

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

Contents

1	Instructions	2
II	Exercise 00 : listpushback	3
III	Exercise 01: listpushfront	5
IV	Exercise 02 : listsize	7
\mathbf{V}	Exercise 03 : listlast	9
\mathbf{VI}	Exercise 04 : listclear	11
VII	Exercice 05: listat	13
VIII	Exercise 06 : listreverse	15
IX	Exercise 07: listforeach	17
\mathbf{X}	Exercise 08: listforeachif	19
XI	Exercise 09: listfind	22
XII	Exercise 10 : listlast	24
XIII	Exercise 11: listremoveif	26
XIV	Exercise 12 : listmerge	28
XV	Exercise 13: listsort	30
XVI	Exercise 14 : sortlistinsert	32
XVII	Exercise 15 : sortedlistmerge	34

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 <u>cannot</u> leave <u>any</u> 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.

Chapter II

Exercise 00: listpushback

	Exercise 00	
/	listpushback	/
Turn-in directory: es	x00/	
Files to turn in: *		
Allowed packages: github.com/42tokyo/ft		
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
}

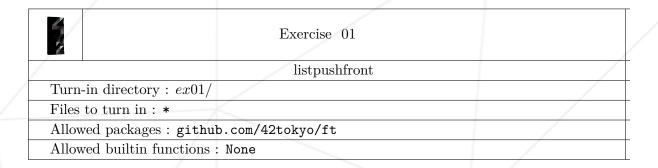
func ListPushBack(1 *List, data interface{}) {
}
```

- Usage
- Output of usage

```
$ go run .
Hello
there
how are you
$
```

Chapter III

Exercise 01: listpushfront



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(1 *List, data interface{}) {
}
```

- Usage
- Output of usage

```
$ go run .
how are you there Hello
$
```

```
package main
import (
    "fmt"

    "piscine"
)

func main() {

    link := &piscine.List{}

    piscine.ListPushFront(link, "Hello")

    piscine.ListPushFront(link, "there")

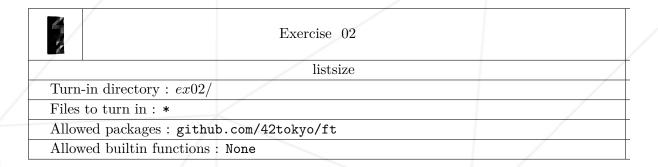
    piscine.ListPushFront(link, "how are you")

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

    fmt.Println()
}
```

Chapter IV

Exercise 02: listsize



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
- Output of usage

```
$ go run .
4
$
```

```
package main
import (
    "fmt"
    "piscine"
)

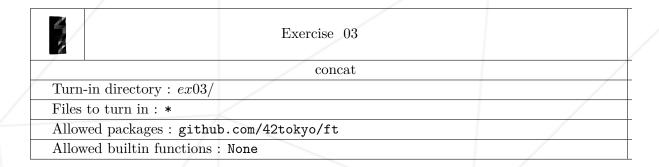
func main() {
    link := &piscine.List{}

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

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

Chapter V

Exercise 03: listlast



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(1 *List) interface{} {
}
```

- Usage
- Output of usage

```
$ go run .
1
<nil>
$
```

```
package main
import (
    "fmt"

    "piscine"
)

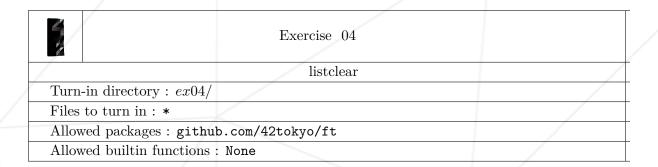
func main() {
    link := &piscine.List{}
    link2 := &piscine.List{}

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

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

Chapter VI

Exercise 04: listclear



Write a function ListClear that deletes all nodes from a linked list 1.

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

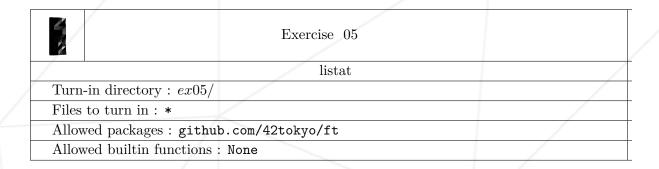
```
func ListClear(1 *List) {
}
```

- Usage
- Output of usage

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

Chapter VII

Exercice 05: listat



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(1 *NodeL, pos int) *NodeL{
}
```

