

import (
        "fmt"
        "piscine"
)

func main() {
        link := &piscine.List{}

        piscine.ListPushBack(link, 1)
        piscine.ListPushBack(link, 2)
        piscine.ListPushBack(link, 3)
        piscine.ListPushBack(link, 4)

        piscine.ListReverse(link)

        it := link.Head

        for it != nil {
                fmt.Println(it.Data)
                it = it.Next
        }

        fmt.Println("Tail", link.Tail)
        fmt.Println("Head", link.Head)
}
```

- Output of usage

```
$ go mod init ex06
$ go run .
4
3
2
1
Tail &{1 <nil>}
Head &{4 0xc42000a140}
$
```

Chapter IX

Exercise 07 : listforeach

	Exercise 07
listforeach	
Turn-in directory : *ex*07/	
Files to turn in : `*`	
Allowed packages : `fmt`	
Allowed builtin functions : `None`	

Write a function ListForEach that applies a function given as argument to the data within each node of the list l.

- The function given as argument must have a pointer as argument: l *List
- Copy the functions Add2_node and Subtract3_node in the same file as the function ListForEach is defined.

- Expected function and structure

```go
type NodeL struct {
        Data interface{}
        Next *NodeL
}

type List struct {
        Head *NodeL
        Tail *NodeL
}

func ListForEach(l *List, f func(*NodeL)) {
}

func Add2_node(node *NodeL) {
        switch node.Data.(type) {
        case int:
                node.Data = node.Data.(int) + 2
        case string:
                node.Data = node.Data.(string) + "2"
        }
}

func Subtract3_node(node *NodeL) {
        switch node.Data.(type) {
        case int:
                node.Data = node.Data.(int) - 3
        case string:
                node.Data = node.Data.(string) + "-3"
        }
}
```

- Usage

```
package main

import (
        "fmt"
        "piscine"
)

func main() {
        link := &piscine.List{}

        piscine.ListPushBack(link, "1")
        piscine.ListPushBack(link, "2")
        piscine.ListPushBack(link, "3")
        piscine.ListPushBack(link, "5")

        piscine.ListForEach(link, piscine.Add2_node)

        it := link.Head
        for it != nil {
                fmt.Println(it.Data)
                it = it.Next
        }
}
```

- Output of usage

```
$ go mod init ex07
$ go run .
12
22
32
52
$
```

Chapter X

Exercise 08 : listforeachif

	Exercise 08
listforeachif	
Turn-in directory : *ex*08/	
Files to turn in : `*`	
Allowed packages : `fmt`	
Allowed builtin functions : `None`	

Write a function ListForEachIf that applies a function given as argument to the data within some of the nodes of the list l.

- This function receives two functions:
  - f is a function that is applied to the node.
  - cond is a function that returns a boolean and it will be used to determine if the function f should be applied to the node.
- The function given as argument must have a pointer *NodeL as argument.

- Expected function and structure

```go
type NodeL struct {
        Data interface{}
        Next *NodeL
}

type List struct {
        Head *NodeL
        Tail *NodeL
}

func IsPositiveNode(node *NodeL) bool {
        switch node.Data.(type) {
        case int, float32, float64, byte:
                return node.Data.(int) > 0
        default:
                return false
        }
}

func IsAlNode(node *NodeL) bool {
        switch node.Data.(type) {
        case int, float32, float64, byte:
                return false
        default:
                return true
        }
}


func ListForEachIf(l *List, f func(*NodeL), cond func(*NodeL) bool) {

}
```

- Usage

```go
package main

import (
        "piscine"
        "fmt"
)

func PrintElem(node *piscine.NodeL) {
        fmt.Println(node.Data)
}

func StringToInt(node *piscine.NodeL) {
        node.Data = 2
}

func PrintList(l *piscine.List) {
        it := l.Head
        for it != nil {
                fmt.Print(it.Data, "->")
                it = it.Next
        }
        fmt.Print("nil","\n")
}

func main() {
        link := &piscine.List{}

        piscine.ListPushBack(link, 1)
        piscine.ListPushBack(link, "hello")
        piscine.ListPushBack(link, 3)
        piscine.ListPushBack(link, "there")
        piscine.ListPushBack(link, 23)
        piscine.ListPushBack(link, "!")
        piscine.ListPushBack(link, 54)

        PrintList(link)

        fmt.Println("--------function applied--------")
        piscine.ListForEachIf(link, PrintElem, piscine.IsPositiveNode)

        piscine.ListForEachIf(link, StringToInt, piscine.IsAlNode)

        fmt.Println("--------function applied--------")
        PrintList(link)

        fmt.Println()
}
```

- Output of usage

```
$ go mod init ex08
$ go run .
1->hello->3->there->23->!->54->nil
--------function applied--------
1
3
23
54
--------function applied--------
1->2->3->2->23->2->54->nil

