**QUESTIONS UNDER LEVEL ONE**

**Write a program that displays the alphabet, with even letters in uppercase, and odd letters in lowercase, followed by a newline ('\n').**

Usage

$ go run. | cat -e

ABcDeFgHiJkLmNoPqRsTuVwXyZ$

$

**Write a program that displays its first argument, if there is one.**

***Usage***

$ go run. hello there

hello

$ go run . "hello there" how are you

hello there

$ go run .

$

**Write a program that displays its last argument, if there is one.**

***Usage***

$ go run . hello there

there

$ go run . "hello there" how are you

you

$ go run . "hello there"

hello there

$ go run .

$

**Write a program that displays "Hello World!" followed by a newline ('\n').**

Usage

$ go run .

Hello World!

$

**Write a program that displays the number of arguments passed to it. This number will be followed by a newline ('\n').**

If there is no argument, the program displays 0 followed by a newline.

Usage

$ go run . 1 2 3 5 7 24

6

$ go run . 6 12 24 | cat -e

3$

$ go run . | cat -e

0$

$

**Write a program that prints the decimal digits in ascending order (from 0 to 9) on a single line.A line is a sequence of characters preceding the end of line character ('\n').**

Usage

$ go run .

0123456789

$

Write a program that displays all digits in descending order, followed by a newline ('\n').

Usage

$ go run .

9876543210

$

**QUESTION FOR LEVEL 2**

**Write a program that takes two string and checks whether it is possible to write the first string with characters from the second string.This rewrite must respect the order in which these characters appear in the second string.If it is possible, the program displays the string followed by a newline ('\n'), otherwise it simply displays nothing.If the number of arguments is different from 2, the program displays nothing.**

Usage

$ go run . 123 123

123

$ go run . faya fgvvfdxcacpolhyghbreda

faya

$ go run . faya fgvvfdxcacpolhyghbred

$ go run . error rrerrrfiiljdfxjyuifrrvcoojh

$ go run . "quarante deux" "qfqfsudf arzgsayns tsregfdgs sjytdekuoixq "

quarante deux

$ go run .

$

**QUESTION FOR LEVEL 3**

**Write a function that returns the first rune of a string.**

func main() {

z01.PrintRune(piscine.FirstRune("Hello!"))

z01.PrintRune(piscine.FirstRune("Salut!"))

z01.PrintRune(piscine.FirstRune("Ola!"))

z01.PrintRune('\n')

}

$ go run .

HSO

$

**Write a program that takes a string and displays its last word, followed by a newline ('\n'). A word is a section of string delimited by spaces or by the start/end of the string. The output will be followed by a newline ('\n').If the number of arguments is different from 1, or if there are no word, the program displays nothing.**

Usage

$ go run . "FOR PONY" | cat -e

PONY$

$ go run . "this ... is sparta, then again, maybe not" | cat -e

not$

$ go run . " "

$ go run . "a" "b"

$ go run . " lorem,ipsum " | cat -e

lorem,ipsum$

$

**The function should have as parameters a slice of integers a []int and a function f func(int, int) int. For each element of the slice, it should apply the function f func(int, int) int, save the result and then print it.**

**Expected function**

func ReduceInt(a []int, f func(int, int) int) {

}

Usage

**Here is a possible program to test your function :**

package main

func main() {

mul := func(acc int, cur int) int {

return acc \* cur

}

sum := func(acc int, cur int) int {

return acc + cur

}

div := func(acc int, cur int) int {

return acc / cur

}

as := []int{500, 2}

ReduceInt(as, mul)

ReduceInt(as, sum)

ReduceInt(as, div)

}

And its output :

$ go run .

1000

502

250

$

**QUESTIONS FOR LEVEL 4**

**Write a program called alphamirror that takes a string as argument and displays this string after replacing each alphabetical character with the opposite alphabetical character.The case of the letter remains unchanged, for example :'a' becomes 'z', 'Z' becomes 'A' 'd' becomes 'w', 'M' becomes 'N' The final result will be followed by a newline ('\n').If the number of arguments is different from 1, the program prints a new line.**

Usage

$ go run . "abc"

zyx$

$

$ go run . "My horse is Amazing." | cat -e

Nb slihv rh Znzarmt.$

$

$ go run .

