Begin 1

Тарафи квадрат a дода шудааст. Периметри он P = 4*a ёфта шавад.

Given the side *a* of a square, find the perimeter *P* of the square: $P = 4 \cdot a$.

```
package main

import "fmt"

func main() {
    var a float32
    fmt.Print("a = ")
    fmt.Scanf("%f", &a)
    p := a * 4
    fmt.Printf("P = %.2f", p)
}
```

Begin 2

Тарафи квадрат a дода шудааст. Масоҳати он $S = a^2$ ёфта шавад.

Given the side *a* of a square, find the area *S* of the square: $S = a^2$.

```
package main

import "fmt"

func main() {
    var a float32
    fmt.Print("a = ")
    fmt.Scanf("%f", &a)
    s := a*a
    fmt.Printf("S = %.2f", s)
}
```

Begin 3

Тарафҳои росткунча a ва b дода шудаанд. Масоҳати он S=a*b ва периметри он P=2*(a+b) ёфта шаванд.

The sides *a* and *b* of a rectangle are given. Find the area $S = a \cdot b$ and the perimeter $P = 2 \cdot (a + b)$ of the rectangle.

```
package main
```

```
import "fmt"

func main() {
    var a, b float32;
    fmt.Print("a = ")
    fmt.Scan(&a)
    fmt.Print("b = ")
    fmt.Scan(&b)
    s := a*b
    p := 2 * (a + b)
    fmt.Printf("S = %.2f\nP = %.2f\n", s, p)
}
```

Begin 4

Диаметри давра d дода шудааст. Дарозии он L=PI*d ёфта шавад. Ба сифати қимати PI 3.14 истифода бурда шавад.

Given the diameter *d* of a circle, find the length *L* of the circle: $L = \pi \cdot d$. Use 3.14 for a value of π.

```
package main

import "fmt"

func main() {
    const PI = 3.14
    var d float32
    fmt.Print("d = ")
    fmt.Scan(&d)
    l := PI * d
    fmt.Printf("L = %.2f\n", l)
}
```

Begin 5

Дарозии рӯяи куб a дода шудааст. Ҳаҷми куб $V=a^3$ ва масоҳати сатҳи болоии он $S=6*a^2$ ёфта шаванд.

Given the edge *a* of a cube, find the volume $V = a^3$ and the surface area $S = 6 \cdot a^2$ of the cube.

```
package main

import (
    "fmt"
    "math"
)

func main() {
    var a float64
```

```
    fmt.Print("a = ")
    fmt.Scanf("%f", &a)
    v := math.Pow(a, 3)
    s := 6 * a * a
    fmt.Printf("V = %.3f\nS = %.3f\n", v, s)
}
```

Begin 6

Дарозиҳои рӯяҳои параллелопипеди росткунча a, b, c дода шудаанд. Ҳаҷми он V=a*b*c ва масоҳати сатҳи болоии он S=2*(a*b+b*c+a*c) ёфта шаванд.

The edges *a*, *b*, *c* of a right parallelepiped are given. Find the volume $V = a \cdot b \cdot c$ and the surface area $S = 2 \cdot (a \cdot b + b \cdot c + a \cdot c)$ of the right parallelepiped.

```
package main

import "fmt"

func main() {
    var a, b, c float32
    fmt.Print("a = ")
    fmt.Scan(&a)
    fmt.Print("b = ")
    fmt.Scan(&b)
    fmt.Print("c = ")
    fmt.Scan(&c)
    v := a * b * c
    s := 2 * (a*b + b*c + a*c)
    fmt.Printf("V = %.3f\nS = %.3f\n", v, s)
}
```

Begin 7

Дарозии давра L ва масоҳати доира S ёфта шаванд, агар радиус R дода шуда бошад: L=2*PI*R, S=PI*R^2. Ба сифати қимати PI 3.14 истифода бурда шавад.

Given the radius *R* of a circle, find the length *L* of the circumference and the area *S* of the circle:
$L = 2 \cdot \pi \cdot R$, $S = \pi \cdot R^2$.Use 3.14 for a value of π.

```
package main

import "fmt"

func main() {
```

```
    const PI = 3.14
    var r float64
    fmt.Print("R = ")
    fmt.Scanf("%f", &r)
    l := 2 * PI * r
    s := PI * r * r
    fmt.Printf("L = %.3f\nS = %.3f\n", l, s)
}
```

Begin 8

Ду ададҳо a ва b дода шудаанд. Қимати миёнаи арифметикии онҳо: (a+b)/2 ёфта шавад.

Given two numbers *a* and *b*, find their *average*: (*a* + *b*)/2.

```
package main

import "fmt"

func main() {
    var a, b float64
    fmt.Print("a = ")
    fmt.Scan(&a)
    fmt.Print("b = ")
    fmt.Scan(&b)
    aMean := (a + b) / 2
    fmt.Printf("aMean = %.2f\n", aMean)
}
```

Begin 9

Ду ададҳои ғайриманфӣ a ва b дода шудаанд. Қимати миёнаи геометрии онҳо ёфта шавад, яъне решаи квадратӣ аз ҳосилизарби онҳо sqrt(a*b).

Given two nonnegative numbers *a* and *b*, find their *geometrical mean* (a square root of their product): $(a \cdot b)^{1/2}$.

```
package main

import (
    "fmt"
    "math"
)

func main() {
    var a, b float64
    fmt.Print("a [positive] = ")
    fmt.Scan(&a)
    fmt.Print("b [positive] = ")
    fmt.Scan(&b)
```

```
    gMean := math.Sqrt(a * b)
    fmt.Printf("gMean = %.2f\n", gMean)
}
```

Begin 10

Ду ададҳои ғайринулӣ дода шудаанд. Сумма, фарқ, ҳосилизарб ва ҳосилитақсими квадратҳои онҳо ёфта шаванд.

Two nonzero numbers are given. Find the sum, the difference, the product, and the quotient of their squares.

```
package main

import "fmt"

func main() {
    var a, b float64
    fmt.Print("A = ")
    fmt.Scan(&a)
    fmt.Print("B = ")
    fmt.Scan(&b)
    sqrA := a * a
    sqrB := b * b
    sum := sqrA + sqrB
    sub := sqrA - sqrB
    mul := sqrA * sqrB
    div := sqrA / sqrB
    fmt.Printf("sum = %.2f\n", sum)
    fmt.Printf("sub = %.2f\n", sub)
    fmt.Printf("mul = %.2f\n", mul)
    fmt.Printf("div = %.2f\n", div)
}
```

Begin 11

Ду ададҳои ғайринулӣ дода шудаанд. Сумма, фарқ, ҳосилизарб ва ҳосилитақсими қиматҳои мутлақи онҳо ёфта шаванд.

Two nonzero numbers are given. Find the sum, the difference, the product, and the quotient of their absolute values.

```
package main

import (
    "fmt"
    "math"
)
```

```
func main() {
    var a, b float64
    fmt.Print("A = ")
    fmt.Scan(&a)
    fmt.Print("B = ")
    fmt.Scan(&b)
    absA := math.Abs(a)
    absB := math.Abs(b)
    sum := absA + absB
    sub := absA - absB
    mul := absA * absB
    div := absA / absB
    fmt.Printf("sum = %.2f\n", sum)
    fmt.Printf("sub = %.2f\n", sub)
    fmt.Printf("mul = %.2f\n", mul)
    fmt.Printf("div = %.2f\n", div)
}
```

Begin 12

Катетҳои секунҷаи росткунҷа a ва b дода шудаанд. Гипотенуза c ва периметри он P ёфта шаванд: $c=sqrt(a^2+b^2)$, $P=a+b+c$.

The legs *a* and *b* of a right triangle are given. Find the hypotenuse *c* and the perimeter *P* of the triangle:
$c = (a^2 + b^2)^{1/2}$, $P = a + b + c$

```
package main

import (
    "fmt"
    "math"
)

func main() {
    var a, b, c, p float64
    fmt.Print("a = ")
    fmt.Scan(&a)
    fmt.Print("b = ")
    fmt.Scan(&b)
    c = math.Sqrt(a*a + b*b)
    p = a + b + c
    fmt.Printf("c = %.2f\n", c)
    fmt.Printf("P = %.2f\n", p)
}
```

Begin 13

Ду доираҳо бо маркази умумӣ ва радиусҳои R_1 ва R_2 ($R_1>R_2$) дода шудаанд. Масоҳатҳои ин доираҳо S_1 ва S_2, ҳамчунин масоҳати ҳалқае S_3, ки радиуси беруниаш баробари R_1 ва

радиуси дохилиаш баробари R_2 ҳастанд, ёфта шаванд: $S_1=PI*R_1^2$, $S_2=PI*R_2^2$, $S_3=S_1-S_2$.

Given the radiuses R_1 and R_2 of two concentric circles ($R_1 > R_2$), find the areas S_1 and S_2 of the circles and the area S_3 of the ring bounded by the circles:
$S_1 = \pi\cdot(R_1)^2$, $S_2 = \pi\cdot(R_2)^2$, $S_3 = S_1 - S_2$.Use 3.14 for a value of π.

```
package main

import "fmt"

func main() {
    const PI = 3.14
    var r1, r2, s1, s2, s3 float64
    fmt.Print("R1 = ")
    fmt.Scan(&r1)
    fmt.Print("R2 = ")
    fmt.Scan(&r2)
    s1 = PI * r1 * r1
    s2 = PI * r2  *r2
    s3 = s1 - s2
    fmt.Printf("S1 = %.2f\n", s1)
    fmt.Printf("S2 = %.2f\n", s2)
    fmt.Printf("S3 = %.2f\n", s3)
}
```

Begin 14

Дарозии давра L дода шудааст. Радиуси он R ва масоҳати доирае S ки, бо ин давра маҳдуд аст, бо назардошти он, ки $L=2*PI*R$, $S=PI*R^2$ аст, ёфта шаванд. Ба сифати қимати PI 3.14 истифода бурда шавад.

Given the length L of a circumference, find the radius R and the area S of the circle. Take into account that $L = 2\cdot\pi\cdot R$, $S = \pi\cdot R^2$. Use 3.14 for a value of π.

```
package main

import "fmt"

func main() {
    const PI = 3.14
    var l, r, s float64
    fmt.Print("L = ")
    fmt.Scanf("%f", &l)
    r = l / (2 * PI)
```

```
    s = PI * r * r
    fmt.Printf("R = %.2f\n", r)
    fmt.Printf("S = %.2f\n", s)
}
```

Begin 15

Масоҳати доира S дода шудааст. Диаметри он D ва дарозии даврае L, ки ин доираро маҳдуд мекунад, бо дарназардошти он, ки L=PI*D, S=PI*D^2/4 аст, ёфта шаванд. Ба сифати қимати PI 3.14 истифода бурда шавад.

Given the area *S* of a circle, find the diameter *D* and the length *L* of the circumference. Take into account that
$L = \pi \cdot D, \quad S = \pi \cdot D^2/4$. Use 3.14 for a value of π.

```
package main

import (
    "fmt"
    "math"
)

func main() {
    const PI = 3.14
    var s, d, l float64
    fmt.Print("S = ")
    fmt.Scanf("%f", &s)
    d = math.Sqrt(4 * s / PI)
    l = PI * d
    fmt.Printf("D = %.2f\n", d)
    fmt.Printf("L = %.2f\n", l)
}
```

Begin 16

Масофаи байни ду нуқта бо координатаҳои додашудаи x_1 ва x_2 дар тири ададӣ ёфта шавад: $|x_2 - x_1|$.

Two points with the coordinates x_1 and x_2 are given on the real axis. Find the distance between these points: $|x_2 - x_1|$.

```
package main

import (
    "fmt"
    "math"
)

func main() {
```

```
    var x1, x2 float64
    fmt.Print("x1 = ")
    fmt.Scan(&x1)
    fmt.Print("x2 = ")
    fmt.Scan(&x2)
    distance := math.Abs(x2 - x1)
    fmt.Printf("distance = %.2f\n", distance)
}
```

Begin 17

Се нуқтаҳо A, B, C дар тири ададӣ дода шудаанд. Дарозиҳои порчаҳои AC ва BC ва суммаи онҳо ёфта шаванд.

Three points *A*, *B*, *C* are given on the real axis. Find the length of *AC*, the length of *BC*, and the sum of these lengths.

```
package main

import (
    "fmt"
    "math"
)

func main() {
    var a, b, c float64
    fmt.Print("A = ")
    fmt.Scan(&a)
    fmt.Print("B = ")
    fmt.Scan(&b)
    fmt.Print("C = ")
    fmt.Scan(&c)
    ac := math.Abs(c - a)
    bc := math.Abs(c - b)
    sum := ac + bc
    fmt.Printf("AC = %.2f\n", ac)
    fmt.Printf("BC = %.2f\n", bc)
    fmt.Printf("AC + BC = %.2f\n", sum)
}
```

Begin 18

Се нуқтаҳо A, B, C дар тири ададӣ дода шудаанд. Нуқтаи C дар байни нуқтаҳои A ва B ҷойгир аст. Ҳосилизарби дарозиҳои порчаҳои AC ва BC ёфта шавад.

Three points *A*, *B*, *C* are given on the real axis, the point *C* is located between the points *A* and *B*. Find the product of the length of *AC* and the length of *BC*.

```
package main
```

```
import (
    "fmt"
    "math"
)

func main() {
    var a, b, c float64
    fmt.Print("A = ")
    fmt.Scan(&a)
    fmt.Print("B = ")
    fmt.Scan(&b)
    fmt.Print("C = ")
    fmt.Scan(&c)
    ac := math.Abs(c - a)
    bc := math.Abs(c - b)
    mul := ac * bc
    fmt.Printf("AC * BC = %.2f\n", mul)
}
```

Begin 19

Координатаҳои ду қуллаҳои муқобили росткунча дода шудаанд:(x_1,y_1), (x_2,y_2). Тарафҳои росткунча ба тирҳои координатӣ параллел мебошанд. Периметр ва масоҳати росткунчаи додашуда ёфта шаванд.

The coordinates (x_1, y_1) and (x_2, y_2) of two opposite vertices of a rectangle are given. Sides of the rectangle are parallel to coordinate axes. Find the perimeter and the area of the rectangle.

```
package main

import (
    "fmt"
    "math"
)

func main() {
    var x1, y1, x2, y2 float64
    fmt.Print("x1 = ")
    fmt.Scan(&x1)
    fmt.Print("y1 = ")
    fmt.Scan(&y1)
    fmt.Print("x2 = ")
    fmt.Scan(&x2)
    fmt.Print("y2 = ")
    fmt.Scan(&y2)
    a := math.Abs(x2 - x1)
    b := math.Abs(y2 - y1)
    p := 2 * (a + b)
    s := a * b
    fmt.Printf("P = %.2f\nS = %.2f\n", p, s)
}
```

Begin 20

Масофаи байни ду нуқтаҳо бо координатаҳои додашудаи (x_1,y_1) ва (x_2,y_2) дар ҳамворӣ ёфта шавад. Масофа аз рӯи формулаи зерин ҳисоб карда мешавад: sqrt($(x_2-x_1)^2+(y_2-y_1)^2$).

The coordinates (x_1, y_1) and (x_2, y_2) of two points are given. Find the distance between the points: $((x_2 - x_1)^2 + (y_2 - y_1)^2)^{1/2}$

```
package main

import (
    "fmt"
    "math"
)

func main() {
    var x1, y1, x2, y2 float64
    fmt.Print("x1 = ")
    fmt.Scan(&x1)
    fmt.Print("y1 = ")
    fmt.Scan(&y1)
    fmt.Print("x2 = ")
    fmt.Scan(&x2)
    fmt.Print("y2 = ")
    fmt.Scan(&y2)
    s := math.Sqrt(math.Pow(x2 - x1, 2) + math.Pow(y2 - y1, 2))
    fmt.Printf("S = %.2f\n", s)
}
```

Integer 1

Масофа L бо сантиметр дода шудааст. Амали тақсими бутунро истифода бурда, миқдори метрҳои пурраро дар он ёбед (1 метр = 100 см).

A distance *L* is given in centimeters. Find the amount of full meters of this distance (1 m = 1000 cm). Use the operator of integer division.

```
package main

import "fmt"

func main() {
    var l int
    fmt.Print("L = ")
    fmt.Scanf("%d", &l)
```

```
    meters := l / 100
    fmt.Printf("meters = %d\n", meters)
}
```

Integer 2

Вазн M бо килограм дода шудааст. Амали тақсими бутунро истифода бурда, миқдори тоннаҳои пурраро дар он ёбед (1 тонна = 1000 кг).

A weight *M* is given in kilograms. Find the amount of full tons of this weight (1 ton = 1000 kg). Use the operator of integer division.

```
package main

import "fmt"

func main() {
    var m int
    fmt.Print("M = ")
    fmt.Scanf("%d", &m)
    tons := m / 1000
    fmt.Printf("tons = %d\n", tons)
}
```

Integer 3

Андозаи файл бо байт дода шудааст. Амали тақсими бутунро истифода бурда, миқдори килобайтҳои пурраро, ки файли мазкур банд мекунад, ёбед (1 килобайт = 1024 байт).

A file size is given in bytes. Find the amount of full Kbytes of this size (1 K = 1024 bytes). Use the operator of integer division.

```
package main

import "fmt"

func main() {
    var bytes int
    fmt.Print("bytes = ")
    fmt.Scanf("%d", &bytes)
    kBytes := bytes / 1024
    fmt.Printf("kBytes = %d\n", kBytes)
}
```

Integer 4

Ададҳои бутуни мусбат A ва B (A>B) дода шудаанд. Дар порчаи дарозиаш A миқдори калонтарини имконпазири порчаҳои дарозиашон B ҷойгиранд. Бо истифодабарии амали тақсими бутун миқдори порчаҳои B-ро, ки дар порчаи A ҷойгиранд, ёбед.

Two positive integers *A* and *B* are given (*A* > *B*). Segment of length *A* contains the greatest possible amount of inside segments of length *B* (without overlaps). Find the amount of segments *B* placed on the segment *A*. Use the operator of integer division.

```
package main

import "fmt"

func main() {
    var a, b int
    fmt.Print("A = ")
    fmt.Scan(&a)
    fmt.Print("B = ")
    fmt.Scan(&b)
    porchaho := a / b;
    fmt.Printf("porchaho = %d\n", porchaho)
}
```

Integer 5

Ададҳои бутуни мусбат A ва B (A>B) дода шудаанд. Дар порчаи дарозиаш A миқдори калонтарини имконпазири порчаҳои дарозиашон B ҷойгиранд. Бо истифодабарии амали гирифтани бақия аз тақсими бутун дарозии қисми бандНАбудаи порчаи A-ро ёбед.

Two positive integers *A* and *B* are given (*A* > *B*). Segment of length *A* contains the greatest possible amount of inside segments of length *B* (without overlaps). Find the length of unused part of the segment *A*. Use the operator of taking the remainder after integer division.

```
package main
```

```
import "fmt"

func main() {
    var a, b int
    fmt.Print("A = ")
    fmt.Scan(&a)
    fmt.Print("B = ")
    fmt.Scan(&b)
    freeSpace := a % b
    fmt.Printf("freeSpace = %d\n", freeSpace)
}
```

Integer 6

Адади дурақама дода шудааст. Дар аввал рақами чапи он (даҳӣ)-ро, сонӣ рақами рости он (воҳид)-ро хориҷ кунед. Барои ёфтани даҳӣ амали тақсими бутунро, барои ёфтани воҳид бошад - амали гирифтани бақия аз тақсимро истифода буред.

A two-digit integer is given. Output its left digit (a tens digit) and then its right digit (a ones digit). Use the operator of integer division for obtaining the tens digit and the operator of taking remainder for obtaining the ones digit.

```
package main

import "fmt"

func main() {
    var number int
    fmt.Print("number [10-99] = ")
    fmt.Scanf("%d", &number)
    dahi := number / 10
    vohid := number % 10
    fmt.Printf("dahi = %d\n", dahi)
    fmt.Printf("vohid = %d\n", vohid)
}
```

Integer 7

Адади дурақама дода шудааст. Сумма ва ҳосилизарби рақамҳои онро ёбед.

A two-digit integer is given. Find the sum and the product of its digits.

```
package main

import "fmt"

func main() {
    var number int
    fmt.Print("number [10-99] = ")
    fmt.Scanf("%d", &number)
    dahi := number / 10
    vohid := number % 10
    sum := dahi + vohid
    mul := dahi * vohid
    fmt.Printf("sum = %d\nmultiplication = %d\n", sum, mul)
}
```

Integer 8

Адади дурақама дода шудааст. Ададеро хориҷ кунед, ки дар натиҷаи ҷойивазкунии рақамҳои адади ибтидоӣ пайдо гардидааст.

A two-digit integer is given. Output an integer obtained from the given one by exchange of its digits.

```
package main

import "fmt"

func main() {
    var number int
    fmt.Print("number [10-99] = ")
    fmt.Scanf("%d", &number)
    dahi := number / 10
    vohid := number % 10
    number = vohid * 10 + dahi
    fmt.Printf("number = %d\n", number)
}
```

Integer 9

Адади серақама дода шудааст. Бо истифодабарии як амали тақсими бутун рақами аввалаи адади мазкур (садӣ)-ро хориҷ кунед.

A three-digit integer is given. Using one operator of integer division find first digit of the given integer (a hundreds digit).

```
package main

import "fmt"
```

```
func main() {
    var number int
    fmt.Print("number [100-999] = ")
    fmt.Scanf("%d", &number)
    sadi := number / 100
    fmt.Printf("sadi = %d\n", sadi)
}
```

Integer 10

Адади сераҷама дода шудааст. Дар аввал рақами охирини он (воҳид)-ро, сонӣ рақами мобайнии он(даҳӣ)-ро хориҷ кунед.

A three-digit integer is given. Output its last digit (a ones digit) and then its middle digit (a tens digit).

```
package main

import "fmt"

func main() {
    var number int
    fmt.Print("number [100-999] = ")
    fmt.Scanf("%d", &number)
    vohid := number % 10
    dahi := number / 10 % 10
    fmt.Printf("vohid = %d\ndahi = %d\n", vohid, dahi)
}
```

Integer 11

Адади сераҷама дода шудааст. Сумма ва ҳосилизарби рақамҳои онро ёбед.

A three-digit integer is given. Find the sum and the product of its digits.

```
package main

import "fmt"

func main() {
    var number int
    fmt.Print("number [100-999] = ")
    fmt.Scanf("%d", &number)
    sadi := number / 100
    dahi := number / 10 % 10
    vohid := number % 10
    sum := sadi + dahi + vohid
    mul := sadi * dahi * vohid
```

```
    fmt.Printf("sum = %d\nmultiplication = %d\n", sum, mul)
}
```

Integer 12

Ададӣ сераҷама дода шудааст. Ададеро хориҷ кунед, ки дар натиҷаи хондани ададӣ ибтидоӣ аз рост ба чап ҳосил шудааст.

A three-digit integer is given. Output an integer obtained from the given one by reading it from right to left.

```
package main

import "fmt"

func main() {
    var number int
    fmt.Print("number [100-999] = ")
    fmt.Scanf("%d", &number)
    sadi := number / 100
    dahi := number / 10 % 10
    vohid := number % 10
    number = vohid * 100 + dahi * 10 + sadi
    fmt.Println("number = ", number)
}
```

Integer 13

Ададӣ сераҷама дода шудааст. Дар он раҷами аз чап якӯмро хат зада, онро аз рост нависед. Ададӣ ҳосилшударо хориҷ кунед.

A three-digit integer is given. Output an integer obtained from the given one by moving its left digit to the right side.

```
package main

import "fmt"

func main() {
    var number int
    fmt.Print("number [100-999] = ")
    fmt.Scanf("%d", &number)
    sadi := number / 100
    dahi := number / 10 % 10
    vohid := number % 10
    number = dahi * 100 + vohid * 10 + sadi
    fmt.Printf("number = %d\n", number)
}
```

Integer 14

Адади серақама дода шудааст. Дар он рақами аз рост якӯмро хат зада, онро аз чап нависед. Адади ҳосилшударо хорич кунед.

A three-digit integer is given. Output an integer obtained from the given one by moving its right digit to the left side.

```
package main

import "fmt"

func main() {
    var number int
    fmt.Print("number [100-999] = ")
    fmt.Scanf("%d", &number)
    sadi := number / 100
    dahi := number / 10 % 10
    vohid := number % 10
    number = vohid * 100 + sadi * 10 + dahi
    fmt.Printf("number = %d\n", number)
}
```

Integer 15

Адади серақама дода шудааст. Ададеро хорич кунед, ки дар натичаи ҷойивазкунии рақамҳои садӣ ва даҳии адади ибтидоӣ ҳосил шудааст (масалан, 123 мешавад 213).

A three-digit integer is given. Output an integer obtained from the given one by exchange a tens digit and a hundreds digit (for example, 123 will be changed to 213).

```
package main

import "fmt"

func main() {
    var number int
    fmt.Print("number [100-999] = ")
    fmt.Scanf("%d", &number)
    sadi := number / 100
    dahi := number / 10 % 10
    vohid := number % 10
    number = dahi * 100 + sadi * 10 + vohid
    fmt.Printf("number = %d\n", number)
}
```

Integer 16

Ададӣ сераҷама дода шудааст. Ададеро хориҷ кунед, ки дар натиҷаи ҷойивазкунии рақамҳои даҳӣ ва воҳидии ададӣ ибтидоӣ ҳосил шудааст.

A three-digit integer is given. Output an integer obtained from the given one by exchange a ones digit and a tens digit (for example, 123 will be changed to 132).

```
package main

import "fmt"

func main() {
    var number int
    fmt.Print("number [100-999] = ")
    fmt.Scanf("%d", &number)
    sadi := number / 100
    dahi := number / 10 % 10
    vohid := number % 10
    number = sadi * 100 + vohid * 10 + dahi
    fmt.Printf("number = %d\n", number)
}
```

Integer 17

Ададӣ бутуни аз 999 калон дода шудааст. Бо истифодабарии як амали тақсими бутун ва як амали гирифтани бақия аз тақсим рақамеро ёбед, ки ба қатори садии ин адад мувофиқ меояд.

An integer greater than 999 is given. Using one operator of integer division and one operator of taking the remainder find a hundreds digit of the given integer.

```
package main

import "fmt"

func main() {
    var number int
    fmt.Print("number [1000:] = ")
    fmt.Scanf("%d", &number)
    sadi := number % 1000 / 100
    fmt.Printf("sadi = %d\n", sadi)
}
```

Integer 18

Ададӣ бутуни аз 999 калон дода шудааст. Бо истифодабарии як амали тақсими бутун ва як амали гирифтани бақия аз тақсим рақамеро ёбед, ки ба қатори ҳазории ин адад мувофиқ меояд.

An integer greater than 999 is given. Using one operator of integer division and one operator of taking the remainder find a thousands digit of the given integer.

```
package main

import "fmt"

func main() {
    var number int
    fmt.Print("number [1000:] = ")
    fmt.Scanf("%d", &number)
    hazori := number / 1000 % 10
    fmt.Printf("hazori = %d\n", hazori)
}
```

Integer 19

Аз аввали шабонарӯз N сония (N-бутун) гузаштааст. Миқдори дақиқаҳои пурраеро, ки аз аввали шабонарӯз гузаштааст, ёбед.

From the beginning of the day *N* seconds have passed (*N* is integer). Find an amount of full minutes passed from the beginning of the day.

```
package main

import "fmt"

func main() {
    var n int
    fmt.Print("N = ")
    fmt.Scanf("%d", &n)
    minutes := n / 60
    fmt.Printf("minutes = %d\n", minutes)
}
```

Integer 20

Аз аввали шабонарӯз N сония (N-бутун) гузаштааст. Миқдори соатҳои пурраеро, ки аз аввали шабонарӯз гузаштааст, ёбед.

From the beginning of the day *N* seconds have passed (*N* is integer). Find an amount of full hours passed from the beginning of the day.

```
package main

import "fmt"

func main() {
    var n int
    fmt.Print("N = ")
    fmt.Scanf("%d", &n)
    hours := n / 3600
    fmt.Printf("hours = %d\n", hours)
}
```

Boolean 1

Адади бутун A дода шудааст. Дурустии гуфтори: «Адади A мусбӣ аст»-ро санҷед.

Given integer *A*, verify the following proposition: "The number *A* is positive".

```
package main

import "fmt"

func main() {
        var a int
        fmt.Print("A = ")
        fmt.Scanf("%d", &a)
        isPositive := a > 0
        fmt.Printf("Positive = %t\n", isPositive)
}
```

Boolean 2

Адади бутун A дода шудааст. Дурустии гуфтори: «Адади A тоқ аст»-ро санҷед.

Given integer *A*, verify the following proposition: "The number *A* is odd".

```
package main

import "fmt"

func main() {
    var a int
    fmt.Print("A = ")
    fmt.Scanf("%d", &a)
    isOdd := a % 2 != 0
    fmt.Printf("Odd = %t\n", isOdd)
}
```

Boolean 3

Адади бутун A дода шудааст. Дурустии гуфтори: «Адади A ҷуфт аст»-ро санҷед.

Given integer *A*, verify the following proposition: "The number *A* is even".

```
package main

import "fmt"

func main() {
    var a int
    fmt.Print("A = ")
    fmt.Scanf("%d", &a)
    isEven := a % 2 == 0
    fmt.Printf("Even = %t\n", isEven)
}
```

Boolean 4

Ду ададҳои бутун дода шудаанд: A, B. Дурустии гуфтори: «Нобаробариҳои A>2 ва B<=3 дурустанд»-ро санҷед.

Given two integers *A* and *B*, verify the following proposition: "The inequalities $A > 2$ and $B \leq 3$ both are fulfilled".

```
package main

import "fmt"

func main() {
    var a, b int
    fmt.Print("A = ")
    fmt.Scan(&a)
```

```
    fmt.Print("B = ")
    fmt.Scan(&b)
    result := a > 2 && b <= 3
    fmt.Printf("result = %t\n", result)
}
```

Boolean 5

Ду ададҳои бутун дода шудаанд: A, B. Дурустии гуфтори: «Нобаробариҳои A>=0 ё B<-2 дурустанд»-ро санҷед.

Given two integers A and B, verify the following proposition: "The inequality $A \geq 0$ is fulfilled or the inequality $B < -2$ is fulfilled".

```
package main

import "fmt"

func main() {
    var a, b int
    fmt.Print("A = ")
    fmt.Scan(&a)
    fmt.Print("B = ")
    fmt.Scan(&b)
    result := a >= 0 || b < -2
    fmt.Printf("result = %t\n", result)
}
```

Boolean 6

Се ададҳои бутун дода шудаанд: A, B, C. Дурустии гуфтори: «Нобаробарии дукаратаи A<B<C дуруст аст»-ро санҷед.

Given three integers A, B, C, verify the following proposition: "The double inequality $A < B < C$ is fulfilled".

```
package main

import "fmt"

func main() {
    var a, b, c int
    fmt.Print("A = ")
    fmt.Scan(&a)
    fmt.Print("B = ")
    fmt.Scan(&b)
    fmt.Print("C = ")
    fmt.Scan(&c)
    result := a < b && b < c
    fmt.Printf("result = %t\n", result)
```

```
}
```

Boolean 7

Се ададҳои бутун дода шудаанд: A, B, C. Дурустии гуфтори: «Адади B дар байни ададҳои A ва C ҷойгир аст»-ро санҷед.

Given three integers *A*, *B*, *C*, verify the following proposition: "The number *B* is between *A* and *C*".

```
package main

import "fmt"

func main() {
    var a, b, c int
    fmt.Print("A = ")
    fmt.Scan(&a)
    fmt.Print("B = ")
    fmt.Scan(&b)
    fmt.Print("C = ")
    fmt.Scan(&c)
    result := a < b && b < c || a > b && b > c
    fmt.Printf("result = %t\n", result)
}
```

Boolean 8

Ду ададҳои бутун дода шудаанд: A, B. Дурустии гуфтори: «Ҳар яке аз ададҳои A ва B тоқ аст»-ро санҷед.

Given two integers *A* and *B*, verify the following proposition: "Each of the numbers *A* and *B* is odd".

```
package main

import "fmt"

func main() {
    var a, b int
    fmt.Print("A = ")
    fmt.Scan(&a)
    fmt.Print("B = ")
    fmt.Scan(&b)
    result := (a % 2 != 0) && (b % 2 != 0)
    fmt.Printf("result = %t\n", result)
}
```

Boolean 9

Ду ададҳои бутун дода шудаанд: A, B. Дурустии гуфтори: «Аққалан яке аз ададҳои A ва B тоқ аст»-ро санҷед.

Given two integers *A* and *B*, verify the following proposition: "At least one of the numbers *A* and *B* is odd".

```
package main

import "fmt"

func main() {
    var a, b int
    fmt.Print("A = ")
    fmt.Scan(&a)
    fmt.Print("B = ")
    fmt.Scan(&b)
    result := (a % 2 != 0) || (b % 2 != 0)
    fmt.Printf("result = %t\n", result)
}
```

Boolean 10

Ду ададҳои бутун дода шудаанд: A, B. Дурустии гуфтори: «Раco яке аз ададҳои A ва B тоқ аст»-ро санҷед.

Given two integers *A* and *B*, verify the following proposition: "Exactly one of the numbers *A* and *B* is odd".

```
package main

import "fmt"

func main() {
    var a, b int
    fmt.Print("A = ")
    fmt.Scan(&a)
    fmt.Print("B = ")
    fmt.Scan(&b)
    result := (a + b) % 2 != 0
    fmt.Printf("result = %t\n", result)
}
```

Boolean 11

Ду ададҳои бутун дода шудаанд: A, B. Дурустии гуфтори: «Ададҳои A ва B ҷуфтии якхела доранд»-ро санҷед.

Given two integers *A* and *B*, verify the following proposition: "The numbers *A* and *B* have equal parity".

```
package main

import "fmt"

func main() {
    var a, b int
    fmt.Print("A = ")
    fmt.Scan(&a)
    fmt.Print("B = ")
    fmt.Scan(&b)
    result := (a + b) % 2 == 0
    fmt.Printf("result = %t\n", result)
}
```

Boolean 12

Се ададҳои бутун дода шудаанд: A, B, C. Дурустии гуфтори: «Ҳар яке аз ададҳои A, B, C мусбат аст»-ро санҷед.

Given three integers *A*, *B*, *C*, verify the following proposition: "Each of the numbers *A*, *B*, *C* is positive".

```
package main

import "fmt"

func main() {
    var a, b, c int
    fmt.Print("A = ")
    fmt.Scan(&a)
    fmt.Print("B = ")
    fmt.Scan(&b)
    fmt.Print("C = ")
    fmt.Scan(&c)
    result := a > 0 && b > 0 && c > 0
    fmt.Printf("result = %t\n", result)
}
```

Boolean 13

Се ададҳои бутун дода шудаанд: A, B, C. Дурустии гуфтори: «Аққалан яке аз ададҳои A, B, C мусбат аст»-ро санҷед.

Given three integers *A*, *B*, *C*, verify the following proposition: "At least one of the numbers *A*, *B*, *C* is positive".

```
package main

import "fmt"

func main() {
    var a, b, c int
```

```
    fmt.Print("A = ")
    fmt.Scan(&a)
    fmt.Print("B = ")
    fmt.Scan(&b)
    fmt.Print("C = ")
    fmt.Scan(&c)
    result := a > 0 || b > 0 || c > 0
    fmt.Printf("result = %t\n", result)
}
```

Boolean 14

Се ададҳои бутун дода шудаанд: A, B, C. Дурустии гуфтори: «Рақо яке аз ададҳои A, B, C мусбат аст»-ро санҷед.

Given three integers *A*, *B*, *C*, verify the following proposition: "Exactly one of the numbers *A*, *B*, *C* is positive".

```
package main

import "fmt"

func main() {
    var a, b, c int
    fmt.Print("A = ")
    fmt.Scan(&a)
    fmt.Print("B = ")
    fmt.Scan(&b)
    fmt.Print("C = ")
    fmt.Scan(&c)
    result := a > 0 && b <= 0 && c <= 0 ||
              a <= 0 && b > 0 && c <= 0 ||
              a <= 0 && b <= 0 && c > 0
    fmt.Printf("result = %t\n", result)
}
```

Boolean 15

Се ададҳои бутун дода шудаанд: A, B, C. Дурустии гуфтори: «Рақо дуто аз ададҳои A, B, C мусбатанд»-ро санҷед.

Given three integers *A*, *B*, *C*, verify the following proposition: "Exactly two of the numbers *A*, *B*, *C* are positive".

```
package main

import "fmt"

func main() {
    var a, b, c int
    fmt.Print("A = ")
    fmt.Scan(&a)
```

```
    fmt.Print("B = ")
    fmt.Scan(&b)
    fmt.Print("C = ")
    fmt.Scan(&c)
    result := a > 0 && b > 0 && c <= 0 ||
              a > 0 && b <= 0 && c > 0 ||
              a <= 0 && b > 0 && c > 0
    fmt.Printf("result = %t\n", result)
}
```

Boolean 16

Адади бутуни мусбӣ дода шудааст. Дурустии гуфтори: «Адади мазкур адади ҷуфти дураҷама аст»-ро санҷед.

Given a positive integer, verify the following proposition: "The integer is a two-digit even number".

```
package main

import "fmt"

func main() {
    var number uint
    fmt.Print("number = ")
    fmt.Scanf("%d", &number)
    var result bool = (number % 2 == 0) && (number >= 10) && (number <= 99)
    fmt.Printf("result = %t\n", result)
}
```

Boolean 17

Адади бутуни мусбӣ дода шудааст. Дурустии гуфтори: «Адади мазкур адади тоқи сераҷама аст»-ро санҷед.

Given a positive integer, verify the following proposition: "The integer is a three-digit odd number".

```
package main

import "fmt"

func main() {
    var number uint
    fmt.Print("number = ")
    fmt.Scanf("%d", &number)
    var result bool = (number % 2 != 0) && (number >= 100) && (number <= 999)
    fmt.Printf("result = %t\n", result)
}
```

Boolean 18

Дурустии гуфтори: «Дар байни се ададҳои додашудаи бутун аққалан як ҷуфти ададҳои мувофиқоянд ҳаст»-ро санҷед.

Verify the following proposition: "Among three given integers there is at least one pair of equal ones".

```
package main

import "fmt"

func main() {
    var a, b, c int
    fmt.Scan(&a, &b, &c)
    var result bool = a == b || b == c || c == a
    fmt.Printf("result = %t\n", result)
}
```

Boolean 19

Дурустии гуфтори: «Дар байни се ададҳои додашудаи бутун аққалан як ҷуфти ададҳои дутарафа муқобил ҳаст»-ро санҷед.

Verify the following proposition: "Among three given integers there is at least one pair of opposite ones".

```
package main

import "fmt"

func main() {
    var a, b, c int
    fmt.Scan(&a, &b, &c)
    var result bool = a == -b || b == -c || c == -a
    fmt.Printf("result = %t\n", result)
}
```

Boolean 20

Адади сераҳама дода шудааст. Дурустии гуфтори: «Ҳамаи рақамҳои адади мазкур гуногунанд»-ро санҷед.

Given a three-digit integer, verify the following proposition: "All digits of the number are different".

```
package main

import "fmt"
```

```
func main() {
    var number int
    fmt.Print("number [100-999] = ")
    fmt.Scanf("%d", &number)
    var sadi int = number / 100
    var dahi int = number / 10 % 10
    var vohid int = number % 10
    var result bool = sadi != dahi && dahi != vohid && vohid != sadi
    fmt.Printf("result = %t\n", result)
}
```

If 1

Ададӣ бутун дода шудааст. Агар он мусбӣ бошад, аз он 8-ро тарҳ кунед; дар ҳолати акс онро тағйир надиҳед. Ададӣ ҳосилшударо хориҷ кунед.

An integer is given. If the integer is positive then decrease it by 8, otherwise do not change it. Output the obtained integer.

```
package main

import "fmt"

func main() {
    var number int
    fmt.Print("number = ")
    fmt.Scanf("%d", &number)
    if number > 0 {
        number -= 8
    }
    fmt.Printf("newNumber = %d\n", number)
}
```

If 2

Ададӣ бутун дода шудааст. Агар он мусбӣ бошад, аз он 8-ро тарҳ кунед; дар ҳолати акс ба он 8-ро ҳамроҳ кунед. Ададӣ ҳосилшударо хориҷ кунед.

An integer is given. If the integer is positive then decrease it by 8, otherwise increase it by 6. Output the obtained integer.

```
package main

import "fmt"
```

```
func main() {
    var number int
    fmt.Print("number = ")
    fmt.Scanf("%d", &number)
    if number > 0 {
        number -= 8
    } else {
        number += 6
    }
    fmt.Printf("newNumber = %d\n", number)
}
```

If 3

Адади бутун дода шудааст. Агар он мусбӣ бошад, аз он 8-ро тарҳ кунед; агар манфӣ бошад, пас ба он 8-ро ҳамроҳ кунед; агар нулӣ бошад, пас онро ба 10 иваз кунед. Адади ҳосилшударо хориҷ кунед.