- Usage
- Output of usage

```
$ go run .
1
how are
<nil>
$
```

```
package main
import (
    "fmt"
    "piscine"
)

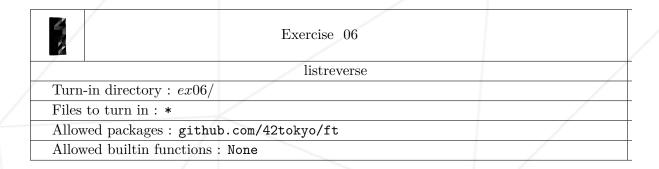
func main() {
    link := &piscine.List{}

    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))
}
```

Chapter VIII

Exercise 06: listreverse



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(1 *List) {
}
```

- Usage
- Output of usage

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

```
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)
}
```

Chapter IX

Exercise 07: listforeach

/

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: 1 *List
- Copy the functions Add2_node and Subtract3_node in the same file as the function ListForEach is defined.
- Expected function and structure
- Usage
- Output of usage

```
$ go run .
12
22
32
52
$
```

```
type NodeL struct {
        Data interface{}
       Next *NodeL
type List struct {
       Head *NodeL
       Tail *NodeL
func ListForEach(1 *List, f func(*NodeL)) {
func Add2_node(node *NodeL) {
       switch node.Data.(type) {
               node.Data = node.Data.(int) + 2
        case string:
               node.Data = node.Data.(string) + "2"
func Subtract3_node(node *NodeL) {
       switch node.Data.(type) {
               node.Data = node.Data.(int) - 3
       case string:
               node.Data = node.Data.(string) + "-3"
```

```
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.ListPushBack(link, "5")

    piscine.ListForEach(link, piscine.Add2_node)

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

Chapter X

Exercise 08: listforeachif

Exercise 08	
listforeachif	
Turn-in directory : $ex08/$	
Files to turn in: *	
Allowed packages: github.com/42tokyo/ft	
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
- Usage
- Output of usage

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

```
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
    }

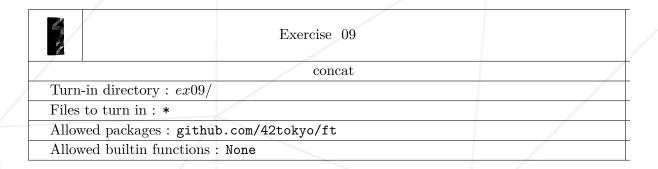
func IsAlNode(node *NodeL) bool {
    switch node.Data.(type) {
        case int, float32, float64, byte:
            return false
            default:
            return false
            default:
            return false
            default:
            return true
    }
}

func ListForEachIf(1 *List, f func(*NodeL), cond func(*NodeL) bool) {
}
```

```
package main
           "piscine"
"fmt"
func PrintElem(node *piscine.NodeL) {
    fmt.Println(node.Data)
func StringToInt(node *piscine.NodeL) {
           node.Data = 2
func PrintList(1 *piscine.List) {
    it := 1.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()
```

Chapter XI

Exercise 09: listfind



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
- Output of usage

```
$ go run .
0xc42000a0a0
hello2
$
```

```
package main
import (
    "fmt"
    "piscine"
)

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

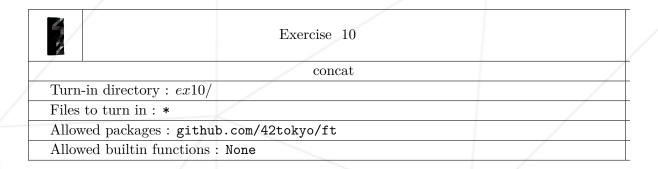
    piscine.ListPushBack(link, "hello")
    piscine.ListPushBack(link, "hello1")
    piscine.ListPushBack(link, "hello2")
    piscine.ListPushBack(link, "hello3")

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

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

Chapter XII

Exercise 10: listlast



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(1 *List) interface{} {
}
```

- Usage
- Output of usage

```
$ go run .
1
<nil>
$
```

• Note: the address may be different in each execution of the program.

```
package main
import (
    "fmt"

    "piscine"
)

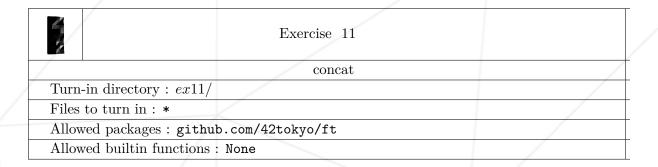
func main() {
        link := &piscine.List{}
        link2 := &piscine.List{}

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

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

Chapter XIII

Exercise 11: listremoveif



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

 \bullet Expected function and structure

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

type List struct {
          Head *NodeL
          Tail *NodeL
}

func ListRemoveIf(1 *List, data_ref interface{}) {
}
```