$

**Write a function called Chunk that receives as parameters a slice, slice []int, and a number size int. The goal of this function is to chunk a slice into many sub slices where each sub slice has the length of size.**

If the size is 0 it should print a newline ('\n').

Expected function

func Chunk(slice []int, size int) {

}

Usage

Here is a possible program to test your function :

package main

func main() {

Chunk([]int{}, 10)

Chunk([]int{0, 1, 2, 3, 4, 5, 6, 7}, 0)

Chunk([]int{0, 1, 2, 3, 4, 5, 6, 7}, 3)

Chunk([]int{0, 1, 2, 3, 4, 5, 6, 7}, 5)

Chunk([]int{0, 1, 2, 3, 4, 5, 6, 7}, 4)

}

And its output :

$ go run .

[]

[[0 1 2] [3 4 5] [6 7]]

[[0 1 2 3 4] [5 6 7]]

[[0 1 2 3] [4 5 6 7]]

$

**Write a function that behaves like the Compare function.**

**Expected function**

func Compare(a, b string) int {

}

Usage

Here is a possible program to test your function :

package main

import (

"fmt"

"piscine"

)

func main() {

fmt.Println(piscine.Compare("Hello!", "Hello!"))

fmt.Println(piscine.Compare("Salut!", "lut!"))

fmt.Println(piscine.Compare("Ola!", "Ol"))

}

And its output :

$ go run .

0

-1

1

$

Instructions

**Write a program that is called doop.The program has to be used with three arguments: A value, An operator, one of : +, -, /, \*, Another value ,In case of an invalid operator, value, number of arguments or an overflow, the programs prints nothing.The program has to handle the modulo and division operations by 0 as shown on the output examples below.**

Usage

$ go run .

$ go run . 1 + 1 | cat -e

2

$

$ go run . hello + 1

$ go run . 1 p 1

$ go run . 1 / 0 | cat -e

No division by 0

$

$ go run . 1 % 0 | cat -e

No modulo by 0

$

$ go run . 9223372036854775807 + 1

$ go run . -9223372036854775809 - 3

$ go run . 9223372036854775807 "\*" 3

$ go run . 1 "\*" 1

1

$ go run . 1 "\*" -1

-1

$

**Write a function that returns the first prime number that is equal or inferior to the int passed as parameter.If there are no prime .inferior to the int passed as parameter the function should return 0.**

**Expected function**

func FindPrevPrime(nb int) int {

}

Usage

Here is a possible program to test your function :

package main

import "fmt"

func main() {

fmt.Println(FindPrevPrime(5))

fmt.Println(FindPrevPrime(4))

}

And its output :

$ go run .

5

3

$

**The function should have as parameters a function, f func(int, int) int a slice of integers, slice []int and an int acc int. For each element of the slice, it should apply the arithmetic function, save the result and print it. The function will be tested with our own functions Add, Sub, and Mul.**

**Expected function**

func FoldInt(f func(int, int) int, a []int, n int) {

}

Usage

Here is a possible program to test your function:

package main

func main() {

table := []int{1, 2, 3}

ac := 93

FoldInt(Add, table, ac)

FoldInt(Mul, table, ac)

FoldInt(Sub, table, ac)

fmt.Println()

table = []int{0}

FoldInt(Add, table, ac)

FoldInt(Mul, table, ac)

FoldInt(Sub, table, ac)

}

And its output :

$ go run .