An integer is given. If the integer is positive then decrease it by 8, if the integer is negative then increase it by 6, if the integer equals 0 then change it to 10. Output the obtained integer.

```
package main

import "fmt"

func main() {
    var number int
    fmt.Print("number = ")
    fmt.Scanf("%d", &number)
    if number > 0 {
        number -= 8
    } else if number < 0 {
        number += 6
    } else {
        number = 10
    }
    fmt.Printf("newNumber = %d\n", number)
}
```

If 4

Се ададҳои бутун дода шудаанд. Миқдори ададҳои мусбиро дар маҷмӯаи ибтидоӣ ёбед.

Three integers are given. Find the amount of positive integers in the input data.

```
package main
```

```
import "fmt"

func main() {
    var x, y, z int
    fmt.Print("number1 = ")
    fmt.Scan(&x)
    fmt.Print("number2 = ")
    fmt.Scan(&y)
    fmt.Print("number3 = ")
    fmt.Scan(&z)
    positives := 0
    if x > 0 { positives++ }
    if y > 0 { positives++ }
    if z > 0 { positives++ }
    fmt.Printf("positives: %d\n", positives)
}
```

If 5

Се ададҳои бутун дода шудаанд. Миқдори ададҳои мусбӣ ва миқдори ададҳои манфиро дар маҷмӯаи ибтидоӣ ёбед.

Three integers are given. Find the amount of positive and amount of negative integers in the input data.

```
package main

import "fmt"

func main() {
    var a, b, c int
    fmt.Scan(&a, &b, &c)
    var positives, negatives uint = 0, 0
    if a > 0 { positives++ }
    if b > 0 { positives++ }
    if c > 0 { positives++ }
    if a < 0 { negatives++ }
    if b < 0 { negatives++ }
    if c < 0 { negatives++ }
    fmt.Printf("positives = %d\nnegatives = %d\n", positives, negatives)
}
```

If 6

Ду ададҳо дода шудаанд. Калонтарини онҳоро ёбед.

Given two real numbers, output the larger value of them.

```
package main

import "fmt"

func main() {
```

```
    var a, b, kalon float64
    fmt.Scan(&a, &b)
    if a > b {
        kalon = a
    } else {
        kalon = b
    }
    fmt.Printf("greater is %.2f\n", kalon)
}
```

If 7

Ду ададҳо дода шудаанд. Рақами тартибии хурдтарини онҳоро ёбед.

Given two real numbers, output the order number of the smaller of them.

```
package main

import "fmt"

func main() {
    var a, b float64
    var index uint
    fmt.Scan(&a, &b)
    if a < b {
        index = 1
    } else {
        index = 2
    }
    fmt.Printf("index = %d\n", index)
}
```

If 8

Ду ададҳо дода шудаанд. Дар аввал калонтарин ва сонӣ хурдтарини онҳоро ёбед.

Given two real numbers, output the larger value and then the smaller value of them.

```
package main

import "fmt"

func main() {
    var a, b, kalon, xurd float64
    fmt.Scan(&a, &b)
    kalon, xurd = a, b
    if kalon < xurd {
        kalon, xurd = b, a
```

```
    }
    fmt.Printf("greater = %.2f\nsmaller = %.2f\n", kalon, xurd)
}
```

If 9

Ду тағйирёбандаҳои типи ҳақиқӣ дода шудаанд: A, B. Қиматҳои тағйирёбандаҳои мазкурро чунон ҷобаҷо кунед, ки дар A қимати хурдтарин ва дар B бошад — қимати калонтарин ҷоргир шавад. Қиматҳои нави тағйирёбандаҳои A ва B-ро хориҷ кунед.

The values of two real variables *A* and *B* are given. Redistribute the values so that *A* and *B* have the smaller and the larger value respectively. Output the new values of the variables *A* and *B*.

```
package main

import "fmt"

func main() {
    var a, b float32
    fmt.Print("A = ")
    fmt.Scan(&a)
    fmt.Print("B = ")
    fmt.Scan(&b)
    if a > b {
        tmp := a
        a = b
        b = tmp
    }
    fmt.Printf("A = %.2f\nB = %.2f\n", a, b)
}
```

If 10

Ду тағйирёбандаҳои типи бутун дода шудаанд: A ва B. Агар қиматҳои онҳо нобаробар бошанд, пас ба ҳар як тағйирёбанда суммаи ин қиматҳоро бахшед, аммо агар баробар бошанд, пас ба тағйирёбандаҳо қиматҳои нулиро бахшед. Қиматҳои нави тағйирёбандаҳои A ва B-ро хориҷ кунед.

The values of two integer variables *A* and *B* are given. If the values are not equal then assign the sum of given values to each

variable, otherwise assign zero value to each variable. Output the new values of the variables *A* and *B*.

```
package main

import "fmt"

func main() {
    var a, b int
    fmt.Print("A = ")
    fmt.Scan(&a)
    fmt.Print("B = ")
    fmt.Scan(&b)
    if a != b {
        a, b = a + b, a + b
    } else {
        a, b = 0, 0
    }
    fmt.Printf("A = %d\nB = %d\n", a, b)
}
```

If 11

Ду тағйирёбандаҳои типи бутун дода шудаанд: A ва B. Агар қиматҳои онҳо нобаробар бошанд, пас ба ҳар як тағйирёбанда калонтарини ин қиматҳоро бахшед, аммо агар баробар бошанд, пас ба тағйирёбандаҳо қиматҳои нулиро бахшед. Қиматҳои нави тағйирёбандаҳои A ва B-ро хориҷ кунед.

The values of two integer variables *A* and *B* are given. If the values are not equal then assign the larger value to each variable, otherwise assign zero value to each variable. Output the new values of the variables *A* and *B*.

```
package main

import "fmt"

func main() {
    var a, b int
    fmt.Print("A = ")
    fmt.Scan(&a)
    fmt.Print("B = ")
    fmt.Scan(&b)
    if a != b {
        kalon := a
        if a < b { kalon = b }
        a, b = kalon, kalon
    } else {
        a, b = 0, 0
```

```
    }
    fmt.Printf("A = %d\nB = %d\n", a, b)
}
```

If 12

Се ададҳо дода шудаанд. Хурдтарини онҳоро хориҷ кунед.

Given three real numbers, output the minimal value of them.

```
package main

import "fmt"

func main() {
    var a, b, c, xurd float32
    fmt.Scan(&a, &b, &c)
    if a < b && a < c {
        xurd = a
    } else if b < c {
        xurd = b
    } else {
        xurd = c
    }
    fmt.Printf("smaller = %.2f\n", xurd)
}
```

If 13

Се ададҳо дода шудаанд. Қимати мобайниро аз байни онҳо ёбед (яъне ададеро, ки дар байни қиматҳои калонтарин ва хурдтарин ҷойгир аст).

Given three real numbers, output the value between the minimum and the maximum.

```
package main

import "fmt"

func main() {
    var a, b, c, kalon, xurd float32
    fmt.Scan(&a, &b, &c)
    if a < b && a < c {
        xurd = a
    } else if b < c {
        xurd = b
    } else {
        xurd = c
    }
    if a > b && a > c {
        kalon = a
    } else if b > c {
```

```
        kalon = b
    } else {
        kalon = c
    }
    bayn := a + b + c - kalon - xurd
    fmt.Printf("bayn = %.2f", bayn)
}
```

If 14

Се ададҳо дода шудаанд. Дар аввал адади хурдтарин ва сонӣ адади калонтаринро хориҷ кунед.

Given three real numbers, output the minimal value and then the maximal value.

```
package main

import "fmt"

func main() {
    var a, b, c, kalon, xurd float32
    fmt.Scan(&a, &b, &c)
    if a < b && a < c {
        xurd = a
    } else if b < c {
        xurd = b
    } else {
        xurd = c
    }
    if a > b && a > c {
        kalon = a
    } else if b > c {
        kalon = b
    } else {
        kalon = c
    }
    fmt.Printf("smaller = %.2f\ngreater = %.2f", xurd, kalon)
}
```

If 15

Се ададҳо дода шудаанд. Суммаи ду ададҳои калонтаринро аз байни онҳо ёбед.

Given three real numbers, output the sum of two largest values.

```
package main

import "fmt"

func main() {
    var a, b, c, xurd float32
```

```
    fmt.Scan(&a, &b, &c)
    if a < b && a < c {
        xurd = a
    } else if b < c {
        xurd = b
    } else {
        xurd = c
    }
    sum := a + b + c - xurd
    fmt.Printf("sum = %.2f\n", sum)
}
```

If 16

Се тағйирёбандаҳои типи ҳақиқӣ дода шудаанд: A, B, C. Агар қиматҳои онҳо аз рӯи афзуншавӣ ҷобаҷо карда шуда бошанд, пас онҳоро ба ду зарб кунед; дар ҳолати акс қимати ҳар як тағйирёбандаро ба муқобилаломаташ иваз кунед. Қиматҳои нави тағйирёбандаҳои A, B, C-ро хориҷ кунед.

The values of three real variables *A*, *B*, *C* are given. If the values are in ascending order then double them, otherwise replace the value of each variable by its opposite value. Output the new values of the variables *A*, *B*, *C*.

```
package main

import "fmt"

func main() {
    var a, b, c float32
    fmt.Print("A = ")
    fmt.Scan(&a)
    fmt.Print("B = ")
    fmt.Scan(&b)
    fmt.Print("C = ")
    fmt.Scan(&c)
    if a < b && b < c {
        a *= 2
        b *= 2
        c *= 2
    } else {
        a, b, c = -a, -b, -c
    }
    fmt.Printf("A = %.2f\nB = %.2f\nC = %.2f\n", a, b, c)
}
```

If 17

Се тағйирёбандаҳои типи ҳақиқӣ дода шудаанд: A, B, C. Агар қиматҳои онҳо аз рӯи афзуншавӣ ва ё камшавӣ ҷобаҷо карда шуда бошанд, пас онҳоро ба ду зарб кунед; дар холати акс қимати ҳар як тағйирёбандаро ба муқобилаломаташ иваз кунед. Қиматҳои нави тағйирёбандаҳои A, B, C-ро хориҷ кунед.

The values of three real variables *A*, *B*, *C* are given. If the values are in ascending or descending order then double them, otherwise replace the value of each variable by its opposite value. Output the new values of the variables *A*, *B*, *C*.

```
package main

import "fmt"

func main() {
    var a, b, c float32
    fmt.Print("A = ")
    fmt.Scan(&a)
    fmt.Print("B = ")
    fmt.Scan(&b)
    fmt.Print("C = ")
    fmt.Scan(&c)
    if a < b && b < c || a > b && b > c {
        a *= 2
        b *= 2
        c *= 2
    } else {
        a, b, c = -a, -b, -c
    }
    fmt.Printf("A = %.2f\nB = %.2f\nC = %.2f\n", a, b, c)
}
```

If 18

Се ададҳои бутун дода шудаанд, ки яке аз онҳо аз ду ададҳои дигарии байни ҳам баробар фарқ мекунад. Рақами тартибии ададифарққунандаро муайян кунед.

Three integers are given. One of them differs from two other equal integers. Output the order number of the integer that differs from the others.

```
package main

import "fmt"
```

```
func main() {
    var a, b, c, index int
    fmt.Scan(&a, &b, &c)
    if (a == b) {
        index = 3
    } else if a == c {
        index = 2
    } else {
        index = 1
    }
    fmt.Printf("index = %d\n", index)
}
```

If 19

Чор ададҳои бутун дода шудаанд, ки яке аз онҳо аз се дадҳои дигарии байни ҳам баробар фарқ мекунад. Рақами тартибии ададӣ фарққунандаро муайян кунед.

Four integers are given. One of them differs from three other equal integers. Output the order number of the integer that differs from the others.

```
package main

import "fmt"

func main() {
    var a, b, c, d, index int
    fmt.Scan(&a, &b, &c, &d)
    if a == b && b == c {
        index = 4
    } else if a == b && b == d {
        index = 3
    } else if a == c && c == d {
        index = 2
    } else {
        index = 1
    }
    fmt.Printf("index = %d\n", index)
}
```

If 20

Дар тири ададӣ се нуқтаҳои: A, B, C ҷойгир шудаанд. Муайян кунед, ки кадоме аз ду нуқтаҳои охирӣ (B ё C) ба нуқтаи A наздиктар ҷой гирифтааст. Ин нуқта ва масофаи онро аз нуқтаи A хориҷ кунед.

Three points *A*, *B*, *C* on the real axis are given. Determine whether *B* or *C* is closer to *A*. Output the nearest point and its distance from *A*.

```
package main

import (
    "fmt"
    "math"
)

func main() {
    var a, b, c float64
    fmt.Print("A = ")
    fmt.Scan(&a)
    fmt.Print("B = ")
    fmt.Scan(&b)
    fmt.Print("C = ")
    fmt.Scan(&c)
    ab := math.Abs(b - a)
    ac := math.Abs(c - a)
    if ab < ac {
        fmt.Printf("closest point: %.2f\nclosest distance: %.2f\n", b, ab)
    } else {
        fmt.Printf("closest point: %.2f\nclosest distance: %.2f\n", c, ac)
    }
}
```

Case 1

Ададӣ бутун дар фосилаи 1–7 дода шудааст. Сатр - номи рӯзи ҳафтаи ба ададӣ мазкур мувофиқояндаро хориҷ кунед (1 — «душанбе», 2 — «сешанбе» ва ғ.).

An integer in the range 1 to 7 is given. Output the name of the respective day of week: 1 — "Monday", 2 — "Tuesday", …, 7 — "Sunday".

```
package main

import "fmt"

func main() {
    var number int
    fmt.Print("number [1-7] = ")
    fmt.Scanf("%d", &number)
    weekDay := "Errorday";
    switch number {
        case 1: weekDay = "Monday"
        case 2: weekDay = "Tuesday"
        case 3: weekDay = "Wednesday"
```

```
        case 4: weekDay = "Thursday"
        case 5: weekDay = "Friday"
        case 6: weekDay = "Saturday"
        case 7: weekDay = "Sunday"
    }
    fmt.Printf("weekDay: %s\n", weekDay)
}
```

Case 2

Адади бутун K дода шудааст. Сатр - тасвири баҳои ба адади K мувофиқояндаро хориҷ кунед (1 — «бад», 2 — «ноком», 3 — «қаноатбахш», 4 — «хуб», 5 — «аъло»). Агар K дар фосилаи 1–5 нахобад, пас сатри «хатогӣ»-ро хориҷ кунед.

Given an integer *K*, output the respective examination mark: 1 — "bad", 2 — "unsatisfactory", 3 — "mediocre", 4 — "good", 5 — "excellent". If *K* is not in the range 1 to 5 then output string "error".

```
package main

import "fmt"

func main() {
    var K int
    fmt.Print("K = ")
    fmt.Scanf("%d", &K)
    mark := "";
    switch K {
        case 1: mark = "bad"
        case 2: mark = "ghayriqanoatbaxsh"
        case 3: mark = "qanoatbaxsh"
        case 4: mark = "xub"
        case 5: mark = "a'lo"
        default: mark = "error"
    }
    fmt.Println("mark:", mark)
}
```

Case 3

Рақами моҳ — адади бутун дар фосилаи 1–12 дода шудааст (1 — январ, 2 — феврал ва ғ.). Номи фасли мувофиқи солро хориҷ кунед («зимистон», «баҳор», «тобистон», «тирамоҳ»).

A number of month is given (as an integer in the range 1 to 12): 1 — January, 2 — February, etc. Output the name of the respective season: "Winter", "Spring", "Summer", "Autumn".

```
package main

import "fmt"

func main() {
    var monthNo int
    fmt.Print("monthNumber [1-12] = ")
    fmt.Scanf("%d", &monthNo)
    seasonName := ""
    switch monthNo {
        case 1, 2, 12: seasonName = "Winter"
        case 3, 4, 5: seasonName = "Spring"
        case 6, 7, 8: seasonName = "Summer"
        case 9, 10, 11: seasonName = "Autumn"
    }
    fmt.Println("seasonName = ", seasonName)
}
```

Case 4

Рақами моҳ — адади бутун дар фосилаи 1–12 дода шудааст (1 — январ, 2 — феврал ва ғ.). Миқдори рӯзҳоро дар ин моҳ барои соли муқаррарӣ муайян кунед.

A number of month is given (as an integer in the range 1 to 12): 1 — January, 2 — February, etc. Output the amount of days in the month for a non-leap year.

```
package main

import "fmt"

func main() {
    var monthNo int
    fmt.Print("monthNumber [1-12] = ")
    fmt.Scanf("%d", &monthNo)
    days := 0
    switch monthNo {
        case 2: days = 28
        case 4, 6, 9, 11: days = 30
        case 1, 3, 5, 7, 8, 10, 12: days = 31
    }
    fmt.Println("days: ", days)
}
```

Case 5

Амалҳои арифметикӣ таҳти ададҳо ба тариқи зайл рақамгузорӣ карда шудаанд: 1 — ҷамъ, 2 — тарҳ, 3 — зарб, 4 — тақсим. Рақами амал N (ададӣ бутун дар фосилаи 1–4) ва ададҳои ҳақиқӣ A ва B (B НОбаробари 0(нул) аст) дода шудаанд. Амали нишондодашударо таҳти ададҳо иҷро карда, натиҷаро хориҷ кунед.

The arithmetic operations are numbered as: 1 — addition, 2 — subtraction, 3 — multiplication, 4 — division. The order number *N* of an operation and two real numbers *A* and *B* are given (*N* is an integer in the range 1 to 4, *B* is not equal to 0). Perform the operation with the operands *A* and *B* and output the result.

```
package main

import "fmt"

func main() {
	var n int
	fmt.Print("N = ")
	fmt.Scan(&n)
	var a, b, result float32
	fmt.Print("A = ")
	fmt.Scan(&a)
	fmt.Print("B = ")
	fmt.Scan(&b)
	fmt.Printf("%.2f", a)
	switch n {
	case 1:
		result = a + b
		fmt.Print(" + ")
	case 2:
		result = a - b
		fmt.Print(" - ")
	case 3:
		result = a * b
		fmt.Print(" * ")
	case 4:
		result = a / b
		fmt.Print(" / ")
	}
	fmt.Printf("%.2f = %.2f\n", b, result)
}
```

Case 6

Бузургиҳои дарозӣ ба тариқи зайл рақамгузорӣ карда шудаанд: 1 — детсиметр, 2 — километр, 3 — метр, 4 —

миллиметр, 5 — сантиметр. Рақами бузургии дарозӣ (адади бутун дар фосилаи 1–5) ва дарозии порча бо ин бузургиҳо (адади ҳақиқӣ) дода шудаанд. Дарозии порчаро бо метр ёбед.

The units of length are numbered as: 1 — decimeter, 2 — kilometer, 3 — meter, 4 — millimeter, 5 — centimeter. The order number *N* of a unit of length and also the length *L* of a segment are given (*N* is an integer in the range 1 to 5, *L* is a real number). Output the length of the segment in meters.

```
package main

import "fmt"

func main() {
        var nomer int
        fmt.Scan(&nomer)
    switch nomer {
        case 1: fmt.Print("Length (in dm):\t")
        case 2: fmt.Print("Length (in km):\t")
        case 3: fmt.Print("Length (in m):\t")
        case 4: fmt.Print("Length (in mm):\t")
        case 5: fmt.Print("Length (in sm):\t")
        }
    var value float64
    fmt.Scan(&value)
    switch nomer {
        case 1: value /= 10
        case 2: value *= 1000
        //case 3:
        case 4: value /= 1000
        case 5: value /= 100
    }
    fmt.Println("Length (in m):\t", value)
}
```

Case 7

Бузургиҳои вазн ба тариқи зайл рақамгузорӣ карда шудаанд: 1 — килограмм, 2 — миллиграмм, 3 — грамм, 4 — тонна, 5 — сентнер. Рақами бузургии вазн (адади бутун дар фосилаи 1–5) ва вазни ҷисм бо ин бузургиҳо (адади ҳақиқӣ) дода шудаанд. Вазни ҷисмро бо килограм ёбед.

The units of weight are numbered as: 1 — kilogram, 2 — milligram, 3 — gram, 4 — ton, 5 — centner (= 100 kilograms).

The order number *N* of a unit of weight and the mass *M* of a solid are given (*N* is an integer in the range 1 to 5, *M* is a real number). Output the mass of the solid in kilograms.

```
package main

import "fmt"

func main() {
        var nomer int
        fmt.Scan(&nomer)
    switch nomer {
        case 1: fmt.Print("Weight (in kg):\t")
        case 2: fmt.Print("Weight (in mg):\t")
        case 3: fmt.Print("Weight (in g):\t")
        case 4: fmt.Print("Weight (in tn):\t")
        case 5: fmt.Print("Weight (in ct):\t")
        }
    var value float64
    fmt.Scan(&value)
    switch nomer {
        //case 1:
        case 2: value /= 1000000
        case 3: value /= 1000
        case 4: value *= 1000
        case 5: value *= 100
    }
    fmt.Println("Weight (in kg):\t", value)
}
```

Case 8

Ду ададҳои бутун дода шудаанд: D (рӯз) ва M (моҳ), ки санаи дурусти соли муқаррариро муайян мекунанд. Қиматҳои D ва M-ро барои як рӯз пеш аз санаи додашуда ёбед.

Given two integers *D* (day) and *M* (month) representing a correct date of a non-leap year, output values *D* and *M* for the previous date.

```
package main

import "fmt"

func main() {
    var d, m int
    fmt.Print("D = ")
    fmt.Scan(&d)
    fmt.Print("M = ")
    fmt.Scan(&m)
    switch d {
```

```
        case 1:
            switch m {
                case 1:
                    d, m = 31, 12
                default:
                    switch m {
                        case 3: d = 28
                        case 5, 7, 10, 12: d = 30
                        default: d = 31
                    }
                    m--
            }
        default: d--
    }
    fmt.Printf("D = %d\t\tM = %d\n", d, m)
}
```

Case 9

Ду ададҳои бутун дода шудаанд: D (рӯз) ва M (моҳ), ки санаи дурусти соли муқаррариро муайян мекунанд. Қиматҳои D ва M-ро барои як рӯз пас аз санаи додашуда ёбед.

Given two integers *D* (day) and *M* (month) representing a correct date of a non-leap year, output values *D* and *M* for the next date.

```
package main

import "fmt"

func main() {
    var d, m int
    fmt.Print("D = ")
    fmt.Scan(&d)
    fmt.Print("M = ")
    fmt.Scan(&m)
    switch m {
        case 1, 3, 5, 7, 8, 10:
            switch d {
                case 31: d, m = 1, m + 1
                default: d++
            }
        case 4, 6, 9, 11:
            switch d {
                case 30: d, m = 1, m + 1
                default: d++
            }
        case 2:
            switch d {
                case 28: d, m = 1, m + 1
                default: d++
            }
        case 12:
```

```
            switch d {
                case 31: d, m = 1, 1
                default: d++
            }
    }
    fmt.Printf("D = %d\t\tM = %d\n", d, m)
}
```

Case 10

Робот ба чаҳор самт ҳаракат карда метавонад («N» — шимол, «W» — ғарб, «S» — ҷануб, «E» — шарқ) ва метавонад се фармонро қабул кунад: 0 — ҳаракатро давом додан, 1 — баргаштан ба чап, –1 — баргаштан ба рост. Аломат C — самти ибтидоии робот ва адади бутун N — фармони ба он фиристодашаванда дода шудаанд. Самти роботро пас аз иҷрои фармони гирифташуда хориҷ кунед.

A robot can move in four directions ("N" — north, "W" — west, "S" — south, "E" — east) and perform three digital instructions: 0 — "move in the former direction", 1 — "turn left", −1 — "turn right". A symbol *C* (an initial direction of the robot) and an integer *N* (an instruction) are given. Output the direction of the robot (as symbol) after performing the instruction.

```
package main

import "fmt"

func main() {
    var (
        c string
        n int
    )
    fmt.Print("Direction: C = ")
    fmt.Scan(&c)
    fmt.Print("N = ")
    fmt.Scan(&n)
    switch c {
        case "N", "n":
            switch n {
                case 1: c = "W"
                case -1: c = "E"
            }
        case "S", "s":
            switch n {
                case 1: c = "E"
                case -1: c = "W"
            }
        case "W", "w":
```

```
            switch n {
                case 1: c = "S"
                case -1: c = "N"
            }
        case "E", "e":
            switch n {
                case 1: c = "N"
                case -1: c = "S"
            }
    }
    fmt.Printf("New Direction: C = %s\n", c)
}
```

Case 11

Локатор ба яке аз тарафҳои олам нигаронида шудааст («N» — шимол, «W» — ғарб, «S» — ҷануб, «E» — шарқ) ва се фармонҳои рақамии гардишро қабул карда метавонад: 1 — гардиш ба чап, –1 — гардиш ба рост, 2 — гардиш ба 180° (қафо). Аломат C — самти ибтидоии локатор ва ададҳои бутун N_1 ва N_2 — ду фармонҳои фиристодашуда дода шудаанд. Самти локаторро пас иҷрои ин фармонҳо хориҷ кунед.

A locator can be focused on the directions "N" (north), "W" (west), "S" (south), "E" (east) and perform three digital instructions: 1 — "turn left", −1 — "turn right", 2 — "turn on 180°"). A symbol *C* (an initial direction of the locator) and two integers N_1 and N_2 (instructions) are given. Output the direction of the locator (as symbol) after performing the instructions.

```
package main

import "fmt"

func main() {
    var (
        c string
        n1, n2 int
    )
    fmt.Print("Direction: C = ")
    fmt.Scan(&c)
    fmt.Print("N1 = ")
    fmt.Scan(&n1)
    fmt.Print("N2 = ")
    fmt.Scan(&n2)

    //1+1 = 2   180
    //1-1 = 0   continue
```

```
    //1+2 = 3   to right
    //-1+1 = 0  continue
    //-1-1 = -2 180
    //-1+2 = 1  to left
    //2+1 = 3   to right
    //2-1 = 1   to left
    //2+2 = 4   continue

    switch n1 + n2 {
        case 1: //to left
            switch c {
                case "N", "n": c = "W"
                case "W", "w": c = "S"
                case "S", "s": c = "E"
                case "E", "e": c = "N"
            }
        case 2, -2: //turn 180
            switch c {
                case "N", "n": c = "S"
                case "S", "s": c = "N"
                case "W", "w": c = "E"
                case "E", "e": c = "W"
            }
        case 3: //to right
            switch c {
                case "N", "n": c = "E"
                case "E", "e": c = "S"
                case "S", "s": c = "W"
                case "W", "w": c = "N"
            }
        //case 0, 4: //continue
    }
    fmt.Printf("New Direction: C = %s\n", c)
}
```

Case 12

Элементҳои давра ба тариқи зайл рақамгузорӣ карда шудаанд: 1 — радиус R, 2 — диаметр D=2·R, 3 — дарозӣ L=2·π·R, 4 — масоҳати доира S=π·R^2. Рақами яке аз ин элементҳо ва қимати он дода шудаанд. Қиматҳои элементҳои боқимондаи давраи мазкурро (бо ҳамон тартиб) хориҷ кунед. Ба сифати қимати π 3.14-ро истифода баред.

Elements of a circle are numbered as: 1 — radius *R*, 2 — diameter *D* = 2·*R*, 3 — length *L* = 2·π·*R* of the circumference, 4 — area *S* = π·*R*^{2}. The order number of one element and its value (as a real number) are given. Output values of other elements in the same order. Use 3.14 for a value of π.

```
package main
```

```
import (
    "fmt"
    "math"
)

func main() {
    const PI = 3.14
    var nomer int
    fmt.Scan(&nomer)
    switch nomer {
        case 1: fmt.Print("R = ")
        case 2: fmt.Print("D = ")
        case 3: fmt.Print("L = ")
        case 4: fmt.Print("S = ")
    }
    var value, r, d, l, s float64
    fmt.Scan(&value)
    switch nomer {
        case 1: r = value
            d = 2 * r;
            l = 2 * PI * r;
            s = PI * r * r;
        case 2: d = value
            r = d / 2;
            l = 2 * PI * r;
            s = PI * r * r;
        case 3: l = value
            r = l / (2 * PI);
            d = 2 * r;
            s = PI * r * r;
        case 4: s = value
            r = math.Sqrt(s / PI);
            d = 2 * r;
            l = 2 * PI * r;
    }
    if nomer != 1 {
        fmt.Printf("R = %.2f\t", r)
    }
    if nomer != 2 {
        fmt.Printf("D = %.2f\t", d)
    }
    if nomer != 3 {
        fmt.Printf("L = %.2f\t", l)
    }
    if nomer != 4 {
        fmt.Printf("S = %.2f\t", s)
    }
}
```

Case 13

Элементҳои секунҷаи росткунҷаи баробарпаҳлӯ ба тариқи зайл рақамгузорӣ карда шудаанд: 1—катет a, 2—гипотенуза c=a*sqrt(2), 3—баландӣ h, ки ба гипотенуза фароварда шудааст (h=c/2), 4—масоҳат S=c*h/2. Рақами яке аз ин элементҳо ва қимати он дода шудаанд. Қиматҳои

элементҳои боқимондаи секунҷаи мазкурро (бо ҳамон тартиб) хорич кунед.

Elements of a right isosceles triangle are numbered as: 1 — leg a, 2 — hypotenuse $c = a \cdot (2)^{1/2}$, 3 — altitude h drawn onto hypotenuse ($h = c/2$), 4 — area $S = c \cdot h/2$. The order number of one element and its value (as a real number) are given. Output values of other elements in the same order.

```
package main

import (
    "fmt"
    "math"
)

func main() {
    var nomer int
    fmt.Scan(&nomer)
    switch nomer {
        case 1: fmt.Print("a = ")
        case 2: fmt.Print("c = ")
        case 3: fmt.Print("h = ")
        case 4: fmt.Print("S = ")
    }
    var value, a, c, h, s float64
    fmt.Scan(&value)
    switch nomer {
        case 1: a = value
            c = a * math.Sqrt(2);
            h = c / 2;
            s = c * h / 2;
        case 2: c = value
            a = c / math.Sqrt(2);
            h = c / 2;
            s = c * h / 2;
        case 3: h = value
            a = 2 * h / math.Sqrt(2);
            c = a * math.Sqrt(2);
            s = c * h / 2;
        case 4: s = value
            h = math.Sqrt(s);
            c = 2 * h;
            a = c / math.Sqrt(2);
    }
    if nomer != 1 {
        fmt.Printf("a = %.2f\t", a)
    }
    if nomer != 2 {
        fmt.Printf("c = %.2f\t", c)
    }
    if nomer != 3 {
        fmt.Printf("h = %.2f\t", h)
    }
    if nomer != 4 {
        fmt.Printf("S = %.2f\t", s)
```

```
    }
}
```

Case 14

Элементҳои секунҷаи росткунҷа ба тариқи зайл рақамгузорӣ карда шудаанд: 1 — тараф a, 2 — радиуси давраи дохилӣ R_1 (R_1=a*sqrt(3)/6), 3 — радиуст давраи берунӣ R_2 (R_2=2*R_1), 4 — масоҳат S=a^2*sqrt(3)/4. Рақами яке аз ин элементҳо ва қимати он дода шудаанд. Қиматҳои элементҳои боқимондаи секунҷаи мазкурро (бо ҳамон тартиб) хориҷ кунед.

Elements of an equilateral triangle are numbered as: 1 — side a, 2 — radius R_1 of inscribed circle ($R_1 = a\cdot(3)^{1/2}/6$), 3 — radius R_2 of circumscribed circle ($R_2 = 2\cdot R_1$), 4 — area $S = a^2\cdot(3)^{1/2}/4$. The order number of one element and its value (as a real number) are given. Output values of other elements in the same order.

```
package main

import (
    "fmt"
    "math"
)

func main() {
    var nomer int
    fmt.Scan(&nomer)
    switch nomer {
        case 1: fmt.Print("a = ")
        case 2: fmt.Print("R1 = ")
        case 3: fmt.Print("R2 = ")
        case 4: fmt.Print("S = ")
    }
    var value, a, r1, r2, s float64
    fmt.Scan(&value)
    switch nomer {
        case 1: a = value
            r1 = a * math.Sqrt(3) / 6;
            r2 = 2 * r1;
            s = a*a * math.Sqrt(3) / 4;
        case 2: r1 = value
            r2 = 2 * r1;
            a = 6 * r1 / math.Sqrt(3);
            s = a*a * math.Sqrt(3) / 4;
        case 3: r2 = value
            r1 = r2 / 2;
            a = 6 * r1 / math.Sqrt(3);
            s = a*a * math.Sqrt(3) / 4;
        case 4: s = value
```

```
            a = math.Sqrt( 4 * s / math.Sqrt(3) );
            r1 = a * math.Sqrt(3) / 6;
            r2 = 2 * r1;
    }
    if nomer != 1 {
        fmt.Printf("a = %.2f\t", a)
    }
    if nomer != 2 {
        fmt.Printf("R1 = %.2f\t", r1)
    }
    if nomer != 3 {
        fmt.Printf("R2 = %.2f\t", r2)
    }
    if nomer != 4 {
        fmt.Printf("S = %.2f\t", s)
    }
}
```

Case 15

Ба аломатҳои қартаҳои бозӣ рақамҳои тартибӣ бахшида шудаанд: 1 - дил, 2 - пашша, 3 - хишт, 4 - таппон. Ба обрӯи қартаҳои аз даҳ боло рақамҳои зерин бахшида шудаанд: 11 - валет, 12 - дама, 13 - шоҳ, 14 - туз. Ду ададҳои бутун дода шудаанд: N - обрӯ (6<=N<=14) ва M - аломати қарта (1<=M<=4). Номи қартаи мувофиқро дар намуди "шаши хишт", "дамаи таппон", "тузи пашша" ва ғ. хориҷ кунед.

The suits of playing cards are numbered as: 1 — spades, 2 — clubs, 3 — diamonds, 4 — hearts. Card values "Jack", "Queen", "King", "Ace" are numbered as 11, 12, 13, 14 respectively. A card value *N* (as an integer in the range 6 to 14) and a suit *M* (as an integer in the range 1 to 4) are given. Output the card description as: "six of diamonds", "queen of spades", etc.

```
package main

import "fmt"

func main() {
    var n, m int
    fmt.Print("N = ")
    fmt.Scan(&n)
    fmt.Print("M = ")
    fmt.Scan(&m)
    var result string = ""
    switch n {
        case 6: result += "six"
        case 7: result += "seven"
        case 8: result += "eight"
```

```
        case 9: result += "nine"
        case 10: result += "ten"
        case 11: result += "jack"
        case 12: result += "queen"
        case 13: result += "king"
        case 14: result += "ace"
    }
    result += " of "
    switch m {
        case 1: result += "spades"
        case 2: result += "clubs"
        case 3: result += "diamonds"
        case 4: result += "hearts"
    }
    fmt.Println(result)
}
```

Case 16

Адади бутун дар фосилаи 20–69, ки синнусолро муайян мекунад (бо солҳо) дода шудааст. Сатр-эзоҳи синнусоли нишондодашуда хориҷ кунед, ки мувофиқоии дурусти калимаи русии «год»-ро таъмин мекунад, масалан: 20 — «двадцать лет», 32 — «тридцать два года», 41 — «сорок один год».

Given an age in years (as an integer in the range 20 to 69), output its alphabetic equivalent as: "twenty years", "thirty-two years", "forty-one years", etc.

```
package main

import "fmt"

func main() {
    var year int
    fmt.Print("year [20-69] = ")
    fmt.Scanf("%d", &year)
    dahi := year / 10
    vohid := year % 10
    var result string = ""
    switch dahi {
        case 2: result += "twenty"
        case 3: result += "thirty"
        case 4: result += "forty"
        case 5: result += "fifty"
        case 6: result += "sixty"
    }
    if vohid != 0 {
        result += "-"
    }
    switch vohid {
        case 1: result += "one"
```

```
        case 2: result += "two"
        case 3: result += "three"
        case 4: result += "four"
        case 5: result += "five"
        case 6: result += "six"
        case 7: result += "seven"
        case 8: result += "eight"
        case 9: result += "nine"
    }
    result += " years"
    fmt.Println(result)
}
```

Case 17

Адади бутун дар фосилаи 10–40, ки миқдори супоришҳои дарсиро аз рӯи баъзе мавзӯъҳо муайян мекунад, дода шудааст. Сатр-эзоҳи миқдори супоришҳои нишондодашударо хориҷ кунед, ки мувофиқоии дурусти ададро бо ибораи русии «учебное задание» таъмин мекунад, масалан: 18 — «восемнадцать учебных заданий», 23 — «двадцать три учебных задания», 31 — «тридцать одно учебное задание».

Given an order number of some training task (as an integer in the range 10 to 40), output its alphabetic equivalent as: "the eighteenth task", "the twenty-third task", "the thirtieth task", etc.

```
package main

import "fmt"

func main() {
    var number int
    fmt.Print("number [10-40] = ")
    fmt.Scanf("%d", &number)
    dahi := number / 10
    vohid := number % 10
    var result string = "the "
    switch dahi {
        case 1:
            switch vohid {
                case 0: result += "tenth"
                case 1: result += "eleventh"
                case 2: result += "twelfth"
                case 3: result += "thirteenth"
                case 4: result += "fourteenth"
                case 5: result += "fifteenth"
                case 6: result += "sixteenth"
                case 7: result += "seventeenth"
                case 8: result += "eighteenth"
```

```
                case 9: result += "nineteenth"
            }
        case 2:
            if vohid == 0 {
                result += "twentieth"
            } else {
                result += "twenty-"
            }
        case 3:
            if vohid == 0 {
                result += "thirtieth"
            } else {
                result += "thirty-"
            }
        case 4:
            if vohid == 0 {
                result += "fortieth"
            } else {
                result += "forty-"
            }
    }
    if dahi != 1 {
        switch vohid {
            case 1: result += "first"
            case 2: result += "second"
            case 3: result += "third"
            case 4: result += "fourth"
            case 5: result += "fifth"
            case 6: result += "sixth"
            case 7: result += "seventh"
            case 8: result += "eighth"
            case 9: result += "ninth"
        }
    }
    result += " task"
    fmt.Println(result)
}
```

Case 18

Ададӣ бутун дар фосилаи 100-999 дода шудааст. Сатр-эзоҳи ададӣ мазкурро хориҷ кунед, масалан: 256 - «дусаду панҷоҳу шаш», 814 - «ҳаштсаду чордаҳ».

Given an integer in the range 100 to 999, output its alphabetic equivalent. For example, 100 — "one hundred", 256 — "two hundred and fifty-six", 814 — "eight hundred and fourteen", 901 — "nine hundred and one".

```
package main

import "fmt"

func main() {
    var number int
```

```
fmt.Print("number [100-999] = ")
fmt.Scanf("%d", &number)
sadi := number / 100
dahi := number / 10 % 10
vohid := number % 10
var result string = ""
switch sadi {
    case 1: result += "one hundred "
    case 2: result += "two hundred "
    case 3: result += "three hundred "
    case 4: result += "four hundred "
    case 5: result += "five hundred "
    case 6: result += "six hundred "
    case 7: result += "seven hundred "
    case 8: result += "eight hundred "
    case 9: result += "nine hundred "
}
if dahi != 0 || vohid != 0 {
    result += "and "
}
switch dahi {
    case 1:
        switch vohid {
            case 0: result += "ten"
            case 1: result += "eleven"
            case 2: result += "twelve"
            case 3: result += "thirteen"
            case 4: result += "fourteen"
            case 5: result += "fifteen"
            case 6: result += "sixteen"
            case 7: result += "seventeen"
            case 8: result += "eighteen"
            case 9: result += "nineteen"
        }
    case 2: result += "twenty"
    case 3: result += "thirty"
    case 4: result += "forty"
    case 5: result += "fifty"
    case 6: result += "sixty"
    case 7: result += "seventy"
    case 8: result += "eighty"
    case 9: result += "ninety"
}
if dahi != 1 {
    if dahi > 1 {
        if vohid != 0 {
            result += "-"
        } else {
            result += " "
        }
    }
    switch vohid {
        case 1: result += "one"
        case 2: result += "two"
        case 3: result += "three"
        case 4: result += "four"
        case 5: result += "five"
        case 6: result += "six"
        case 7: result += "seven"
        case 8: result += "eight"
        case 9: result += "nine"
    }
```

```
    }
    fmt.Println(result)
}
```

Case 19

Дар тақвими шарқӣ даври 60-сола қабул гардидааст, ки аз зердаврҳои 12-сола иборат аст ва номҳои рангҳоро ифода мекунанд: сабз, сурх, зард, сафед ва сиёҳ. Дар ҳар як зердавр солҳо номҳои ҳайвонҳоро доро мебошанд: муш, гов, паланг, харгӯш, аждаҳор, мор, асп, гӯсфанд, маймун, мурғ, саг ва хук. Аз рӯи рақами сол номи он солро муайян кунед, агар соли 1984 — оғози даври: «соли муши сабз» бошад.