$
```

Chapter XI

Exercise 09 : listfind

Exercise 09
listfind
Turn-in directory : *ex*09/
Files to turn in : `*`
Allowed packages : `fmt`
Allowed builtin functions : `None`

Write a function ListFind that returns the address of the first node in the list l that is determined to be equal to ref by the function CompStr.

- For this exercise the function CompStr must be used.
- Expected function and structure

```
type NodeL struct {
        Data interface{}
        Next *NodeL
}

type List struct {
        Head *NodeL
        Tail *NodeL
}

func CompStr(a, b interface{}) bool {
        return a == b
}

func ListFind(l *List, ref interface{}, comp func(a, b interface{}) bool) *interface{} {

}
```

- Usage

```go
package main

import (
        "fmt"
        "piscine"
)

func main() {
        link := &piscine.List{}

        piscine.ListPushBack(link, "hello")
        piscine.ListPushBack(link, "hello1")
        piscine.ListPushBack(link, "hello2")
        piscine.ListPushBack(link, "hello3")

        found := piscine.ListFind(link, interface{}("hello2"), piscine.CompStr)

        fmt.Println(found)
        fmt.Println(*found)
}
```

- Output of usage

```
$ go mod init ex09
$ go run .
0xc42000a0a0
hello2
$
```

Chapter XII

Exercise 10 : listremoveif

	Exercise 10
listremoveif	
Turn-in directory : *ex*10/	
Files to turn in : `*`	
Allowed packages : `fmt`	
Allowed builtin functions : `None`	

Write a function ListRemoveIf that removes all elements that are equal to the data_ref in the argument of the function.

- Expected function and structure

```
type NodeL struct {
        Data interface{}
        Next *NodeL
}

type List struct {
        Head *NodeL
        Tail *NodeL
}

func ListRemoveIf(l *List, data_ref interface{}) {

}
```

- Usage

```go
package main

import (
        "fmt"
        "piscine"
)

func PrintList(l *piscine.List) {
        it := l.Head
        for it != nil {
                fmt.Print(it.Data, " -> ")
                it = it.Next
        }

        fmt.Print(nil, "\n")
}

func main() {
        link := &piscine.List{}
        link2 := &piscine.List{}

        fmt.Println("----normal state----")
        piscine.ListPushBack(link2, 1)
        PrintList(link2)
        piscine.ListRemoveIf(link2, 1)
        fmt.Println("------answer-----")
        PrintList(link2)
        fmt.Println()

        fmt.Println("----normal state----")
        piscine.ListPushBack(link, 1)
        piscine.ListPushBack(link, "Hello")
        piscine.ListPushBack(link, 1)
        piscine.ListPushBack(link, "There")
        piscine.ListPushBack(link, 1)
        piscine.ListPushBack(link, 1)
        piscine.ListPushBack(link, "How")
        piscine.ListPushBack(link, 1)
        piscine.ListPushBack(link, "are")
        piscine.ListPushBack(link, "you")
        piscine.ListPushBack(link, 1)
        PrintList(link)

        piscine.ListRemoveIf(link, 1)
        fmt.Println("------answer-----")
        PrintList(link)
}
```

- Output of usage

```
$ go mod init ex10
$ go run .
----normal state----
1 -> <nil>
------answer-----
<nil>

----normal state----
1 -> Hello -> 1 -> There -> 1 -> 1 -> How -> 1 -> are -> you -> 1 -> <nil>
------answer-----
Hello -> There -> How -> are -> you -> <nil>
$
```

Chapter XIII

Exercise 11 : listmerge

	Exercise 11
listmerge	
Turn-in directory : *ex*11/	
Files to turn in : `*`	
Allowed packages : `fmt`	
Allowed builtin functions : `None`	

Write a function ListMerge that places elements of a list l2 at the end of another list l1.

- New elements should not be created!
- Expected function and structure

```
type NodeL struct {
        Data interface{}
        Next *NodeL
}

type List struct {
        Head *NodeL
        Tail *NodeL
}

func ListMerge(l1 *List, l2 *List) {

}
```

- Usage

```go
package main

import (
        "fmt"
        "piscine"
)

func PrintList(l *piscine.List) {
        it := l.Head
        for it != nil {
                fmt.Print(it.Data, " -> ")
                it = it.Next
        }
        fmt.Print(nil, "\n")
}

func main() {
        link := &piscine.List{}
        link2 := &piscine.List{}

        piscine.ListPushBack(link, "a")
        piscine.ListPushBack(link, "b")
        piscine.ListPushBack(link, "c")
        piscine.ListPushBack(link, "d")
        fmt.Println("-----first List------")
        PrintList(link)

        piscine.ListPushBack(link2, "e")
        piscine.ListPushBack(link2, "f")
        piscine.ListPushBack(link2, "g")
        piscine.ListPushBack(link2, "h")
        fmt.Println("-----second List------")
        PrintList(link2)

        fmt.Println("-----Merged List-----")
        piscine.ListMerge(link, link2)
        PrintList(link)
}
```

- Output of usage