99

558

87

93

0

93

$

**Write a program that takes 3 arguments, the first argument is a string in which a letter (the 2nd argument) will be replaced by another one (the 3rd argument).If the number of arguments is different from 3, the program displays nothing. If the second argument is not contained in the first one (the string) then the program rewrites the string followed by a newline ('\n').**

Usage

$ go run . "hella there" "a" "o"

hello there

$ go run . "hallo thara" "a" "e"

hello there

$ go run . "abcd" "z" "l"

abcd

$ go run . "something" "a" "o" "b" "c"

$

**Write a program that displays a number's multiplication table. The parameter will always be a strictly positive number that fits in an int. Said number multiplied by 9 will also fit in an int.**

Usage

$ go run . 9

1 x 9 = 9

2 x 9 = 18

3 x 9 = 27

4 x 9 = 36

5 x 9 = 45

6 x 9 = 54

7 x 9 = 63

8 x 9 = 72

9 x 9 = 81

$ go run . 19

1 x 19 = 19

2 x 19 = 38

3 x 19 = 57

4 x 19 = 76

5 x 19 = 95

6 x 19 = 114

7 x 19 = 133

8 x 19 = 152

9 x 19 = 171

$ go run .

$

**QUESTION FOR LEVEL 5**

**Write a program that takes two string and displays, without doubles, the characters that appear in both string, in the order they appear in the first one.The display will be followed by a newline ('\n'). If the number of arguments is different from 2, the program displays nothing.**

Usage

$ go run . "padinton" "paqefwtdjetyiytjneytjoeyjnejeyj"

padinto

$ go run . ddf6vewg64f twthgdwthdwfteewhrtag6h4ffdhsd

df6ewg4

$

**Write a program that determines if a given number is a power of 2. A number n is a power of 2 if it meets the following condition: n = 2 m where m is another integer number.For example, 4 = 2 2 or 16 = 2 4 are power of 2 numbers while 24 is not.This program must print true if the given number is a power of 2, otherwise it has to print false. If there is more than one or no argument, the program should print nothing.When there is only one argument, it will always be a positive valid int.**

Usage :

$ go run . 2 | cat -e

true$

$ go run . 64

true

$ go run . 513

false

$ go run .

$ go run . 64 1024

$

**Write a program that transforms a string passed as argument in its Pig Latin version.The rules used by Pig Latin are as follows: If a word begins with a vowel, just add "ay" to the end. If it begins with a consonant, then we take all consonants before the first vowel and we put them on the end of the word and add "ay" at the end. Only the latin vowels will be considered as vowel(aeiou).If the word has no vowels, the program should print "No vowels".If the number of arguments is different from one, the program prints nothing.**

Usage

$ go run .

$ go run . pig | cat -e

igpay$

$ go run . Is | cat -e

Isay$

$ go run . crunch | cat -e

unchcray$

$ go run . crnch | cat -e

No vowels$

$ go run . something else | cat -e

$

**Write a program that takes a number as argument, and prints it in binary value without a newline at the end. If the the argument is not a number, the program should print 00000000.**

Usage :

$ go run . 1

00000001$

$ go run . 192

11000000$

$ go run . a

00000000$

$ go run . 1 1

$ go run .

$

**Write a function that takes a byte, reverses it bit by bit (as shown in the example) and returns the result.**

**Expected function**

func ReverseBits(oct byte) byte {

}

Example:

1 byte

0010 0110 || / 0110 0100

**Write a program called rn. The objective is to convert a number, given as an argument, into a roman number and print it with its roman number calculation.**

**The program should have a limit of 4000. In case of an invalid number, for example "hello" or 0 the program should print ERROR: cannot convert to roman digit.**

**Roman Numerals reminder:**

**I 1**

**V 5**

**X 10**

**L 50**

**C 100**

**D 500**

**M 1000**

**For example, the number 1732 would be denoted MDCCXXXII in Roman numerals. However, Roman numerals are not a purely additive number system. In particular, instead of using four symbols to represent a 4, 40, 9, 90, etc. (i.e., IIII, XXXX, VIIII, LXXXX, etc.), such numbers are instead denoted by preceding the symbol for 5, 50, 10, 100, etc., with a symbol indicating subtraction. For example, 4 is denoted IV, 9 as IX, 40 as XL, etc.**