One of the Asian calendars uses 60-years periods divided into 12-years cycles, which are associated with a color: green, red, yellow, white, black. Each year in a cycle is connected with some animal: rat, cow, tiger, hare, dragon, snake, horse, sheep, monkey, hen, dog, pig. Given some year (as positive integer), output its name provided that 1984 is "The Green Rat`s year".

```
package main

import "fmt"

func main() {
    var year int
    const BEGINPOINT = 1984
    fmt.Scanf("%d", &year)
    var result string = "The "
    var farq, color, animal int
    if year >= BEGINPOINT {
        farq = year - BEGINPOINT
        color = farq % 60
        switch {
            case color < 12: result += "Green "
            case color < 24: result += "Red "
            case color < 36: result += "Yellow "
            case color < 48: result += "White "
            default: result += "Black "
        }
        animal = farq % 12
    } else {
        farq = BEGINPOINT - year - 1
        color = farq % 60
        switch {
            case color < 12: result += "Black "
            case color < 24: result += "White "
```

```
            case color < 36: result += "Yellow "
            case color < 48: result += "Red "
            default: result += "Green "
        }
        animal = farq % 12 + 12
    }
    switch animal {
        case 0, 23: result += "Rat"
        case 1, 22: result += "Cow"
        case 2, 21: result += "Tiger"
        case 3, 20: result += "Hare"
        case 4, 19: result += "Dragon"
        case 5, 18: result += "Snake"
        case 6, 17: result += "Horse"
        case 7, 16: result += "Sheep"
        case 8, 15: result += "Monkey"
        case 9, 14: result += "Hen"
        case 10, 13: result += "Dog"
        case 11, 12: result += "Pig"
    }
    result += "'s year"
    fmt.Println(result)
}
```

Case 20

Ду ададҳои бутун дода шудаанд: D(рӯз) ва M(моҳ), ки санаи дурустро муайян мекунанд. Аломати бурҷи дувоздаҳгонаи ба ин сана мувофиқро хориҷ кунед: «Далв»(20.1–18.2), «Ҳут»(19.2–20.3), «Ҳамал»(21.3–19.4), «Савр»(20.4–20.5), «Ҷавзо»(21.5–21.6), «Саратон»(22.6–22.7), «Асад»(23.7–22.8), «Сунбула»(23.8–22.9), «Мизон»(23.9–22.10), «Ақраб»(23.10–22.11), «Қавс»(23.11–21.12), «Ҷадӣ»(22.12–19.1).

Given two integers *D* (day) and *M* (month) that represent a correct date, output the zodiacal name corresponding to this date: "Aquarius" 20.1–18.2, "Pisces" 19.2–20.3, "Aries" 21.3–19.4, "Taurus" 20.4–20.5, "Gemini" 21.5–21.6, "Cancer" 22.6–22.7, "Leo" 23.7–22.8, "Virgo" 23.8–22.9, "Libra" 23.9–22.10, "Scorpio" 23.10–22.11, "Sagittarius" 23.11–21.12, "Capricorn" 22.12–19.1.

```
package main

import "fmt"

func main() {
```

```
var d, m int
fmt.Print("D = ")
fmt.Scan(&d)
fmt.Print("M = ")
fmt.Scan(&m)
var zodiacalName string = ""
switch m {
    case 1:
        switch {
            case d >= 20: zodiacalName = "Aquarius"
            default: zodiacalName = "Capricorn"
        }
    case 2:
        switch {
            case d >= 19: zodiacalName = "Pisces"
            default: zodiacalName = "Aquarius"
        }
    case 3:
        switch {
            case d >= 21: zodiacalName = "Aries"
            default: zodiacalName = "Pisces"
        }
    case 4:
        switch {
            case d >= 20: zodiacalName = "Taurus"
            default: zodiacalName = "Aries"
        }
    case 5:
        switch {
            case d >= 21: zodiacalName = "Gemini"
            default: zodiacalName = "Taurus"
        }
    case 6:
        switch {
            case d >= 22: zodiacalName = "Cancer"
            default: zodiacalName = "Gemini"
        }
    case 7:
        switch {
            case d >= 23: zodiacalName = "Leo"
            default: zodiacalName = "Cancer"
        }
    case 8:
        switch {
            case d >= 23: zodiacalName = "Virgo"
            default: zodiacalName = "Leo"
        }
    case 9:
        switch {
            case d >= 23: zodiacalName = "Libra"
            default: zodiacalName = "Virgo"
        }
    case 10:
        switch {
            case d >= 23: zodiacalName = "Scorpio"
            default: zodiacalName = "Libra"
        }
    case 11:
        switch {
            case d >= 23: zodiacalName = "Sagittarius"
            default: zodiacalName = "Scorpio"
        }
```

```
        case 12:
            switch {
                case d >= 22: zodiacalName = "Capricorn"
                default: zodiacalName = "Sagittarius"
            }
    }
    fmt.Println(zodiacalName)
}
```

For 1

Ду ададҳои бутуни K ва N (N>0) дода шудаанд. Адади K-ро N маротиба хориҷ кунед.

Given integers *K* and *N* (*N* > 0), output the number *K* *N* times.

```
package main

import "fmt"

func main() {
    var k, n int
    fmt.Print("K = ")
    fmt.Scan(&k)
    fmt.Print("N = ")
    fmt.Scan(&n)
    for i := 0; i < n; i++ {
        fmt.Printf("%d\t", k)
    }
}
```

For 2

Ду ададҳои бутуни A ва B (A<B) дода шудаанд. Ҳамаи ададҳои бутуни дар байни ададҳои A ва B ҷойгирбударо бо тартиби афзуншавӣ (дар якҷоягӣ бо худи ададҳои A ва B), ҳамчунин миқдори ин ададҳоро N хориҷ кунед.

Given two integers *A* and *B* (*A* < *B*), output in ascending order all integers in the range *A* to *B* (including *A* and *B*). Also output the amount *N* of these integers.

```
package main

import "fmt"

func main() {
    var a, b int
```

```
    fmt.Print("A = ")
    fmt.Scan(&a)
    fmt.Print("B = ")
    fmt.Scan(&b)
    n := 0
    for i := a; i <= b; i++ {
        fmt.Printf("%d\t", i)
        n++;
    }
    fmt.Printf("\nN = %d\n", n)
}
```

For 3

Ду ададҳои бутуни A ва B (A<B) дода шудаанд. Ҳамаи ададҳои бутуни дар байни ададҳои A ва B ҷойгирбударо бо тартиби камшавӣ (ба истиснои ададҳои A ва B), ҳамчунин миқдори ин ададҳоро N хориҷ кунед.

Given two integers *A* and *B* (*A* < *B*), output in descending order all integers in the range *A* to *B* (excluding *A* and *B*). Also output the amount *N* of these integers.

```
package main

import "fmt"

func main() {
    var a, b int
    fmt.Print("A = ")
    fmt.Scan(&a)
    fmt.Print("B = ")
    fmt.Scan(&b)
    n := 0
    for i := b-1; i > a; i-- {
        fmt.Printf("%d\t", i)
        n++
    }
    fmt.Printf("\nN = %d\n", n)
}
```

For 4

Адади ҳақиқӣ — нархи 1 кг қанд дода шудааст. Нархи 1,2,…,10 кг қандро хориҷ кунед.

Given the price of 1 kg of sweets (as a real number), output the cost of 1, 2, …, 10 kg of these sweets.

```
package main
```

```
import "fmt"

func main() {
    var oneKgPrice float32
    fmt.Print("price of one kg:  ")
    fmt.Scanf("%f", &oneKgPrice)
    price := oneKgPrice
    for i := 1; i <= 10; i++ {
        fmt.Printf("%.2f\t", price)
        price += oneKgPrice
    }
}
```

For 5

Адади ҳақиқӣ — нархи 1 кг қанд дода шудааст. Нархи 0.1, 0.2, …, 1 кг қандро хориҷ кунед.

Given the price of 1 kg of sweets (as a real number), output the cost of 0.1, 0.2, …, 1 kg of these sweets.

```
package main

import "fmt"

func main() {
    var oneKgPrice, price float64
    fmt.Print("price of one kg:\t")
    fmt.Scanf("%f", &oneKgPrice)
    for i := 0.1; i <= 1; i += 0.1 {
        price = i * oneKgPrice
        fmt.Printf("%.2f\t", price)
    }
}
```

For 6

Адади ҳақиқӣ - нархи 1 кг қанд дода шудааст. Нархи 1.2, 1.4, .. , 2 кг қандро хориҷ кунед.

Given the price of 1 kg of sweets (as a real number), output the cost of 1.2, 1.4, …, 2 kg of these sweets.

```
package main

import "fmt"

func main() {
    var oneKgPrice, price float64
    fmt.Print("price of one kg:\t")
    fmt.Scanf("%f", &oneKgPrice)
```

```
    for i := 1.2; i <= 2; i += 0.2 {
        price = i * oneKgPrice
        fmt.Printf("%.2f\t", price)
    }
}
```

For 7

Ду ададҳои бутуни A ва B (A<B) дода шудаанд. Суммаи ҳамаи ададҳои бутунро аз A то B дар якҷоягӣ ёбед.

Given two integers *A* and *B* (*A* < *B*), find the sum of all integers in the range *A* to *B* inclusive.

```
package main

import "fmt"

func main() {
    var a, b int
    fmt.Print("A = ")
    fmt.Scan(&a)
    fmt.Print("B = ")
    fmt.Scan(&b)
    sum := 0
    for i := a; i <= b; i++ {
        sum += i
    }
    fmt.Printf("sum = %d\n", sum)
}
```

For 8

Ду ададҳои бутуни A ва B (A<B) дода шудаанд. Ҳосилизарби ҳамаи ададҳои бутунро аз A то B дар якҷоягӣ ёбед.

Given two integers *A* and *B* (*A* < *B*), find the product of all integers in the range *A* to *B* inclusive.

```
package main

import "fmt"

func main() {
    var a, b int
    fmt.Print("A = ")
    fmt.Scan(&a)
    fmt.Print("B = ")
    fmt.Scan(&b)
    mul := 1
```

```
    for i := a; i <= b; i++ {
        mul *= i
    }
    fmt.Printf("multiplication = %d\n", mul)
}
```

For 9

Ду ададҳои бутуни A ва B (A<B) дода шудаанд. Суммаи квадратҳои ҳамаи ададҳои бутунро аз A то B дар якҷоягӣ ёбед.

Given two integers *A* and *B* (*A* < *B*), find the sum of squares of all integers in the range *A* to *B* inclusive.

```
package main

import "fmt"

func main() {
    var a, b int
    fmt.Print("A = ")
    fmt.Scan(&a)
    fmt.Print("B = ")
    fmt.Scan(&b)
    sum := 0
    for i := a; i <= b; i++ {
        sum += i*i
    }
    fmt.Printf("result = %d\n", sum)
}
```

For 10

Ададӣ бутуни N(>0) дода шудааст. Суммаи 1+1/2+1/3+…+1/N (ададӣ ҳақиқӣ)-ро ёбед.

Given an integer *N* (> 0), find the value of a following sum (as a real number): 1 + 1/2 + 1/3 + … + 1/*N*

```
package main

import "fmt"

func main() {
    var n int
    fmt.Print("N = ")
    fmt.Scanf("%d", &n)
    var sum float64 = 0
    for i := 1; i <= n; i++ {
        sum += 1 / float64(i)
```

```
    }
    fmt.Printf("sum = %.6f\n", sum)
}
```

For 11

Ададu бутуни N(>0) дода шудааст. Суммаи $N^2+(N+1)^2+(N+2)^2+\ldots+(2\cdot N)^2$(ададu бутун)-ро ёбед.

Given an integer N (> 0), find the value of a following sum (as an integer): $N^2 + (N + 1)^2 + (N + 2)^2 + \ldots + (2\cdot N)^2$

```
package main

import "fmt"

func main() {
    var n int
    fmt.Print("N = ")
    fmt.Scanf("%d", &n)
    sum := 0
    for i := 0; i <= n; i++ {
        sum += (n + i) * (n + i)
    }
    fmt.Printf("result = %d\n", sum)
}
```

For 12

Ададu бутуни N(>0) дода шудааст. ҲосилиЗарби 1.1 · 1.2 · 1.3 · ... (N зарбкунанда)-ро ёбед.

Given an integer N (> 0), find the value of a following product of N factors: 1.1 · 1.2 · 1.3 · …

```
package main

import "fmt"

func main() {
    var n int
    fmt.Print("N = ")
    fmt.Scanf("%d", &n)
    var mul, number float64 = 1, 1
    for i := 0; i < n; i++ {
        number += 0.1
        mul *= number
    }
    fmt.Printf("result = %.6f\n", mul)
}
```

For 13

Ададӣ бутуни N (>0) дода шудааст. Қимати ифодаи 1.1 - 1.2 + 1.3 - ... (N ҷамъшавандаҳо, аломатҳои иваз шуда меистанд)-ро ёбед. Операторӣ шартиро истифода набаред.

Given an integer *N* (> 0), find the value of the following expression of *N* terms with alternating signs:
1.1 − 1.2 + 1.3 − … Do not use conditional statements.

```
package main

import "fmt"

func main() {
    var n int
    fmt.Print("N = ")
    fmt.Scanf("%d", &n)
    var result, number float64 = 0, 1.1
    alomat := 1
    for i := 0; i < n; i++ {
        result += float64(alomat) * number
        number += 0.1
        alomat *= -1
    }
    fmt.Printf("result = %.2f\n", result)
}
```

For 14

Ададӣ бутуни N (>0) дода шудааст. Квадрати ададӣ мазкурро ёбед, барои ҳисобкунии он формулаи зеринро истифода баред: N^2=1+3+5+...+(2·N-1). Пас аз ҳамроҳкунии ҳар як ҷамъшаванда ба сумма қимати ҷории суммаро хориҷ кунед (дар натиҷа квадратҳои ҳамаи ададҳои бутун аз 1 то N хориҷ карда мешаванд).

Given an integer *N* (> 0), compute N^2 by means of the formula $N^2 = 1 + 3 + 5 + \ldots + (2 \cdot N - 1)$Output the value of the sum after addition of each term. As a result, squares of all integers in the range 1 to *N* will be output.

```
package main

import "fmt"
```

```
func main() {
    var n int
    fmt.Print("N = ")
    fmt.Scanf("%d", &n)
    sum := 0
    for i := 1; i <= n; i++ {
        sum += 2 * i - 1
        fmt.Printf("%d\t", sum)
    }
}
```

For 15

Адади ҳақиқии A ва адади бутуни N(>0) дода шудаанд. A дар дараҷаи N-ро ёбед: A^N=A*A* ...*A (ададҳои A - N маротиба бо ҳам зарб мешаванд).

Given a real number *A* and an integer *N* (> 0), find *A* raised to the power *N* (i. e., the product of *N* values of *A*): A^N = *A*·*A*· … ·*A*

```
package main

import "fmt"

func main() {
    var (
        a float64
        n int
        degree float64 = 1
    )
    fmt.Print("A = ")
    fmt.Scan(&a)
    fmt.Print("N = ")
    fmt.Scan(&n)
    for i := 0; i < n; i++ {
        degree *= a;
    }
    fmt.Printf("degree = %.2f\n", degree)
}
```

For 16

Адади ҳақиқии A ва адади бутуни N(>0) дода шудаанд. Бо истифодабарии як давр ҳамаи дараҷаҳои бутуни адади A-ро аз 1 то N хориҷ кунед.

A real number *A* and an integer *N* (> 0) are given. Using one loop-statement compute and output powers A^K for all integer exponents *K* in the range 1 to *N*.

```
package main

import "fmt"

func main() {
    var (
        a float64
        n int
        degree float64 = 1
    )
    fmt.Print("A = ")
    fmt.Scan(&a)
    fmt.Print("N = ")
    fmt.Scan(&n)
    for i := 1; i <= n; i++ {
        degree *= a;
        fmt.Printf("%.2f\t", degree)
    }
}
```

For 17

Адади ҳақиқии A ва ададди бутуни N(>0) дода шудаанд. Бо истифодабарии як давр суммаи $1+A+A^2+A^3+...+A^N$-ро ёбед.

A real number *A* and an integer *N* (> 0) are given. Using one loop-statement compute the sum $1 + A + A^2 + A^3 + \ldots + A^N$

```
package main

import "fmt"

func main() {
    var (
        a float64
        n int
        sum float64
        degree float64 = 1
    )
    fmt.Print("A = ")
    fmt.Scan(&a)
    fmt.Print("N = ")
    fmt.Scan(&n)
    for i := 0; i <= n; i++ {
        sum += degree
        degree *= a;
    }
    fmt.Printf("result = %.2f\n", sum)
}
```

For 18

Адади ҳақиқии A ва адади бутуни N(>0) дода шудаанд. Бо истифодабарии як давр қимати ифодаи $1–A+A^2–A^3+ ... + (–1)^N*A^N$-ро ёбед. Операторӣ шартиро истифода набаред.

A real number *A* and an integer *N* (> 0) are given. Using one loop-statement compute the expression
$1 − A + A^2 − A^3 + … + (−1)^N·A^N$Do not use conditional statements.

```
package main

import "fmt"

func main() {
    var (
        a float64
        n int
        result float64
        degree float64 = 1
        alomat int = 1
    )
    fmt.Print("A = ")
    fmt.Scan(&a)
    fmt.Print("N = ")
    fmt.Scan(&n)
    for i := 0; i <= n; i++ {
        result += float64(alomat) * degree
        degree *= a
        alomat *= -1
    }
    fmt.Printf("result = %.2f\n", result)
}
```

For 19

Адади бутуни N(>0) дода шудааст. ҲосилиЗарби N!=1*2*...*N (N–факториал)-ро ёбед. Барои роҳ надодан ба пуршавии қатори ададҳои бутун ин зарбкуниро ба воситаи тағйирёбандаи ҳақиқӣ ҳисоб кунед ва онро чун адади ҳақиқӣ хориҷ кунед.

Given an integer *N* (> 0), find the value of a following product:*N*! = 1·2·…·*N*(*N–factorial*). To avoid the integer

overflow, compute the product using a real variable and output the result as a real number.

```
package main

import "fmt"

func main() {
    var n int
    fmt.Print("N = ")
    fmt.Scanf("%d", &n)
    var fact float64 = 1
    for i := 2; i <= n; i++ {
        fact *= float64(i)
    }
    fmt.Printf("%d! = %.2f\n", n, fact)
}
```

For 20

Ададӣ бутуни N(>0) дода шудааст. Бо истифодабарии як давр суммаи 1!+2!+3!+ ... +N!(ифодаи N! — N–факториал — ҳосилизарби ҳамаи ададҳои бутунро аз 1 то N ифода мекунад: N!=1*2* ... *N)-ро ёбед. Барои роҳ надодан ба пуршавии қатори ададҳои бутун ҳисобкуниро бо ёрии тағйирёбандаҳои ҳақиқӣ гузаронед ва натиҷаро чун ададӣ ҳақиқӣ хориҷ кунед.

An integer *N* (> 0) is given. Using one loop-statement compute the sum 1! + 2! + 3! + … + *N*!, where *N*! (*N–factorial*) is the product of all integers in the range 1 to *N*: $N! = 1 \cdot 2 \cdot \ldots \cdot N$. To avoid the integer overflow, compute the sum using real variables and output the result as a real number.

```
package main

import "fmt"

func main() {
    var n int
    fmt.Print("N = ")
    fmt.Scanf("%d", &n)
    var sum, fact float64 = 0, 1
    for i := 1; i <= n; i++ {
        fact *= float64(i)
        sum += fact
    }
    fmt.Printf("result = %.2f\n", sum)
}
```

While 1

Ададҳои мусбии A ва B (A>B) дода шудаанд. Дар порчаи дарозиаш A миқдори калонтарини имконпазири порчаҳои дарозиашон B ҷой дода шудаанд. Амалҳои зарб ва тақсимро истифода набурда, дарозии қисми банднабудаи порчаи A-ро ёбед.

Two positive real numbers *A* and *B* (*A* > *B*) are given. A segment of length *A* contains the greatest possible amount of segments of length *B* (without overlaps). Not using multiplication and division, find the length of unused part of the segment *A*.

```
package main

import "fmt"

func main() {
    var a, b float32
    fmt.Print("A = ")
    fmt.Scan(&a)
    fmt.Print("B = ")
    fmt.Scan(&b)
    freeSpace := a
    for freeSpace >= b {
        freeSpace -= b
    }
    fmt.Printf("free space: %.2f\n", freeSpace)
}
```

While 2

Ададҳои мусбии A ва B (A>B) дода шудаанд. Дар порчаи дарозиаш A миқдори калонтарини имконпазири порчаҳои дарозиашон B ҷой дода шудаанд. Амалҳои зарб ва тақсимро истифода набурда, миқдори порчаҳои B-ро, ки дар порчаи A ҷойгиранд, ёбед.

Two positive real numbers *A* and *B* (*A* > *B*) are given. A segment of length *A* contains the greatest possible amount of

segments of length *B* (without overlaps). Not using multiplication and division, find the amount of segments *B*, which are placed on the segment *A*.

```
package main

import "fmt"

func main() {
    var a, b float32
    fmt.Print("A = ")
    fmt.Scan(&a)
    fmt.Print("B = ")
    fmt.Scan(&b)
    porchaho := 0
    for a >= b {
        a -= b
        porchaho++
    }
    fmt.Printf("porchaho = %d\n", porchaho)
}
```

While 3

Ададҳои мусбии бутуни N ва K дода шудаанд. Фақат амалҳои ҷамъ ва тарҳро истифода бурда, қимати тақсими бутуни N-ро ба K, ҳамчунин боқимондаи ин тақсимро ёбед.

Two positive integers *N* and *K* are given. Using addition and subtraction only, find a quotient of the integer division *N* on *K* and also a remainder after this division.

```
package main

import "fmt"

func main() {
    var n, k int
    fmt.Print("N = ")
    fmt.Scan(&n)
    fmt.Print("K = ")
    fmt.Scan(&k)
    div := 0
    for n >= k {
        n -= k
        div++
    }
    mod := n
    fmt.Printf("div = %d\nmod = %d\n", div, mod)
}
```

While 4

Ададӣ бутуни N(>0) дода шудааст. Агар он дараҷаи ададӣ 3 бошад, пас True-ро хориҷ кунед, агар набошад — False-ро хориҷ кунед.

An integer *N* (> 0) is given. If it equals 3 raised to some integer power then output true, otherwise output false.

```
package main

import "fmt"

func main() {
    var n int
    fmt.Print("N = ")
    fmt.Scanf("%d", &n)
    degree3 := 1
    for degree3 < n {
        degree3 *= 3
    }
    result := degree3 == n
    fmt.Printf("result = %t\n", result)
}
```

While 5

Ададӣ бутуни N(>0), ки якчанд дараҷаи ададӣ 2 аст, дода шудааст: $N = 2^K$. Ададӣ бутуни K — нишондиҳандаи ин дараҷаро ёбед.

Given an integer *N* (> 0) that equals 2 raised to some integer power: $N = 2^K$, find the exponent *K* of the power.

```
package main

import "fmt"

func main() {
    var n int
    fmt.Print("N = ")
    fmt.Scanf("%d", &n)
    var degree, k = 1, 0
    for degree < n {
        k++
        degree *= 2
    }
    fmt.Printf("K = %d\n", k)
}
```

While 6

Адади бутуни N(>0) дода шудааст. Фактириали дукаратаи N-ро ёбед: N!!=N·(N–2)·(N–4)·… (зарбшавандаи охирин ба 2 баробар аст, агар N — ҷуфт бошад ва ба 1 баробар аст, агар N — тоқ бошад). Барои роҳ надодан ба пуршавии қатори ададҳои бутун ин зарбкуниро бо ёрии тағйирёбандаи ҳақиқӣ иҷро кунед ва онро чун адади ҳақиқӣ хориҷ кунед.

Given an integer N (> 0), compute the *double factorial of N*: $N!! = N\cdot(N-2)\cdot(N-4)\cdot\ldots$,where the last factor equals 2 if N is an even number, and 1 otherwise. To avoid the integer overflow, compute the double factorial using a real variable and output the result as a real number.

```
package main

import "fmt"

func main() {
    var n int
    fmt.Print("N = ")
    fmt.Scanf("%d", &n)
    var fact2, tmp float64 = 1, float64(n)
    for tmp > 0 {
        fact2 *= tmp
        tmp -= 2
    }
    fmt.Printf("%d!! = %.2f\n", n, fact2)
}
```

While 7

Адади бутуни N(>0) дода шудааст. Адади бутуни мусбии хурдтарини K-ро ёбед, ки квадрати он аз N калон аст: K^2>N. Функсияи азрешабарории квадратиро истифода набаред.

Given an integer N (> 0), find the smallest positive integer K such that its square is greater than N: $K^2 > N$. Do not use the operation of extracting a root.

```
package main

import "fmt"

func main() {
    var n int
    fmt.Print("N = ")
```

```
    fmt.Scanf("%d", &n)
    k := 1
    for k * k <= n {
        k++
    }
    fmt.Printf("K = %d\n", k)
}
```

While 8

Адади бутуни N(>0) дода шудааст. Адади бутуни калонтарини K-ро ёбед, ки квадрати он аз N калон нест: $K^2 \leq N$. Функсияи азрешабарории квадратиро истифода набаред.

Given an integer N (> 0), find the largest integer K such that its square is not greater than N: $K^2 \leq N$. Do not use the operation of extracting a root.

```
package main

import "fmt"

func main() {
    var n int
    fmt.Print("N = ")
    fmt.Scanf("%d", &n)
    k := 1
    for k * k <= n {
        k++
    }
    k--
    fmt.Printf("K = %d\n", k)
}
```

While 9

Адади бутуни N(>1) дода шудааст. Адади бутуни хурдтарини K-ро ёбед, ки нобаробарии зеринро ичро мекунад $3^K > N$.

Given an integer N (> 1), find the smallest integer K such that the inequality $3^K > N$ is fulfilled.

```
package main

import "fmt"

func main() {
```

```
    var n int
    fmt.Print("N = ")
    fmt.Scanf("%d", &n)
    k, degree := 0, 1
    for degree <= n {
        degree *= 3
        k++
    }
    fmt.Printf("K = %d\n", k)
}
```

While 10

Ададӣ бутуни N(>1) дода шудааст. Ададӣ бутуни калонтарини K-ро ёбед, ки нобаробарии зеринро иҷро мекунад $3^K<N$.

Given an integer N (> 1), find the largest integer K such that the inequality $3^K < N$ is fulfilled.

```
package main

import "fmt"

func main() {
    var n int
    fmt.Print("N = ")
    fmt.Scanf("%d", &n)
    k, degree := 0, 1
    for degree < n {
        degree *= 3
        k++
    }
    k--
    fmt.Printf("K= %d\n", k)
}
```

While 11

Ададӣ бутуни N (>1) дода шудааст. Аз ададҳои бутуни K хурдтаринашро хориҷ кунед, ки барои он суммаи 1 + 2 + … + K калон ё баробари N мешавад ва ҳамчунин худи ин суммаро низ хориҷ кунед.

An integer N (> 1) is given. Find the smallest integer K such that the sum 1 + 2 + … + K is greater than or equal to N. Output K and the corresponding sum.

```
package main
```

```
import "fmt"

func main() {
    var n int
    fmt.Print("N = ")
    fmt.Scanf("%d", &n)
    k, sum := 0, 0
    for sum < n {
        k++
        sum += k
    }
    fmt.Printf("K = %d\t\tsum = %d\n", k, sum)
}
```

While 12

Адади бутуни N (>1) дода шудааст. Аз ададҳои бутуни K калонтаринашро хориҷ кунед, ки барои он суммаи 1+2+…+K хурд ё баробари N мешавад ва ҳамчунин ин суммаро низ хориҷ кунед.

An integer *N* (> 1) is given. Find the largest integer *K* such that the sum 1 + 2 + … + *K* is less than or equal to *N*. Output *K* and the corresponding sum.

```
package main

import "fmt"

func main() {
    var n int
    fmt.Print("N = ")
    fmt.Scanf("%d", &n)
    k, sum := 0, 0
    for sum <= n {
        k++
        sum += k
    }
    sum -= k
    k--
    fmt.Printf("K = %d\t\tsum = %d\n", k, sum)
}
```

While 13

Адади A(>1) дода шудааст. Аз ададҳои бутуни K хурдтаринашро хориҷ кунед, ки барои он суммаи 1+1/2+…+1/K калони A мешавад ва ҳамчунин худи ин суммаро низ хориҷ кунед.

A real number *A* (> 1) is given. Find the smallest integer *K* such that the sum 1 + 1/2 + … + 1/*K* is greater than *A*. Output *K* and the corresponding sum.

```
package main

import "fmt"

func main() {
    var a float64
    fmt.Print("A = ")
    fmt.Scanf("%f", &a)
    var sum float64 = 0
    k := 0
    for sum <= a {
        k++
        sum += 1 / float64(k)
    }
    fmt.Printf("K = %d\t\tsum = %.5f\n", k, sum)
}
```

While 14

Адади A(>1) дода шудааст. Аз ададҳои бутуни K калонтаринашро хориҷ кунед, ки барои он суммаи 1+1/2+..+1/K хурди A мешавад ва ҳамчунин худи ин суммаро низ хориҷ кунед.

A real number *A* (> 1) is given. Find the largest integer *K* such that the sum 1 + 1/2 + … + 1/*K* is less than *A*. Output *K* and the corresponding sum.

```
package main

import "fmt"

func main() {
    var a float64
    fmt.Print("A = ")
    fmt.Scanf("%f", &a)
    var sum float64 = 0
    k := 0
    for sum < a {
        k++
        sum += 1 / float64(k)
    }
    sum -= 1 / float64(k)
    k--
    fmt.Printf("K = %d\t\tsum = %.5f\n", k, sum)
}
```

While 15

Пасандози аввалин дар бонк ба 1000 сомонӣ баробар аст. Пас аз ҳар моҳ андозаи пасандоз ба P фоиз аз суммаи мавҷуда зиёд мешавад (P — адади ҳақиқӣ, 0<P<25). Аз рӯи P-и додашуда муайян кунед, ки пас аз чанд моҳ андозаи пасандоз аз 1100 сомонӣ зиёд мешавад. Миқдори моҳҳои ёфташуда K (адади бутун) ва андозаи ниҳоии пасандоз S (адади ҳақиқӣ)-ро ёбед.

A principal of 1000 euro is invested at a rate of *P* percent compounded annually. A real number *P* is given, 0 < *P* < 25. Find, how many years *K* it will take for an investment to exceed 1100 euro. Output *K* (as an integer) and the compound amount *S* of the principal at the end of *K* years (as a real number).

```
package main

import "fmt"

func main() {
    var deposit float64 = 1000
    var p float64
    fmt.Print("P = ")
    fmt.Scanf("%f", &p)
    month := 0
    for deposit <= 1100 {
        month++
        deposit += deposit / 100 * p
    }
    fmt.Printf("K = %d\t\tS = %.2f\n", month, deposit)
}
```

While 16

Варзишгар-лижарон машқро бо давидани 10 км дар рӯзи аввал сар кард. Ҳар як рӯзи оянда он дарозии давишро ба P фоизи рӯзи пешина зиёд кард (P - адади ҳақиқӣ, 0<P<50). Аз рӯи P-и додашуда муайян кунед, ки пас аз чанд рӯз давиши натиҷавии лижарон барои ҳамаи рӯзҳо аз 200 км зиёд мешавад. Миқдори ёфташудаи рӯзҳо K (адади бутун) ва давиши натиҷавӣ S (адади ҳақиқӣ)-ро ёбед.

The skier began trainings having run 10 km. Each next day he increased the run distance by *P* percent from the distance of the last day. A real number *P* is given, 0 < *P* < 50). Find, how many days *K* it will take for a total run to exceed 200 km. Output *K* (as an integer) and the total run *S* (as a real number).

```
package main

import "fmt"

func main() {
    var masofa float64 = 10
    var p float64
    fmt.Print("P = ")
    fmt.Scanf("%f", &p)
    days := 1
    var sum float64 = masofa
    for sum <= 200 {
        days++
        masofa += masofa / 100 * p
        sum += masofa
    }
    fmt.Printf("K = %d\t\tS = %.3f\n", days, sum)
}
```

While 17

Адади бутуни N(>0) дода шудааст. Бо истифодабарии амалҳои тақсими бутун ва гирифтани бақия аз ҳосили тақсим ҳамаи рақамҳои онро сар карда аз тарафи рост (қатори воҳидӣ) хориҷ кунед.

Given an integer *N* (> 0), output all digits of the number *N* starting from the right digit (a ones digit). Use the operators of integer division and taking the remainder after integer division.

```
package main

import "fmt"

func main() {
    var n int
    fmt.Print("N = ")
    fmt.Scanf("%d", &n)
    for n > 0 {
        fmt.Printf("%d\t", n % 10)
        n /= 10
    }
}
```

While 18

Ададӣ бутуни N(>0) дода шудааст. Бо истифодабарии амалҳои тақсими бутун ва гирифтани бақия аз ҳосили тақсим миқдор ва суммаи рақамҳои онро ёбед.

Given an integer *N* (> 0), find the amount and the sum of its digits. Use the operators of integer division and taking the remainder after integer division.

```
package main

import "fmt"

func main() {
    var n int
    fmt.Print("N = ")
    fmt.Scanf("%d", &n)
    count, sum := 0, 0
    for n > 0 {
        sum += n % 10
        n /= 10
        count++
    }
    fmt.Printf("count = %d\t\tsum = %d\n", count, sum)
}
```

While 19

Ададӣ бутуни N(>0) дода шудааст. Бо истифодабарии амалҳои тақсими бутун ва гирифтани бақия аз ҳосили тақсим ададеро ёбед, ки ҳангоми хондани ададӣ N аз рост ба чап ҳосил гардидааст.

An integer *N* (> 0) is given. Output an integer obtained from the given one by reading it from right to left. Use the operators of integer division and taking the remainder after integer division.

```
package main

import "fmt"

func main() {
    var n int
    fmt.Print("N = ")
    fmt.Scanf("%d", &n)
    chappa, tmp := 0, n
    for tmp > 0 {
```

```
        chappa = chappa * 10 + tmp % 10
        tmp /= 10
    }
    fmt.Printf("chappa(%d) = %d\n", n, chappa)
}
```

While 20

Адади бутуни N(>0) дода шудааст. Бо ёрии амалҳои тақсими бутун ва гирифтани бақия аз ҳосили тақсим муайян кунед, ки дар навишти адади N рақами «2» ҳаст ё не. Агар бошад, пас True-ро хориҷ кунед, вагарна — False-ро хориҷ кунед.

An integer *N* (> 0) is given. Determine whether its decimal representation contains a digit "2" or not, and output true or false respectively. Use the operators of integer division and taking the remainder after integer division.

```
package main

import "fmt"

func main() {
    var n int
    fmt.Print("N = ")
    fmt.Scanf("%d", &n)
    has2 := false
    for n > 0 {
        if n % 10 == 2 {
            has2 = true
            break
        }
        n /= 10
    }
    fmt.Printf("has2 = %t\n", has2)
}
```

Series 1

Даҳ то ададҳои ҳақиқӣ дода шудаанд. Суммаи онҳоро ёбед.

Given ten real numbers, find their sum.

```
package main

import "fmt"
```

```
func main() {
    var number float32
    var sum float32 = 0
    for i := 0; i < 10; i++ {
        fmt.Scan(&number)
        sum += number
    }
    fmt.Printf("sum = %.2f\n", sum)
}
```

Series 2

Даҳ то ададҳои ҳақиқӣ дода шудаанд. Ҳосилизарби онҳоро ёбед.

Given ten real numbers, find their product.

```
package main

import "fmt"

func main() {
    var number float32
    var mul float32 = 1
    for i := 0; i < 10; i++ {
        fmt.Scan(&number)
        mul *= number
    }
    fmt.Printf("multiplication = %.2f\n", mul)
}
```

Series 3

Даҳ то ададҳои ҳақиқӣ дода шудаанд. Қимати миёнаи арифметикии онҳоро ёбед.

Given ten real numbers, find their average.

```
package main

import "fmt"

func main() {
    var number float32
    var sum float32 = 0
    for i := 0; i < 10; i++ {
        fmt.Scan(&number)
        sum += number
    }
    aMean := sum / 10
    fmt.Printf("aMean = %.2f\n", aMean)
}
```

Series 4

Адади бутуни N ва маҷмӯъ аз N ададҳои ҳақиқӣ дода шудаанд. Сумма ва ҳосилизарби ададҳоро аз маҷмӯи мазкур хориҷ кунед.

An integer *N* and a sequence of *N* real numbers are given. Output the sum and the product of all elements of this sequence.

```
package main

import "fmt"

func main() {
    var n int
    fmt.Print("N = ")
    fmt.Scan(&n)
    var number float32
    var sum, mul float32 = 0, 1
    for i := 0; i < n; i++ {
        fmt.Scan(&number)
        sum += number
        mul *= number
    }
    fmt.Printf("sum = %.2f\n", sum)
    fmt.Printf("multiplication = %.2f\n", mul)
}
```

Series 5

Адади бутуни N ва маҷмӯъ аз N ададҳои ҳақиқии мусбӣ дода шудаанд. Қисмҳои бутуни ҳамаи ададҳоро (чун ададҳои ҳақиқӣ бо қисми даҳии нулӣ) аз маҷмӯи мазкур бо ҳамин тартиб хориҷ кунед, ҳамчунин суммаи ҳамаи қисмҳои бутунро низ хориҷ кунед.

An integer *N* and a sequence of *N* positive real numbers are given. Output in the same order the integer parts of all elements of this sequence (as real numbers with zero fractional part). Also output the sum of all integer parts.

```
package main

import (
    "fmt"
    "math"
)
```

```
func main() {
    var n int
    fmt.Print("N = ")
    fmt.Scan(&n)
    var number, sum, butun float64
    for n > 0 {
        fmt.Scan(&number)
        butun = math.Floor(number)
        sum += butun
        fmt.Printf("%.2f\t", butun)
        n--
    }
    fmt.Printf("\nsum = %.2f\n", sum)
}
```

Series 6

Адади бутуни N ва мачмӯъ аз N ададҳои ҳақиқии мусбӣ дода шудаанд. Қисмҳои даҳии ҳамаи ададҳоро (чун ададҳои ҳақиқӣ бо қисми бутуни нулӣ), ҳамчунин ҳосилизарби ҳамаи қисмҳои даҳиро хорич кунед.

An integer *N* and a sequence of *N* positive real numbers are given. Output in the same order the fractional parts of all elements of this sequence (as real numbers with zero integer part). Also output the product of all fractional parts.

```
package main

import (
    "fmt"
    "math"
)

func main() {
    var n int
    fmt.Print("N = ")
    fmt.Scan(&n)
    var number, kasri float64
    var mul float64 = 1
    for n > 0 {
        fmt.Scan(&number)
        kasri = number - math.Floor(number)
        fmt.Printf("%.2f\t", kasri)
        mul *= kasri
        n--
    }
    fmt.Printf("\nmultiplication = %.6f\n", mul)
}
```

Series 7

Адади бутуни N ва маҷмӯъ аз N ададҳои ҳақиқӣ дода шудаанд. Қиматҳои яклухткардашудаи ҳамаи ададҳоро аз маҷмӯи мазкур (чун ададҳои бутун), ҳамчунин суммаи қиматҳои яклухткардашударо низ хориҷ кунед.

An integer *N* and a sequence of *N* real numbers are given. Output in the same order the rounded values of all elements of this sequence to the nearest whole number (as integers). Also output the sum of all rounded values.

```
package main

import (
    "fmt"
    "math"
)

func main() {
    var n int
    fmt.Print("N = ")
    fmt.Scan(&n)
    var (
        number float64
        rounded, sum int
    )
    for n > 0 {
        fmt.Scan(&number)
        rounded = int(math.Round(number))
        fmt.Printf("%.d\t", rounded)
        sum += rounded
        n--
    }
    fmt.Printf("\nsum = %d\n", sum)
}
```

Series 8

Адади бутуни N ва маҷмӯъ аз N ададҳои бутун дода шудаанд. Ҳамаи ададҳои ҷуфтро аз маҷмӯи мазкур бо ҳамон тартиб ва миқдори чунин ададҳо К-ро хориҷ кунед.

An integer *N* and a sequence of *N* integers are given. Output in the same order all even-valued elements of the sequence and also their amount *K*.

```
package main

import "fmt"
```

```
func main() {
    var n, number, k int
    fmt.Print("N = ")
    fmt.Scan(&n)
    for n > 0 {
        fmt.Scan(&number)
        if number % 2 == 0 {
            fmt.Printf("%d\t", number)
            k++
        }
        n--
    }
    fmt.Printf("\nK = %d\n", k)
}
```

Series 9

Адади бутуни N ва маҷмӯъ аз N ададҳои бутун дода шудаанд. Рақамҳои ҳамаи ададҳои тоқро аз маҷмӯи мазкур бо ҳамон тартиб ва миқдори чунин ададҳо K-ро хориҷ кунед.

An integer *N* and a sequence of *N* integers are given. Output in the same order the order numbers of all odd-valued elements of the sequence and also their amount *K*.

```
package main

import "fmt"

func main() {
    var n, number, k int
    fmt.Print("N = ")
    fmt.Scan(&n)
    index := 1
    for index <= n {
        fmt.Scan(&number)
        if number % 2 != 0 {
            fmt.Printf("%d\t", index)
            k++
        }
        index++
    }
    fmt.Printf("\nK = %d\n", k)
}
```

Series 10

Адади бутуни N ва маҷмӯъ аз N ададҳои бутун дода шудаанд. Агар дар маҷмӯъ адади мусбӣ бошад, пас True-ро хориҷ кунед; дар ҳолати акс бошад, False-ро хориҷ кунед.