```
$ go mod init ex11
$ go run .
-----first List------
a -> b -> c -> d -> <nil>
-----second List------
e -> f -> g -> h -> <nil>
-----Merged List-----
a -> b -> c -> d -> e -> f -> g -> h -> <nil>
$
```

Chapter XIV

Exercise 12 : listsort

	Exercise 12
listsort	
Turn-in directory : *ex*12/	
Files to turn in : `*`	
Allowed packages : `fmt`	
Allowed builtin functions : `None`	

Write a function ListSort that sorts the nodes of a linked list by ascending order.

- The NodeI structure will be the only one used.
- Expected function and structure

```go
type NodeI struct {
        Data int
        Next *NodeI
}

func ListSort(l *NodeI) *NodeI {

}
```

- Usage

```
package main

import (
        "fmt"
        "piscine"
)

func PrintList(l *piscine.NodeI) {
        it := l
        for it != nil {
                fmt.Print(it.Data, " -> ")
                it = it.Next
        }
        fmt.Print(nil, "\n")
}

func listPushBack(l *piscine.NodeI, data int) *piscine.NodeI {
        n := &piscine.NodeI{Data: data}

        if l == nil {
                return n
        }
        iterator := l
        for iterator.Next != nil {
                iterator = iterator.Next
        }
        iterator.Next = n
        return l
}

func main() {
        var link *piscine.NodeI

        link = listPushBack(link, 5)
        link = listPushBack(link, 4)
        link = listPushBack(link, 3)
        link = listPushBack(link, 2)
        link = listPushBack(link, 1)

        PrintList(piscine.ListSort(link))
}
```

- Output of usage

```
$ go mod init ex12
$ go run .
1 -> 2 -> 3 -> 4 -> 5 -> <nil>
$
```

Chapter XV

Exercise 13 : sortlistinsert

Exercise 13
sortlistinsert
Turn-in directory : *ex*13/
Files to turn in : `*`
Allowed packages : `fmt`
Allowed builtin functions : `None`

Write a function SortListInsert that inserts data_ref in the linked list l while keeping the list sorted in ascending order.

- During the tests the list passed as an argument will be already sorted.
- Expected function

```
func SortListInsert(l *NodeI, data_ref int) *NodeI{

}
```

- Usage

```
package main

import (
        "fmt"
        "piscine"
)

func PrintList(l *piscine.NodeI) {
        it := l
        for it != nil {
                fmt.Print(it.Data, " -> ")
                it = it.Next
        }
        fmt.Print(nil, "\n")
}

func listPushBack(l *piscine.NodeI, data int) *piscine.NodeI {
        n := &piscine.NodeI{Data: data}

        if l == nil {
                return n
        }
        iterator := l
        for iterator.Next != nil {
                iterator = iterator.Next
        }
        iterator.Next = n
        return l
}

func main() {

        var link *piscine.NodeI

        link = listPushBack(link, 1)
        link = listPushBack(link, 4)
        link = listPushBack(link, 9)

        PrintList(link)

        link = piscine.SortListInsert(link, -2)
        link = piscine.SortListInsert(link, 2)
        PrintList(link)
}
```

- Output of usage

```
$ go mod init ex13
$ go run .
1 -> 4 -> 9 -> <nil>
-2 -> 1 -> 2 -> 4 -> 9 -> <nil>
$
```

Chapter XVI

Exercise 14 : sortedlistmerge

	Exercise 14
sortedlistmerge	
Turn-in directory : *ex*14/	
Files to turn in : `*`	
Allowed packages : `fmt`	
Allowed builtin functions : `None`	

Write a function SortedListMerge that merges two lists n1 and n2 in ascending order.

- During the tests n1 and n2 will already be initially sorted.
- Expected function

```
func SortedListMerge(n1 *NodeI, n2 *NodeI) *NodeI {

}
```

- Usage

```
package main

import (
        "fmt"
        "piscine"
)

func PrintList(l *piscine.NodeI) {
        it := l
        for it != nil {
                fmt.Print(it.Data, " -> ")
                it = it.Next
        }
        fmt.Print(nil, "\n")
}

func listPushBack(l *piscine.NodeI, data int) *piscine.NodeI {
        n := &piscine.NodeI{Data: data}

        if l == nil {
                return n
        }
        iterator := l
        for iterator.Next != nil {
                iterator = iterator.Next
        }
        iterator.Next = n
        return l
}

func main() {
        var link *piscine.NodeI
        var link2 *piscine.NodeI

        link = listPushBack(link, 3)
        link = listPushBack(link, 5)
        link = listPushBack(link, 7)

        link2 = listPushBack(link2, -2)
        link2 = listPushBack(link2, 9)

        PrintList(piscine.SortedListMerge(link2, link))
}
```

- Output of usage

```
$ go mod init ex14
$ go run .
-2 -> 3 -> 5 -> 7 -> 9 -> <nil>
$
```