**The following table gives the Roman numerals for the first few positive integers.**

**1 I 11 XI 21 XXI**

**2 II 12 XII 22 XXII**

**3 III 13 XIII 23 XXIII**

**4 IV 14 XIV 24 XXIV**

**5 V 15 XV 25 XXV**

**6 VI 16 XVI 26 XXVI**

**7 VII 17 XVII 27 XXVII**

**8 VIII 18 XVIII 28 XXVIII**

**9 IX 19 XIX 29 XXIX**

**10 X 20 XX 30 XXX**

Usage

$ go run . hello

ERROR: cannot convert to roman digit

$ go run . 123

C+X+X+I+I+I

CXXIII

$ go run . 999

(M-C)+(C-X)+(X-I)

CMXCIX

$ go run . 3999

M+M+M+(M-C)+(C-X)+(X-I)

MMMCMXCIX

$ go run . 4000

ERROR: cannot convert to roman digit

$

**Write a function that takes a byte, swaps its halfs (like the example) and returns the result.**

**Expected function**

func SwapBits(octet byte) byte {

}

Example:

1 byte

0100 | 0001

\ /

/ \

0001 | 0100

**Write a program that takes two string and displays, without doubles, the characters that appear in either one of the string.The display will be in the same order that the characters appear on the command line and will be followed by a newline ('\n').If the number of arguments is different from 2, then the program displays a newline ('\n').**

Usage

$ go run . zpadinton paqefwtdjetyiytjneytjoeyjnejeyj | cat -e

zpadintoqefwjy$

$

$ go run . ddf6vewg64f gtwthgdwthdwfteewhrtag6h4ffdhsd | cat -e

df6vewg4thras$

$

$ go run . rien "cette phrase ne cache rien" | cat -e

rienct phas$

$

$ go run . | cat -e

$

$ go run . rien | cat -e

$

**QUESTION FOR LEVEL 6**

**Write a program that takes two string representing two strictly positive integers that fit in an int.The program displays their greatest common divisor followed by a newline ('\n').If the number of arguments is different from 2, the program displays nothing.All arguments tested will be positive int values.**

Usage

$ go run . 42 10 | cat -e

2$

$ go run . 42 12

6

$ go run . 14 77

7

$ go run . 17 3

1

$ go run .

$ go run . 50 12 4

$

**Write a program that takes a positive (or zero) number expressed in base 10, and that displays it in base 16 (with lowercase letters) followed by a newline ('\n'). If the number of arguments is different from 1, the program displays nothing. Error cases have to be handled as shown in the example below.**

Usage

$ go run . 10

a

$ go run . 255

ff

$ go run . 5156454

4eae66

$ go run .

$ go run . "123 132 1" | cat -e

ERROR$

$

**Write a program called repeat\_alpha that takes a string and displays it repeating each alphabetical character as many times as its alphabetical index.The result must be followed by a newline ('\n'). 'a' becomes 'a', 'b' becomes 'bb', 'e' becomes 'eeeee', etc...If the number of arguments is different from 1, the program displays nothing.**

Usage

$ go run . abc | cat -e

abbccc

$ go run . Choumi. | cat -e

CCChhhhhhhhooooooooooooooouuuuuuuuuuuuuuuuuuuuummmmmmmmmmmmmiiiiiiiii.$

$ go run . "abacadaba 01!" | cat -e

abbacccaddddabba 01!$

$ go run .

$ go run . "" | cat -e

$

$

**Write a function SortWordArr that sorts by ascii (in ascending order) a string slice.**

**Expected function**

func SortWordArr(a []string) {

}

Usage

Here is a possible program to test your function :

package main

import (

"fmt"

"piscine"

)

func main() {

result := []string{"a", "A", "1", "b", "B", "2", "c", "C", "3"}

piscine.SortWordArr(result)

fmt.Println(result)

}

And its output :

$ go run .