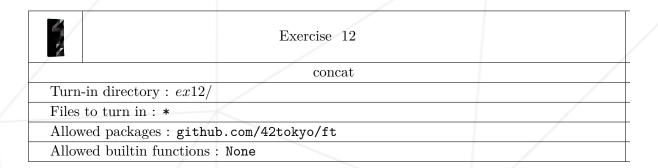
- Usage
- Output of usage

```
$ go run .
----normal state----
1 -> <ni1>
-----answer-----
<ni1>
----normal state----
1 -> Hello -> 1 -> There -> 1 -> 1 -> How -> 1 -> are -> you -> 1 -> <ni1>
-----answer-----
Hello -> There -> How -> are -> you -> <ni1>
$
```

```
package main
func PrintList(l *piscine.List) {
            it := 1.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, "How")
piscine.ListPushBack(link, "are")
piscine.ListPushBack(link, "you")
piscine.ListPushBack(link, 1)
            PrintList(link)
            piscine.ListRemoveIf(link, 1)
            fmt.Println("----answer-
            PrintList(link)
```

Chapter XIV

Exercise 12: listmerge



Write a function ListMerge that places elements of a list 12 at the end of another list 11.

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

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

type List struct {
          Head *NodeL
          Tail *NodeL
}

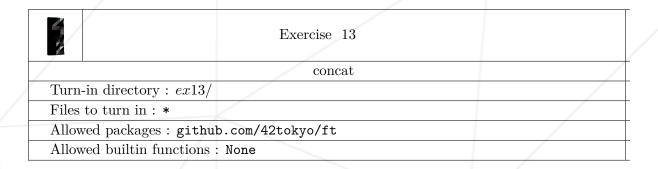
func ListMerge(l1 *List, 12 *List) {
}
```

- Usage
- Output of usage

```
$ 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 XV

Exercise 13: listsort



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

```
type NodeI struct {
    Data int
    Next *NodeI
}
func ListSort(1 *NodeI) *NodeI {
}
```

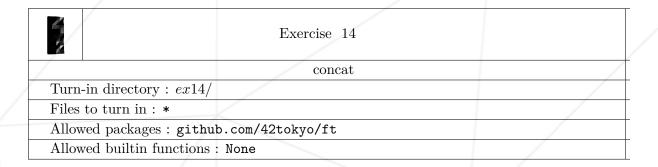
- Usage
- Output of usage

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

```
package main
func PrintList(l *piscine.NodeI) {
         for it != nil {
                 fmt.Print(it.Data, " -> ")
it = it.Next
         fmt.Print(nil, "\n")
func listPushBack(1 *piscine.NodeI, data int) *piscine.NodeI {
        n := &piscine.NodeI{Data: data}
         if 1 == nil {
                  return n
         for iterator.Next != nil {
                 iterator = iterator.Next
         iterator.Next = n
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))
```

Chapter XVI

Exercise 14: sortlistinsert



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(1 *NodeI, data_ref int) *NodeI{
}
```

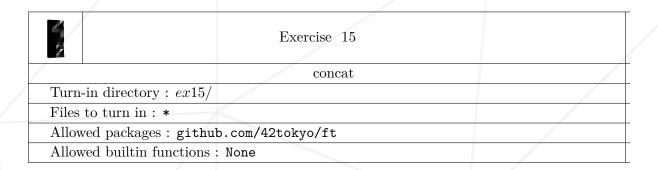
- Usage
- Output of usage

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

```
package main
func PrintList(l *piscine.NodeI) {
          for it != nil {
                    fmt.Print(it.Data, " -> ")
it = it.Next
          fmt.Print(nil, "\n")
\label{eq:func_listPushBack(l *piscine.NodeI, data int) *piscine.NodeI { } \\ n := \&piscine.NodeI \{ \mbox{Data: data} \}
          if 1 == nil {
                    return n
          for iterator.Next != nil {
                    iterator = iterator.Next
          iterator.Next = n
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)
```

Chapter XVII

Exercise 15: sortedlistmerge



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
- Output of usage

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

```
package main
func PrintList(l *piscine.NodeI) {
         for it != nil {
                   fmt.Print(it.Data, " -> ")
it = it.Next
         fmt.Print(nil, "\n")
func listPushBack(1 *piscine.NodeI, data int) *piscine.NodeI {
         n := &piscine.NodeI{Data: data}
         if 1 == nil {
         iterator := 1
for iterator.Next != nil {
                   iterator = iterator.Next
         iterator.Next = n
         return 1
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))
```