An integer *N* and a sequence of *N* integers are given. Output the logical value true if the sequence contains positive-valued elements, otherwise output false.

```
package main

import "fmt"

func main() {
    var n, number int
    fmt.Print("N = ")
    fmt.Scan(&n)
    hasPositive := false
    for n > 0 {
        fmt.Scan(&number)
        if !hasPositive && number > 0 {
            hasPositive = true
        }
        n--
    }
    fmt.Printf("has positive:\t%t\n", hasPositive)
}
```

Series 11

Ададҳои бутуни K, N ва маҷмӯъ аз N ададҳои бутун дода шудаанд. Агар дар маҷмӯъ адади аз K хурд бошад, пас True-ро хориҷ кунед; дар ҳолати акс False-ро хориҷ кунед.

Integers *K*, *N* and a sequence of *N* integers are given. Output the logical value false if the sequence contains elements of value less than *K*, otherwise output false.

```
package main

import "fmt"

func main() {
    var k, n, number int
    fmt.Print("K = ")
    fmt.Scan(&k)
    fmt.Print("N = ")
    fmt.Scan(&n)
    lessK := false
    for n > 0 {
        fmt.Scan(&number)
        if !lessK && number < k {
            lessK = true
        }
        n--
    }
    fmt.Printf("has less then %d:\t%t\n", k, lessK)
```

```
}
```

Series 12

Маҷмӯи ададҳои бутуни ғайринулӣ дода шудааст; нишонаи баанҷомрасии он - адади 0 аст. Миқдори ададҳоро дар маҷмӯъ хориҷ кунед.

A sequence of nonzero integers terminated by zero is given (the final zero is not an element of the sequence). Output the length of the sequence.

```
package main

import "fmt"

func main() {
    var number, count int
    for {
        fmt.Scan(&number)
        if (number == 0) {
            break
        }
        count++
    }
    fmt.Printf("count = %d\n", count)
}
```

Series 13

Маҷмӯи ададҳои бутуни ғайринулӣ дода шудааст; нишонаи баанҷомрасии он - адади 0 аст. Суммаи ҳамаи ададҳои ҷуфти мусбиро аз маҷмӯи мазкур хориҷ кунед. Агар адади талабкардашуда дар маҷмӯъ набошад, пас 0(нул)-ро хориҷ кунед.

A sequence of nonzero integers terminated by zero is given. Output the sum of all positive-valued elements of the sequence. If the sequence does not contain the required elements then output 0.

```
package main

import "fmt"

func main() {
    var number, sum int
```

```
    for {
        fmt.Scan(&number)
        if (number == 0) {
            break
        }
        if number > 0 && (number % 2 == 0) {
            sum += number
        }
    }
    fmt.Printf("sum = %d\n", sum)
}
```

Series 14

Адади бутуни K ва маҷмӯъ аз ададҳои бутуни ғайринулӣ дода шудаанд; нишонаи баанҷомрасии он - адади 0 аст. Миқдори ададҳои аз K хурдро дар маҷмӯъ хориҷ кунед.

An integer *K* and a sequence of nonzero integers terminated by zero are given (the final zero is not an element of the sequence). Output the amount of elements whose value less than *K*.

```
package main

import "fmt"

func main() {
    var k, number, count int
    fmt.Print("K = ")
    fmt.Scan(&k)
    for {
        fmt.Scan(&number)
        if number == 0 {
            break
        }
        if number < k {
            count++
        }
    }
    fmt.Printf("count = %d\n", count)
}
```

Series 15

Адади бутуни K ва маҷмӯи ададҳои бутуни ғайринулӣ дода шудаанд; нишонаи баанҷомрасии он - адади 0 аст. Рақами адади аввалини аз K калонро дар маҷмӯъ хориҷ кунед. Агар чунин адад набошад, пас 0(нул)-ро хориҷ кунед.

An integer *K* and a sequence of nonzero integers terminated by zero are given (the final zero is not an element of the sequence). Output the order number of the first element whose value greater than *K*. If the sequence does not contain the required elements then output 0.

```
package main

import "fmt"

func main() {
    var k, number, index int
    fmt.Print("K = ")
    fmt.Scan(&k)
    i := 1
    for {
        fmt.Scan(&number)
        if number == 0 {
            break
        }
        if index == 0 && number > k {
            index = i
        }
        i++
    }
    fmt.Printf("index = %d\n", index)
}
```

Series 16

Адади бутуни K ва мачмӯи ададҳои бутуни ғайринулӣ дода шудаанд; нишонаи баанҷомрасии он - адади 0 аст. Рақами адади охирини аз K калонро дар мачмӯъ хориҷ кунед. Агар чунин адад набошад, пас 0(нул)-ро хориҷ кунед.

An integer *K* and a sequence of nonzero integers terminated by zero are given (the final zero is not an element of the sequence). Output the order number of the last element whose value greater than *K*. If the sequence does not contain the required elements then output 0.

```
package main

import "fmt"

func main() {
    var k, number, index int
    fmt.Print("K = ")
    fmt.Scan(&k)
```

```
    i := 1
    for {
        fmt.Scan(&number)
        if number == 0 {
            break
        }
        if number > k {
            index = i
        }
        i++
    }
    fmt.Printf("index = %d\n", index)
}
```

Series 17

Адади ҳақиқии B, адади бутуни N ва маҷмӯъ аз N ададҳои ҳақиқии афзуншавандаи батартибгузошташуда дода шудаанд. Элементҳои маҷмӯъро якҷоя бо адади B хориҷ кунед, дар ин ҳангом тартиби ададҳои хориҷшавандаро нигоҳ доред.

A real number *B*, an integer *N* and a sequence of *N* real numbers are given. The values of elements of the sequence are in ascending order. Output the number *B* jointly with the elements of the sequence so that all output numbers were in ascending order.

```
package main

import "fmt"

func main() {
    var (
        b, number float64
        n int
        printed = false
    )
    fmt.Print("B = ")
    fmt.Scan(&b)
    fmt.Print("N = ")
    fmt.Scan(&n)
    for n > 0 {
        fmt.Scan(&number)
        if !printed && b < number {
            fmt.Printf("%.2f\t", b)
            printed = true
        }
        fmt.Printf("%.2f\t", number)
        n--
    }
    if !printed {
```

```
        fmt.Printf("%.2f\n", b)
    }
}
```

Series 18

Адади бутуни N ва маҷмӯъ аз N ададҳои бутуни афзуншавандаи батартибгузошташуда дода шудаанд. Маҷмӯи мазкур элементҳои якхеларо низ доро буда метавонад. Ҳамаи элементҳои гуногуни маҷмӯи мазкурро бо ҳамон тартиб хориҷ кунед.

An integer *N* and a sequence of *N* integers are given. The values of elements of the sequence are in ascending order, the sequence may contain equal elements. Output in the same order all distinct elements of the sequence.

```
package main

import "fmt"

func main() {
    var n, number, prev int
    fmt.Print("N = ")
    fmt.Scan(&n)
    firstTime := true
    for n > 0 {
        fmt.Scan(&number)
        if firstTime {
            fmt.Printf("%d  ", number)
            firstTime = false
        } else if (number != prev) {
            fmt.Printf("%d  ", number)
        }
        prev = number
        n--
    }
}
```

Series 19

Адади бутуни N (> 1) ва маҷмӯъ аз N ададҳои бутун дода шудаанд. Он элементҳои маҷмӯъро, ки онҳо аз ҳамсояи чапи худ хурд ҳастанд ва ҳамчунин миқдори чунин элементҳо K-ро хориҷ кунед.

An integer *N* (> 1) and a sequence of *N* integers are given. Output the elements that are less than their left-side neighbor. Also output the amount *K* of such elements.

```
package main

import "fmt"

func main() {
    var n, number, prev, k int
    fmt.Print("N = ")
    fmt.Scan(&n)
    fmt.Scan(&prev)
    for n > 1 {
        fmt.Scan(&number)
        if number < prev {
            fmt.Printf("%d  ", number)
            k++
        }
        prev = number
        n--
    }
    fmt.Printf("\nK = %d\n", k)
}
```

Series 20

Адади бутуни N (> 1) ва маҷмӯъ аз N ададҳои бутун дода шудаанд. Он элементҳои маҷмӯъро, ки онҳо аз ҳамсояи рости худ хурд ҳастанд ва миқдори чунин элементҳо K-ро хориҷ кунед.

An integer *N* (> 1) and a sequence of *N* integers are given. Output the elements that are less than their right-side neighbor. Also output the amount *K* of such elements.

```
package main

import "fmt"

func main() {
    var n, number, prev, k int
    fmt.Print("N = ")
    fmt.Scan(&n)
    fmt.Scan(&prev)
    for n > 1 {
        fmt.Scan(&number)
        if prev < number {
            fmt.Printf("%d  ", prev)
            k++
        }
        prev = number
```

```
        n--
    }
    fmt.Printf("\nK = %d\n", k)
}
```

Proc 1

Протседураи PowerA3(A, B)-ро тасвир кунед, ки дараҷаи сеюми адади A-ро ҳисоб мекунад ва онро ба тағйирёбандаи B бозмегардонад (A - параметри дохилшаванда, B - параметри хориҷшаванда; ҳар дуи параметрҳо ҳақиқӣ мебошанд). Бо ёрии ин протседура дараҷаҳои сеюми панҷ ададҳои додашударо ёбед.

Write a procedure PowerA3(*A*, *B*) that computes the third degree of a real number *A* and assigns the result to a real variable *B* (*A* is an input parameter, *B* is an output parameter). Using this procedure, find the third degree of five given real numbers.

```
package main

import "fmt"

func PowerA3(a float32, b *float32) {
    *b = a * a * a
}

func main() {
    var a, b float32
    for i := 0; i < 5; i++ {
        fmt.Print("A = ")
        fmt.Scan(&a)
        PowerA3(a, &b)
        fmt.Printf("B = %.2f\n\n", b)
    }
}
```

Proc 2

Протседураи PowerA234(A, B, C, D)-ро тасвир кунед, ки дараҷаи дуюм, сеюм ва чоруми адади A-ро ҳисоб карда, ин дараҷаҳоро мувофиқан ба тағйирёбандаҳои B, C ва D бозмегадонад (A - параметри дохилшаванда, B, C, D -

параметрҳои хориҷшаванда; ҳамаи параметрҳо ҳақиқӣ мебошанд). Бо ёрии ин протседура дараҷаи дуюм, сеюм ва чорӯми панҷ ададҳои додашударо ёбед.

Write a procedure PowerA234(*A*, *B*, *C*, *D*) that computes the second, the third, and the fourth degrees of a real number *A* and assigns the results to real variables *B*, *C*, and *D* respectively (*A* is an input parameter, *B*, *C*, *D* are output parameters). Using this procedure, find the second, the third, and the fourth degrees of five given real numbers.

```
package main

import "fmt"

/*
func PowerA234(a float32, b *float32, c *float32, d *float32) {
    *b = a * a
    *c = *b * a
    *d = *c * a
}
*/

/*
func PowerA234(a float32) (b float32, c float32, d float32) {
    b = a * a
    c = b * a
    d = c * a
    return
}
*/

func PowerA234(a float32) (float32, float32, float32) {
    b := a * a
    c := b * a
    d := c * a
    return b, c, d
}

func main() {
    var a, b, c, d float32
    for i := 0; i < 5; i++ {
        fmt.Print("A = ")
        fmt.Scan(&a)
        //PowerA234(a, &b, &c, &d)
        b, c, d = PowerA234(a)
        fmt.Printf("B = %.2f\tC = %.2f\tD = %.2f\n\n", b, c, d)
    }
}
```

Proc 3

Протседураи Mean(X, Y, AMean, GMean)-ро тасвир кунед, ки қимати миёнаи арифметикӣ AMean=(X+Y)/2 ва қимати миёнаи геометрӣ GMean=sqrt(X*Y)-и ду ададҳои мусбии X ва Y-ро ҳисоб мекунад (X ва Y - параметрҳои дохилшаванда, AMean ва GMean - параметрҳои хориҷшавандаи типи ҳақиқӣ). Бо ёрии ин протседура қиматҳои миёнаи арифметикӣ ва геометриро барои ҷуфтҳои ададҳои (A,B), (A,C), (A,D) ёбед, агар A, B, C, D дода шуда бошанд.

Write a procedure Mean(*X*, *Y*, *AMean*, *GMean*) that computes the *arithmetical mean AMean* = (*X*+*Y*)/2 and the *geometrical mean GMean* = $(X \cdot Y)^{1/2}$ of two positive numbers *X* and *Y* (*X* and *Y* are input parameters, *AMean* and *GMean* are output parameters; all parameters are the real-valued ones). Using this procedure, find the arithmetical mean and the geometrical mean of pairs (*A*, *B*), (*A*, *C*), (*A*, *D*) provided that real numbers *A*, *B*, *C*, *D* are given.

```
package main

import "fmt"
import "math"

/*
func Mean(x float64, y float64, aMean *float64, gMean *float64) {
    *aMean = (x + y) / 2
    *gMean = math.Sqrt(x * y)
}
*/

/*
func Mean(x float64, y float64) (aMean float64, gMean float64) {
    aMean = (x + y) / 2
    gMean = math.Sqrt(x * y)
    return
}
*/

func Mean(x float64, y float64) (float64, float64) {
    aMean := (x + y) / 2
    gMean := math.Sqrt(x * y)
    return aMean, gMean
}

func main() {
    var a, b, c, d, aM, gM float64
    fmt.Print("A = ")
```

```
    fmt.Scan(&a)
    fmt.Print("B = ")
    fmt.Scan(&b)
    fmt.Print("C = ")
    fmt.Scan(&c)
    fmt.Print("D = ")
    fmt.Scan(&d)
    //Mean(a, b, &aM, &gM)
    aM, gM = Mean(a, b)
    fmt.Print("Mean(A, B, AMean, GMean):\t")
    fmt.Printf("AMean = %.2f\tGMean = %.2f\n", aM, gM)
    //Mean(a, c, &aM, &gM)
    aM, gM = Mean(a, c)
    fmt.Print("Mean(A, C, AMean, GMean):\t")
    fmt.Printf("AMean = %.2f\tGMean = %.2f\n", aM, gM)
    //Mean(a, d, &aM, &gM)
    aM, gM = Mean(a, d)
    fmt.Print("Mean(A, D, AMean, GMean):\t")
    fmt.Printf("AMean = %.2f\tGMean = %.2f\n", aM, gM)
}
```

Proc 4

Протседураи TrianglePS(a,P,S)-ро тасвир кунед, ки аз рӯи тарафи a-и секунҷаи баробартараф периметр P=3*a ва масоҳати он S=a^2*sqrt(3)/4-ро ҳособ мекунад (a - параметри дохилшаванда, P ва S - параметрҳои хориҷшаванда; ҳамаи параметрҳо ҳақиқӣ мебошанд). Бо ёрии ин протседура периметрҳо ва масоҳатҳои се секунҷаҳои баробартарафро бо тарафҳои додашуда ёбед.

Write a procedure TrianglePS(*a*, *P*, *S*) that computes the perimeter $P = 3 \cdot a$ and the area $S = a^2 \cdot (3)^{1/2}/4$ of an equilateral triangle with the side *a* (*a* is an input parameter, *P* and *S* are output parameters, all parameters are the real-valued ones). Using this procedure, find the perimeters and the areas of three triangles with the given lengths of the sides.

```
package main

import (
    "fmt"
    "math"
)

/*
func TrianglePS(a float64, p *float64, s *float64) {
    *p = 3 * a
    *s = a*a * math.Sqrt(3) / 4
}
```

```
*/

/*
func TrianglePS(a float64) (p float64, s float64) {
    p = 3 * a
    s = a*a * math.Sqrt(3) / 4
    return
}
*/

func TrianglePS(a float64) (float64, float64) {
    p := 3 * a
    s := a*a * math.Sqrt(3) / 4
    return p, s
}

func main() {
    var a, p, s float64
    for i := 0; i < 3; i++ {
        fmt.Print("a = ")
        fmt.Scan(&a)
        //TrianglePS(a, &p, &s)
        p, s = TrianglePS(a)
        fmt.Printf("P = %.2f\tS = %.2f\n\n", p, s)
    }
}
```

Proc 5

Протседураи RectPS(x_1,y_1,x_2,y_2,P,S)-ро тасвир кунед, ки периметр P ва масоҳати S росткунҷаро бо тарафҳои ба тирҳои координатӣ параллел аз рӯи координатаҳои қуллаҳои муқобили он (x_1,y_1), (x_2,y_2) ҳисоб мекунад (x_1,y_1,x_2,y_2 - параметрҳои дохилшаванда, P ва S - параметрҳои хориҷшавандаи типи ҳақиқӣ). Бо ёрии ин протседура периметрҳо ва масоҳатҳои се росткунҷаҳоро бо қуллаҳои муқобили додашуда ёбед.

Write a procedure RectPS(x_1, y_1, x_2, y_2, P, S) that computes the perimeter P and the area S of a rectangle whose opposite vertices have coordinates (x_1, y_1) and (x_2, y_2) and sides are parallel to coordinate axes (x_1, y_1, x_2, y_2 are input parameters, P and S are output parameters, all parameters are the real-valued ones). Using this procedure, find the perimeters and the areas of three rectangles with the given opposite vertices.

```
package main
```

```
import (
    "fmt"
    "math"
)

/*
func RectPS(x1 float64, y1 float64, x2 float64, y2 float64, p *float64, s
*float64) {
    a := math.Abs(x2 - x1)
    b := math.Abs(y2 - y1)
    *p = 2 * (a + b)
    *s = a * b
}
*/

/*
func RectPS(x1 float64, y1 float64, x2 float64, y2 float64) (p float64, s
float64) {
    a := math.Abs(x2 - x1)
    b := math.Abs(y2 - y1)
    p = 2 * (a + b)
    s = a * b
    return
}
*/

func RectPS(x1 float64, y1 float64, x2 float64, y2 float64) (float64,
float64) {
    a := math.Abs(x2 - x1)
    b := math.Abs(y2 - y1)
    p := 2 * (a + b)
    s := a * b
    return p, s
}

func main() {
    var x1, y1, x2, y2, p, s float64
    for i := 0; i < 3; i++ {
        fmt.Scan(&x1, &y1, &x2, &y2)
        //RectPS(x1, y1, x2, y2, &p, &s)
        p, s = RectPS(x1, y1, x2, y2)
        fmt.Printf("P = %.2f\t\tS = %.2f\n\n", p, s)
    }
}
```

Proc 6

Протседураи DigitCountSum(K,C,S)-ро тасвир кунед, ки миқдори рақамҳои C ададӣ бутуни мусбии K, ҳамчунин суммаи онҳо S-ро ҳисоб мекунад (K - параметри дохилшаванда, C ва S - параметрҳои хориҷшавандаи типи бутун). Бо ёрии ин протседура миқдор ва суммаи рақамҳоро барои ҳар яке аз панҷ ададҳои бутуни додашуда ёбед.

Write a procedure DigitCountSum(*K*, *C*, *S*) that finds the amount *C* of digits in the decimal representation of a positive integer *K* and also the sum *S* of its digits (*K* is an input parameter, *C* and *S* are output parameters, all parameters are the integer ones). Using this procedure, find the amount and the sum of digits for each of five given integers.

```
package main

import "fmt"

/*
func DigitCountSum(k uint, c *uint, s *uint) {
    *c, *s = 0, 0
    for k > 0 {
        (*c)++
        *s += k % 10
        k /= 10
    }
}
*/

/*
func DigitCountSum(k uint) (c uint, s uint) {
    c, s = 0, 0
    for k > 0 {
        c++
        s += k % 10
        k /= 10
    }
    return
}
*/

func DigitCountSum(k uint) (uint, uint) {
    var c, s uint = 0, 0
    for k > 0 {
        c++
        s += k % 10
        k /= 10
    }
    return c, s
}

func main() {
    var k, count, sum uint
    for i := 0; i < 5; i++ {
        fmt.Printf("K%d = ", i+1)
        fmt.Scan(&k)
        //DigitCountSum(k, &count, &sum)
        count, sum = DigitCountSum(k)
        fmt.Printf("count = %d\t\tsum = %d\n\n", count, sum)
    }
}
```

Proc 7

Протседураи InvertDigits(K)-ро тасвир кунед, ки тартиби ҷойгиршавии рақамҳои ададӣ бутуни мусбии K-ро ба баръакс иваз мекунад (K - параметри типи бутун аст, ки дар як вақт ҳам дохилшаванда ва ҳам хориҷшаванда мебошад). Бо ёрии ин протседура тартиби ҷойгиравии рақамҳоро барои ҳар яке аз панҷ ададҳои бутуни додашуда ба баръакс иваз кунед.

Write a procedure InvDigits(*K*) that inverts the order of digits of a positive integer *K* (*K* is an input and output integer parameter). Using this procedure, invert the order of digits for each of five given integers.

```
package main

import "fmt"

/*
func InvDigits(k *uint) {
    tmp := *k
    *k = 0
    for tmp > 0 {
        *k = (*k) * 10 + tmp % 10
        tmp /= 10
    }
}
*/

/*
func InvDigits(k uint) uint {
    var tmp uint = 0
    for k > 0 {
        tmp = tmp * 10 + k % 10
        k /= 10
    }
    return tmp
}
*/

func InvDigits(k uint) (tmp uint) {
    tmp = 0
    for k > 0 {
        tmp = tmp * 10 + k % 10
        k /= 10
    }
    return
}

func main() {
    var k uint
    for i := 1; i <= 5; i++ {
        fmt.Printf("K%d = ", i)
        fmt.Scan(&k)
```

```
        //InvDigits(&k)
        k = InvDigits(k)
        fmt.Printf("K%d = %d\n\n", i, k)
    }
}
```

Proc 8

Протседураи AddRightDigit(D, K)-ро тасвир кунед, ки ба ададди бутуни мусбии K рақами D-ро аз рост ҳамроҳ мекунад (D - параметри дохилшавандаи типи бутун аст, ки дар фосилаи 0-9 мехобад, K - параметри типи бутун аст, ки дар як вақт ҳам дохилшаванда ва ҳам хориҷшаванда мебошад). Бо ёрии ин протседура пайдарпай ба ададди додашудаи K рақамҳои додашудаи D_1 ва D_2-ро аз рост ҳамроҳ кунед ва натиҷаи ҳар як ҳамроҳкуниро хориҷ кунед.

Write a procedure AddRightDigit(*D*, *K*) that adds a digit *D* to the right side of the decimal representation of a positive integer *K* (*D* is an input integer parameter with the value in the range 0 to 9, *K* is an input and output integer parameter). Having input an integer *K* and two one-digit numbers D_1, D_2 and using two calls of this procedure, sequentially add the given digits D_1, D_2 to the right side of the given K and output the result of each adding.

```
package main

import "fmt"

/*
func AddRightDigit(d uint, k *uint) {
    *k = (*k) * 10 + d
}
*/

/*
func AddRightDigit(d uint, k uint) uint {
    return k * 10 + d
}
*/

func AddRightDigit(d uint, k uint) (tmp uint) {
    tmp = k * 10 + d
    return
}
```

```
func main() {
    var k, d uint
    fmt.Print("K = ")
    fmt.Scan(&k)
    for i := 1; i <= 2; i++ {
        fmt.Printf("D%d = ", i)
        fmt.Scan(&d)
        //AddRightDigit(d, &k)
        k = AddRightDigit(d, k)
        fmt.Printf("K = %d\n\n", k)
    }
}
```

Proc 9

Протседураи AddLeftDigit(D, K)-ро тасвир кунед, ки ба ададди бутуни мусбии K рақами D-ро аз чап ҳамроҳ мекунад (D - параметри дохилшавандаи типи бутун, ки дар фосилаи 1-9 мехобад, K - параметри типи бутун аст, ки дар як вақт ҳам дохилшаванда ва ҳам хориҷшаванда мебошад). Бо ёрии ин протседура пайдарпай ба ададди додашудаи K рақамҳои додашудаи D_1 ва D_2-ро аз чап ҳамроҳ кунед ва натиҷаи ҳар як ҳамроҳкуниро хориҷ кунед.

Write a procedure AddLeftDigit(*D*, *K*) that adds a digit *D* to the left side of the decimal representation of a positive integer *K* (*D* is an input integer parameter with the value in the range 0 to 9, *K* is an input and output integer parameter). Having input an integer *K* and two one-digit numbers D_1, D_2 and using two calls of this procedure, sequentially add the given digits D_1, D_2 to the left side of the given K and output the result of each adding.

```
package main

import "fmt"

/*
func AddLeftDigit(d uint, k *uint) {
    tmp := *k
    for tmp > 0 {
        d *= 10
        tmp /= 10
    }
    *k += d
}
*/
```

```
/*
func AddLeftDigit(d uint, k uint) uint {
    tmp := k
    for tmp > 0 {
        d *= 10
        tmp /= 10
    }
    return k + d
}
*/

func AddLeftDigit(d uint, k uint) (tmp uint) {
    tmp = k
    for k > 0 {
        d *= 10
        k /= 10
    }
    tmp += d
    return
}

func main() {
    var k, d uint
    fmt.Print("K = ")
    fmt.Scan(&k)
    for i := 1; i <= 2; i++ {
        fmt.Printf("D%d = ", i)
        fmt.Scan(&d)
        //AddLeftDigit(d, &k)
        k = AddLeftDigit(d, k)
        fmt.Printf("K = %d\n\n", k)
    }
}
```

Proc 10

Протседураи Swap(X, Y)-ро тасвир кунед, ки таркиби тағйирёбандаҳои X ва Y-ро иваз мекунад (X ва Y - параметрҳои ҳақиқӣ, ки дар як вақт ҳам дохилшаванда ва ҳам хориҷшаванда мебошанд). Бо ёрии он барои тағйирёбандаҳои додашудаи A, B, C, D пайдарпай таркиби ҷуфтҳои ададҳои зеринро иваз кунед: A ва B, C ва D, B ва C ва қиматҳои нави A, B, C, D-ро хориҷ кунед.

Write a procedure Swap(*X*, *Y*) that exchanges the values of variables *X* and *Y* (*X* and *Y* are input and output real-valued parameters). Having input integers *A*, *B*, *C*, *D* and using three calls of this procedure, sequentially exchange the values of the

pairs *A* and *B*, *C* and *D*, *B* and *C*. Output the new values of *A*, *B*, *C*, *D*.

```
package main

import "fmt"

/*
func Swap(x *float32, y *float32) {
    tmp := *x
    *x = *y
    *y = tmp
}
*/

func Swap(x float32, y float32) (float32, float32) {
    return y, x
}

func main() {
    var a, b, c, d float32
    fmt.Print("A = ")
    fmt.Scan(&a)
    fmt.Print("B = ")
    fmt.Scan(&b)
    fmt.Print("C = ")
    fmt.Scan(&c)
    fmt.Print("D = ")
    fmt.Scan(&d)
    //Swap(&a, &b)
    //Swap(&c, &d)
    //Swap(&b, &c)
    a, b = Swap(a, b)
    c, d = Swap(c, d)
    b, c = Swap(b, c)
    fmt.Printf("\nA = %.2f\nB = %.2f\nC = %.2f\nD = %.2f\n", a, b, c, d)
}
```

Proc 11

Протседураи Minmax(X, Y)-ро тасвир кунед, ки ба тағйирёбандаи X қимати хурдтарини тағйирёбандаҳои X ва Y, ба тағйирёбандаи Y - қимати калонтарини ин тағйирёбандаҳоро сабт мекунад (X ва Y - параметрҳои ҳақиқӣ мебошанд, ки дар як вақт ҳам дохилшаванда ва ҳам хориҷшаванда мебошанд). Чор даъвати ин протседураро истифода бурда, қиматҳои хурдтарин ва калонтарини ададҳои додашудаи A, B, C, D-ро ёбед.

Write a procedure Minmax(*X*, *Y*) that exchanges, if necessary, the values of two variables *X* and *Y* so that $X \le Y$ (*X* and *Y* are

input and output real-valued parameters). Using four calls of this procedure, find the minimal value and the maximal value among given real numbers *A*, *B*, *C*, *D*.

```
package main

import "fmt"

/*
func Swap(x *float32, y *float32) {
    tmp := *x
    *x = *y
    *y = tmp
}
func Minmax(x *float32, y *float32) {
    if *x > *y {
        Swap(x, y)
    }
}
*/

/*
func Minmax(x float32, y float32) (float32, float32) {
    if x > y {
        return y, x
    }
    return x, y
}
*/

func Minmax(x float32, y float32) (min float32, max float32) {
    if x < y {
        min, max = x, y
    } else {
        min, max = y, x
    }
    return
}

func main() {
    var a, b, c, d float32
    fmt.Print("A = ")
    fmt.Scan(&a)
    fmt.Print("B = ")
    fmt.Scan(&b)
    fmt.Print("C = ")
    fmt.Scan(&c)
    fmt.Print("D = ")
    fmt.Scan(&d)
    //Minmax(&a, &b)
    //Minmax(&c, &d)
    //Minmax(&a, &c)
    //Minmax(&b, &d)
    a, b = Minmax(a, b)
    c, d = Minmax(c, d)
    a, c = Minmax(a, c)
    b, d = Minmax(b, d)
    fmt.Printf("Min = %.2f\t\tMax = %.2f\n", a, d)
}
```

Proc 12

Протседураи SortInc3(A, B, C)-ро тасвир кунед, ки таркиби тағйирёбандаҳои A, B, C чунон иваз мекунад, ки қиматҳои онҳо афзуншаванда ба тартиб гузошта мешаванд (A, B, C - параметрҳои ҳақиқӣ, ки дар як вақт ҳам дохилшаванда ва ҳам хориҷшаванда мебошанд). Бо ёрии ин протседура ду маҷмӯи додашудаи аз се ададҳо иборатбудаи: (A_1,B_1,C_1) ва (A_2,B_2,C_2)-ро афзуншаванда ба тартиб гузоред.

Write a procedure SortInc3(*A*, *B*, *C*) that interchanges, if necessary, the values of three variables *A*, *B*, *C* so that $A \le B \le C$ (*A*, *B*, *C* are input and output real-valued parameters). Using this procedure, sort each of two given triples of real numbers (A_1, B_1, C_1) and (A_2, B_2, C_2) in ascending order.

```
package main

import "fmt"

/*
func Swap(x *float32, y *float32) {
    tmp := *x
    *x = *y
    *y = tmp
}
func Minmax(x *float32, y *float32) {
    if *x > *y {
        Swap(x, y)
    }
}
func SortInc3(a *float32, b *float32, c *float32) {
    Minmax(a, b)
    Minmax(a, c)
    Minmax(b, c)
}
*/

func Minmax(x float32, y float32) (float32, float32) {
    if x > y {
        return y, x
    }
    return x, y
}
/*
func SortInc3(a float32, b float32, c float32) (x float32, y float32, z
float32) {
    a, b = Minmax(a, b)
    a, c = Minmax(a, c)
    b, c = Minmax(b, c)
    x, y, z = a, b, c
```

```
    return
}
*/
func SortInc3(a float32, b float32, c float32) (float32, float32, float32) {
    a, b = Minmax(a, b)
    a, c = Minmax(a, c)
    b, c = Minmax(b, c)
    return a, b, c
}

func main() {
    var a, b, c float32
    for i := 1; i <= 2; i++ {
        fmt.Printf("A%d = ", i)
        fmt.Scan(&a)
        fmt.Printf("B%d = ", i)
        fmt.Scan(&b)
        fmt.Printf("C%d = ", i)
        fmt.Scan(&c)
        //SortInc3(&a, &b, &c)
        a, b, c = SortInc3(a, b, c)
        fmt.Printf("A%d = %.2f\tB%d = %.2f\tC%d = %.2f\n\n", i, a, i, b, i,
c)
    }
}
```

Proc 13

Протседураи SortDec3(A,B,C)-ро тасвир кунед, ки таркиби тағйирёбандаҳои A,B,C-ро чунон иваз мекунад, ки қиматҳои онҳо камшаванда ба тартиб гузошта мешаванд (A,B,C - параметрҳои ҳақиқӣ, ки дар як вақт ҳам дохилшаванда ва ҳам хориҷшаванда мебошанд). Бо ёрии ин протседура ду маҷмӯи додашудаи аз се ададҳо иборатбудаи: (A_1,B_1,C_1) ва (A_2,B_2,C_2)-ро камшаванда ба тартиб гузоред.

Write a procedure SortDec3(*A*, *B*, *C*) that interchanges, if necessary, the values of three variables *A*, *B*, *C* so that $A \geq B \geq C$ (*A*, *B*, *C* are input and output real-valued parameters). Using this procedure, sort each of two given triples of real numbers (A_1, B_1, C_1) and (A_2, B_2, C_2) in descending order.

```
package main

import "fmt"

/*
func Swap(x *float32, y *float32) {
    tmp := *x
    *x = *y
    *y = tmp
```

```
}
func Maxmin(x *float32, y *float32) {
    if *x < *y {
        Swap(x, y)
    }
}
func SortDec3(a *float32, b *float32, c *float32) {
    Maxmin(a, b)
    Maxmin(a, c)
    Maxmin(b, c)
}
*/

func Maxmin(x float32, y float32) (float32, float32) {
    if x < y {
        return y, x
    }
    return x, y
}
/*
func SortDec3(a float32, b float32, c float32) (x float32, y float32, z
float32) {
    a, b = Maxmin(a, b)
    a, c = Maxmin(a, c)
    b, c = Maxmin(b, c)
    x, y, z = a, b, c
    return
}
*/
func SortDec3(a float32, b float32, c float32) (float32, float32, float32) {
    a, b = Maxmin(a, b)
    a, c = Maxmin(a, c)
    b, c = Maxmin(b, c)
    return a, b, c
}

func main() {
    var a, b, c float32
    for i := 1; i <= 2; i++ {
        fmt.Printf("A%d = ", i)
        fmt.Scan(&a)
        fmt.Printf("B%d = ", i)
        fmt.Scan(&b)
        fmt.Printf("C%d = ", i)
        fmt.Scan(&c)
        //SortDec3(&a, &b, &c)
        a, b, c = SortDec3(a, b, c)
        fmt.Printf("A%d = %.2f\tB%d = %.2f\tC%d = %.2f\n\n", i, a, i, b, i,
c)
    }
}
```

Proc 14

Протседураи ShiftRight3(A,B,C)-ро тасвир кунед, ки кӯчиши рости давриро ичро мекунад: қимати A ба B мекӯчад, қимати B - ба C ва қимати C - ба A мекӯчад (A,B,C -

параметрҳои ҳақиқӣ, ки дар як вақт ҳам дохилшаванда ва ҳам хориҷшаванда мебошанд). Бо ёрии ин протседура кӯчиши рости давриро барои ду маҷмӯъҳои додашудаи аз се ададҳо иборатбудаи: (A_1,B_1,C_1) ва (A_2,B_2,C_2) иҷро кунед.

Write a procedure ShiftRight3(*A*, *B*, *C*) that performs a *right cyclic shift* by assigning the initial values of *A*, *B*, *C* to variables *B*, *C*, *A* respectively (*A*, *B*, *C* are input and output real-valued parameters). Using this procedure, perform the right cyclic shift for each of two given triples of real numbers: (A_1, B_1, C_1) and (A_2, B_2, C_2).

```
package main

import "fmt"

/*
func ShiftRight3(a *float32, b *float32, c *float32) {
    tmp := *c
    *c = *b
    *b = *a
    *a = tmp
}
*/

func ShiftRight3(a float32, b float32, c float32) (float32, float32, float32)
{
    return c, a, b
}

func main() {
    var a, b, c float32
    for i := 1; i <= 2; i++ {
        fmt.Printf("A%d = ", i)
        fmt.Scan(&a)
        fmt.Printf("B%d = ", i)
        fmt.Scan(&b)
        fmt.Printf("C%d = ", i)
        fmt.Scan(&c)
        //ShiftRight3(&a, &b, &c)
        a, b, c = ShiftRight3(a, b, c)
        fmt.Printf("\nA%d = %.2f\nB%d = %.2f\nC%d = %.2f\n\n", i, a, i, b, i,
c)
    }
}
```

Proc 15

Протседураи ShiftLeft3(A,B,C)-ро тасвир кунед, ки кӯчиши чапи давриро иҷро мекунад: қимати A ба C мекӯчад, қимати

C - ба B ва қимати B - ва A мекӯчад (A,B,C - параметрҳои ҳақиқӣ, ки дар як вақт ҳам дохилшаванда ва ҳам хориҷшаванда мебошанд). Бо ёрии ин протседура кӯчиши чапи давриро барои ду маҷмӯъҳои додашудаи аз се ададҳо иборатбудаи: (A_1,B_1,C_1) ва (A_2,B_2,C_2) иҷро кунед.

Write a procedure ShiftLeft3(*A*, *B*, *C*) that performs a *left cyclic shift* by assigning the initial values of *A*, *B*, *C* to variables *C*, *A*, *B* respectively (*A*, *B*, *C* are input and output real-valued parameters). Using this procedure, perform the left cyclic shift for each of two given triples of real numbers: (A_1, B_1, C_1) and (A_2, B_2, C_2).

```
package main

import "fmt"

/*
func ShiftLeft3(a *float32, b *float32, c *float32) {
    tmp := *c
    *c = *a
    *a = *b
    *b = tmp
}
*/

func ShiftLeft3(a float32, b float32, c float32) (float32, float32, float32)
{
    return b, c, a
}

func main() {
    var a, b, c float32
    for i := 1; i <= 2; i++ {
        fmt.Printf("A%d = ", i)
        fmt.Scan(&a)
        fmt.Printf("B%d = ", i)
        fmt.Scan(&b)
        fmt.Printf("C%d = ", i)
        fmt.Scan(&c)
        //ShiftLeft3(&a, &b, &c)
        a, b, c = ShiftLeft3(a, b, c)
        fmt.Printf("\nA%d = %.2f\nB%d = %.2f\nC%d = %.2f\n\n", i, a, i, b, i,
c)
    }
}
```

Proc 16

Функсияи типи бутуни Sign(X)-ро тасвир кунед, ки барои адади ҳақиқии X қиматҳои зеринро бозмегардонад: -1, агар X<0; 0, агар X=0; 1, агар X>0. Бо ёрии ин функсия қимати ифодаи Sign(A)+Sign(B)-ро барои ададҳои ҳақиқии додашудаи A ва B ҳисоб кунед.

Write an integer function Sign(*X*) that returns the following value: −1, if *X* < 0; 0, if *X* = 0; 1, if *X* > 0(X is a real-valued parameter). Using this function, evaluate an expression Sign(*A*) + Sign(*B*) for given real numbers *A* and *B*.

```
package main

import "fmt"

func Sign(x float32) int {
    if x < 0 {
        return -1
    } else if x > 0 {
        return 1
    }
    return 0
}

func main() {
    var a, b float32
    fmt.Print("A = ")
    fmt.Scan(&a)
    fmt.Print("B = ")
    fmt.Scan(&b)
    result := Sign(a) + Sign(b)
    fmt.Printf("result = %d\n", result)
}
```

Proc 17

Функсияи типи бутуни RootsCount(A,B,C)-ро тасвир кунед, ки миқдори решаҳои муодилаи квадратии $A*x^2+B*x+C=0$-ро муайян мекунад (A,B,C - параметрҳои ҳақиқӣ, A<>0). Бо ёрии он миқдори решаҳоро барои ҳар яке аз се муодилаҳои квадратӣ бо коэффитсиентҳои додашуда ёбед. Миқдори решаҳо аз рӯи қимати дискриминант муайян мегардад: $D=B^2-4*A*C$.

Write an integer function RootCount(*A*, *B*, *C*) that returns the amount of roots of the quadratic equation $A \cdot x^2 + B \cdot x + C = 0$ (*A*, *B*, *C* are real-valued parameters, $A \neq 0$). Using this function, find the amount of roots for each of three quadratic equations with given coefficients. Note that the amount of roots is determined by the value of a *discriminant*: $D = B^2 - 4 \cdot A \cdot C$

```
package main

import "fmt"

func RootCount(a float32, b float32, c float32) int {
    d := b*b - 4*a*c
    roots := 0
    if d > 0 {
        roots += 2
    } else if d == 0 {
        roots++
    }
    return roots
}

func main() {
    var a, b, c float32
    for i := 1; i <= 3; i++ {
        fmt.Printf("A%d = ", i)
        fmt.Scan(&a)
        fmt.Printf("B%d = ", i)
        fmt.Scan(&b)
        fmt.Printf("C%d = ", i)
        fmt.Scan(&c)
        fmt.Printf("RootCount(A%d, B%d, C%d) = %d\n\n", i, i, i, RootCount(a,
b, c))
    }
}
```

Proc 18

Функсияи типи ҳақиқии CircleS(R)-ро тасвир кунед, ки масоҳати доираи радиусаш R (R-ҳақиқӣ)-ро меёбад. Бо ёрии ин функсия масоҳати се доираҳоро бо радиусҳои додашуда ёбед. Масоҳати доираи радиусаш R аз рӯи формулаи s=pi*R^2 муайян карда мешавад. Ба сифати қимати pi: 3.14-ро истифода баред.