[1 2 3 A B C a b c]

$

**QUESTION LEVEL 7**

**Write a program that takes a positive integer as argument and displays the sum of all prime numbers inferior or equal to it followed by a newline ('\n').**

**If the number of arguments is different from 1, or if the argument is not a positive number, the program displays 0 followed by a newline.**

Usage

$ go run . 5

10

$ go run . 7

17

$ go run . -2

0

$ go run . 0

0

$ go run .

0

$ go run . 5 7

0

$

**Write a program that takes a positive int and displays its prime factors, followed by a newline ('\n'). Factors must be displayed in ascending order and separated by \*.If the number of arguments is different from 1, if the argument is invalid, or if the integer does not have a prime factor, the program displays nothing.**

Usage

$ go run . 225225

3\*3\*5\*5\*7\*11\*13

$ go run . 8333325

3\*3\*5\*5\*7\*11\*13\*37

$ go run . 9539

9539

$ go run . 804577

804577

$ go run . 42

2\*3\*7

$ go run . a

$ go run . 0

$ go run . 1

$

**Write a program named hiddenp that takes two string and that, if the first string is hidden in the second one, displays 1 followed by a newline ('\n'), otherwise it displays 0 followed by a newline.Let s1 and s2 be string. It is considered that s1 is hidden in s2 if it is possible to find each character from s1 in s2, in the same order as they appear in s1.If s1 is an empty string, it is considered hidden in any string.If the number of arguments is different from 2, the program displays nothing.**

Usage

$ go run . "fgex.;" "tyf34gdgf;'ektufjhgdgex.;.;rtjynur6" | cat -e

1$

$ go run . "abc" "2altrb53c.sse" | cat -e

1$

$ go run . "abc" "btarc" | cat -e

0$

$ go run . "DD" "DABC" | cat -e

0$

$ go run .

$

**Write a program that takes a string as a parameter, and prints its words in reverse, followed by a newline. A word is a sequence of alphanumerical characters. If the number of arguments is different from 1, the program will display nothing.In the parameters that are going to be tested, there will not be any extra spaces. (meaning that there will not be additional spaces at the beginning or at the end of the string and that words will always be separated by exactly one space).**

Usage

$ go run . "the time of contempt precedes that of indifference"

indifference of that precedes contempt of time the

$ go run . "abcdefghijklm"

abcdefghijklm

$ go run . "he stared at the mountain"

mountain the at stared he

$ go run . "" | cat-e

$

$

**Write a program that takes a string and displays this string after rotating it one word to the left.Thus, the first word becomes the last, and others stay in the same order.A word is a sequence of alphanumerical characters.Words will be separated by only one space in the output.If the number of arguments is different from 1, the program displays a newline**.

Usage

$ go run . "abc " | cat -e

abc$

$ go run . "Let there be light"

there be light Let

$ go run . " AkjhZ zLKIJz , 23y"

zLKIJz , 23y AkjhZ

$ go run . | cat -e

$

$

**Write a function that receives a string and a separator and returns a slice of strings that results of splitting the string s by the separator sep.**

**Expected function**

func Split(s, sep string) []string {

}

Usage

Here is a possible program to test your function :

package main

import (

"fmt"

"piscine"

)

func main() {

s := "HelloHAhowHAareHAyou?"

fmt.Printf("%#v\n", piscine.Split(s, "HA"))

}

And its output :

$ go run .

[]string{"Hello", "how", "are", "you?"}

$

**Write a function that takes two arguments: s: a numeric string in a given base.base: a string representing all the different digits that can represent a numeric value.And return the integer value of s in the given base.If the base is not valid it returns 0.Validity rules for a base :A base must contain at least 2 characters.Each character of a base must be unique**.**A base should not contain + or - characters.String number must contain only elements that are in base.Only valid string numbers will be tested.The function does not have to manage negative numbers.**

**Expected function**

func AtoiBase(s string, base string) int {

}

Usage

Here is a possible program to test your function :

package main

import (

"fmt"

"piscine"

)

func main() {

fmt.Println(piscine.AtoiBase("125", "0123456789"))

fmt.Println(piscine.AtoiBase("1111101", "01"))

fmt.Println(piscine.AtoiBase("7D", "0123456789ABCDEF"))

fmt.Println(piscine.AtoiBase("uoi", "choumi"))

fmt.Println(piscine.AtoiBase("bbbbbab", "-ab"))

}

And its output :

$ go run .

125

125

125

125

0

$

**QUESTION LEVEL 8**

**Write a function that simulates the behavior of the Itoa function in Go. Itoa transforms a number represented as anint in a number represented as a string.For this exercise the handling of the signs + or - does have to be taken into account.**

**Expected function**

func Itoa(n int) string {

}

Usage

Here is a possible program to test your function :

package main

import (

"fmt"

"piscine"

)

func main() {

fmt.Println(piscine.Itoa(12345))

fmt.Println(piscine.Itoa(0))

fmt.Println(piscine.Itoa(-1234))

fmt.Println(piscine.Itoa(987654321))

}

And its output :

$ go run .

12345

0

-1234

987654321

$

**QUESTION LEVEL 9**

**Write a program that takes an undefined number of string in arguments. For each argument, if the expression is correctly bracketed, the program prints on the standard output OK followed by a newline ('\n'), otherwise it prints Error followed by a newline.Symbols considered as brackets are parentheses ( and ), square brackets [ and ] and curly braces { and }. Every other symbols are simply ignored.An opening bracket must always be closed by the good closing bracket in the correct order. A string which does not contain any bracket is considered as a correctly bracketed string.If there is no argument, the program must print nothing.**

Usage

$ go run . '(johndoe)' | cat -e

OK$

$ go run . '([)]' | cat -e

Error$

$ go run . '' '{[(0 + 0)(1 + 1)](3\*(-1)){()}}' | cat -e

OK$

OK$

$ go run .

$

**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

Here is a possible program to test your function :

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))

}

And its output :

$ go run .

4

$

**Write a program that takes an undefined number of arguments which could be considered as options and writes on the standard output a representation of those options as groups of bytes followed by a newline ('\n'). An option is an argument that begins with a - and that can have multiple characters which could be : -abcdefghijklmnopqrstuvwxyz All options are stocked in a single int and each options represents a bit of that int, and should be stocked like this : - 00000000 00000000 00000000 00000000, - \*\*\*\*\*\*zy xwvutsrq ponmlkji hgfedcba Launching the program without arguments or with the -h flag activated must print all the valid options on the standard output, as shown on one of the following examples. Please note the -h flag has priority over the others flags when it is called first in one of the arguments. (See the examples) A wrong option must print Invalid Option followed by a newline.**

Usage

$ go run . | cat -e

options: abcdefghijklmnopqrstuvwxyz$

$ go run . -abc -ijk | cat -e

00000000 00000000 00000111 00000111$

$ go run . -z | cat -e

00000010 00000000 00000000 00000000$

$ go run . -abc -hijk | cat -e

options: abcdefghijklmnopqrstuvwxyz$

$ go run . -h | cat -e

options: abcdefghijklmnopqrstuvwxyz$

$ go run . -zh | cat -e

00000010 00000000 00000000 10000000$

$ go run . -z -h | cat -e

options: abcdefghijklmnopqrstuvwxyz$

$ go run . -hhhhhh | cat -e

options: abcdefghijklmnopqrstuvwxyz$

$ go run . -eeeeee | cat -e

00000000 00000000 00000000 00010000$

$ go run . -% | cat -e

Invalid Option$

$ go run . - | cat -e

Invalid Option$

$

**Write a function that takes (arr [10]byte), and displays the memory as in the example.After displaying the memory the function must display all the ASCII graphic characters. The non printable characters must be replaced by a dot.The ASCII graphic characters are any characters intended to be written, printed, or otherwise displayed in a form that can be read by humans, present on the ASCII encoding.**

**Expected function**

func PrintMemory(arr [10]byte) {

}

Usage

Here is a possible program to test your function :

package main

import "piscine"

func main() {

piscine.PrintMemory([10]byte{'h', 'e', 'l', 'l', 'o', 16, 21, '\*'})