Write a real-valued function CircleS(*R*) that returns the area of a circle of radius *R* (*R* is a real number). Using this function, find the areas of three circles of given radiuses. Note that the area of

a circle of radius *R* can be found by formula $S = \pi \cdot R^2$. Use 3.14 for a value of π.

```
package main

import "fmt"

func CircleS(r float32) float32 {
    return 3.14 * r * r
}

func main() {
    var r, s float32
    for i := 1; i <= 3; i++ {
        fmt.Printf("R%d = ", i)
        fmt.Scan(&r)
        s = CircleS(r)
        fmt.Printf("S%d = %.2f\n\n", i, s)
    }
}
```

Proc 19

Функсияи типи ҳақиқии RingS(R_1,R_2)-ро тасвир кунед, ки масоҳати ҳалқаи дар байни ду даврaҳои марказашон умумӣ, ки радиусҳои R_1 ва R_2-ро доростанд (R_1 ва R_2 - ҳақиқианд, $R_1>R_2$), меёбад. Бо ёрии он масоҳатҳои се ҳалқаҳоро, ки барои онҳо радиусҳои дохилӣ ва беруна дода шудаанд, ёбед. Аз формулаи масоҳати доираи радиусаш R: s=pi*R^2 истифода баред. Ба сифати қимати pi: 3.14-ро истифода баред.

Write a real-valued function RingS(R_1, R_2) that returns the area of a ring bounded by a concentric circles of radiuses R_1 and R_2 (R_1 and R_2 are real numbers, $R_1 > R_2$). Using this function, find the areas of three rings with given outer and inner radiuses. Note that the area of a circle of radius *R* can be found by formula $S = \pi \cdot R^2$. Use 3.14 for a value of π.

```
package main

import "fmt"

func RingS(r1 float32, r2 float32) float32 {
    const PI = 3.14
    s1 := PI * r1 * r1
    s2 := PI * r2 * r2
```

```
    return s1 - s2
}

func main() {
    var r1, r2, s float32
    for i := 1; i <= 3; i++ {
        fmt.Printf("R1 = ")
        fmt.Scan(&r1)
        fmt.Printf("R2 = ")
        fmt.Scan(&r2)
        s = RingS(r1, r2)
        fmt.Printf("RingS(%.2f, %.2f) = %.2f\n\n", r1, r2, s)
    }
}
```

Proc 20

Функсияи TriangleP(a,h)-ро тасвир кунед, ки периметри секунҷаи баробарпаҳлӯро аз рӯи асоси он a ва баландии h, ки ба асос фароварда шудааст (a ва h - ҳақиқианд) меёбад. Бо ёрии ин функсия периметрҳои се секунҷаҳоро, ки барои онҳо асосҳо ва баландиҳо дода шудаанд, ёбед. Барои ёфтани тарафи паҳлӯии секунҷа b теоремаи Пифагорро истифода баред: $b^2=(a / 2)^2+h^2$.

Write a real-valued function TriangleP(*a*, *h*) that returns the perimeter of an isosceles triangle with given base *a* and altitude *h* (*a* and *h* are real numbers). Using this function, find the perimeters of three triangles with given bases and altitudes. Note that the leg *b* of an isosceles triangle can be found by the *Pythagorean theorem*:$b^2 = (a/2)^2 + h^2$

```
package main

import (
    "fmt"
    "math"
)

func TriangleP(a float64, h float64) float64 {
    b := math.Sqrt(math.Pow(a/2, 2) + h*h)
    return a + 2*b
}

func main() {
    var a, h, p float64
    for i := 1; i <= 3; i++ {
        fmt.Printf("a%d = ", i)
        fmt.Scan(&a)
        fmt.Printf("h%d = ", i)
```

```
        fmt.Scan(&h)
        p = TriangleP(a, h)
        fmt.Printf("P%d = %.2f\n\n", i, p)
    }
}
```

Minmax 1

Адади бутуни N ва маҷмӯъ аз N ададҳо дода шудаанд. Элементҳои хурдтарин ва калонтарини маҷмӯи мазкурро ёбед ва бо тартиби нишондодашуда хориҷ кунед.

An integer *N* and a sequence of *N* real numbers are given. Find the minimal element and the maximal element of the sequence (that is, elements with the minimal value and the maximal value respectively).

```
package main

import "fmt"

func main() {
    var (
        n int
        number, min, max float32
        inited bool
    )
    fmt.Print("N = ")
    fmt.Scan(&n)
    for i := 0; i < n; i++ {
        fmt.Scan(&number)
        if !inited {
            min, max, inited = number, number, true
        } else if number < min {
            min = number
        } else if number > max {
            max = number
        }
    }
    fmt.Printf("Min = %.2f\nMax = %.2f\n", min, max)
}
```

Minmax 2

Адади бутуни N ва маҷмӯъ аз N росткунҷаҳо бо тарафҳои додашуда — ҷуфтҳои ададҳо (a, b) дода шудаанд. Масоҳати хурдтарини росткунҷаро аз маҷмӯи мазкур ёбед.

An integer *N* and a sequence of *N* rectangles are given. Each rectangle is defined by a pair of its sides (*a*, *b*). Find the rectangle with the minimal area and output the value of its area.

```
package main

import "fmt"

func main() {
    var (
        n int
        a, b, s, minS float32
        inited bool
    )
    fmt.Print("N = ")
    fmt.Scan(&n)
    for i := 1; i <= n; i++ {
        fmt.Printf("a%d = ", i)
        fmt.Scan(&a)
        fmt.Printf("b%d = ", i)
        fmt.Scan(&b)
        s = a * b
        if !inited {
            minS, inited = s, true
        } else if s < minS {
            minS = s
        }
    }
    fmt.Printf("minS = %.2f\n", minS)
}
```

Minmax 3

Ададӣ бутуни N ва маҷмӯъ аз N росткунҷаҳо бо тарафҳои додашуда - ҷуфтҳои ададҳо (a, b) дода шудаанд. Периметри калонтарини росткунҷаро аз маҷмӯи мазкур ёбед.

An integer *N* and a sequence of *N* rectangles are given. Each rectangle is defined by a pair of its sides (*a*, *b*). Find the rectangle with the maximal perimeter and output the value of its perimeter.

```
package main

import "fmt"

func main() {
    var (
        n int
        a, b, p, maxP float32
        inited bool
    )
```

```
    fmt.Print("N = ")
    fmt.Scan(&n)
    for i := 1; i <= n; i++ {
        fmt.Printf("a%d = ", i)
        fmt.Scan(&a)
        fmt.Printf("b%d = ", i)
        fmt.Scan(&b)
        p = 2 * (a + b)
        if !inited {
            maxP, inited = p, true
        } else if p > maxP {
            maxP = p
        }
    }
    fmt.Printf("maxP = %.2f\n", maxP)
}
```

Minmax 4

Адади бутуни N ва мачмӯъ аз N ададҳо дода шудаанд. Рақами элементи хурдтаринро аз мачмӯи мазкур ёбед.

An integer *N* and a sequence of *N* real numbers are given. Find the order number of the minimal element of the sequence.

```
package main

import "fmt"

func main() {
    var (
        n, minIndex int
        number, min float32
        inited bool
    )
    fmt.Print("N = ")
    fmt.Scan(&n)
    for i := 1; i <= n; i++ {
        fmt.Scan(&number)
        if !inited {
            min, minIndex, inited = number, i, true
        } else if number < min {
            min, minIndex = number, i
        }
    }
    fmt.Printf("minIndex = %d\n", minIndex)
}
```

Minmax 5

Адади бутуни N ва мачмӯъ аз N ҷуфти ададҳо (m, v) — маълумот дар бораи вазн m ва ҳачми v ҷузъиёт, ки аз материалҳои гуногун тайёр карда шудаанд, дода шудаанд.

Рақами ҷузъиётеро, ки аз материали дорои зичии калонтарин тайёр карда шудааст, ҳамчунин бузургии ин зичии калонтаринро хориҷ кунед. Зичӣ P аз рӯи формулаи P = m/v ҳисоб карда мешавад.

An integer *N* and a sequence of *N* pairs of real numbers (*m*, *v*) are given. Each pair of given numbers contains the weight *m* and the volume *v* of a detail that is made of some homogeneous material. Output the order number of a detail that is made of the material with maximal density. Also output the corresponding density. Note that the density *P* can be found by formula *P* = *m*/*v*.

```
package main

import "fmt"

func main() {
    var (
        n, maxIndex int
        m, v, p, maxP float32
        inited bool
    )
    fmt.Print("N = ")
    fmt.Scan(&n)
    for i := 1; i <= n; i++ {
        fmt.Printf("m%d = ", i)
        fmt.Scan(&m)
        fmt.Printf("v%d = ", i)
        fmt.Scan(&v)
        p = m / v
        if !inited {
            maxP, maxIndex, inited = p, i, true
        } else if p > maxP {
            maxP, maxIndex = p, i
        }
    }
    fmt.Printf("index = %d\t\tmaxP = %.2f\n", maxIndex, maxP)
}
```

Minmax 6

Адади бутуни N ва маҷмӯъ аз N ададҳои ҳақиқӣ дода шудаанд. Рақами элементи якӯми хурдтарин ва рақами элементи охирини калонтаринро аз маҷмӯи мазкур ёбед ва онҳоро бо тартиби нишондодашуда хориҷ кунед.

An integer *N* and a sequence of *N* integers are given. Find the order numbers of the first minimal element and the last maximal element of the sequence.

```
package main

import "fmt"

func main() {
    var n, number, min, max, minIndex, maxIndex int
    fmt.Print("N = ")
    fmt.Scan(&n)
    for i := 1; i <= n; i++ {
        fmt.Scan(&number)
        if i == 1 {
            min, max = number, number
            minIndex, maxIndex = i, i
        } else if number >= max {
            max, maxIndex = number, i
        } else if number < min {
            min, minIndex = number, i
        }
    }
    fmt.Printf("minIndex = %d\t\tmaxIndex = %d\n", minIndex, maxIndex)
}
```

Minmax 7

Адади бутуни N ва маҷмӯъ аз N ададҳои ҳақиқӣ дода шудаанд. Рақами элементи якӯми калонтарин ва рақами элементи охирини хурдтаринро аз маҷмӯи мазкур ёбед ва онҳоро бо тартиби нишондодашуда хориҷ кунед.

An integer *N* and a sequence of *N* integers are given. Find the order numbers of the first maximal element and the last minimal element of the sequence.

```
package main

import "fmt"

func main() {
    var n, number, min, max, minIndex, maxIndex int
    fmt.Print("N = ")
    fmt.Scan(&n)
    for i := 1; i <= n; i++ {
        fmt.Scan(&number)
        if i == 1 {
            min, max = number, number
            minIndex, maxIndex = i, i
        } else if number > max {
            max, maxIndex = number, i
```

```
        } else if number <= min {
            min, minIndex = number, i
        }
    }
    fmt.Printf("maxIndex = %d\t\tminIndex = %d\n", maxIndex, minIndex)
}
```

Minmax 8

Адади бутуни N ва маҷмӯъ аз N ададҳои ҳақиқӣ дода шудаанд. Рақамҳои элементҳои хурдтарини якӯм ва охиринро аз маҷмӯи мазкур ёбед ва онҳоро бо тартиби нишондодашуда хориҷ кунед.

An integer *N* and a sequence of *N* integers are given. Find the order numbers of the first and the last minimal elements of the sequence.

```
package main

import "fmt"

func main() {
    var n, number, min, firstMinIndex, lastMinIndex int
    var inited bool
    fmt.Print("N = ")
    fmt.Scan(&n)
    for i := 1; i <= n; i++ {
        fmt.Scan(&number)
        if !inited {
            min, firstMinIndex, lastMinIndex, inited = number, i, i, true
        } else if number <= min {
            if number < min {
                min, firstMinIndex, lastMinIndex = number, i, i
            } else if number == min {
                lastMinIndex = i
            }
        }
    }
    fmt.Printf("firstMinIndex = %d\t\tlastMinIndex = %d\n", firstMinIndex,
lastMinIndex)
}
```

Minmax 9

Адади бутуни N ва маҷмӯъ аз N ададҳои ҳақиқӣ дода шудаанд. Рақамҳои элементҳои калонтарини якӯм ва охиринро аз маҷмӯи мазкур ёбед ва онҳоро бо тартиби нишондодашуда хориҷ кунед.

An integer *N* and a sequence of *N* integers are given. Find the order numbers of the first and the last maximal elements of the sequence.

```
package main

import "fmt"

func main() {
    var n, number, max, firstMaxIndex, lastMaxIndex int
    fmt.Print("N = ")
    fmt.Scan(&n)
    for i := 1; i <= n; i++ {
        fmt.Scan(&number)
        if i == 1 {
            max, firstMaxIndex, lastMaxIndex = number, i, i
        } else if number >= max {
            if number > max {
                max, firstMaxIndex, lastMaxIndex = number, i, i
            } else if number == max {
                lastMaxIndex = i
            }
        }
    }
    fmt.Printf("firstMaxIndex = %d\t\tlastMinIndex = %d\n", firstMaxIndex,
lastMaxIndex)
}
```

Minmax 10

Адади бутуни N ва мачмӯъ аз N ададҳои ҳақиқӣ дода шудаанд. Рақами элементи якӯми экстремалӣ (яъне калонтарин ё хурдтарин)-ро аз мачмӯи мазкур ёбед.

An integer *N* and a sequence of *N* integers are given. Find the order number of the first *extremal* (that is, minimal or maximal) element of the sequence.

```
package main

import "fmt"

func main() {
    var n, number, min, max, minIndex, maxIndex int
    fmt.Print("N = ")
    fmt.Scan(&n)
    for i := 1; i <= n; i++ {
        fmt.Scan(&number)
        if i == 1 {
            min, max, minIndex, maxIndex = number, number, i, i
        } else if number > max {
            max, maxIndex = number, i
        } else if number < min {
```

```
            min, minIndex = number, i
        }
    }
    if minIndex < maxIndex {
        fmt.Println(minIndex)
    } else {
        fmt.Println(maxIndex)
    }
}
```

Minmax 11

Адади бутуни N ва маҷмӯъ аз N ададҳои ҳақиқӣ дода шудаанд. Рақами элементи охирини экстремалӣ (яъне калонтарин ё хурдтарин)-ро аз маҷмӯи мазкур ёбед.

An integer *N* and a sequence of *N* integers are given. Find the order number of the last *extremal* (that is, minimal or maximal) element of the sequence.

```
package main

import "fmt"

func main() {
    var n, number, min, max, minIndex, maxIndex int
    fmt.Print("N = ")
    fmt.Scan(&n)
    for i := 1; i <= n; i++ {
        fmt.Scan(&number)
        if i == 1 {
            min, max, minIndex, maxIndex = number, number, i, i
        } else if number >= max {
            max, maxIndex = number, i
        } else if number <= min {
            min, minIndex = number, i
        }
    }
    if minIndex > maxIndex {
        fmt.Println(minIndex)
    } else {
        fmt.Println(maxIndex)
    }
}
```

Minmax 12

Адади бутуни N ва маҷмӯъ аз N ададҳо дода шудаанд. Адади мусбии хурдтаринро аз маҷмӯи мазкур ёбед. Агар дар маҷмӯъ адади мусбӣ набошад, пас 0(нул)-ро хориҷ кунед.

An integer *N* and a sequence of *N* real numbers are given. Output the minimal positive number contained in the sequence. If the sequence does not contain positive numbers then output 0.

```
package main

import "fmt"

func main() {
    var (
        n int
        number, minPositive float32
        inited bool
    )
    fmt.Print("N = ")
    fmt.Scan(&n)
    for i := 1; i <= n; i++ {
        fmt.Scan(&number)
        if number > 0 {
            if !inited {
                minPositive, inited = number, true
            } else if number < minPositive {
                minPositive = number
            }
        }
    }
    fmt.Printf("minPositive = %.2f\n", minPositive)
}
```

Minmax 13

Адади бутуни N ва маҷмӯъ аз N ададҳои бутун дода шудаанд. Рақами адади токи калонтарини аввалинро аз маҷмӯи мазкур ёбед. Агар адади тоқ дар маҷмӯъ набошад, пас 0(нул)-ро хориҷ кунед.

An integer *N* and a sequence of *N* integers are given. Output the order number of the first maximal odd number contained in the sequence. If the sequence does not contain odd numbers then output 0.

```
package main

import "fmt"

func main() {
    var n, number, maxOdd, maxIndex int
    var inited bool
    fmt.Print("N = ")
    fmt.Scan(&n)
```

```
    for i := 1; i <= n; i++ {
        fmt.Scan(&number)
        if number % 2 != 0 {
            if !inited {
                maxOdd, maxIndex, inited = number, i, true
            } else if number > maxOdd {
                maxOdd, maxIndex = number, i
            }
        }
    }
    fmt.Printf("maxOddIndex = %d\n", maxIndex)
}
```

Minmax 14

Адади B(>0) ва маҷмӯъ аз даҳ ададҳо дода шудаанд. Аз дохили он элементҳои маҷмӯъ, ки аз B калон ҳастанд, элементи хурдтаринашро, ҳамчунин рақами тартибии онро хориҷ кунед. Агар дар маҷмӯъ адади аз B калон набошад, пас ду маротиба 0(нул)-ро хориҷ кунед.

A positive real number *B* and a sequence of 10 real numbers are given. Find the minimum among elements that are greater than *B* and output its value and its order number. If the sequence does not contain elements greater than *B* then output 0 twice (the first zero as a real number, the second zero as an integer).

```
package main

import "fmt"

func main() {
    var (
        b, number, min float32
        minIndex int
        inited bool
    )
    fmt.Print("B = ")
    fmt.Scan(&b)
    for i := 1; i <= 10; i++ {
        fmt.Scan(&number)
        if number > b {
            if !inited {
                min, minIndex, inited = number, i, true
            } else if number < min {
                min, minIndex = number, i
            }
        }
    }
    fmt.Printf("min = %.2f\t\tminIndex = %d\n", min, minIndex)
}
```

Minmax 15

Ададҳои B, C (B>0, C>B) ва маҷмӯъ аз даҳ ададҳо дода шудаанд. Аз дохили он элементҳои маҷмӯъ, ки дар фосилаи (B,C) мехобанд, калонтаринашро, ҳамчунин рақами тартибии онро хориҷ кунед. Агар дар маҷмӯъ ададе талабкардашуда набошад, пас ду маротиба 0(нул)-ро хориҷ кунед.

Two real numbers *B*, *C* (0 < *B* < *C*) and a sequence of 10 real numbers are given. Find the maximum among elements that are in the interval (*B*, *C*) and output its value and its order number. If the sequence does not contain elements in the interval (*B*, *C*) then output 0 twice (the first zero as a real number, the second zero as an integer).

```
package main

import "fmt"

func main() {
    var (
        b, c, number, max float32
        maxIndex int
        inited bool
    )
    fmt.Print("B = ")
    fmt.Scan(&b)
    fmt.Print("C = ")
    fmt.Scan(&c)
    for i := 1; i <= 10; i++ {
        fmt.Scan(&number)
        if b < number && number < c {
            if !inited {
                max, maxIndex, inited = number, i, true
            } else if number > max {
                max, maxIndex = number, i
            }
        }
    }
    fmt.Printf("max = %.2f\t\tminIndex = %d\n", max, maxIndex)
}
```

Minmax 16

Адади бутуни N ва маҷмӯъ аз N ададҳои бутун дода шудаанд. Миқдори элементҳоеро, ки пеш аз элементи хурдтарини аввалин ҷойгир шудаанд, ёбед.

An integer *N* and a sequence of *N* integers are given. Find the amount of the elements that are located before the first minimal element.

```
package main

import "fmt"

func main() {
    var n, number, min, minIndex int
    fmt.Print("N = ")
    fmt.Scan(&n)
    for i := 1; i <= n; i++ {
        fmt.Scan(&number)
        if i == 1 {
            min, minIndex = number, i
        } else if number < min {
            min, minIndex = number, i
        }
    }
    fmt.Printf("elements = %d\n", minIndex-1)
}
```

Minmax 17

Адади бутуни N ва маҷмӯъ аз N ададҳои бутун дода шудаанд. Миқдори элементҳоеро, ки пас аз элементи калонтарини охирин ҷойгир шудаанд, ёбед.

An integer *N* and a sequence of *N* integers are given. Find the amount of the elements that are located after the last maximal element.

```
package main

import "fmt"

func main() {
    var n, number, max, maxIndex int
    fmt.Print("N = ")
    fmt.Scan(&n)
    for i := 1; i <= n; i++ {
        fmt.Scan(&number)
        if i == 1 {
            max, maxIndex = number, i
        } else if number >= max {
```

```
            max, maxIndex = number, i
        }
    }
    fmt.Printf("elements = %d\n", n - maxIndex)
}
```

Minmax 18

Адади бутуни N ва маҷмӯъ аз N ададҳои бутун дода шудаанд. Миқдори элементҳоеро, ки дар байни элементҳои калонтарини аввалин ва охирин ҷойгир шудаанд, ёбед. Агар дар маҷмӯъ як то элементи калонтарин мавҷуд бошад, пас 0(нул)-ро хориҷ кунед.

An integer *N* and a sequence of *N* integers are given. Find the amount of the elements that are located between the first and the last maximal element. If the sequence contains the unique maximal element then output 0.

```
package main

import "fmt"

func main() {
    var n, number, max, firstMaxIndex, lastMaxIndex int
    fmt.Print("N = ")
    fmt.Scan(&n)
    for i := 1; i <= n; i++ {
        fmt.Scan(&number)
        if i == 1 {
            max, firstMaxIndex, lastMaxIndex = number, i, i
        } else if number > max {
            max, firstMaxIndex, lastMaxIndex = number, i, i
        } else if number == max {
            lastMaxIndex = i
        }
    }
    elements := 0
    if lastMaxIndex > firstMaxIndex {
        elements = lastMaxIndex - firstMaxIndex - 1
    }
    fmt.Printf("elements = %d\n", elements)
}
```

Minmax 19

Адади бутуни N ва маҷмӯъ аз N ададҳои бутун дода шудаанд. Миқдори элементҳои хурдтаринро аз маҷмӯи мазкур ёбед.

An integer *N* and a sequence of *N* integers are given. Find the amount of the minimal elements of the sequence.

```
package main

import "fmt"

func main() {
    var n, number, min, count int
    fmt.Print("N = ")
    fmt.Scan(&n)
    for i := 0; i < n; i++ {
        fmt.Scan(&number)
        if i == 0 {
            min, count = number, 1
        } else if number < min {
            min, count = number, 1
        } else if number == min {
            count++
        }
    }
    fmt.Printf("count = %d\n", count)
}
```

Minmax 20

Адади бутуни N ва маҷмӯъ аз N ададҳои бутун дода шудаанд. Миқдори умумии элементҳои экстремалӣ (яъне калонтарин ва хурдтарин)-ро аз маҷмӯи мазкур ёбед.

An integer *N* and a sequence of *N* integers are given. Find the total amount of all *extremal* (that is, minimal or maximal) elements of the sequence.

```
package main

import "fmt"

func main() {
    var n, number, min, max, minCount, maxCount int
    fmt.Print("N = ")
    fmt.Scan(&n)
    for i := 0; i < n; i++ {
        fmt.Scan(&number)
        if i == 0 {
            min, max, minCount, maxCount = number, number, 1, 1

        } else if number < min {
            min, minCount = number, 1
        } else if number == min {
            minCount++

        } else if number > max {
```

```
            max, maxCount = number, 1
        } else if number == max {
            maxCount++
        }
    }
    fmt.Printf("count = %d\n", minCount + maxCount)
}
```

Array 1

Адади бутуни N(>0) дода шудааст. Массиви типи бутуни андозаи N-ро тартиб дода, хориҷ кунед, ки N-то ададҳои тоқи мусбии аввалинро доро бошад: 1, 3, 5, … .

Given an integer *N* (> 0), create and output an array of *N* integers that are the first positive odd numbers: 1, 3, 5, … .

```
package main

import "fmt"

func main() {
    var n int
    fmt.Print("N = ")
    fmt.Scan(&n)
    arr := make([]int, n)
    for index, _ := range arr {
        arr[index] = 2 * index + 1
    }
    fmt.Println(arr)
}
```

Array 2

Адади бутуни N(>0) дода шудааст. Массиви типи бутуни андозаи N-ро тартиб дода, хориҷ кунед, ки дараҷаҳои адади 2(ду)-ро аз якӯм то N-ӯм доро бошад: 2, 4, 8, 16, … .

Given an integer *N* (> 0), create and output an array of *N* integers that are the first positive integer powers of 2: 2, 4, 8, 16, … .

```
package main

import "fmt"

func main() {
```

```
    var n int
    fmt.Print("N = ")
    fmt.Scanf("%d", &n)
    var arr []int = make([]int, n)
    for index, _ := range arr {
        if index == 0 {
            arr[index] = 2
        } else {
            arr[index] = arr[index-1] + arr[index-1]
        }
    }
    fmt.Println(arr)
}
```

Array 3

Адади бутуни N(>1), ҳамчунин аъзои аввалин A ва фарқи прогрессияи арифметикӣ D дода шудаанд. Массиви андозаи N-ро тартиб дода, хориҷ кунед, ки N-то аъзоҳои аввалаи прогрессияи зеринро доро бошад: A, A+D, A+2*D, A+3*D, … .

An integer N (> 1), the first term A and the common difference D of an *arithmetic sequence* are given (A and D are real numbers). Create and output an array of N real numbers that are the initial terms of this sequence:
$A, \quad A + D, \quad A + 2 \cdot D, \quad A + 3 \cdot D, \quad \ldots .$

```
package main

import "fmt"

func main() {
    var (
        n int
        a, d float32
    )
    fmt.Print("N = ")
    fmt.Scan(&n)
    fmt.Print("A = ")
    fmt.Scan(&a)
    fmt.Print("D = ")
    fmt.Scan(&d)
    var arr []float32 = make([]float32, n)
    for index, _ := range arr {
        if index == 0 {
            arr[index] = a
        } else {
            arr[index] = arr[index-1] + d
        }
    }
    for _, value := range arr {
```

```
        fmt.Printf("%.2f\t", value)
    }
}
```

Array 4

Адади бутуни N(>1), ҳамчунин аъзои аввала A ва маърачи прогрессияи геометрӣ Q дода шудаанд. Массиви андозаи N-ро тартиб дода, хориҷ кунед, ки N-то аъзоҳои аввалаи прогрессияи зеринро доро бошад: A, A*Q, A*Q^2, A*Q^3, … .

An integer *N* (> 1), the first term *A* and the common ratio *R* of a *geometric sequence* are given (*A* and *D* are real numbers). Create and output an array of *N* real numbers that are the initial terms of this sequence: A, $A \cdot R$, $A \cdot R^2$, $A \cdot R^3$, … .

```
package main

import "fmt"

func main() {
    var (
        n int
        a, d float32
    )
    fmt.Print("N = ")
    fmt.Scan(&n)
    fmt.Print("A = ")
    fmt.Scan(&a)
    fmt.Print("D = ")
    fmt.Scan(&d)
    var array [] float32 = make([]float32, n)
    for index, _ := range array {
        if index == 0 {
            array[index] = a
        } else {
            array[index] = array[index-1] * d
        }
    }
    for _, value := range array {
        fmt.Printf("%.2f\t", value)
    }
}
```

Array 5

Адади бутуни N (>2) дода шудааст. Массиви типи бутуни андозаи N-ро тартиб дода, хориҷ кунед, ки N-то элементҳои

аввалаи пайдарпайии ададҳои Фибоначчиро доро бошад: F[k]: F[1]=1, F[2]=1, F[k]=F[k–2]+F[k–1], k=3, 4, … .

Given an integer N (> 2), create and output an array of N integers that are the initial terms of a sequence $\{F_K\}$ of the *Fibonacci numbers*:
$F_1 = 1, \quad F_2 = 1, \quad F_K = F_{K-2} + F_{K-1}, \quad K = 3, 4, \ldots .$

```
package main

import "fmt"

func main() {
    var n int
    fmt.Print("N = ")
    fmt.Scanf("%d", &n)
    var array []int = make([]int, n)
    array[0], array[1] = 1, 1
    for index := 2; index < n; index++ {
        array[index] = array[index-1] + array[index-2]
    }
    fmt.Println(array)
}
```

Array 6

Ададҳои бутуни N(>2), A ва B дода шудаанд. Массиви типи бутуни андозаи N-ро тартиб дода, хориҷ кунед, ки элементи якӯми он баробари A буда, дуюмаш баробари B аст ва ҳар як элементи оянда баробари суммаи ҳамаи элементҳои пешина аст.

Given three integers N (> 2), A, B, create and output an array of N integers such that the first element is equal to A, the second one is equal to B, and each subsequent element is equal to the sum of all previous ones.

```
package main

import "fmt"

func main() {
    var n, a, b int
    fmt.Print("N = ")
    fmt.Scan(&n)
    fmt.Print("A = ")
    fmt.Scan(&a)
    fmt.Print("B = ")
```

```
    fmt.Scan(&b)
    var array []int = make([]int, n)
    array[0], array[1] = a, b
    var sum int = a + b
    for index := 2; index < n; index++ {
        array[index] = sum
        sum += array[index]
    }
    fmt.Println(array)
}
```

Array 7

Массиви андозаи N дода шудааст. Элементҳои онро бо тартиби баръакс хориҷ кунед.

Given an array of *N* real numbers, output its elements in inverse order.

```
package main

import "fmt"

func main() {
    var n int
    fmt.Print("N = ")
    fmt.Scan(&n)
    var array []float32 = make([]float32, n)
    for index, _ := range array {
        fmt.Scan(&array[index])
    }
    for index := len(array)-1; index >= 0; index-- {
        fmt.Printf("%.2f\t", array[index])
    }
}
```

Array 8

Массиви типи бутуни андозаи N дода шудааст. Ҳамаи ададҳои тоқи дар массиви мазкур мавҷударо бо тартиби афзуншавии рақамҳои тартибиашон хориҷ кунед ва ҳамчунин миқдори онҳоро низ хориҷ кунед.

Given an array of *N* integers, output all odd numbers contained in the array in ascending order of their indices. Also output the amount *K* of odd numbers contained in the array.

```
package main

import "fmt"
```

```
func main() {
    var n, k int
    fmt.Print("N = ")
    fmt.Scan(&n)
    var array []int = make([]int, n)
    for index, _ := range array {
        fmt.Scan(&array[index])
    }
    for _, value := range array {
        if value % 2 != 0 {
            fmt.Printf("%d\t", value)
            k++
        }
    }
    fmt.Printf("\nK = %d\n", k)
}
```

Array 9

Массиви типи бутуни андозаи N дода шудааст. Ҳамаи ададҳои ҷуфти дар массиви мазкур мавҷударо бо тартиби камшавии рақамҳои тартибиашон хориҷ кунед ва ҳамчунин миқдори онҳоро низ хориҷ кунед.

Given an array of *N* integers, output all even numbers contained in the array in descending order of their indices. Also output the amount *K* of even numbers contained in the array.

```
package main

import "fmt"

func main() {
    var n, k int
    fmt.Print("N = ")
    fmt.Scan(&n)
    var array []int = make([]int, n)
    for index, _ := range array {
        fmt.Scan(&array[index])
    }
    for index := len(array) - 1; index >= 0; index-- {
        if array[index] % 2 == 0 {
            fmt.Printf("%d\t", array[index])
            k++
        }
    }
    fmt.Printf("\nK = %d\n", k)
}
```

Array 10

Массиви типи бутуни андозаи N дода шудааст. Дар аввал ҳамаи ададҳои ҷуфти дар массиви мазкур мавҷударо бо тартиби афзуншавии рақамҳои тартибиашон, сонӣ ҳамаи ададҳои тоқро бо тартиби камшавии рақамҳои тартибиашон хориҷ кунед.

Given an array of *N* integers, output all even numbers contained in the array in ascending order of their indices and then output all odd numbers contained in the array in descending order of their indices.

```
package main

import "fmt"

func main() {
    var n int
    fmt.Print("N = ")
    fmt.Scan(&n)
    var array []int = make([]int, n)
    for index, _ := range array {
        fmt.Scan(&array[index])
    }
    for _, value := range array {
        if value % 2 == 0 {
            fmt.Printf("%d\t", value)
        }
    }
    for index := len(array) - 1; index >= 0; index-- {
        if array[index] % 2 != 0 {
            fmt.Printf("%d\t", array[index])
        }
    }
}
```

Array 11

Массиви A бо андозаи N ва ададӣ бутуни K (1≤K≤N) дода шудаанд. Элементҳои массивро бо рақамҳои тартибии ба ададӣ K каратӣ хориҷ кунед: A[k], A[2*k], A[3*k], … . Операторӣ шартиро истифода набаред.

An array *A* of *N* real numbers and an integer *K* ($1 \le K \le N$) are given. Output array elements with order numbers that are multiples of *K*: A_K, $A_{2 \cdot K}$, $A_{3 \cdot K}$, … . Do not use conditional statements.

```
package main

import "fmt"

func main() {
    var n, k int
    fmt.Print("N = ")
    fmt.Scan(&n)
    // var a []float32 = make([]float32, n)
    a := make([]float32, n)
    for index, _ := range a {
        fmt.Scan(&a[index])
    }
    fmt.Print("K = ")
    fmt.Scan(&k)
    for index := k-1; index < n; index += k {
        fmt.Printf("%.2f\t", a[index])
    }
}
```

Array 12

Массиви A бо андозаи N (N - адади ҷуфт) дода шудааст. Элементҳои онро, ки дорои рақамҳои тартибии ҷуфт ҳастанд, бо тартиби афзуншавии рақамҳои тартибӣ хориҷ кунед: A[2], A[4], A[6], .., A[n]. Оператори шартиро истифода набаред.

An array A of N real numbers is given (N is an even number). Output array elements with even order numbers in ascending order of indices: A_2, A_4, A_6, …, A_N. Do not use conditional statements.

```
package main

import "fmt"

func main() {
    var n int
    fmt.Print("N = ")
    fmt.Scan(&n)
    a := make([]float32, n)
    for index, _ := range a {
        fmt.Scan(&a[index])
    }
    for index := 1; index < n; index += 2 {
        fmt.Printf("%.2f\t", a[index])
    }
}
```

Array 13

Массиви A бо андозаи N (N — адади тоқ) дода шудааст. Элементҳои онро, ки дорои рақамҳои тартибии тоқ ҳастанд, бо тартиби камшавии рақамҳои тартибӣ хориҷ кунед: A[n], A[n-2], A[n-4], …,A[1]. Операторӣ шартиро истифода набаред.

An array *A* of *N* real numbers is given (*N* is an odd number). Output array elements with odd order numbers in descending order of indices: A_N, A_{N-2}, A_{N-4}, …, A_1. Do not use conditional statements.

```
package main

import "fmt"

func main() {
    var n int
    fmt.Print("N = ")
    fmt.Scan(&n)
    a := make([]float32, n)
    for index, _ := range a {
        fmt.Scan(&a[index])
    }
    for index := len(a) - 1; index >= 0; index -= 2 {
        fmt.Printf("%.2f\t", a[index])
    }
}
```

Array 14

Массиви A бо андозаи N дода шудааст. Дар аввал элементҳои онро, ки дорои рақамҳои тартибии ҷуфт ҳастанд (бо тартиби афзуншавии рақамҳои тартибӣ), сонӣ элементҳоеро, ки дорои рақамҳои тартибии тоқ ҳастанд (ҳамчунин бо тартиби афзуншавии рақамҳои тартибӣ), хориҷ кунед: A[2], A[4], A[6], …, A[1], A[3], A[5], … .

An array *A* of *N* real numbers is given. Output array elements with even order numbers (in ascending order of indices) and then output array elements with odd order numbers (in ascending order of indices too):
A_2, A_4, A_6, …, A_1, A_3, A_5, … .Do not use conditional statements.

```
package main

import "fmt"

func main() {
    var n int
    fmt.Print("N = ")
    fmt.Scan(&n)
    a := make([]float32, n)
    for index, _ := range a {
        fmt.Scan(&a[index])
    }
    for index := 1; index < n; index += 2 {
        fmt.Printf("%.2f\t", a[index])
    }
    for index := 0; index < n; index += 2 {
        fmt.Printf("%.2f\t", a[index])
    }
}
```

Array 15

Массиви A бо андозаи N дода шудааст. Дар аввал элементҳои онро, ки дорои рақамҳои тартибии тоқ ҳастанд, бо тартиби афзуншавии рақамҳои тартибӣ, сонӣ элементҳоеро, ки дорои рақамҳои тартибии ҷуфт ҳастанд, бо тартиби камшавии рақамҳои тартибӣ хориҷ кунед. A[1], A[3], A[5], .., A[6], A[4], A[2]. Операторӣ шартиро истифода набаред.

An array A of N real numbers is given. Output array elements with odd order numbers in ascending order of indices and then output array elements with even order numbers in descending order of indices: A_1, A_3, A_5, …, A_6, A_4, A_2.Do not use conditional statements.

```
package main

import "fmt"

func main() {
    var n int
    fmt.Print("N = ")
    fmt.Scan(&n)
    a := make([]float32, n)
    for index, _ := range a {
        fmt.Scan(&a[index])
    }
    for index := 0; index < n; index += 2 {
        fmt.Printf("%.2f\t", a[index])
    }
```

```
    index := len(a) - 1
    if n % 2 != 0 {
        index--
    }
    for ; index >= 0; index -= 2 {
        fmt.Printf("%.2f\t", a[index])
    }
}
```

Array 16

Массиви A бо андозаи N дода шудааст. Элементҳои онро бо тартиби зерин хориҷ кунед: A[1], A[n], A[2], A[n-1], A[3], A[n-2], .. .

An array A of N real numbers is given. Output array elements as follows: A_1, A_N, A_2, A_{N-1}, A_3, A_{N-2}, ...

```
package main

import "fmt"

func main() {
    var n int
    fmt.Print("N = ")
    fmt.Scan(&n)
    a := make([]float32, n)
    for index, _ := range a {
        fmt.Scan(&a[index])
    }
    from, to := 0, len(a) - 1
    for from < to {
        fmt.Printf("%.2f\t%.2f\t", a[from], a[to])
        from++
        to--
    }
    if from == to {
        fmt.Printf("%.2f\t", a[from])
    }
}
```

Array 17

Массиви A бо андозаи N дода шудааст. Элементҳои онро бо тартиби зерин хориҷ кунед: A[1], A[2], A[n], A[n-1], A[3], A[4], A[n-2], A[n-3], .. .

An array A of N real numbers is given. Output array elements as follows: A_1, A_2, A_N, A_{N-1}, A_3, A_4, A_{N-2}, A_{N-3}, ...

```
package main
```

```
import "fmt"

func main() {
    var n int
    fmt.Print("N = ")
    fmt.Scan(&n)
    a := make([]float32, n)
    for index, _ := range a {
        fmt.Scan(&a[index])
    }
    from, to := 0, len(a) - 1
    for {
        if from > to { break }
        fmt.Printf("%.2f\t", a[from])
        from++

        if from > to { break }
        fmt.Printf("%.2f\t", a[from])
        from++

        if from > to { break }
        fmt.Printf("%.2f\t", a[to])
        to--

        if from > to { break }
        fmt.Printf("%.2f\t", a[to])
        to--
    }
}
```

Array 18

Массиви A, ки дорои ададҳои ғайринулӣ аст, бо андозаи 10 дода шудааст. Қимати элементи аввалини онро аз он элементҳои A[k]-и он, ки нобаробарии A[k]<A[10]-ро қаноат мекунонанд, хориҷ кунед. Агар чунин элемент набошад, пас 0(нул)-ро хориҷ кунед.

An array A of 10 nonzero integers is given. Output the value of the first element A_K that satisfies the following inequality: $A_K < A_{10}$. If the array does not contain such elements then output 0.

```
package main

import "fmt"

func main() {
    var a [10]int
    for index, _ := range a {
        fmt.Scan(&a[index])
    }
```

```
    var value int
    for index := 0; index < 9; index++ {
        if a[index] < a[9] {
            value = a[index]
            break
        }
    }
    fmt.Printf("%d\n", value)
}
```

Array 19

Массиви типи бутуни A бо андозаи 10 дода шудааст. Рақами тартибии элементи охиринро аз он элементҳои A[k]-и он, ки нобаробарии дукаратаи A[1]<A[k]<A[10]-ро қаноат мекунонанд, хориҷ кунед. Агар чунин элемент набошад, пас 0(нул)-ро хориҷ кунед.

An array A of 10 integers is given. Output the order number of the last element A_K that satisfies the following double inequality: $A_1 < A_K < A_{10}$. If the array does not contain such elements then output 0.

```
package main

import "fmt"

func main() {
    var a [10]int
    for index, _ := range a {
        fmt.Scan(&a[index])
    }
    var nomer int
    for index := 1; index < 9; index++ {
        if a[0] < a[index] && a[index] < a[9] {
            nomer = index + 1
        }
    }
    fmt.Printf("%d\n", nomer)
}
```

Array 20

Массиви андозаи N ва ададҳои бутуни K ва L (1≤K≤L≤N) дода шудаанд. Суммаи элементҳои массивро бо рақамҳои тартибии аз K то L дар якҷоягӣ ёбед.

An array of *N* real numbers and two integers *K* and *L* ($1 \le K \le L \le N$) are given. Find the sum of array elements with the order numbers in the range *K* to *L* inclusively.

```
package main

import "fmt"

func main() {
    var n, k, l int
    fmt.Print("N = ")
    fmt.Scan(&n)
    array := make([]float32, n)
    for index, _ := range array {
        fmt.Scan(&array[index])
    }
    fmt.Print("K = ")
    fmt.Scan(&k)
    fmt.Print("L = ")
    fmt.Scan(&l)
    var sum float32
    for index := k-1; index < l; index++ {
        sum += array[index]
    }
    fmt.Printf("sum = %.2f\n", sum)
}
```

Matrix 1

Ададҳои бутуни мусбии M ва N дода шудаанд. Матритсаи типи бутуни андозаи M x N-ро тартиб диҳед, ки дар он ҳамаи элементҳои сатри I-ӯм дорои қимати 10•I (I = 1, …, M) мебошанд.

Given two positive integers *M* and *N*, create and output an $M \times N$ matrix of integers such that all its elements of the *I*-th row are equal to $10 \cdot I$ ($I = 1, \ldots, M$).

```
package main

import "fmt"

func main() {
    var m, n int
    fmt.Print("M = ")
    fmt.Scan(&m)
    fmt.Print("N = ")
    fmt.Scan(&n)
    var matrix [][]int = make([][]int, m)
    for row, _ := range matrix {
```

```
        matrix[row] = make([]int, n)
        for col, _ := range matrix[row] {
            matrix[row][col] = 10 * (row + 1)
        }
    }
    for _, array := range matrix {
        for _, value := range array {
            fmt.Printf("%d\t", value)
        }
        fmt.Println()
    }
}
```

Matrix 2

Ададҳои бутуни мусбии M ва N дода шудаанд. Матритсаи типи бутуни андозаи M x N-ро тартиб диҳед, ки дар он ҳамаи элементҳои сутуни J-ӯм дорои қимати 5•J (J = 1, …, N) мебошанд.

Given two positive integers *M* and *N*, create and output an *M* × *N* matrix of integers such that all its elements of the *J*-th column are equal to 5·*J* (*J* = 1, …, *N*).

```
package main

import "fmt"

func main() {
    var m, n int
    fmt.Print("M = ")
    fmt.Scan(&m)
    fmt.Print("N = ")
    fmt.Scan(&n)
    var matrix [][]int = make([][]int, m)
    for row, _ := range matrix {
        matrix[row] = make([]int, n)
        for col, _ := range matrix[row] {
            matrix[row][col] = 5 * (col + 1)
        }
    }
    for _, array := range matrix {
        for _, value := range array {
            fmt.Printf("%d\t", value)
        }
        fmt.Println()
    }
}
```

Matrix 3

Ададҳои бутуни мусбии M, N ва мачмӯъ аз M ададҳо дода шудаанд. Матритсаи андозаи M x N-ро тартиб диҳед, ки дар он дар ҳар як сутун ҳамаи ададҳои мачмӯи ибтидоӣ (бо ҳамон тартиб) бошанд.

Two positive integers *M*, *N* and a sequence of *M* real numbers are given. Create and output an $M \times N$ matrix of real numbers such that each of its columns contains all numbers from the given sequence (in the same order).

```
package main

import "fmt"

func main() {
    var m, n int
    fmt.Print("M = ")
    fmt.Scan(&m)
    fmt.Print("N = ")
    fmt.Scan(&n)
    var array []float32 = make([]float32, m)
    for index, _ := range array {
        fmt.Scan(&array[index])
    }
    var matrix [][]float32 = make([][]float32, m)
    for row, _ := range matrix {
        matrix[row] = make([]float32, n)
        for col, _ := range matrix[row] {
            matrix[row][col] = array[row]
        }
    }
    fmt.Println()
    for _, array := range matrix {
        for _, value := range array {
            fmt.Printf("%.2f\t", value)
        }
        fmt.Println()
    }
}
```

Matrix 4

Ададҳои бутуни мусбии M, N ва мачмӯъ аз N ададҳо дода шудаанд. Матритсаи андозаи M x N-ро тартиб диҳед, ки дар он дар ҳар як сатр ҳамаи ададҳои мачмӯи ибтидоӣ (бо ҳамон тартиб) бошанд.

Two positive integers *M*, *N* and a sequence of *M* real numbers are given. Create and output an $M \times N$ matrix of real numbers

such that each of its rows contains all numbers from the given sequence (in the same order).

```
package main

import "fmt"

func main() {
    var m, n int
    fmt.Print("M = ")
    fmt.Scan(&m)
    fmt.Print("N = ")
    fmt.Scan(&n)
    var array []float32 = make([]float32, n)
    for index, _ := range array {
        fmt.Scan(&array[index])
    }
    var matrix [][]float32 = make([][]float32, m)
    for row, _ := range matrix {
        matrix[row] = make([]float32, n)
        for col, _ := range matrix[row] {
            matrix[row][col] = array[col]
        }
    }
    fmt.Println()
    for _, array := range matrix {
        for _, value := range array {
            fmt.Printf("%.2f\t", value)
        }
        fmt.Println()
    }
}
```

Matrix 5

Ададҳои бутуни мусбии M, N, ададиD ва маҷмӯи дорои M ададҳо дода шудаанд. Матритсаи андозаи M x N-ро тартиб диҳед, ки дар он сутуни аввалин бо маҷмӯи ададҳои ибтидоӣ мувофиқат мекунад, элементҳои ҳар як сутуни оянда ба суммаи элементи сутуни пешина ва ададиD баробаранд (дар натиҷа ҳар як сатри матритса элементҳои прогрессияи арифметикиро доро мешавад).

Two positive integers *M* and *N*, a real number *D*, and a sequence of *M* real numbers are given. Create and output an $M \times N$ matrix of real numbers such that its first column contains all numbers from the given sequence (in the same order), and elements of each next column are equal to the sum of the corresponding

element of the previous column and the number *D* (so each row of the matrix will be an *arithmetic sequence* with the common difference *D*).

```
package main

import "fmt"

func main() {
    var (
        m, n int
        d float32
    )
    fmt.Print("M = ")
    fmt.Scan(&m)
    fmt.Print("N = ")
    fmt.Scan(&n)
    fmt.Print("D = ")
    fmt.Scan(&d)
    var array []float32 = make([]float32, m)
    for index, _ := range array {
        fmt.Scan(&array[index])
    }
    var matrix [][]float32 = make([][]float32, m)
    for row, _ := range matrix {
        matrix[row] = make([]float32, n)
        for col, _ := range matrix[row] {
            if col == 0 {
                matrix[row][col] = array[row]
            } else {
                matrix[row][col] = matrix[row][col-1] + d
            }
        }
    }
    fmt.Println()
    for _, array := range matrix {
        for _, value := range array {
            fmt.Printf("%.2f\t", value)
        }
        fmt.Println()
    }
}
```

Matrix 6

Ададҳои бутуни мусбии M, N, ададӣ Q ва маҷмӯи дорои N ададҳо дода шудаанд. Матритсаи андозаи M x N-ро тартиб диҳед, ки дар он сатри аввал бо маҷмӯи ададҳои ибтидоӣ мувофиқ аст, элементҳои ҳар як сатри оянда бошанд, ба ҳосилизарби элементи мувофиқи сатри пешина ва Q баробаранд (дар натиҷа ҳар як сутуни матритса элементҳои прогрессияи геометриро доро мешавад).

Two positive integers *M* and *N*, a real number *D*, and a sequence of *M* real numbers are given. Create and output an $M \times N$ matrix of real numbers such that its first row contains all numbers from the given sequence (in the same order), and elements of each next row are equal to the sum of the corresponding element of the previous row and the number *R* (so each column of the matrix will be a *geometric sequence* with the common ratio *R*).

```
package main

import "fmt"

func main() {
    var (
        m, n int
        d float32
    )
    fmt.Print("M = ")
    fmt.Scan(&m)
    fmt.Print("N = ")
    fmt.Scan(&n)
    fmt.Print("D = ")
    fmt.Scan(&d)
    var array []float32 = make([]float32, n)
    for index, _ := range array {
        fmt.Scan(&array[index])
    }
    var matrix [][]float32 = make([][]float32, m)
    for row, _ := range matrix {
        matrix[row] = make([]float32, n)
        for col, _ := range matrix[row] {
            if row == 0 {
                matrix[row][col] = array[col]
            } else {
                matrix[row][col] = matrix[row-1][col] * d
            }
        }
    }
    fmt.Println()
    for _, array := range matrix {
        for _, value := range array {
            fmt.Printf("%.2f\t", value)
        }
        fmt.Println()
    }
}
```

Matrix 7

Матритсаи андозаи M x N ва адади бутуни K (1 <= K <= M) дода шудаанд. Элементҳои сатри K-ӯми матритсаи мазкурро хориҷ кунед.

An $M \times N$ matrix of real numbers and an integer K are given ($1 \le K \le M$). Output elements of the matrix row with the order number K.

```
package main

import "fmt"

func main() {
    var m, n, k int
    fmt.Print("M = ")
    fmt.Scan(&m)
    fmt.Print("N = ")
    fmt.Scan(&n)
    var matrix [][]float32 = make([][]float32, m)
    for row, _ := range matrix {
        matrix[row] = make([]float32, n)
        for col, _ := range matrix[row] {
            fmt.Scan(&matrix[row][col])
        }
    }
    fmt.Print("K = ")
    fmt.Scan(&k)
    fmt.Println()
    for col := 0; col < n; col++ {
        fmt.Printf("%.2f\t", matrix[k-1][col])
    }
}
```

Matrix 8

Матритсаи андозаи M x N ва адади бутуни K (1 <= K <= N) дода шудаанд. Элементҳои сутуни K-ӯми матритсаи мазкурро хориҷ кунед.

An $M \times N$ matrix of real numbers and an integer K are given ($1 \le K \le M$). Output elements of the matrix column with the order number K.

```
package main

import "fmt"

func main() {
    var m, n, k int
    fmt.Print("M = ")
```

```
    fmt.Scan(&m)
    fmt.Print("N = ")
    fmt.Scan(&n)
    var matrix [][]float32 = make([][]float32, m)
    for row, _ := range matrix {
        matrix[row] = make([]float32, n)
        for col, _ := range matrix[row] {
            fmt.Scan(&matrix[row][col])
        }
    }
    fmt.Print("K = ")
    fmt.Scan(&k)
    fmt.Println()
    for row := 0; row < m; row++ {
        fmt.Printf("%.2f\t", matrix[row][k-1])
    }
}
```

Matrix 9

Матритсаи андозаи M x N дода шудааст. Элементҳои онро, ки дар сатрҳои дорои рақамҳои тартибии ҷуфт (2, 4, …) ҷойгиранд, хориҷ кунед. Хориҷкунии элементҳоро аз рӯи сатрҳо иҷро кунед, оператори шартиро истифода набаред.

An $M \times N$ matrix of real numbers is given. Output elements of its rows with even order numbers (2, 4, …). An output of matrix elements must be performed in the order of rows. Do not use conditional statements.

```
package main

import "fmt"

func main() {
    var m, n int
    fmt.Print("M = ")
    fmt.Scan(&m)
    fmt.Print("N = ")
    fmt.Scan(&n)
    var matrix [][]float32 = make([][]float32, m)
    for row, _ := range matrix {
        matrix[row] = make([]float32, n)
        for col, _ := range matrix[row] {
            fmt.Scan(&matrix[row][col])
        }
    }
    fmt.Println()
    for row := 1; row < m; row += 2 {
        for col, _ := range matrix[row] {
            fmt.Printf("%.2f\t", matrix[row][col])
        }
        fmt.Println()
    }
```

```
}
```

Matrix 10

Матритсаи андозаи M x N дода шудааст. Элементҳои онро, ки дар сутунҳои дорои рақамҳои тартибии тоқ (1, 3, …) ҷойгиранд, хориҷ кунед. Хориҷкунии элементҳоро аз рӯи сутунҳо иҷро кунед, оператори шартиро истифода набаред.

An $M \times N$ matrix of real numbers is given. Output elements of its columns with odd order numbers (1, 3, …). An output of matrix elements must be performed in the order of columns. Do not use conditional statements.

```
package main

import "fmt"

func main() {
    var m, n int
    fmt.Print("M = ")
    fmt.Scan(&m)
    fmt.Print("N = ")
    fmt.Scan(&n)
    var matrix [][]float32 = make([][]float32, m)
    for row, _ := range matrix {
        matrix[row] = make([]float32, n)
        for col, _ := range matrix[row] {
            fmt.Scan(&matrix[row][col])
        }
    }
    fmt.Println()
    for row, _ := range matrix {
        for col := 0; col < n; col += 2 {
            fmt.Printf("%.2f\t", matrix[row][col])
        }
        fmt.Println()
    }
}
```

Matrix 11

Матритсаи андозаи M x N дода шудааст. Элементҳои онро аз рӯи тартиби зерин хориҷ кунед: сатри аввал аз чап ба рост, сатри дуюм аз рост ба чап, сатри сеюм аз чап ба рост, сатри чорӯм аз рост ба чап ва ғ.

An $M \times N$ matrix of real numbers is given. Output elements of the matrix in the following order: the first row from left to right, the second row from right to left, the third row from left to right, the fourth row from right to left, and so on.

```
package main

import "fmt"

func main() {
    var m, n int
    fmt.Print("M = ")
    fmt.Scan(&m)
    fmt.Print("N = ")
    fmt.Scan(&n)
    var matrix [][]float32 = make([][]float32, m)
    for row, _ := range matrix {
        matrix[row] = make([]float32, n)
        for col, _ := range matrix[row] {
            fmt.Scan(&matrix[row][col])
        }
    }
    var col, modifier int
    for row, _ := range matrix {
        if row % 2 == 0 {
            col, modifier = 0, 1
        } else {
            col, modifier = n - 1, -1
        }
        for {
            if col < 0 || col >= n { break }
            fmt.Printf("%.2f\t", matrix[row][col])
            col += modifier
        }
        fmt.Println()
    }
}
```

Matrix 12

Матритсаи андозаи M x N дода шудааст. Элементҳои онро аз рӯи тартиби зерин хориҷ кунед: сутуни аввал аз боло ба поён, сутуни дуюм аз поён ба боло, сутуни сеюм аз боло ба поён, сутуни чорӯм аз поён ба боло ва ғ.

An $M \times N$ matrix of real numbers is given. Output elements of the matrix in the following order: the first column from up to down, the second column from down to up, the third column from up to down, the fourth column from down to up, and so on.

```
package main
```

```
import "fmt"

func main() {
    var m, n int
    fmt.Print("M = ")
    fmt.Scan(&m)
    fmt.Print("N = ")
    fmt.Scan(&n)
    var matrix [][]float32 = make([][]float32, m)
    for row, _ := range matrix {
        matrix[row] = make([]float32, n)
        for col, _ := range matrix[row] {
            fmt.Scan(&matrix[row][col])
        }
    }
    var row, modifier int
    for col := 0; col < n; col++ {
        if col % 2 == 0 {
            row, modifier = 0, 1
        } else {
            row, modifier = m - 1, -1
        }
        for {
            if row < 0 || row >= m { break }
            fmt.Printf("%.2f\t", matrix[row][col])
            row += modifier
        }
        fmt.Println()
    }
}
```

Matrix 13

Матритсаи квадратии A бо тартиби M дода шудааст. Аз элементи $A_{1,1}$ сар карда, элементҳои онро ба тариқи зайл хориҷ кунед («аз рӯи гӯшаҳо»): ҳамаи элементҳои сатри якӯм; элементҳои сутуни охирин, ба ғайр аз элементи якӯм (аллакай хориҷ шудааст); элементҳои боқимондаи сатри дуюм; элементҳои боқимондаи сутуни пешазохирон ва ғ.; охирон элементи $A_{M,1}$ хориҷ карда мешавад.

A real-valued square matrix *A* of order *M* is given. Starting with the element $A_{1,1}$, output its elements as follows: all elements of the first row, all elements of the *M*-th column except the element $A_{1,M}$ (which is already output), all remaining elements of the second row, all remaining elements of the (*M*−1)-th column, and so on; the element $A_{M,1}$ must be output in the end. All rows must

be output from left to right, all columns must be output from up to down.

```
package main

import "fmt"

func main() {
    var m int
    fmt.Print("M = ")
    fmt.Scan(&m)
    var a [][]float32 = make([][]float32, m)
    for row, _ := range a {
        a[row] = make([]float32, m)
        for col, _ := range a[row] {
            fmt.Scan(&a[row][col])
        }
    }
    fmt.Println()
    var col int
    for row, _ := range a {
        col = m - 1 - row;
        for j := 0; j <= col; j++ {
            fmt.Printf("%.2f\t", a[row][j])
        }
        fmt.Println()
        for j := row + 1; j < m; j++ {
            fmt.Printf("%.2f\t", a[j][col])
        }
        fmt.Println()
    }
}
```

Matrix 14

Матритсаи квадратии A бо тартиби M дода шудааст. Аз элементи $A_{1,1}$ сар карда, элементҳои онро бо тартиби зерин хориҷ кунед («аз рӯи гӯшаҳо»): ҳамаи элементҳои сутуни якӯм; элементҳои сатри охирин, ба ғайр аз элементи якӯм (аллакай хориҷ шудааст); элементҳои боқимондаи сутуни дуюм; элементҳои боқимондаи сатри пешазохирон ва ғ.; охирон элементи $A_{1,M}$ хориҷ карда мешавад.

A real-valued square matrix *A* of order *M* is given. Starting with the element $A_{1,1}$, output its elements as follows: all elements of the first column, all elements of the *M*-th row except the element $A_{M,1}$ (which is already output), all remaining elements of the second column, all remaining elements of the (*M*−1)-th row, and

so on; the element $A_{1,M}$ must be output in the end. All rows must be output from left to right, all columns must be output from up to down.

```
package main

import "fmt"

func main() {
    var m int
    fmt.Print("M = ")
    fmt.Scan(&m)
    var a [][]float32 = make([][]float32, m)
    for row, _ := range a {
        a[row] = make([]float32, m)
        for col, _ := range a[row] {
            fmt.Scan(&a[row][col])
        }
    }
    fmt.Println()
    var row int
    for col := 0; col < m; col++ {
        row = m - 1 - col;
        for i := 0; i <= row; i++ {
            fmt.Printf("%.2f\t", a[i][col])
        }
        fmt.Println()
        for i := col + 1; i < m; i++ {
            fmt.Printf("%.2f\t", a[row][i])
        }
        fmt.Println()
    }
}
```

Matrix 15

Матритсаи квадратии A бо тартиби M (M — адади тоқ) дода шудааст. Аз элементи $A_{1,1}$ сар карда ва аз рӯи ақрабаки соат ҷой иваз намуда, ҳамаи элементҳои онро ба таври симпеч хориҷ кунед: сатри якӯм, сутуни охирон, сатри охирон бо тартиби баръакс, сутуни якӯм бо тартиби баръакс, элементҳои боқимондаи сатри дуюм ва ғ.; охирон элементи марказии матритса хориҷ карда мешавад.

A real-valued square matrix *A* of order *M* is given (*M* is an odd number). Starting with the element $A_{1,1}$ and moving clockwise, output all matrix elements *in the spiral order*: the first row from left to right, the last column from up to down, the last row from

right to left, the first column from down to up, all remaining elements of the second row (from left to right), and so on; the central element of the matrix must be output in the end.

```
package main

import "fmt"

func main() {
    var m int
    fmt.Print("M = ")
    fmt.Scan(&m)
    var a [][]float32 = make([][]float32, m)
    for row, _ := range a {
        a[row] = make([]float32, m)
        for col, _ := range a[row] {
            fmt.Scan(&a[row][col])
        }
    }
    fmt.Println()
    var index, limit int = 0, m / 2 + m % 2
    for row := 0; row < limit; row++ {
        index = m - row - 1
        for col := row; col <= index; col++ {
            fmt.Printf("%.2f\t", a[row][col])
        }
        fmt.Println()
        for col := row+1; col <= index; col++ {
            fmt.Printf("%.2f\t", a[col][index])
        }
        fmt.Println()
        for col := index-1; col >= row; col-- {
            fmt.Printf("%.2f\t", a[index][col])
        }
        fmt.Println()
        for col := index-1; col > row; col-- {
            fmt.Printf("%.2f\t", a[col][row])
        }
        fmt.Println()
    }
}
```

Matrix 16

Матритсаи квадратии A бо тартиби M (M — адади тоқ) дода шудааст. Аз элементи $A_{1,1}$ сар карда ва бар муқобили ақрабаки соат ҷой иваз намуда, ҳамаи элементҳои онро ба таври симпеч хориҷ кунед: сутуни якӯм, сатри охирон, сутуни охирон бо тартиби баръакс, сатри якӯм бо тартиби баръакс, элементҳои боқимондаи сутуни дуюм ва ғ.; охирон элементи марказии матритса хориҷ карда мешавад.

A real-valued square matrix *A* of order *M* is given (*M* is an odd number). Starting with the element $A_{1,1}$ and moving counter-clockwise, output all matrix elements *in the spiral order*: the first column from up to down, the last row from left to right, the last column from down to up, the first row from right to left, all remaining elements of the second column (from up to down), and so on; the central element of the matrix must be output in the end.

```
package main

import "fmt"

func main() {
    var m int
    fmt.Print("M = ")
    fmt.Scan(&m)
    var a [][]float32 = make([][]float32, m)
    for row, _ := range a {
        a[row] = make([]float32, m)
        for col, _ := range a[row] {
            fmt.Scan(&a[row][col])
        }
    }
    fmt.Println()
    var index, limit int = 0, m / 2 + m % 2
    for col := 0; col < limit; col++ {
        index = m - 1 - col
        for row := col; row <= index; row++ {
            fmt.Printf("%.2f\t", a[row][col])
        }
        fmt.Println()
        for row := col+1; row <= index; row++ {
            fmt.Printf("%.2f\t", a[index][row])
        }
        fmt.Println()
        for row := index-1; row >= col; row-- {
            fmt.Printf("%.2f\t", a[row][index])
        }
        fmt.Println()
        for row := index-1; row > col; row-- {
            fmt.Printf("%.2f\t", a[col][row])
        }
        fmt.Println()
    }
}
```

Matrix 17

Матритсаи андозаи M x N ва адади бутуни K (1 <= K <= M) дода шудаанд. Сумма ва ҳосилизарби элементҳои сатри K-ӯми матритсаи мазкурро ёбед.

An $M \times N$ matrix of real numbers and an integer K are given ($1 \le K \le M$). Find the sum and the product of elements of the matrix row with the order number K.

```
package main

import "fmt"

func main() {
    var m, n, k int
    fmt.Print("M = ")
    fmt.Scan(&m)
    fmt.Print("N = ")
    fmt.Scan(&n)
    var matrix [][]float32 = make([][]float32, m)
    for row, _ := range matrix {
        matrix[row] = make([]float32, n)
        for col, _ := range matrix[row] {
            fmt.Scan(&matrix[row][col])
        }
    }
    fmt.Print("K = ")
    fmt.Scan(&k)
    var sum, mul float32 = 0, 1
    for col := 0; col < n; col++ {
        sum += matrix[k-1][col]
        mul *= matrix[k-1][col]
    }
    fmt.Printf("sum = %.2f\t\tmultiplication = %.2f\n", sum, mul)
}
```

Matrix 18

Матритсаи андозаи M x N ва адади бутуни K (1 <= K <= N) дода шудаанд. Сумма ва ҳосилизарби элементҳои сутуни K-ӯми матритсаи мазкурро ёбед.

An $M \times N$ matrix of real numbers and an integer K are given ($1 \le K \le N$). Find the sum and the product of elements of the matrix column with the order number K.

```
package main

import "fmt"

func main() {
    var m, n, k int
    fmt.Print("M = ")
    fmt.Scan(&m)
    fmt.Print("N = ")
    fmt.Scan(&n)
    var matrix [][]float32 = make([][]float32, m)
```

```
    for row, _ := range matrix {
        matrix[row] = make([]float32, n)
        for col, _ := range matrix[row] {
            fmt.Scan(&matrix[row][col])
        }
    }
    fmt.Print("K = ")
    fmt.Scan(&k)
    var sum, mul float32 = 0, 1
    for row := 0; row < m; row++ {
        sum += matrix[row][k-1]
        mul *= matrix[row][k-1]
    }
    fmt.Printf("sum = %.2f\t\tmultiplication = %.2f\n", sum, mul)
}
```

Matrix 19

Матритсаи андозаи M x N дода шудааст. Барои ҳар як сатри матритса суммаи элементҳояшро ёбед.

An $M \times N$ matrix of real numbers is given. Find the sum of elements for each matrix row.

```
package main

import "fmt"

func main() {
    var m, n int
    fmt.Print("M = ")
    fmt.Scan(&m)
    fmt.Print("N = ")
    fmt.Scan(&n)
    var matrix [][]float32 = make([][]float32, m)
    for row, _ := range matrix {
        matrix[row] = make([]float32, n)
        for col, _ := range matrix[row] {
            fmt.Scan(&matrix[row][col])
        }
    }
    fmt.Println()
    var sum float32
    for row, _ := range matrix {
        sum = 0
        for col, _ := range matrix[row] {
            sum += matrix[row][col]
        }
        fmt.Printf("%.2f\t", sum)
    }
}
```

Matrix 20

Матритсаи андозаи M x N дода шудааст. Барои ҳар як сутуни матритса хосилизарби элементҳояшро ёбед.

An $M \times N$ matrix of real numbers is given. Find the product of elements for each matrix column.

```
package main

import "fmt"

func main() {
    var m, n int
    fmt.Print("M = ")
    fmt.Scan(&m)
    fmt.Print("N = ")
    fmt.Scan(&n)
    var matrix [][]float32 = make([][]float32, m)
    for row, _ := range matrix {
        matrix[row] = make([]float32, n)
        for col, _ := range matrix[row] {
            fmt.Scan(&matrix[row][col])
        }
    }
    fmt.Println()
    var mul float32
    for col := 0; col < n; col++ {
        mul = 1
        for row := 0; row < m; row++ {
            mul *= matrix[row][col]
        }
        fmt.Printf("%.2f\t", mul)
    }
}
```

String 1

Рамзи C дода шудааст. Коди онро (яъне рақами тартибиро дар ҷадвали кодҳо) хориҷ кунед.

Given a character *C*, output its numeric value in the character set.

```
package main

import "fmt"

func main() {
    var c string;
    fmt.Print("C = ")
    fmt.Scan(&c)
    var n rune = []rune(c)[0]
```

```
    fmt.Printf("N = %d\n", n)
}
```

String 2

Ададӣ бутуни N (32 <= N <= 126) дода шудааст. Рамзеро, ки кодаш ба N баробар аст, хориҷ кунед.

Given an integer N ($32 \le N \le 126$), output a character with the numeric value N in the character set.

```
package main

import "fmt"

func main() {
    var n int
    fmt.Print("N = ")
    fmt.Scan(&n)
    var c = string(n)
    fmt.Printf("C = %s", c)
}
```

String 3

Рамзи C дода шудааст. Ду рамзеро хориҷ кунед, ки якӯми онҳо дар ҷадвали кодҳо пеш аз рамзи C аст, дуюмаш бошад, пас аз рамзи C аст.

Given a character C, output two characters: the first character precedes C in the character set, the second one follows C in the character set.

```
package main

import "fmt"

func main() {
    var c string
    fmt.Print("C = ")
    fmt.Scan(&c)
    var code rune = []rune(c)[0]
    prev, next := string(code - 1), string(code + 1)
    fmt.Printf("Prev = %s\t\tNext = %s\n", prev, next)
}
```

String 4

Ададӣ бутуни N (1 <= N <= 26) дода шудааст. N-то ҳарфҳои калони аввалини алифбои лотиниро хориҷ кунед.

Given an integer *N* (1 ≤ *N* ≤ 26), output *N* first *capital* (that is, uppercase) letters of the English alphabet ("A", "B", "C", and so on).

```
package main

import "fmt"

func main() {
    var n int
    fmt.Print("N = ")
    fmt.Scanf("%d", &n)
    code := 65
    for i := 0; i < n; i++ {
        fmt.Printf("%s\t", string(code + i))
    }
}
```

String 5

Ададӣ бутуни N (1 <= N <= 26) дода шудааст. N-то ҳарфҳои хурди охирини алифбои лотиниро бо тартиби баръакс (аз ҳарфи «z» сар карда) хориҷ кунед.

Given an integer *N* (1 ≤ *N* ≤ 26), output *N* last *small* (that is, lowercase) letters of the English alphabet in inverse order ("z", "y", "x", and so on).

```
package main

import "fmt"

func main() {
    var n int
    fmt.Print("N = ")
    fmt.Scanf("%d", &n)
    code := 122
    for i := 0; i < n; i++ {
        fmt.Printf("%s\t", string(code - i))
    }
}
```

String 6

Рамзи C дода шудааст, ки рақам ё ҳарф (лотинӣ ё тоҷикӣ)-ро тасвир мекунад. Агар C рақам бошад, пас сатри «digit»-ро хориҷ кунед, агар ҳарфи лотинӣ бошад — сарти «lat»-ро хориҷ кунед, агар тоҷикӣ бошад — сатри «toj»-ро хориҷ кунед.

A character *C* representing a digit or a letter of the Latin alphabet is given. If *C* is a digit then output the string "digit", if *C* is a capital letter then output the string "capital", otherwise output the string "small".

```
package main

import "fmt"

func main() {
    var c string
    fmt.Print("C = ")
    fmt.Scan(&c)
    var code rune = []rune(c)[0]
    if code >= 65 && code <= 90 || code >= 97 && code <= 122 {
        fmt.Println("lat")
    } else if code >= 1040 && code <= 1103 {
        fmt.Println("rus")
    } else if code >= 48 && code <= 57 {
        fmt.Println("digit")
    }
}
```

String 7

Сатре, ки холӣ нест, дода шудааст. Кодҳои рамзҳои аввалин ва охирини онро хориҷ кунед.

Given a nonempty string, output numeric values of its first and last character in the character set.

```
package main

import "fmt"

func main() {
    var str string
    fmt.Print("string:\t")
    fmt.Scan(&str)
    var codes []rune = []rune(str)
    fmt.Println(codes[0], codes[len(codes) - 1])
}
```

String 8

Ададӣ бутуни N (> 0) ва рамзи C дода шудаанд. Сатри дарозии N-ро хориҷ кунед, ки он аз рамзҳои C таркиб ёфтааст.

Given an integer *N* (> 0) and a character *C*, output a string that is of length *N* and contains characters *C*.

```
package main

import "fmt"

func main() {
    var (
        n int
        c, str string
    )
    fmt.Print("N = ")
    fmt.Scan(&n)
    fmt.Print("C = ")
    fmt.Scan(&c)
    for i := 0; i < n; i++ {
        str += c
    }
    fmt.Println(str)
}
```

String 9

Ададӣ ҷуфти N (> 0) ва рамзҳои C_1 ва C_2 дода шудаанд. Сатри дарозии N-ро хориҷ кунед, ки он аз рамзҳои пайиҳамояндаи C_1 ва C_2, аз C_1 сар карда, таркиб ёфтааст.

Given an even integer *N* (> 0) and two characters C_1, C_2, output a string that is of length *N*, begins with C_1, and contains alternating characters C_1 and C_2.

```
package main

import "fmt"

func main() {
    var (
        n int
        c1, c2, str string
    )
    fmt.Print("N = ")
    fmt.Scan(&n)
    fmt.Print("C1 = ")
```

```
    fmt.Scan(&c1)
    fmt.Print("C2 = ")
    fmt.Scan(&c2)
    for i := 0; i < n; i += 2 {
        str += c1 + c2
    }
    fmt.Println(str)
}
```

String 10

Сатр дода шудааст. Сатреро хориҷ кунед, ки худи ҳамон рамзҳоро дорост, аммо бо тартиби баръакс ҷойгир шудаанд.

Given a string, output a new string that contains the given string characters in inverse order.

```
package main

import "fmt"

func reverse(str string) string {
    var codes []rune = []rune(str)
    from, to := 0, len(codes) - 1
    var tmp rune
    for from < to {
        tmp = codes[from]
        codes[from] = codes[to]
        codes[to] = tmp
        from++
        to--
    }
    return string(codes)
}

func main() {
    var str string
    fmt.Print("string:\t")
    fmt.Scan(&str)
    str = reverse(str)
    fmt.Println(str)
}
```

String 11

Сатри S, ки холӣ нест, дода шудааст. Сатреро хориҷ кунед, ки рамзҳои сатри S-ро дорост, ки дар байни онҳо яктогӣ фосила гузошта шудаст.

Given a nonempty string, output a new string that contains the given string characters separated by a blank character.

```
package main

import "fmt"

func main() {
    var s string
    fmt.Print("S = ")
    fmt.Scan(&s)
    var codes []rune = []rune(s)
    n := len(codes)
    var newCodes []rune = make([]rune, 2 * n - 1)
    var index int = 0
    var space rune = []rune(" ")[0]
    for i := 0; i < n; i++ {
        newCodes[index] = codes[i]
        index++
        if i < n-1 {
            newCodes[index] = space
            index++
        }
    }
    s = string(newCodes)
    fmt.Println(s)
}
```

String 12

Сатри нохолии (холӣ нест) S ва ададu бутуни N (> 0) дода шудаанд. Сатреро хориҷ кунед, ки рамзҳои сатри S-ро дорост, ки дар байни онҳо N-тогӣ рамзҳои «*» (ситорача) гузошта шудаанд.

Given a nonempty string and an integer *N* (> 0), output a new string that contains the given string characters separated by *N* characters "*".

```
package main

import "fmt"

func main() {
    var s string
    var n int
    fmt.Print("S = ")
    fmt.Scan(&s)
    fmt.Print("N = ")
    fmt.Scan(&n)
    codes := []rune(s)
    darozi := len(codes)
    var newCodes []rune = make([]rune, darozi + (darozi-1)*n)
    var index int = 0
    var space rune = []rune("*")[0]
    for i := 0; i < darozi; i++ {
        newCodes[index] = codes[i]
        index++
```

```
        if i < darozi-1 {
            for j := 0; j < n; j++ {
                newCodes[index] = space
                index++
            }
        }
    }
    s = string(newCodes)
    fmt.Println(s)
}
```

String 13

Сатр дода шудааст. Миқдори рақамҳои дар он мавҷударо ҳисоб кунед.

Given a string, find the amount of digits in the string.

```
package main

import "fmt"

func main() {
    var str string
    fmt.Print("string:\t")
    fmt.Scan(&str)
    digits := 0
    var codes []rune = []rune(str)
    for index, _ := range codes {
        if codes[index] >= 48 && codes[index] <= 57 {
            digits++
        }
    }
    fmt.Printf("digits = %d\n", digits)
}
```

String 14

Сатр дода шудааст. Миқдори ҳарфҳои калони лотинии дар он мавҷударо ҳисоб кунед.

Given a string, find the amount of Latin capital letters in the string.

```
package main

import "fmt"

func main() {
    var str string
    fmt.Print("string:\t")
    fmt.Scan(&str)
    LATS := 0
```

```
    var codes []rune = []rune(str)
    for index, _ := range codes {
        if codes[index] >= 65 && codes[index] <= 90 {
            LATS++
        }
    }
    fmt.Printf("LATINS = %d\n", LATS)
}
```

String 15

Сатр дода шудааст. Миқдори умумии ҳарфҳои хурди лотинӣ ва тоҷикии дар он мавҷударо ҳисоб кунед.

Given a string, find the amount of Latin letters in the string.

```
package main

import "fmt"

func main() {
    var str string
    fmt.Print("string:\t")
    fmt.Scan(&str)
    letters := 0
    var codes []rune = []rune(str)
    for index, _ := range codes {
        if codes[index] >= 1072 && codes[index] <= 1103 ||
           codes[index] >= 97 && codes[index] <= 122 {
            letters++
        }
    }
    fmt.Printf("letters = %d\n", letters)
}
```

String 16

Сатр дода шудааст. Ҳамаи ҳарфҳои калони лотинии дар он мавҷударо ба хурд табдил диҳед.

Given a string, convert all Latin capital letters of the string to lowercase.

```
package main

import "fmt"

func toLowerLat(str string) string {
    var codes []rune = []rune(str)
    const (
        GREAT_A rune = 65
        GREAT_Z rune = 90
        SMALL_A rune = 97
```

```
        SMALL_Z rune = 122
    )
    var farq rune
    for index, _ := range codes {
        if codes[index] >= GREAT_A && codes[index] <= GREAT_Z {
            farq = codes[index] - GREAT_A
            codes[index] = SMALL_A + farq
        }
    }
    return string(codes)
}

func main() {
    var str string
    fmt.Print("string:\t")
    fmt.Scan(&str)
    str = toLowerLat(str)
    fmt.Printf("string = %s\n", str)
}
```

String 17

Сатр дода шудааст. Ҳамаи ҳарфҳои хурд (ҳам лотинӣ ва ҳам тоҷикӣ)-и дар он мавҷударо ба калон табдил диҳед.

Given a string, convert all Latin small letters of the string to uppercase.

```
package main

import "fmt"

func toUpper(str string) string {
    var codes []rune = []rune(str)
    const (
        GREAT_LAT_A, GREAT_LAT_Z rune = 65, 90
        SMALL_LAT_A, SMALL_LAT_Z rune = 97, 122

        GREAT_RUS_A, GREAT_RUS_Z rune = 1040, 1071
        SMALL_RUS_A, SMALL_RUS_Z rune = 1072, 1103
    )
    var farq rune
    for index, _ := range codes {
        if codes[index] >= SMALL_LAT_A && codes[index] <= SMALL_LAT_Z {
            farq = codes[index] - SMALL_LAT_A
            codes[index] = GREAT_LAT_A + farq
        } else if codes[index] >= SMALL_RUS_A && codes[index] <= SMALL_RUS_Z
{
            farq = codes[index] - SMALL_RUS_A
            codes[index] = GREAT_RUS_A + farq
        }
    }
    return string(codes)
}

func main() {
    var str string
```

```
    fmt.Print("string:\t")
    fmt.Scan(&str)
    str = toUpper(str)
    fmt.Printf("string = %s\n", str)
}
```

String 18

Сатр дода шудааст. Ҳамаи ҳарфҳои хурд (ҳам лотинӣ ва ҳам тоҷикӣ)-и дар он мавҷударо ба калон табдил диҳед, ҳарфҳои калонро бошад — ба хурд табдил диҳед.

Given a string, convert all Latin capital letters of the string to lowercase and all Latin small letters of the string to uppercase.

```
package main

import "fmt"

func changeRegister(str string) string {
    var codes []rune = []rune(str)
    const (
        GREAT_LAT_A, GREAT_LAT_Z rune = 65, 90
        SMALL_LAT_A, SMALL_LAT_Z rune = 97, 122

        GREAT_RUS_A, GREAT_RUS_Z rune = 1040, 1071
        SMALL_RUS_A, SMALL_RUS_Z rune = 1072, 1103
    )
    var farq rune
    for index, _ := range codes {
        if codes[index] >= SMALL_LAT_A && codes[index] <= SMALL_LAT_Z {
            farq = codes[index] - SMALL_LAT_A
            codes[index] = GREAT_LAT_A + farq
        } else if codes[index] >= SMALL_RUS_A && codes[index] <= SMALL_RUS_Z
{
            farq = codes[index] - SMALL_RUS_A
            codes[index] = GREAT_RUS_A + farq

        } else if codes[index] >= GREAT_LAT_A && codes[index] <= GREAT_LAT_Z
{
            farq = codes[index] - GREAT_LAT_A
            codes[index] = SMALL_LAT_A + farq
        } else if codes[index] >= GREAT_RUS_A && codes[index] <= GREAT_RUS_Z
{
            farq = codes[index] - GREAT_RUS_A
            codes[index] = SMALL_RUS_A + farq
        }
    }
    return string(codes)
}

func main() {
    var str string
    fmt.Print("string:\t")
    fmt.Scan(&str)
    str = changeRegister(str)
```

```
    fmt.Printf("string = %s\n", str)
}
```

String 19

Сатр дода шудааст. Агар он навишти ададӣ бутунро нишон диҳад, пас 1(як)-ро хориҷ кунед, агар ададӣ ҳақиқӣ (бо қисми касрӣ)-ро нишон диҳад — 2(ду)-ро хориҷ кунед; агар сатрро ба адад табдил додан имконнопазир бошад, пас 0(нул)-ро хориҷ кунед. Ба назар гиред, ки қисми касрии ададӣ ҳақиқӣ аз қисми бутунаш бо нуқтаи даҳӣ «.» ҷудо карда мешавад.

A string is given. If the string represents an integer then output 1, if the string represents a real number (with nonzero fractional part) then output 2, otherwise output 0. A fractional part of a real number is preceded by the *decimal point* ".".

```
package main

import "fmt"

func main() {
    var str string
    fmt.Print("string:\t")
    fmt.Scan(&str)
    var codes []rune = []rune(str)
    var point rune = []rune(".")[0]
    var index, darozi = 0, len(codes)
    var negative, points, digits int
    if codes[0] == []rune("-")[0] {
        index++
        negative = 1
    }
    for index < darozi {
        if codes[index] == point {
            points++
        } else if codes[index] >= 48 && codes[index] <= 57 {
            digits++
        }
        index++
    }
    if negative + digits == darozi {
        fmt.Println("1")
    } else if points == 1 && (negative + digits + points == darozi) {
        fmt.Println("2")
    } else {
        fmt.Println("0")
    }
}
```

String 20

Ададӣ бутуни мусбӣ дода шудааст. Рамзҳоеро хориҷ кунед, ки рақамҳои ин ададро тасвир мекунанд (бо тартиби аз чап ба рост).