}

And its output :

$ go run . | cat -e

68 65 6c 6c$

6f 10 15 2a$

00 00$

hello..\*..$

$

**Write a program that takes a string which contains an equation written in Reverse Polish Notation (RPN) as its first argument, that evaluates the equation, and that prints the result on the standard output followed by a newline ('\n').Reverse Polish Notation is a mathematical notation in which every operator follows all of its operands. In RPN, every operator encountered evaluates the previous 2 operands, and the result of this operation then becomes the first of the two operands for the subsequent operator. Operands and operators must be spaced by at least one space.**

**The following operators must be implemented : +, -, \*, /, and %.**

**If the string is not valid or if there is not exactly one argument, Error must be printed on the standard output followed by a newline. If the string has extra spaces it is still considered valid.**

**All the given operands must fit in a int.**

**Examples of formulas converted in RPN:**

3 + 4 >> 3 4 + ((1 \_ 2) \_ 3) - 4 >> 1 2 \_ 3 \_ 4 - or 3 1 2 \* \_ 4 - 50 \_ (5 - (10 / 9)) >> 5 10 9 / - 50 \*

Here is how to evaluate a formula in RPN:

1 2 \* 3 \* 4 -

2 3 \* 4 -

6 4 -

2

Or:

3 1 2 \* \* 4 -

3 2 \* 4 -

6 4 -

2

Usage

$ go run . "1 2 \* 3 \* 4 +" | cat -e

10$

$ go run . "1 2 3 4 +" | cat -e

Error$

$ go run . | cat -e

Error$

$ go run . " 1 3 \* 2 -" | cat -e

1

$ go run . " 1 3 \* ksd 2 -" | cat -e

Error$

$

**Esoteric programming languages (esolang) are designed to test the boundaries of computer programming design, as proofs of concept, as software art, as a hacking interface or simply as a joke. One such esoteric language is Brainfuck. It was created by Urban MÃ¼ller in 1993. It is a minimalist language consisting of 8 simple commands. It is Turing complete, but is not intended for practical use. It exists to amuse and challenge programmers.**

**QUESTION LEVEL 10**

**Write a Brainfuck interpreter program.The source code will be given as the first parameter, and will always be valid with fewer than 4096 operations.Your Brainfuck interpreter will consist of an array of 2048 bytes, all initialized to 0, with a pointer to the first byte.Every operator consists of a single character:**

**>: increment the pointer.**

**<: decrement the pointer.**

**+: increment the pointed byte.**

**-: decrement the pointed byte.**

**.: print the pointed byte to the standard output.**

**[: if the pointed byte is 0, then instead of moving onto the next command, skip to the command after the matching ].**

**]: if the pointed byte is not 0, then instead of moving onto the next command, move back to the command after the matching [.**

**Any other character is a comment**.

Usage

$ go run . "++++++++++[>+++++++>++++++++++>+++>+<<<<-]>++.>+.+++++++..+++.>++.<<+++++++++++++++.>.+++.------.--------.>+.>." | cat -e

Hello World!$

$ go run . "+++++[>++++[>++++H>+++++i<<-]>>>++\n<<<<-]>>--------.>+++++.>." | cat -e

Hi$

$ go run . "++++++++++[>++++++++++>++++++++++>++++++++++<<<-]>---.>--.>-.>++++++++++." | cat -e

abc$

$ go run .

$

**Write a function that: converts an int value to a string using the specified base in the argument and that returns this stringThe base is expressed as an int, from 2 to 16. The characters comprising the base are the digits from 0 to 9, followed by uppercase letters from A to F.For example, the base 4 would be the equivalent of "0123" and the base 16 would be the equivalent of "0123456789ABCDEF".**

**If the value is negative, the resulting string has to be preceded by a minus sign -.Only valid inputs will be tested.**

Expected function

func ItoaBase(value, base int) string {

}

listremoveif

Instructions

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

Here is a possible program to test your function :

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)

}

And its output :

$ 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>

$