Given a positive integer, output all digit characters in the decimal representation of the integer (from left to right).

```
package main

import "fmt"

func int2string(number int) string {
    isNegative := false
    if number < 0 {
        isNegative = true
        number *= -1
    }
    start := []rune("0")[0]
    var array []rune
    for number > 0 {
        array = append(array, start + rune(number % 10))
        number /= 10
    }
    if isNegative {
        array = append(array, []rune("-")[0])
    }
    from, to := 0, len(array) - 1
    for from < to {
        tmp := array[from]
        array[from] = array[to]
        array[to ] = tmp
        from++
        to--
    }
    return string(array)
}

func main() {
    var number int
    fmt.Scanf("%d", &number)
    var str string = int2string(number)
    darozi := len(str)
    for index := 0; index < darozi; index++ {
        fmt.Printf("%c\t", str[index])
    }
}
```

Param 1

Описать функцию MinElem(*A*, *N*) целого типа, находящую минимальный элемент целочисленного массива *A* размера *N*. С помощью этой функции найти минимальные элементы массивов *A*, *B*, *C* размера N_A, N_B, N_C соответственно.

Write an integer function MinElem(*A*, *N*) that returns the value of the minimal element of an array *A* of *N* integers. Using this function, find the minimal elements of three given arrays *A*, *B*, *C* whose sizes are N_A, N_B, N_C respectively.

```
package main

import "fmt"

func MinElem(a []int, n int) int {
    min := a[0]
    for index := 1; index < n; index++ {
        if a[index] < min {
            min = a[index]
        }
    }
    return min
}

func main() {
    var n int
    for i := 1; i <= 3; i++ {
        fmt.Printf("N%d = ", i)
        fmt.Scan(&n)
        array := make([]int, n)
        for j := 0; j < n; j++ {
            fmt.Scan(&array[j])
        }
        fmt.Printf("Min = %d\n\n", MinElem(array, n))
    }
}
```

Param 2

Описать функцию MaxNum(*A*, *N*) целого типа, находящую номер максимального элемента вещественного массива *A* размера *N*. С помощью этой функции найти номера максимальных элементов массивов *A*, *B*, *C* размера N_A, N_B, N_C соответственно.

Write an integer function MaxNum(*A*, *N*) that returns the order number of the maximal element of an array *A* of *N* real numbers. Using this function, find the order numbers of the maximal elements of three given arrays *A*, *B*, *C* whose sizes are N_A, N_B, N_C respectively.

```
package main

import "fmt"

func MaxNum(a []float32, n int) int {
    num := 0
    for index := 1; index < n; index++ {
        if a[index] > a[num] {
            num = index
        }
    }
    return num + 1
}

func main() {
    var n int
    for i := 1; i <= 3; i++ {
        fmt.Printf("N%d = ", i)
        fmt.Scan(&n)
        var array []float32 = make([]float32, n)
        for j := 0; j < n; j++ {
            fmt.Scan(&array[j])
        }
        fmt.Printf("MaxNum = %d\n\n", MaxNum(array, n))
    }
}
```

Param 3

Описать процедуру MinmaxNum(*A*, *N*, *NMin*, *NMax*), находящую номера минимального и максимального элемента вещественного массива *A* размера *N*. Выходные параметры целого типа: *NMin* (номер минимального элемента) и *NMax* (номер максимального элемента). С помощью этой процедуры найти номера минимальных и максимальных элементов массивов *A*, *B*, *C* размера N_A, N_B, N_C соответственно.

Write a procedure MinmaxNum(*A*, *N*, *NMin*, *NMax*) that finds the order numbers *NMin* and *NMax* of the minimal and the maximal element of an array *A* of *N* real numbers

(integers *NMin* and *NMax* are output parameters). Using this procedure, find the order numbers of the minimal and the maximal elements of three given arrays *A*, *B*, *C* whose sizes are N_A, N_B, N_C respectively.

```
package main

import "fmt"

func MinmaxNum(a []float32, n int) (int, int) {
    minIndex, maxIndex := 0, 0
    for index := 1; index < n; index++ {
        if a[index] > a[maxIndex] {
            maxIndex = index
        }
        if a[index] < a[minIndex] {
            minIndex = index
        }
    }
    return minIndex+1, maxIndex+1
}

func main() {
    var n, maxNum, minNum int
    for i := 1; i <= 3; i++ {
        fmt.Printf("N%d = ", i)
        fmt.Scan(&n)
        var array []float32 = make([]float32, n)
        for j := 0; j < n; j++ {
            fmt.Scan(&array[j])
        }
        minNum, maxNum = MinmaxNum(array, n)
        fmt.Printf("minNum = %d\t\tmaxNum = %d\n\n", minNum, maxNum)
    }
}
```

Param 4

Описать процедуру Inv(*A*, *N*), меняющую порядок следования элементов вещественного массива *A* размера *N* на обратный (*инвертирование* массива). Массив *A* является входным и выходным параметром. С помощью этой процедуры инвертировать массивы *A*, *B*, *C* размера N_A, N_B, N_C соответственно.

Write a procedure Inv(*A*, *N*) that changes the order of elements of an array *A* of *N* real numbers to inverse one (the array *A* is an input and output parameter). Using this procedure, change order

of elements of arrays *A*, *B*, *C* whose sizes are N_A, N_B, N_C respectively.

```
package main

import "fmt"

func Inv(a []float32, n int) {
    from, to := 0, n - 1
    for from < to {
        tmp := a[from]
        a[from] = a[to]
        a[to] = tmp
        from++
        to--
    }
}

func main() {
    var n int
    for i := 1; i <= 3; i++ {
        fmt.Printf("N%d = ", i)
        fmt.Scan(&n)
        var array []float32 = make([]float32, n)
        for j := 0; j < n; j++ {
            fmt.Scan(&array[j])
        }
        Inv(array, n)
        for index, _ := range array {
            fmt.Printf("%.2f\t", array[index])
        }
        fmt.Println("\n")
    }
}
```

Param 5

Описать процедуру Smooth1(*A*, *N*), выполняющую *сглаживание* вещественного массива *A* размера *N* следующим образом: элемент A_K заменяется на среднее арифметическое первых *K* исходных элементов массива *A*. Массив *A* является входным и выходным параметром. С помощью этой процедуры выполнить пятикратное сглаживание данного массива *A* размера *N*, выводя результаты каждого сглаживания.

Write a procedure Smooth1(*A*, *N*) that performs *smoothing* an array *A* of *N* real numbers as follows: each element A_K is replaced with the average of initial values of *K* first elements of

the given array *A*. The array *A* is an input and output parameter. Using five calls of this procedure, perform smoothing a given array *A* of *N* real numbers five times successively; output array elements after each smoothing.

```
package main

import "fmt"

func Smooth1(a []float32, n int) {
    var sum float32
    for index, _ := range a {
        sum += a[index]
        a[index] = sum / float32(index + 1)
    }
}

func printArray(a []float32, n int) {
    for index, _ := range a {
        fmt.Printf("%.2f\t", a[index])
    }
    fmt.Println()
}

func main() {
    var n int
    fmt.Print("N = ")
    fmt.Scan(&n)
    var array []float32 = make([]float32, n)
    for index, _ := range array {
        fmt.Scan(&array[index])
    }
    fmt.Println()
    for i := 0; i < 5; i++ {
        Smooth1(array, n)
        printArray(array, n)
    }
}
```

Param 6

Описать процедуру Smooth2(*A*, *N*), выполняющую *сглаживание* вещественного массива *A* размера *N* следующим образом: элемент A_1 не изменяется, элемент A_K ($K = 2, \ldots, N$) заменяется на полусумму исходных элементов A_{K-1} и A_K. Массив *A* является входным и выходным параметром. С помощью этой процедуры выполнить пятикратное сглаживание данного массива *A* размера *N*, выводя результаты каждого сглаживания.

Write a procedure Smooth2(*A*, *N*) that performs *smoothing* an array *A* of *N* real numbers as follows: an element A_1 remains unchanged; elements A_K (K = 2, …, N) is replaced with the average of initial values of elements A_{K-1} and A_K. The array *A* is an input and output parameter. Using five calls of this procedure, perform smoothing a given array *A* of *N* real numbers five times successively; output array elements after each smoothing.

```
package main

import "fmt"

func Smooth2(a []float32, n int) {
    var prev, curr float32
    prev = a[0]
    for index := 1; index < n; index++ {
        curr = a[index]
        a[index] = (prev + curr) / 2.0
        prev = curr
    }
}

func printArray(a []float32, n int) {
    for index, _ := range a {
        fmt.Printf("%.2f\t", a[index])
    }
    fmt.Println()
}

func main() {
    var n int
    fmt.Print("N = ")
    fmt.Scan(&n)
    var array []float32 = make([]float32, n)
    for index, _ := range array {
        fmt.Scan(&array[index])
    }
    fmt.Println()
    for i := 0; i < 5; i++ {
        Smooth2(array, n)
        printArray(array, n)
    }
}
```

Param 7

Описать процедуру Smooth3(*A*, *N*), выполняющую *сглаживание* вещественного массива *A* размера *N* следующим образом: каждый элемент массива заменяется на его среднее арифметическое с соседними элементами

(при вычислении среднего арифметического используются *исходные* значения соседних элементов). Массив *A* является входным и выходным параметром. С помощью этой процедуры выполнить пятикратное сглаживание данного массива *A* размера *N*, выводя результаты каждого сглаживания.

Write a procedure Smooth3(*A*, *N*) that performs *smoothing* an array *A* of *N* real numbers as follows: each array element is replaced with the average of initial values of this element and its neighbors. The array *A* is an input and output parameter. Using five calls of this procedure, perform smoothing a given array *A* of *N* real numbers five times successively; output array elements after each smoothing.

```
package main

import "fmt"

func Smooth3(a []float32, n int) {
    if n == 1 { return }
    var prev, curr float32
    for index, _ := range a {
        curr = a[index]
        if index == 0 {
            a[index] = (curr + a[index+1]) / 2.0
        } else if index == n - 1 {
            a[index] = (prev + curr) / 2.0
        } else {
            a[index] = (prev + curr + a[index+1]) / 3.0
        }
        prev = curr
    }
}

func printArray(a []float32, n int) {
    for index, _ := range a {
        fmt.Printf("%.2f\t", a[index])
    }
    fmt.Println()
}

func main() {
    var n int
    fmt.Print("N = ")
    fmt.Scan(&n)
    var array []float32 = make([]float32, n)
    for index, _ := range array {
        fmt.Scan(&array[index])
    }
    fmt.Println()
```

```
    for i := 0; i < 5; i++ {
        Smooth3(array, n)
        printArray(array, n)
    }
}
```

Param 8

Описать процедуру RemoveX(A, N, X), удаляющую из целочисленного массива A размера N элементы, равные целому числу X. Массив A и число N являются входными и выходными параметрами. С помощью этой процедуры удалить числа X_A, X_B, X_C из массивов A, B, C размера N_A, N_B, N_C соответственно и вывести размер и содержимое полученных массивов.

Write a procedure RemoveX(A, N, X) that removes all elements equal an integer X from an array A of N integers. The array A and its size N are input and output parameters. Using this procedure, remove elements with given values X_A, X_B, X_C from three given arrays A, B, C of size N_A, N_B, N_C respectively and output the new size and elements of each changed array.

```
package main

import "fmt"

func RemoveX(a *[]int, n *int, x int) {
    for index := 0; index < *n; {
        if (*a)[index] == x {
            *a = append((*a)[:index], (*a)[index+1:]...)
            (*n)--
        } else { index++ }
    }
}

func main() {
    var x, n int
    for i := 1; i <= 3; i++ {
        fmt.Printf("X%d = ", i)
        fmt.Scan(&x)
        fmt.Printf("N%d = ", i)
        fmt.Scan(&n)
        var array []int = make([]int, n)
        for index, _ := range array {
            fmt.Scan(&array[index])
        }
        RemoveX(&array, &n, x)
        fmt.Println(array)
    }
```

```
}
```

Param 9

Описать процедуру RemoveForInc(*A*, *N*), удаляющую из вещественного массива *A* размера *N* «лишние» элементы так, чтобы оставшиеся элементы оказались упорядоченными по возрастанию: первый элемент не удаляется, второй элемент удаляется, если он меньше первого, третий — если он меньше предыдущего элемента, оставленного в массиве, и т. д. Например, массив 5.5, 2.5, 4.6, 7.2, 5.8, 9.4 должен быть преобразован к виду 5.5, 7.2, 9.4. Массив *A* и число *N* являются входными и выходными параметрами. С помощью этой процедуры преобразовать массивы *A*, *B*, *C* размера N_A, N_B, N_C соответственно и вывести размер и содержимое полученных массивов.

Write a procedure RemoveForInc(*A*, *N*) that removes some elements from an array *A* of *N* real numbers so that the values of elements being remained were in ascending order: the first element remains unchanged, the second element must be removed if its value is less than the value of the first one, the third element must be removed if its value is less than the value of the previous element being remained, and so on. For instance, the array of elements 5.5, 2.5, 4.6, 7.2, 5.8, 9.4 must be changed to 5.5, 7.2, 9.4. All procedure parameters are input and output ones. Using this procedure, change three given arrays *A*, *B*, *C* whose sizes are N_A, N_B, N_C respectively and output the new size and elements of each changed array.

```
package main

import "fmt"

func RemoveForInc(a *[]float32, n *int) {
    for index := 1; index < *n; {
        if (*a)[index] < (*a)[index-1] {
            *a = append((*a)[:index], (*a)[index+1:] ...)
            (*n)--
```

```
        } else {
            index++
        }
    }
}

func printArray(a []float32, n int) {
    for index, _ := range a {
        fmt.Printf("%.2f\t", a[index])
    }
    fmt.Println()
}

func main() {
    var n int
    for i := 1; i <= 3; i++ {
        fmt.Printf("N%d = ", i)
        fmt.Scan(&n)
        var array []float32 = make([]float32, n)
        for index, _ := range array {
            fmt.Scan(&array[index])
        }
        RemoveForInc(&array, &n);
        printArray(array, n)
    }
}
```

Param 10

Описать процедуру DoubleX(*A*, *N*, *X*), дублирующую в целочисленном массиве *A* размера *N* элементы, равные целому числу *X*. Массив *A* и число *N* являются входными и выходными параметрами. С помощью этой процедуры продублировать числа X_A, X_B, X_C в массивах *A*, *B*, *C* размера N_A, N_B, N_C соответственно и вывести размер и содержимое полученных массивов.

Write a procedure DoubleX(*A*, *N*, *X*) that doubles occurrences of all elements equal an integer *X* for an array *A* of *N* integers. The array *A* and its size *N* are input and output parameters. Using this procedure, double occurrences of elements with given values X_A, X_B, X_C for three given arrays *A*, *B*, *C* of size N_A, N_B, N_C respectively and output the new size and elements of each changed array.

```
package main

import "fmt"
```

```
func DoubleX(a *[]int, n *int, x int) {
    for index := 0; index < *n; index++ {
        if (*a)[index] == x {
            *a = append((*a)[:index+1], (append([]int{x},
(*a)[index+1:]...))...)
            index++
            (*n)++
        }
    }
}

func main() {
    var x, n int
    for i := 1; i <= 3; i++ {
        fmt.Printf("X%d = ", i)
        fmt.Scan(&x)
        fmt.Printf("N%d = ", i)
        fmt.Scan(&n)
        var array []int = make([]int, n)
        for index, _ := range array {
            fmt.Scan(&array[index])
        }
        DoubleX(&array, &n, x)
        fmt.Println(array)
    }
}
```

Param 11

Описать процедуру SortArray(*A*, *N*), выполняющую сортировку по возрастанию вещественного массива *A* размера *N*. Массив *A* является входным и выходным параметром. С помощью этой процедуры отсортировать массивы *A*, *B*, *C* размера N_A, N_B, N_C соответственно.

Write a procedure SortArray(*A*, *N*) that sorts an array *A* of *N* real numbers in ascending order. The array *A* is an input and output parameter. Using this procedure, sort three given arrays *A*, *B*, *C* of size N_A, N_B, N_C respectively.

```
package main

import "fmt"

func SortArray(a []float32, n int) {
    // Bubble Sort
    for i := 0; i < n-1; i++ {
        for j := 1; j < n-i; j++ {
            if a[j-1] > a[j] {
                tmp := a[j-1]
                a[j-1] = a[j]
                a[j] = tmp
            }
```

```
            }
        }
    }

    func printArray(a []float32, n int) {
        for index, _ := range a {
            fmt.Printf("%.2f\t", a[index])
        }
        fmt.Println()
    }

    func main() {
        var n int
        for i := 1; i <= 3; i++ {
            fmt.Printf("N%d = ", i)
            fmt.Scan(&n)
            var array []float32 = make([]float32, n)
            for index, _ := range array {
                fmt.Scan(&array[index])
            }
            SortArray(array, n)
            printArray(array, n)
        }
    }
```

Param 12

Описать процедуру SortIndex(*A*, *N*, *I*), формирующую для вещественного массива *A* размера *N* *индексный массив I* — массив целых чисел того же размера, содержащий номера элементов массива *A* в том порядке, который соответствует возрастанию элементов массива *A* (сам массив *A* при этом не изменяется). Индексный массив *I* является выходным параметром. С помощью этой процедуры создать индексные массивы для массивов *A*, *B*, *C* размера N_A, N_B, N_C соответственно.

Write a procedure SortIndex(*A*, *N*, *I*) that creates an *index array I* for an array *A* of *N* real numbers. The index array contains order numbers of elements of array *A* so that they correspond to array elements in ascending order of their values (the array *A* remains unchanged). The index array *I* is an output parameter. Using this procedure, create index arrays for three given arrays *A*, *B*, *C* of size N_A, N_B, N_C respectively.

```
package main
```

```
import "fmt"

func SortIndex(a []float32, n int, i *[]int) {
    *i = make([]int, n)
    for index, _ := range *i {
        (*i)[index] = index
    }
    for k := 0; k < n-1; k++ {
        for j := 1; j < n-k; j++ {
            if a[(*i)[j-1]] > a[(*i)[j]] {
                tmp := (*i)[j-1]
                (*i)[j-1] = (*i)[j]
                (*i)[j] = tmp
            }
        }
    }
    for index, _ := range *i {
        (*i)[index]++
    }
}

func main() {
    var n int
    for i := 1; i <= 3; i++ {
        fmt.Printf("N%d = ", i)
        fmt.Scan(&n)
        var array []float32 = make([]float32, n)
        for index, _ := range array {
            fmt.Scan(&array[index])
        }
        var indexs []int
        SortIndex(array, n, &indexs)
        fmt.Println(indexs)
    }
}
```

Param 13

Описать процедуру Hill(*A*, *N*), меняющую порядок элементов вещественного массива *A* размера *N* на следующий: наименьший элемент массива располагается на первом месте, наименьший из оставшихся элементов — на последнем, следующий по величине располагается на втором месте, следующий — на предпоследнем и т. д. (в результате график значений элементов будет напоминать *холм*). Массив *A* является входным и выходным параметром. С помощью этой процедуры преобразовать массивы *A*, *B*, *C* размера N_A, N_B, N_C соответственно.

Write a procedure Hill(*A*, *N*) that changes order of elements of an array *A* of *N* real numbers as follows: the minimal element of

the array must be the first one, an element, whose value is the next to minimal value, must be the last one, an element with the next value must be the second one, and so on (as a result, the diagram of values of the array elements will be similar to a *hill*). The array *A* is an input and output parameter. Using this procedure, change three given arrays *A*, *B*, *C* of size N_A, N_B, N_C respectively.

```
package main

import "fmt"

func Hill(a []float32, n int) {
    if n == 1 { return }
    from, to := 0, n-1
    for iter := n/2; iter > 0; iter-- {
        for index := to; index > from; index-- {
            if a[index-1] > a[index] {
                tmp := a[index-1]
                a[index-1] = a[index]
                a[index] = tmp
            }
        }
        from++
        for index := from; index < to; index++ {
            if a[index] < a[index+1] {
                tmp := a[index+1]
                a[index+1] = a[index]
                a[index] = tmp
            }
        }
        to--
    }
}

func printArray(a []float32, n int) {
    for index, _ := range a {
        fmt.Printf("%.2f\t", a[index])
    }
    fmt.Println()
}

func main() {
    var n int
    for i := 1; i <= 3; i++ {
        fmt.Printf("N%d = ", i)
        fmt.Scan(&n)
        var array []float32 = make([]float32, n)
        for index, _ := range array {
            fmt.Scan(&array[index])
        }
        Hill(array, n)
        printArray(array, n)
    }
}
```

Param 14

Описать процедуру Split1(A, N_A, B, N_B, C, N_C), формирующую по вещественному массиву A размера N_A два вещественных массива B и C размера N_B и N_C соответственно; при этом массив B содержит все элементы массива A с нечетными порядковыми номерами (1, 3, …), а массив C — все элементы массива A с четными номерами (2, 4, …). Массивы B и C и числа N_B и N_C являются выходными параметрами. Применить эту процедуру к данному массиву A размера N_A и вывести размер и содержимое полученных массивов B и C.

Write a procedure Split1(A, N_A, B, N_B, C, N_C) that copies elements of an array A of N_A real numbers to arrays B and C so that the array B contains all elements of the array A with odd order numbers (1, 3, …) and the array C contains all elements of the array A with even order numbers (2, 4, …). The arrays B, C and their sizes N_B, N_C are output parameters. Apply this procedure to a given array A of size N_A and output the size and the elements for each of the resulting arrays B and C.

```
package main

import "fmt"

func Split1(a []float32, n int, b *[]float32, nb *int, c *[]float32, nc *int)
{
    *nb = n / 2 + n % 2
    *nc = n / 2
    *b = make([]float32, *nb)
    *c = make([]float32, *nc)
    bIndex, cIndex := 0, 0
    for index, _ := range a {
        if index % 2 == 0 {
            (*b)[bIndex] = a[index]
            bIndex++
        } else {
            (*c)[cIndex] = a[index]
            cIndex++
        }
    }
}

func printArray(a []float32, n int) {
    for index, _ := range a {
```

```
            fmt.Printf("%.2f\t", a[index])
        }
        fmt.Println()
    }

    func main() {
        var nA, nB, nC int
        fmt.Printf("NA = ")
        fmt.Scan(&nA)
        var a []float32 = make([]float32, nA)
        for index, _ := range a {
            fmt.Scan(&a[index])
        }
        var b, c []float32
        Split1(a, nA, &b, &nB, &c, &nC)
        fmt.Println()
        fmt.Printf("NB = %d\t", nB)
        printArray(b, nB)
        fmt.Printf("NC = %d\t", nC)
        printArray(c, nC)
    }
```

Param 15

Описать процедуру Split2(A, N_A, B, N_B, C, N_C), формирующую по целочисленному массиву A размера N_A два целочисленных массива B и C размера N_B и N_C соответственно; при этом массив B содержит все четные числа из массива A, а массив C — все нечетные числа (в том же порядке). Массивы B и C и числа N_B и N_C являются выходными параметрами. Применить эту процедуру к данному массиву A размера N_A и вывести размер и содержимое полученных массивов B и C.

Write a procedure Split2(A, N_A, B, N_B, C, N_C) that copies elements of an array A of N_A integers to arrays B and C so that the array B contains all elements whose values are even numbers and the array C contains all elements whose values are odd numbers (in the same order). The arrays B, C and their sizes N_B, N_C are output parameters. Apply this procedure to a given array A of size N_A and output the size and the elements for each of the resulting arrays B and C.

```
package main

import "fmt"
```

```
func Split2(a []int, n int, b *[]int, nb *int, c *[]int, nc *int) {
    for index, _ := range a {
        if a[index] % 2 == 0 {
            *b = append(*b, a[index])
            (*nb)++
        } else {
            *c = append(*c, a[index])
            (*nc)++
        }
    }
}

func main() {
    var nA, nB, nC int
    fmt.Printf("NA = ")
    fmt.Scan(&nA)
    var a []int = make([]int, nA)
    for index, _ := range a {
        fmt.Scan(&a[index])
    }
    var b, c []int
    Split2(a, nA, &b, &nB, &c, &nC)
    fmt.Println()
    fmt.Printf("NB = %d\t", nB)
    fmt.Println(b)
    fmt.Printf("NC = %d\t", nC)
    fmt.Println(c)
}
```

Param 16

Описать процедуру ArrayToMatrRow(*A*, *K*, *M*, *N*, *B*), формирующую по вещественному массиву *A* размера *K* матрицу *B* размера *M* × *N* (матрица заполняется элементами массива *A* по строкам). «Лишние» элементы массива игнорируются; если элементов массива недостаточно, то оставшиеся элементы матрицы полагаются равными 0. Двумерный массив *B* является выходным параметром. С помощью этой процедуры на основе данного массива *A* размера *K* и целых чисел *M* и *N* сформировать матрицу *B* размера *M* × *N*.

Write a procedure ArrayToMatrRow(*A*, *K*, *M*, *N*, *B*) that copies elements of an array *A* of *K* real numbers to an *M* × *N* matrix *B* (by rows). "Superfluous" array elements must be ignored; if the size of the array is less than the amount of matrix elements then zero value must be assigned to remaining matrix elements. Two-

dimensional array *B* is an output parameter. Having input an array *A* of size *K*, integers *M*, *N* and using this procedure, create a matrix *B* and output its elements.

```
package main

import "fmt"

func ArrayToMatrRow(a []float32, k int, m int, n int, b *[][]float32) {
    *b = make([][]float32, m)
    index := 0
    for row, _ := range *b {
        (*b)[row] = make([]float32, n)
        for col, _ := range (*b)[row] {
            if (index < k) {
                (*b)[row][col] = a[index]
                index++
            } else {
                (*b)[row][col] = 0
            }
        }
    }
}

func main() {
    var k, m, n int
    fmt.Print("K = ")
    fmt.Scan(&k)
    var array []float32 = make([]float32, k)
    for index, _ := range array {
        fmt.Scan(&array[index])
    }
    fmt.Print("M = ")
    fmt.Scan(&m)
    fmt.Print("N = ")
    fmt.Scan(&n)
    var matrix [][]float32
    ArrayToMatrRow(array, k, m, n, &matrix)
    for row, _ := range matrix {
        for col, _ := range matrix[row] {
            fmt.Printf("%.2f\t", matrix[row][col])
        }
        fmt.Println()
    }
}
```

Param 17

Описать процедуру ArrayToMatrCol(*A*, *K*, *M*, *N*, *B*), формирующую по вещественному массиву *A* размера *K* матрицу *B* размера *M* × *N* (матрица заполняется элементами массива *A* по столбцам). «Лишние» элементы массива игнорируются; если элементов массива недостаточно, то

оставшиеся элементы матрицы полагаются равными 0. Двумерный массив *B* является выходным параметром. С помощью этой процедуры на основе данного массива *A* размера *K* и целых чисел *M* и *N* сформировать матрицу *B* размера $M \times N$.

Write a procedure ArrayToMatrCol(*A*, *K*, *M*, *N*, *B*) that copies elements of an array *A* of *K* real numbers to an $M \times N$ matrix *B* (by columns). "Superfluous" array elements must be ignored; if the size of the array is less than the amount of matrix elements then zero value must be assigned to remaining matrix elements. Two-dimensional array *B* is an output parameter. Having input an array *A* of size *K*, integers *M*, *N* and using this procedure, create a matrix *B* and output its elements.

```
package main

import "fmt"

func ArrayToMatrCol(a []float32, k int, m int, n int, b *[][]float32) {
    *b = make([][]float32, m)
    index := 0
    for row, _ := range *b {
        (*b)[row] = make([]float32, n)
    }
    for col := 0; col < n; col++ {
        for row := 0; row < m; row++ {
            if (index < k) {
                (*b)[row][col] = a[index]
                index++
            } else {
                (*b)[row][col] = 0
            }
        }
    }
}

func main() {
    var k, m, n int
    fmt.Print("K = ")
    fmt.Scan(&k)
    var array []float32 = make([]float32, k)
    for index, _ := range array {
        fmt.Scan(&array[index])
    }
    fmt.Print("M = ")
    fmt.Scan(&m)
    fmt.Print("N = ")
    fmt.Scan(&n)
    var matrix [][]float32
    ArrayToMatrCol(array, k, m, n, &matrix)
```

```
    for row, _ := range matrix {
        for col, _ := range matrix[row] {
            fmt.Printf("%.2f\t", matrix[row][col])
        }
        fmt.Println()
    }
}
```

Param 18

Описать процедуру Chessboard(*M*, *N*, *A*), формирующую по целым положительным числам *M* и *N* матрицу *A* размера *M* × *N*, которая содержит числа 0 и 1, расположенные в «шахматном» порядке, причем $A_{1,1} = 0$. Двумерный целочисленный массив *A* является выходным параметром. С помощью этой процедуры по данным целым числам *M* и *N* сформировать матрицу *A* размера *M* × *N*.

Write a procedure Chessboard(*M*, *N*, *A*) that creates an *M* × *N* matrix *A* whose elements are integers 0 and 1, which are arranged in "chessboard" order, and $A_{1,1} = 0$. Two-dimensional array *A* is an output parameter. Having input integers *M*, *N* and using this procedure, create an *M* × *N* matrix *A*.

```
package main

import "fmt"

func Chessboard(m int, n int, a *[][]int) {
    (*a) = make([][]int, m)
    for row, _ := range *a {
        (*a)[row] = make([]int, n)
        for col, _ := range (*a)[row] {
            if (row+col) % 2 == 0 {
                (*a)[row][col] = 0
            } else {
                (*a)[row][col] = 1
            }
        }
    }
}

func main() {
    var m, n int
    fmt.Print("M = ")
    fmt.Scan(&m)
    fmt.Print("N = ")
    fmt.Scan(&n)
    var matrix [][]int;
    Chessboard(m, n, &matrix)
    for row, _ := range matrix {
```

```
        fmt.Println(matrix[row])
    }
}
```

Param 19

Описать функцию Norm1(*A*, *M*, *N*) вещественного типа, вычисляющую *норму* вещественной матрицы *A* размера *M* × *N*: Norm1(*A*, *M*, *N*) = max $\{|A_{1,J}| + |A_{2,J}| + \ldots + |A_{M,J}|\}$,где максимум берется по всем *J* от 1 до *N*. Для данной матрицы *A* размера *M* × *N* найти Norm1(*A*, *K*, *N*), *K* = 1, …, *M*.

Write a real-valued function Norm1(*A*, *M*, *N*) that computes the *norm* of an *M* × *N* matrix *A* of real numbers using the formula Norm1(*A*, *M*, *N*) = max $\{|A_{1,J}| + |A_{2,J}| + \ldots + |A_{M,J}|\}$,where the maximum is being found over *J* = 1, …, *N*. Having input an *M* × *N* matrix *A*, output Norm1(*A*, *K*, *N*), *K* = 1, …, *M*.

```
package main

import "fmt"

func Norm1(a [][]float32, m int, n int) float32 {
    var max, sum float32
    for col := 0; col < n; col++ {
        sum = 0
        for row := 0; row < m; row++ {
            sum += a[row][col]
        }
        if col == 0 {
            max = sum
        } else if sum > max {
            max = sum
        }
    }
    return max;
}

func main() {
    var m, n int
    fmt.Print("M = ")
    fmt.Scan(&m)
    fmt.Print("N = ")
    fmt.Scan(&n)
    var matrix [][]float32 = make([][]float32, m)
    for row, _ := range matrix {
        matrix[row] = make([]float32, n)
        for col, _ := range matrix[row] {
            fmt.Scan(&matrix[row][col])
        }
```

```
    }
    for row, _ := range matrix {
        fmt.Printf("Norm1(A, %d, %d) = %.2f\n", row+1, n, Norm1(matrix,
row+1, n))
    }
}
```

Param 20

Описать функцию Norm2(*A*, *M*, *N*) вещественного типа, вычисляющую *норму* вещественной матрицы *A* размера *M* × *N*: Norm2(*A*, *M*, *N*) = max $\{|A_{I,1}| + |A_{I,2}| + \ldots + |A_{I,N}|\}$,где максимум берется по всем *I* от 1 до *M*. Для данной матрицы *A* размера *M* × *N* найти Norm2(*A*, *K*, *N*), *K* = 1, …, *M*.

Write a real-valued function Norm2(*A*, *M*, *N*) that computes the *norm* of an *M* × *N* matrix *A* of real numbers using the formula Norm2(*A*, *M*, *N*) = max $\{|A_{I,1}| + |A_{I,2}| + \ldots + |A_{I,N}|\}$,where the maximum is being found over *I* = 1, …, *M*. Having input an *M* × *N* matrix *A*, output Norm2(*A*, *K*, *N*), *K* = 1, …, *M*.

```
package main

import "fmt"

func Norm2(a [][]float32, m int, n int) float32 {
    var max, sum float32
    for row := 0; row < m; row++ {
        sum = 0
        for col := 0; col < n; col++ {
            sum += a[row][col]
        }
        if row == 0 {
            max = sum
        } else if sum > max {
            max = sum
        }
    }
    return max;
}

func main() {
    var m, n int
    fmt.Print("M = ")
    fmt.Scan(&m)
    fmt.Print("N = ")
    fmt.Scan(&n)
    var matrix [][]float32 = make([][]float32, m)
    for row, _ := range matrix {
        matrix[row] = make([]float32, n)
```

```
        for col, _ := range matrix[row] {
            fmt.Scan(&matrix[row][col])
        }
    }
    for row, _ := range matrix {
        fmt.Printf("Norm1(A, %d, %d) = %.2f\n", row+1, n, Norm2(matrix,
row+1, n))
    }
}
```

Recur 1

Описать рекурсивную функцию Fact(*N*) вещественного типа, вычисляющую значение *факториала**N*! = 1·2·…·*N*(*N* > 0 — параметр целого типа). С помощью этой функции вычислить факториалы пяти данных чисел.

Write a recursive real-valued function Fact(*N*) that returns the value of *N-factorial*:*N*! = 1·2·…·*N*,where *N* (> 0) is an integer parameter. Using this function, output factorials of five given integers.

```
package main

import "fmt"

func Fact(n int) float64 {
    if n < 2 {
        return 1
    }
    return float64(n) * Fact(n - 1)
}

func main() {
    var n int
    for i := 1; i <= 5; i++ {
        fmt.Printf("N%d = ", i)
        fmt.Scan(&n)
        fmt.Printf("%d! = %.1f\n", n, Fact(n))
    }
}
```

Recur 2

Описать рекурсивную функцию Fact2(*N*) вещественного типа, вычисляющую значение *двойного*

факториала$N!! = N\cdot(N-2)\cdot(N-4)\cdot\ldots$($N > 0$ — параметр целого типа; последний сомножитель в произведении равен 2, если N — четное число, и 1, если N — нечетное). С помощью этой функции вычислить двойные факториалы пяти данных чисел.

Write a recursive real-valued function Fact2(N) that returns the value of *double factorial* of N:$N!! = N\cdot(N-2)\cdot(N-4)\cdot\ldots$,where N (> 0) is an integer parameter; the last factor of the product equals 2 if N is an even number, and 1 otherwise. Using this function, output double factorials of five given integers.

```
package main

import "fmt"

func Fact2(n int) float64 {
    if n < 2 {
        return 1
    }
    return float64(n) * Fact2(n - 2)
}

func main() {
    var n int
    for i := 1; i <= 5; i++ {
        fmt.Printf("N%d = ", i)
        fmt.Scan(&n)
        fmt.Printf("%d! = %.1f\n", i, Fact2(n))
    }
}
```

Recur 3

Описать рекурсивную функцию PowerN(X, N) вещественного типа, находящую значение N-й степени числа X по формулам: $X^0 = 1$,
$X^N = (X^{N/2})^2$ при *четных* $N > 0$, $X^N = X\cdot X^{N-1}$ при *нечетных* $N > 0$,
$X^N = 1/X^{-N}$ при $N < 0$($X \neq 0$ — вещественное число, N — целое; в формуле для четных N должна использоваться операция *целочисленного деления*). С помощью этой функции найти значения X^N для данного X при пяти данных значениях N.

Write a recursive real-valued function PowerN(*X*, *N*) that returns the power X^N ($X \neq 0$ is a real number, *N* is an integer) calculated as follows: $X^0 = 1$,
$X^N = (X^{N\ div\ 2})^2$ if *N* is a positive even number,
$X^N = X \cdot X^{N-1}$ if *N* is a positive odd number,
$X^N = 1/X^{-N}$ if $N < 0$,where "div" denotes the operator of *integer division*. Using this function, output powers X^N for a given real number *X* and five given integers *N*.

```
package main

import "fmt"

func PowerN(x float64, n int) float64 {
    if n == 0 {
        return 1
    }
    if n < 0 {
        return 1 / PowerN(x, -n)
    }
    if n % 2 == 0 {
        half := PowerN(x, n/2)
        return half * half
    }
    return x * PowerN(x, n-1)
}

func main() {
    var (
        x float64
        n int
    )
    fmt.Print("X = ")
    fmt.Scan(&x)
    for i := 1; i <= 5; i++ {
        fmt.Printf("N%d = ", i)
        fmt.Scan(&n)
        fmt.Printf("PowerN(%.2f, %d) = %E\n", x, n, PowerN(x, n))
    }
}
```

Recur 4

Описать рекурсивную функцию Fib1(*N*) целого типа, вычисляющую *N*-й элемент последовательности *чисел Фибоначчи* (*N* — целое число):
$F_1 = F_2 = 1$, $F_K = F_{K-2} + F_{K-1}$, $K = 3, 4, \ldots$.С помощью этой функции найти пять чисел Фибоначчи с данными

номерами, и вывести эти числа вместе с количеством рекурсивных вызовов функции Fib1, потребовавшихся для их нахождения.

Write a recursive integer function Fib1(*N*) that returns the Fibonacci number F_N (*N* is a positive integer). The *Fibonacci numbers* F_K are defined as:
$F_1 = F_2 = 1$, $F_K = F_{K-2} + F_{K-1}$, $K = 3, 4, \ldots$ .Using the function Fib1, find the Fibonacci numbers F_N for five given integers *N*; output the value of each Fibonacci number and also the amount of the recursive function calls, which are required for its calculation.

```
package main

import "fmt"

var count int

func Fib1(n int) int {
    count++
    if n <= 2 {
        return 1
    }
    return Fib1(n-1) + Fib1(n-2)
}

func main() {
    var n, element int
    for i := 1; i <= 5; i++ {
        fmt.Printf("N%d = ", i)
        fmt.Scan(&n)
        count = 0
        element = Fib1(n)
        fmt.Printf("Element = %d\t\tCount = %d\n", element, count)
    }
}
```

Recur 5

Описать рекурсивную функцию Fib2(*N*) целого типа, вычисляющую *N*-й элемент последовательности *чисел Фибоначчи* (*N* — целое число):
$F_1 = F_2 = 1$, $F_K = F_{K-2} + F_{K-1}$, $K = 3, 4, \ldots$.Считать, что номер *N* не превосходит 20. Для уменьшения количества рекурсивных вызовов по сравнению с функцией Fib1 (см.

задание Recur4) создать вспомогательный массив для хранения *уже вычисленных* чисел Фибоначчи и обращаться к нему при выполнении функции Fib2. С помощью функции Fib2 найти пять чисел Фибоначчи с данными номерами.

Write a recursive integer function Fib2(*N*) that returns the Fibonacci number F_N (*N* is a positive integer). The *Fibonacci numbers* F_K are defined as:
$F_1 = F_2 = 1$, $F_K = F_{K-2} + F_{K-1}$, $K = 3, 4, \ldots$ .The integer *N* is assumed to be not greater than 20. Decrease the amount of recursive calls of the function Fib2 (in comparison with the Fib1 function from the task Recur4) by means of using an additional array of integers that should store the Fibonacci numbers *having been calculated*. Using the Fib2 function, output the Fibonacci numbers F_N for five given integers *N*.

```
package main

import "fmt"

var array [20]int

func Fib2(n int) int {
    if n <= 2 {
        return 1
    }
    if array[n-1] == 0 {
        array[n - 1] = Fib2(n-1) + Fib2(n-2)
    }
    return array[n-1]
}

func main() {
    var n, element int
    for i := 1; i <= 5; i++ {
        fmt.Printf("N%d = ", i)
        fmt.Scan(&n)
        for index, _ := range array {
            array[index] = 0
        }
        element = Fib2(n)
        fmt.Printf("Fib2(%d) = %d\n", n, element)
    }
}
```

Recur 6

Описать рекурсивную функцию Combin1(*N*, *K*) целого типа, находящую *C*(*N*, *K*) — *число сочетаний* из *N* элементов по *K* — с помощью рекуррентного соотношения:
C(*N*, 0) = C(*N*, *N*) = 1,
C(*N*, *K*) = *C*(*N* − 1, *K*) + *C*(*N* − 1, *K* − 1) при
0 < *K* < *N*.Параметры функции — целые числа; *N* > 0,
0 ≤ *K* ≤ *N*. Дано число *N* и пять различных значений *K*. Вывести числа *C*(*N*, *K*) вместе с количеством рекурсивных вызовов функции Combin1, потребовавшихся для их нахождения.

Write a recursive integer function Combin1(*N*, *K*) that returns *C*(*N*, *K*) (the *number of combinations* of *N* objects taken *K* at a time) using the following recursive relations (*N* and *K* are integers, *N* > 0, 0 ≤ *K* ≤ *N*): *C*(*N*, 0) = C(*N*, *N*) = 1, *C*(*N*, *K*) = *C*(*N* − 1, *K*) + *C*(*N* − 1, *K* − 1) if 0 < *K* < *N*.Using the function Combin1, find the numbers *C*(*N*, *K*) for a given integer *N* and five given integers *K*; output the value of each number and also the amount of the recursive function calls, which are required for its calculation.

```
package main

import "fmt"

var count int

func Combin1(n, k int) int {
    count++
    if k == 0 || k == n {
        return 1
    }
    return Combin1(n-1, k) + Combin1(n-1, k-1)
}

func main() {
    var n, k, comb int
    fmt.Print("N = ")
    fmt.Scan(&n)
    for i := 1; i <= 5; i++ {
        fmt.Printf("K%d = ", i)
        fmt.Scan(&k)
        count = 0
        comb = Combin1(n, k)
        fmt.Printf("Combin(%d, %d) = %d\t\tCount = %d\n", n, k, comb, count)
    }
```

```
}
```

Recur 7

Описать рекурсивную функцию Combin2(*N*, *K*) целого типа, находящую *C*(*N*, *K*) — *число сочетаний* из *N* элементов по *K* — с помощью рекуррентного соотношения:
C(*N*, 0) = C(*N*, *N*) = 1,
C(*N*, *K*) = *C*(*N* − 1, *K*) + *C*(*N* − 1, *K* − 1) при 0 < *K* < *N*.Параметры функции — целые числа; *N* > 0, 0 ≤ *K* ≤ *N*. Считать, что параметр *N* не превосходит 20. Для уменьшения количества рекурсивных вызовов по сравнению с функцией Combin1 (см. задание Recur6) описать вспомогательный двумерный массив для хранения *уже вычисленных* чисел *C*(*N*, *K*) и обращаться к нему при выполнении функции Combin2. С помощью функции Combin2 найти числа *C*(*N*, *K*) для данного значения *N* и пяти различных значений *K*.

Write a recursive integer function Combin2(*N*, *K*) that returns *C*(*N*, *K*) (the *number of combinations* of *N* objects taken *K* at a time) using the following recursive relations (*N* and *K* are integers, *N* > 0, 0 ≤ *K* ≤ *N*): *C*(*N*, 0) = C(*N*, *N*) = 1, *C*(*N*, *K*) = *C*(*N* − 1, *K*) + *C*(*N* − 1, *K* − 1) if 0 < *K* < *N*.The integer *N* is assumed to be not greater than 20. Decrease the amount of recursive calls of the function Combin2 (in comparison with the Combin1 function from the task Recur6) by means of using an additional two-dimensional array of integers that should store the numbers *C*(*N*, *K*) *having been calculated*. Using the Combin2 function, output the numbers *C*(*N*, *K*) for a given integer *N* and five given integers *K*.

```
package main

import "fmt"

var matrix [20][20]int

func Combin2(n, k int) int {
```

```
    if k == 0 || k == n {
        return 1
    }
    if (matrix[n-1][k-1] == 0) {
        matrix[n-1][k-1] = Combin2(n-1, k) + Combin2(n-1, k-1)
    }
    return matrix[n-1][k-1]
}

func main() {
    var n, k, comb int
    fmt.Print("N = ")
    fmt.Scan(&n)
    for i := 1; i <= 5; i++ {
        fmt.Printf("K%d = ", i)
        fmt.Scan(&k)
        for row, _ := range matrix {
            for col, _ := range matrix[row] {
                matrix[row][col] = 0
            }
        }
        comb = Combin2(n, k)
        fmt.Printf("Combin(%d, %d) = %d\n", n, k, comb)
    }
}
```

Recur 8

Описать рекурсивную функцию RootK(X, K, N) вещественного типа, находящую приближенное значение корня K-й степени из числа X по формуле:
$Y_0 = 1, \quad Y_{N+1} = Y_N - (Y_N - X/(Y_N)^{K-1})/K$,где Y_N обозначает RootK(X, K, N) при фиксированных X и K. Параметры функции: X (> 0) — вещественное число, K (> 1) и N (> 0) — целые. С помощью функции RootK найти для данного числа X приближенные значения его корня K-й степени при шести данных значениях N.

Write a recursive real-valued function RootK(X, K, N) that returns an approximate value of a K-th root of X using the following formulas:
$Y_0 = 1, \quad Y_{N+1} = Y_N - (Y_N - X/(Y_N)^{K-1})/K$,where X (> 0) is a real number, K (> 1), N (> 0) are integers, Y_N denotes RootK(X, K, N) for a fixed values of X and K. Using this function, output approximate values of a K-th root of X for a given X, K and six integers N.

```
package main

import "fmt"

func PowerN(x float64, n int) float64 {
    if n == 0 {
        return 1
    }
    if n < 0 {
        return 1 / PowerN(x, -n)
    }
    if n % 2 == 0 {
        half := PowerN(x, n/2)
        return half * half
    }
    return x * PowerN(x, n-1)
}

func RootK(x float64, k int, n int) float64 {
    if n == 0 {
        return 1
    }
    prev := RootK(x, k, n-1)
    return prev - (prev - x / PowerN(prev, k-1)) / float64(k)
}

func main() {
    var (
        x, root float64
        k, n int
    )
    fmt.Print("X = ")
    fmt.Scan(&x)
    fmt.Print("K = ")
    fmt.Scan(&k)
    for i := 1; i <= 6; i++ {
        fmt.Printf("N%d = ", i)
        fmt.Scan(&n)
        root = RootK(x, k, n)
        fmt.Printf("%.8f\n\n", root)
    }
}
```

Recur 9

Описать рекурсивную функцию GCD(*A*, *B*) целого типа, находящую *наибольший общий делитель* (НОД, greatest common divisor) двух целых положительных чисел *A* и *B*, используя *алгоритм Евклида*: НОД(*A*, *B*) = НОД(*B*, *A* mod *B*), $B \neq 0$; НОД(*A*, 0) = *A*,где «mod» обозначает операцию взятия остатка от деления. С помощью этой функции найти НОД(*A*, *B*), НОД(*A*, *C*), НОД(*A*, *D*), если даны числа *A*, *B*, *C*, *D*.

Write a recursive integer function GCD(*A*, *B*) that returns the *greatest common divisor* of two positive integers *A* and *B*. Use the *Euclidean algorithm*: GCD(*A*, *B*) = GCD(*B*, *A* mod *B*), if $B \neq 0$; GCD(*A*, 0) = *A*,where "mod" denotes the operator of taking the remainder after integer division. Using this function, find the greatest common divisor for each of pairs (*A*, *B*), (*A*, *C*), (*A*, *D*) provided that integers *A*, *B*, *C*, *D* are given.

```
package main

import "fmt"

func GCD(a, b int) int {
    if b == 0 {
        return a
    }
    return GCD(b, a % b)
}

func main() {
    var a, b, c, d int
    fmt.Print("A = ")
    fmt.Scan(&a)
    fmt.Print("B = ")
    fmt.Scan(&b)
    fmt.Print("C = ")
    fmt.Scan(&c)
    fmt.Print("D = ")
    fmt.Scan(&d)
    fmt.Printf("GCD(%d, %d) = %d\n", a, b, GCD(a, b))
    fmt.Printf("GCD(%d, %d) = %d\n", a, c, GCD(a, c))
    fmt.Printf("GCD(%d, %d) = %d\n", a, d, GCD(a, d))
}
```

Recur 10

Описать рекурсивную функцию DigitSum(*K*) целого типа, которая находит сумму цифр целого числа *K*, не используя оператор цикла. С помощью этой функции найти суммы цифр для пяти данных целых чисел.

Write a recursive integer function DigitSum(*K*) that returns the sum of digits of an integer *K* (the loop statements should not be used). Using this function, output the sum of digits for each of five given integers.

```
package main
```

```
import "fmt"

func DigitSum(number int) int {
    if number == 0 {
        return 0
    }
    if number < 0 {
        number *= -1
    }
    return number % 10 + DigitSum(number / 10)
}

func main() {
    var k int
    for i := 1; i <= 5; i++ {
        fmt.Printf("K%d = ", i)
        fmt.Scan(&k)
        fmt.Printf("DigitSum(%d) = %d\n", k, DigitSum(k))
    }
}
```

Recur 11

Описать рекурсивную функцию MaxElem(A, N) целого типа, которая находит максимальный элемент целочисленного массива A размера N ($1 \le N \le 10$), не используя оператор цикла. С помощью этой функции найти максимальные элементы массивов A, B, C размера N_A, N_B, N_C соответственно.

Write a recursive integer function MaxElem(A, N) that returns the maximal element of an array A of N integers ($1 \le N \le 10$; the loop statements should not be used). Using this function, output the maximal elements of three given arrays A, B, C whose sizes are N_A, N_B, N_C respectively.

```
package main

import "fmt"

func MaxElem(a []int, n int) int {
    if n <= 0 {
        return 0
    }
    maximal := a[n-1]
    if n > 0 {
        data := MaxElem(a, n-1)
        if data > maximal {
            maximal = data
        }
    }
```

```
    return maximal
}

func main() {
    var n int
    for i := 1; i <= 3; i++ {
        fmt.Printf("N%d = ", i)
        fmt.Scan(&n)
        var array []int = make([]int, n)
        for index, _ := range array {
            fmt.Scan(&array[index])
        }
        maximal := MaxElem(array, n)
        fmt.Printf("maximal = %d\n\n", maximal)
    }
}
```

Пакети stack

TNode.go

```
package stack

type TNode struct {
    Data int
    Next *TNode
}
```

TStack.go

```
package stack

import "fmt"

type TStack struct {
    Top *TNode
}

func (s *TStack) Make() {
    var n, data int
    fmt.Print("How many nodes?\t")
    fmt.Scan(&n)
    for i := 0; i < n; i++ {
        fmt.Scan(&data)
        s.Push(data)
    }
}

func (s *TStack) ArrayMake(array []int) {
    for index, _ := range array {
        s.Push(array[index])
    }
}

func (s TStack) Display() {
    if s.IsEmpty() { return }
    for i := s.Top; i != nil; i = i.Next {
```

```
        fmt.Print(i.Data)
        if i.Next != nil {
            fmt.Printf(" %s ", string(9472))
        }
    }
    fmt.Println(" >nil")
}

func (s *TStack) Push(d int) {
    var newNode *TNode = new(TNode)
    newNode.Data = d
    newNode.Next = s.Top
    s.Top = newNode
}

func (s *TStack) Pop() int {
    var tmpNode *TNode = s.Top
    s.Top = s.Top.Next
    return tmpNode.Data
}

func (s TStack) IsEmpty() bool {
    return s.Top == nil
}

func (s TStack) Peek() int {
    return s.Top.Data
}
```

Пакети queue

TNode.go

```
package queue

type TNode struct {
    Data int
    Next *TNode
}
```

TQueue.go

```
package queue

import "fmt"

type TQueue struct {
    Head *TNode
    Tail *TNode
}

func (s *TQueue) Make() {
    var n, data int
    fmt.Print("How many nodes?\t")
    fmt.Scan(&n)
    for i := 0; i < n; i++ {
        fmt.Scan(&data)
```

```
            s.Enqueue(data)
        }
    }

    func (s TQueue) Display() {
        if s.IsEmpty() { return }
        for i := s.Head; i != nil; i = i.Next {
            fmt.Print(i.Data)
            if i.Next != nil {
                fmt.Printf(" %s ", string(9472))
            }
        }
        fmt.Println(" >nil")
    }

    func (s *TQueue) Enqueue(d int) {
        var newNode *TNode = new(TNode)
        newNode.Data = d
        if s.Head == nil {
            s.Head, s.Tail = newNode, newNode
        } else {
            s.Tail.Next = newNode
            s.Tail = newNode
        }
    }

    func (s *TQueue) Dequeue() int {
        var tmpNode = s.Head
        s.Head = s.Head.Next
        if s.Head == nil {
            s.Tail = nil
        }
        return tmpNode.Data
    }

    func (s TQueue) IsEmpty() bool {
        return s.Head == nil
    }

    func (s TQueue) First() int {
        return s.Head.Data
    }

    func (s TQueue) Last() int {
        return s.Tail.Data
    }
```

Dynamic 1

Дан адрес P_1 записи типа TNode, содержащей поле Data (целого типа) и поле Next (типа PNode — указателя на TNode). Эта запись связана полем Next со следующей

записью того же типа. Вывести значения полей Data обеих записей, а также адрес P_2 следующей записи.

An address P_1 of a record of TNode type is given. The record consists of the *Data* field (of integer type) and the *Next* field (of PNode type that refers to a variable of TNode type). The given record is linked by its *Next* field with the next record of the same type. Output the value of the *Data* field for each record and the address P_2 of the record that follows the given one.

```
package main

import (
    "fmt"
    "./stack"
)

func main() {
    var P1 *stack.TNode = new(stack.TNode)
    fmt.Print("P1.Data = ")
    fmt.Scan(&(P1.Data))

    P1.Next = new(stack.TNode)
    fmt.Print("P1.Next.Data = ")
    fmt.Scan(&(P1.Next.Data))

    var P2 *stack.TNode = P1.Next
    fmt.Printf("\nP1.Data = %d\n", P1.Data)
    fmt.Printf("P2.Data = %d\n", P2.Data)
    fmt.Printf("P2 = %p", P2)
}
```

Dynamic 2

Дан адрес P_1 записи типа TNode. Эта запись связана полем Next со следующей записью того же типа, она, в свою очередь, — со следующей, и так далее до записи, поле Next которой равно NULL (таким образом, возникает *цепочка* связанных записей). Вывести значения полей Data для всех элементов цепочки, *длину цепочки* (т. е. число ее элементов) и адрес ее последнего элемента.

An address P_1 of a record of TNode type is given. The record is linked by its *Next* field with the next record of the same type, that record is linked with the next one, and so on, until the last

record whose *Next* field equals NULL (as a result, we obtain a *chain* of linked records). Output the value of the *Data* field for each chain component, the chain *length* (that is, the amount of its components) and the address of the last chain component.

```
package main

import (
    "fmt"
    "./stack"
)

func main() {
    var st *stack.TStack = new(stack.TStack)
    st.Make()
    st.Display()
    fmt.Println()
    var (
        n int
        lastNode *stack.TNode
    )
    for i := st.Top; i != nil; i = i.Next {
        n++
        fmt.Printf("%d\t", i.Data)
        lastNode = i
    }
    fmt.Printf("\nN = %d\n", n)
    fmt.Printf("lastNode = %p\n", lastNode)
    st.Display()
}
```

Dynamic 3

Дано число D и указатель P_1 на вершину непустого стека. Добавить элемент со значением D в стек и вывести адрес P_2 новой вершины стека.

An integer D and a pointer P_1 to the top of a nonempty stack are given. Push a component with the value D onto the stack and output the address P_2 of a new top of the stack.

```
package main

import (
    "fmt"
    "./stack"
)

func main() {
    var d int
    fmt.Print("D = ")
    fmt.Scan(&d)
```

```
    var st stack.TStack
    st.Make()
    st.Display()
    st.Push(d)
    st.Display()
}
```

Dynamic 4

Дано число N (> 0) и набор из N чисел. Создать стек, содержащий исходные числа (последнее число будет вершиной стека), и вывести указатель на его вершину.

An integer N (> 0) and a sequence of N integers are given. Create a stack that contains N components with the given values (a component with the last value must be the top of the stack) and output a pointer to the top of the stack.

```
package main

import (
    "fmt"
    "./stack"
)

func main() {
    var n int
    fmt.Print("N = ")
    fmt.Scan(&n)
    var array []int = make([]int, n)
    for index, _ := range array {
        fmt.Scan(&array[index])
    }
    var st stack.TStack
    st.ArrayMake(array)
    fmt.Printf("P1 = %p\n", st.Top)
    st.Display()
}
```

Dynamic 5

Дан указатель P_1 на вершину непустого стека. Извлечь из стека первый (верхний) элемент и вывести его значение D, а также адрес P_2 новой вершины стека. Если после извлечения элемента стек окажется пустым, то положить P_2 = NULL. После извлечения элемента из стека освободить память, занимаемую этим элементом.

A pointer P_1 to the top of a nonempty stack is given. Pop the top component off the stack and output its value D and the address P_2 of a new top of the stack. If the stack will be empty after popping the component then P_2 must be equal to NULL. After popping the component release the memory allocated for this component.

```
package main

import (
    "fmt"
    "./stack"
)

func main() {
    var st stack.TStack
    st.Make()
    st.Display()
    fmt.Println()
    d := st.Pop()
    fmt.Printf("D = %d\n", d)
    fmt.Printf("P2 = %p\n", st.Top)
    st.Display()
}
```

Dynamic 6

Дан указатель P_1 на вершину стека, содержащего не менее десяти элементов. Извлечь из стека первые девять элементов и вывести их значения. Вывести также адрес новой вершины стека. После извлечения элементов из стека освобождать память, которую они занимали.

A pointer P_1 to the top of a stack is given; the stack contains at least ten components. Pop the first nine components off the stack and output their values and the address P_2 of a new top of the stack. After popping components release the memory allocated for these components.

```
package main

import (
    "fmt"
    "./stack"
)

func main() {
```

```
    var st stack.TStack
    st.Make()
    st.Display()
    fmt.Println()
    for i := 0; i < 9; i++ {
        fmt.Printf("%d\t", st.Pop())
    }
    fmt.Printf("\nP2 = %p\n", st.Top)
    st.Display()
}
```

Dynamic 7

Дан указатель P_1 на вершину стека (если стек пуст, то P_1 = NULL). Извлечь из стека все элементы и вывести их значения. Вывести также количество извлеченных элементов N (для пустого стека вывести 0). После извлечения элементов из стека освобождать память, которую они занимали.

A pointer P_1 to the top of a stack is given (if the stack is empty then P_1 equals NULL). Pop all components off the stack and output their values. Also output the amount of popped components (if the stack is empty then output 0). After popping components release the memory allocated for these components.

```
package main

import (
    "fmt"
    "./stack"
)

func main() {
    var st stack.TStack
    st.Make()
    st.Display()
    fmt.Println()
    var n int
    for !st.IsEmpty() {
        fmt.Printf("%d\t", st.Pop())
        n++
    }
    fmt.Printf("\nN = %d\n", n)
}
```

Dynamic 8

Даны указатели P_1 и P_2 на вершины двух непустых стеков. Переместить все элементы из первого стека во второй (в результате элементы первого стека будут располагаться во втором стеке в порядке, обратном исходному) и вывести адрес новой вершины второго стека. Операции выделения и освобождения памяти не использовать.

Two pointers P_1 and P_2 that refer to the tops of two nonempty stacks are given. Move all components from the first stack into the second one (as a result, all components of the first stack will be contained within the second stack in inverse order). Output the address of a new top of the second stack. Do not use operations of allocating and freeing memory.

```
package main

import (
    "fmt"
    "./stack"
)

func main() {
    var st1, st2 stack.TStack
    fmt.Println("Stack #1")
    st1.Make()
    st1.Display()
    fmt.Println("Stack #2")
    st2.Make()
    st2.Display()
    fmt.Println()
    for !st1.IsEmpty() {
        st2.Push(st1.Pop())
    }
    fmt.Printf("\nP3 = %p\n", st2.Top)
    st2.Display()
}
```

Dynamic 9

Даны указатели P_1 и P_2 на вершины двух непустых стеков. Перемещать элементы из первого стека во второй, пока значение вершины первого стека не станет четным (перемещенные элементы первого стека будут располагаться во втором стеке в порядке, обратном исходному). Если в первом стеке нет элементов с четными

значениями, то переместить из первого стека во второй все элементы. Вывести адреса новых вершин первого и второго стека (если первый стек окажется пустым, то вывести для него константу NULL). Операции выделения и освобождения памяти не использовать.

Two pointers P_1 and P_2 that refer to the tops of two nonempty stacks are given. Move components from the first stack into the second one until the value of the top component of the first stack is equal to an even number (as a result, all components having been moved will be contained within the second stack in inverse order). If the first stack contains no components with even values then move all its components. Output the address of a new top for each stack (if the first stack will be empty then output NULL for this stack). Do not use operations of allocating and freeing memory.

```
package main

import (
    "fmt"
    "./stack"
)

func main() {
    var st1, st2 stack.TStack
    fmt.Println("Stack #1")
    st1.Make()
    st1.Display()
    fmt.Println("Stack #2")
    st2.Make()
    st2.Display()
    fmt.Println()
    for !st1.IsEmpty() && (st1.Peek() % 2 != 0) {
        st2.Push(st1.Pop())
    }
    fmt.Printf("\nP3 = %p\n", st1.Top)
    st1.Display()
    fmt.Printf("\nP4 = %p\n", st2.Top)
    st2.Display()
}
```

Dynamic 10

Дан указатель P_1 на вершину непустого стека. Создать два новых стека, переместив в первый из них все элементы

исходного стека с четными значениями, а во второй — с нечетными (элементы в новых стеках будут располагаться в порядке, обратном исходному; один из этих стеков может оказаться пустым). Вывести адреса вершин полученных стеков (для пустого стека вывести NULL). Операции выделения и освобождения памяти не использовать.

A pointer P_1 to the top of a nonempty stack is given. Create two new stacks by moving the given stack components whose values are even (odd) numbers into the first (second) new stack respectively. As a result, all components having been moved will be contained within each new stack in inverse order; one of the new stacks may be empty. Output the address of the top for each new stack (if one of the new stacks will be empty then output NULL for this stack). Do not use operations of allocating and freeing memory.

```
package main

import (
    "fmt"
    "./stack"
)

func main() {
    var st, st1, st2 stack.TStack
    st.Make()
    st.Display()
    fmt.Println()
    for !st.IsEmpty() {
        if st.Peek() % 2 == 0 {
            st1.Push(st.Pop())
        } else {
            st2.Push(st.Pop())
        }
    }
    fmt.Printf("P2 = %p\n", st1.Top)
    st1.Display()
    fmt.Printf("P3 = %p\n", st2.Top)
    st2.Display()
}
```

Dynamic 11

Дан указатель P_1 на вершину стека (если стек пуст, то P_1 = NULL). Также дано число N (> 0) и набор из N чисел.

Описать тип TStack — запись с одним полем Top типа PNode (поле указывает на *вершину* стека) — и процедуру Push(*S*, *D*), которая добавляет в стек *S* новый элемент со значением *D* (*S* — входной и выходной параметр типа TStack, *D* — входной параметр целого типа). С помощью процедуры Push добавить в исходный стек данный набор чисел (последнее число будет вершиной стека) и вывести адрес новой вершины стека.

A pointer P_1 to the top of a stack is given (if the stack is empty then P_1 equals NULL). Also an integer N (> 0) and a sequence of N integers are given. Define a new type called TStack that is a record with one field, *Top*, of PNode type (the field refers to the top of a stack). Also write a procedure Push(*S*, *D*) that pushes a new component with the value *D* onto a stack *S* (a record *S* of TStack type is an input and output parameter, an integer *D* is an input parameter). Using this procedure, push all elements of the given sequence onto the given stack (the last number must be the value of the top component). Output the address of a new top of the stack.

```
package main

import (
    "fmt"
    "./stack"
)

func main() {
    var st stack.TStack
    st.Make()
    st.Display()
    fmt.Print()
    var n, data int
    fmt.Print("N = ")
    fmt.Scan(&n)
    for i := 0; i < n; i++ {
        fmt.Scan(&data)
        st.Push(data)
    }
    fmt.Printf("\nP2 = %p\n", st.Top)
    st.Display()
}
```

Dynamic 12

Дан указатель P_1 на вершину стека, содержащего не менее пяти элементов. Используя тип TStack (см. задание Dynamic11), описать функцию Pop(*S*) целого типа, которая извлекает из стека *S* первый (верхний) элемент, возвращает его значение и освобождает память, которую занимал извлеченный элемент (*S* — входной и выходной параметр типа TStack). С помощью функции Pop извлечь из исходного стека пять элементов и вывести их значения. Вывести также указатель на новую вершину стека (если результирующий стек окажется пустым, то этот указатель должен быть равен NULL).

A pointer P_1 to the top of a stack is given; the stack contains at least five components. Using the TStack type (see Dynamic11), write an integer function Pop(*S*) that pops the top component off a stack *S*, releases memory allocated for this component and returns its value (a record *S* of TStack type is an input and output parameter). Using this function, pop five components off the given stack and output their values. Also output a pointer that refers to a new top of the stack (if the stack will be empty then this pointer must be equal to NULL).

```
package main

import (
    "fmt"
    "./stack"
)

func main() {
    var st stack.TStack
    st.Make()
    st.Display()
    fmt.Println()
    for i := 0; i < 5; i++ {
        fmt.Printf("%d\t", st.Pop())
    }
    fmt.Printf("\nP2 = %p\n", st.Top)
    st.Display()
}
```

Dynamic 13

Дан указатель P_1 на вершину стека. Используя тип TStack (см. задание Dynamic11), описать функции StackIsEmpty(*S*) логического типа (возвращает true, если стек *S* пуст, и false в противном случае) и Peek(*S*) целого типа (возвращает значение вершины непустого стека *S*, не удаляя ее из стека). В обеих функциях переменная *S* является входным параметром типа TStack. С помощью этих функций, а также функции Pop из задания Dynamic12, извлечь из исходного стека пять элементов (или все содержащиеся в нем элементы, если их менее пяти) и вывести их значения. Вывести также значение функции StackIsEmpty для результирующего стека и, если результирующий стек не является пустым, значение и адрес его новой вершины.

A pointer P_1 to the top of a stack is given. Using the TStack type (see Dynamic11), write two functions: a logical function StackIsEmpty(*S*) that returns true if a stack *S* is empty, and false otherwise, and an integer function Peek(*S*) that returns the value of the top component of the stack *S*. A record *S* of TStack type is an input parameter for each function. Using these functions and the Pop function from the task Dynamic12, pop five components (or all stack components if their amount is less than five) off the given stack and output their values. Also output the return value of the StackIsEmpty function for the resulting stack. At last, in the case of the nonempty resulting stack, output the value and the address of its top component.

```
package main

import (
    "fmt"
    "./stack"
)

func main() {
    var st stack.TStack
    st.Make()
    st.Display()
    fmt.Println()
    for i := 0; !st.IsEmpty() && i < 5; i++ {
        fmt.Printf("%d\t", st.Pop())
```

```
    }
    fmt.Printf("\nStackIsEmpty: %t\n", st.IsEmpty())
    if !st.IsEmpty() {
        fmt.Printf("Peek: %d\n", st.Peek())
        fmt.Printf("P2 = %p\n", st.Top)
        st.Display()
    }
}
```

Dynamic 14

Дан набор из 10 чисел. Создать очередь, содержащую данные числа в указанном порядке (первое число будет размещаться в начале очереди, последнее — в конце), и вывести указатели P_1 и P_2 на начало и конец очереди.

A sequence of 10 integers is given. Create a queue that contains components with the given values (a component with the first value must be the head of the queue, a component with the last value must be the tail of the queue) and output pointers P_1 and P_2 to the head and tail of the queue respectively.

```
package main

import (
    "fmt"
    "./queue"
)

func main() {
    var (
        number int
        qu queue.TQueue
    )
    for i := 0; i < 10; i++ {
        fmt.Scan(&number)
        qu.Enqueue(number)
    }
    fmt.Printf("Head = %p\nTail = %p\n", qu.Head, qu.Tail)
    qu.Display()
}
```

Dynamic 15

Дан набор из 10 чисел. Создать две очереди: первая должна содержать числа из исходного набора с нечетными номерами (1, 3, …, 9), а вторая — с четными (2, 4, …, 10); порядок чисел в каждой очереди должен совпадать с

порядком чисел в исходном наборе. Вывести указатели на начало и конец первой, а затем второй очереди.

A sequence of 10 integers is given. Create two queues; the first one must contain the given integers with odd order numbers (1, 3, …, 9), the second one must contain the given integers with even order numbers (2, 4, …, 10). Output pointers to the head and tail of the first queue and then output pointers to the head and tail of the second one.

```
package main

import (
    "fmt"
    "./queue"
)

func main() {
    var (
        number int
        qu1, qu2 queue.TQueue
    )
    for i := 1; i <= 10; i++ {
        fmt.Scan(&number)
        if i % 2 != 0 {
            qu1.Enqueue(number)
        } else {
            qu2.Enqueue(number)
        }
    }
    fmt.Println("\nQueue #1")
    fmt.Printf("Head = %p\t\tTail = %p\n", qu1.Head, qu1.Tail)
    qu1.Display()
    fmt.Println("\nQueue #2")
    fmt.Printf("Head = %p\t\tTail = %p\n", qu2.Head, qu2.Tail)
    qu2.Display()
}
```

Dynamic 16

Дан набор из 10 чисел. Создать две очереди: первая должна содержать все нечетные, а вторая — все четные числа из исходного набора (порядок чисел в каждой очереди должен совпадать с порядком чисел в исходном наборе). Вывести указатели на начало и конец первой, а затем второй очереди (одна из очередей может оказаться пустой; в этом случае вывести для нее две константы NULL).

A sequence of 10 integers is given. Create two queues; the first one must contain the given integers with odd values (in the same order), the second one must contain the given integers with even values (in the same order). Output pointers to the head and tail of the first queue and then output pointers to the head and tail of the second one (if one of the queues will be empty then output NULL twice for this queue).

```
package main

import (
    "fmt"
    "./queue"
)

func main() {
    var (
        number int
        qu1, qu2 queue.TQueue
    )
    for i := 1; i <= 10; i++ {
        fmt.Scan(&number)
        if number % 2 != 0 {
            qu1.Enqueue(number)
        } else {
            qu2.Enqueue(number)
        }
    }
    fmt.Println("\nQueue #1")
    fmt.Printf("Head = %p\t\tTail = %p\n", qu1.Head, qu1.Tail)
    qu1.Display()
    fmt.Println("\nQueue #2")
    fmt.Printf("Head = %p\t\tTail = %p\n", qu2.Head, qu2.Tail)
    qu2.Display()
}
```

Dynamic 17

Дано число D и указатели P_1 и P_2 на начало и конец очереди (если очередь является пустой, то $P_1 = P_2 = \text{NULL}$). Добавить элемент со значением D в конец очереди и вывести новые адреса начала и конца очереди.

An integer D and pointers P_1 and P_2 to the head and tail of a queue are given (if the queue is empty then the pointers equal NULL). Add a component with the value D to the end of the

queue and output the new addresses of the head and tail of the queue.

```
package main

import (
    "fmt"
    "./queue"
)

func main() {
    var (
        d int
        qu queue.TQueue
    )
    fmt.Print("D = ")
    fmt.Scan(&d)
    qu.Make()
    qu.Display()
    fmt.Println()
    qu.Enqueue(d)
    fmt.Printf("P3 = %p\t\tP4 = %p\n", qu.Head, qu.Tail)
    qu.Display()
}
```

Dynamic 18

Дано число D и указатели P_1 и P_2 на начало и конец очереди, содержащей не менее двух элементов. Добавить элемент со значением D в конец очереди и извлечь из очереди первый (начальный) элемент. Вывести значение извлеченного элемента и новые адреса начала и конца очереди. После извлечения элемента из очереди освободить память, занимаемую этим элементом.

An integer D and pointers P_1 and P_2 to the head and tail of a queue are given; the queue contains at least two components. Add a component with the value D to the end of the queue and remove the first component from the front of the queue. Output the value of the component being removed and also output the new addresses of the head and tail of the queue. After removing the component release the memory allocated for this component.

```
package main

import (
    "fmt"
```

```
    "./queue"
)

func main() {
    var (
        d int
        qu queue.TQueue
    )
    fmt.Print("D = ")
    fmt.Scan(&d)
    qu.Make()
    qu.Display()
    fmt.Println()
    qu.Enqueue(d)
    fmt.Printf("Dequeued: %d\n", qu.Dequeue())
    fmt.Printf("P3 = %p\t\tP4 = %p\n", qu.Head, qu.Tail)
    qu.Display()
}
```

Dynamic 19

Дано число N (> 0) и указатели P_1 и P_2 на начало и конец непустой очереди. Извлечь из очереди N начальных элементов и вывести их значения (если очередь содержит менее N элементов, то извлечь все ее элементы). Вывести также новые адреса начала и конца очереди (для пустой очереди дважды вывести NULL). После извлечения элементов из очереди освобождать память, которую они занимали.

An integer N (> 0) and pointers P_1 and P_2 to the head and tail of a nonempty queue are given. Remove N initial components from the queue and output their values (if the queue contains less than N components then remove all its components). Also output the new addresses of the head and tail of the queue (if the resulting queue will be empty then output NULL twice). After removing components release the memory allocated for them.

```
package main

import (
    "fmt"
    "./queue"
)

func main() {
    var n int
    fmt.Print("N = ")
```

```
    fmt.Scan(&n)
    var qu queue.TQueue
    qu.Make()
    qu.Display()
    fmt.Println()
    for i := 0; !qu.IsEmpty() && i < n; i++ {
        fmt.Printf("%d\t", qu.Dequeue())
    }
    fmt.Printf("\nHead = %p\t\tTail = %p\n", qu.Head, qu.Tail)
    qu.Display()
}
```

Dynamic 20

Даны указатели P_1 и P_2 на начало и конец непустой очереди. Извлекать из очереди элементы, пока значение начального элемента очереди не станет четным, и выводить значения извлеченных элементов (если очередь не содержит элементов с четными значениями, то извлечь все ее элементы). Вывести также новые адреса начала и конца очереди (для пустой очереди дважды вывести NULL). После извлечения элементов из очереди освобождать память, которую они занимали.

Pointers P_1 and P_2 to the head and tail of a nonempty queue are given. Remove components from the front of the queue until the value of the head of the queue is equal to an even number; output values of all components being removed (if the queue contains no components with even values then remove all its components). Also output the new addresses of the head and tail of the queue (if the resulting queue will be empty then output NULL twice). After removing components release the memory allocated for them.

```
package main

import (
    "fmt"
    "./queue"
)

func main() {
    var qu queue.TQueue
    qu.Make()
    qu.Display()
    fmt.Println()
```

```
    for !qu.IsEmpty() && (qu.First() % 2 != 0) {
        fmt.Printf("%d\t", qu.Dequeue())
    }
    fmt.Printf("\nHead = %p\t\tTail = %p\n", qu.Head, qu.Tail)
    qu.Display()
}
```

Пакети tree

TNode.go

```
package tree

type TNode struct {
    Data int
    Left *TNode
    Right *TNode
    Parent *TNode
}
```

Field.go

```
package tree

import "fmt"

const (
    SPACE int8 = 0
    LEFT int8 = 1
    CENTER int8 = 2
    RIGHT int8 = 3
    DATA int8 = 4
)

type Field struct {
    nodes [][]*TNode
        chars [][]int8
        height int
        width int
        cellWidth int
        left string
        center string
        right string
}

func (f *Field) Init(node *TNode, height int, width int) {
    f.height = height
    f.width = width
    f.left = string(9484)
        f.center = string(9472)
        f.right = string(9488)
    f.cellWidth = 4
    f.nodes = make([][]*TNode, f.height)
    f.chars = make([][]int8, f.height)
    for row := 0; row < f.height; row++ {
        f.nodes[row] = make([]*TNode, f.width)
        f.chars[row] = make([]int8, f.width)
    }
```

```
    var column int = 0
    f.fillNodes(node, 0, &column)
    f.fillChars()
}

func (f *Field) fillNodes(node *TNode, row int, col *int) {
    if node == nil { return }
    f.fillNodes(node.Left, row+1, col)
    f.nodes[row][*col] = node
    (*col)++
    f.fillNodes(node.Right, row+1, col)
}

func (f *Field) fillChars() {
    for row := 0; row < f.height; row++ {
        for col := 0; col < f.width; col++ {
            if f.nodes[row][col] == nil { continue }
            f.chars[row][col] = DATA
            if f.nodes[row][col].Left != nil {
                for k := col-1; k >= 0; k-- {
                    if f.nodes[row+1][k] != nil {
                        f.chars[row][k] = LEFT
                        break
                    }
                    f.chars[row][k] = CENTER
                }
            }
            if f.nodes[row][col].Right != nil {
                for col++; col < f.width; col++ {
                    if f.nodes[row+1][col] != nil {
                        f.chars[row][col] = RIGHT
                        break
                    }
                    f.chars[row][col] = CENTER
                }
            }
        }
    }
}

func (f Field) getNextNode(row int, col int) *TNode {
    for col++; col < f.width; col++ {
        if f.nodes[row][col] != nil {
            return f.nodes[row][col]
        }
    }
    return nil
}

func getLength(number int) int {
    var len int = 0
    if number < 0 {
        len++
        number *= -1
    }
    for number > 0 {
        len++
        number /= 10
    }
    return len
}
```

```
func printChars(str string, count int) {
    for count > 0 {
        fmt.Print(str)
        count--
    }
}

func (f Field) Display() {
    var (
        nextNode *TNode
        data, spaces, half int
    )
    for row, _ := range f.chars {
        for col, _ := range f.chars[row] {
            switch f.chars[row][col] {
                case SPACE: printChars(" ", f.cellWidth)
                case LEFT:
                    nextNode = f.getNextNode(row, col)
                    data = nextNode.Data
                    spaces = f.cellWidth - getLength(data)
                    half = spaces / 2
                    printChars(" ", half)
                    printChars(f.left, 1)
                    printChars(f.center, f.cellWidth - 1)
                case CENTER: printChars(f.center, f.cellWidth)
                case RIGHT:
                    printChars(f.center, f.cellWidth - 1)
                    printChars(f.right, 1)
                    printChars(" ", spaces - half)
                case DATA:
                    if f.nodes[row][col].Left == nil {
                        data = f.nodes[row][col].Data
                        spaces = f.cellWidth - getLength(data)
                        half = spaces / 2
                        printChars(" ", half)
                    }
                    fmt.Print(data)
                    if f.nodes[row][col].Right == nil {
                        printChars(" ", spaces - half)
                    }
            }
        }
        fmt.Println()
    }
}
```

Tree.go

```
package tree

import "fmt"
import "math/rand"

type Tree struct {
    root *TNode
    current *TNode
    field *Field
    level int
    nodeCount int
}

func (t *Tree) Make() {
```

```
    if t.root == nil {
        t.root = new(TNode)
        t.nodeCount++
        fmt.Print("root's Data: ")
        fmt.Scan(&(t.root.Data))
        t.current = t.root
    }
    var answer int
    fmt.Print("Where to go? (0-Exit; 1-Left; 2-Right; 3-Parent):\t")
    fmt.Scan(&answer)
    if answer != 1 && answer != 2 {
        switch answer {
            case 0: t.current = t.root; return
            case 3: t.current = t.current.Parent;
        }
        t.Make()
        return
    }
    var newNode *TNode = new(TNode)
    t.nodeCount++
    fmt.Print("Data = ")
    fmt.Scan(&(newNode.Data))
    newNode.Parent = t.current
    if answer == 1 {
        t.current.Left = newNode
    } else {
        t.current.Right = newNode
    }
    t.current = newNode
    t.Make()
}

func (t *Tree) AutoMake(count int) {
    if count <= 0 {
        t.current = t.root
        return
    }
    if t.root == nil {
        t.root = new(TNode)
        t.root.Data = rand.Intn(90) + 10
        t.nodeCount++
        count--
        t.current = t.root
        if count <= 0 { return }
    }
    var direction int
    for {
        direction = rand.Intn(3) + 1
        if !(direction == 3 && t.current.Parent == nil) {
            break
        }
    }
    var exit = false
    switch direction {
        case 1:
            if t.current.Left != nil {
                t.current = t.current.Left
                exit = true
            }
        case 2:
            if t.current.Right != nil {
                t.current = t.current.Right
```

```
                exit = true
            }
        case 3:
            t.current = t.current.Parent
            exit = true
    }
    if exit {
        t.AutoMake(count)
        return
    }
    var newNode *TNode = new(TNode)
    t.nodeCount++
    count--
    newNode.Data = rand.Intn(90) + 10
    newNode.Parent = t.current
    if direction == 1 {
        t.current.Left = newNode
    } else {
        t.current.Right = newNode
    }
    t.current = newNode
    t.AutoMake(count)
}

func (t Tree) GetNodeCount() int {
    return t.nodeCount
}

func (t* Tree) GetLevel() int {
    if t.level == 0 {
        t.setLevel(0)
    }
    return t.level;
}

func (t* Tree) setLevel(level int) {
    if (t.current == nil) {
        t.current = t.root
        return
    }
    if level > t.level {
        t.level = level
    }
    var current *TNode = t.current
    t.current = current.Left
    t.setLevel(level + 1)
    t.current = current.Right
    t.setLevel(level + 1)
}

func (t Tree) Display() {
    if t.root == nil { return }
    if t.field == nil {
        t.field = new(Field)
        t.field.Init(t.root, t.GetLevel()+1, t.GetNodeCount())
    }
    t.field.Display()
}

func (t Tree) GetDataCount(data int) int {
    if t.current == nil {
        t.current = t.root
```

```
        return 0
    }
    var current *TNode = t.current
    var count = 0
    if t.current.Data == data {
        count++
    }
    t.current = current.Left
    count += t.GetDataCount(data)
    t.current = current.Right
    count += t.GetDataCount(data)
    return count
}

func (t Tree) GetDataSum() int {
    if t.current == nil {
        t.current = t.root
        return 0
    }
    var current *TNode = t.current
    sum := current.Data
    t.current = current.Left
    sum += t.GetDataSum()
    t.current = current.Right
    sum += t.GetDataSum()
    return sum
}

func (t Tree) GetLeftCount() int {
    if t.current == nil {
        t.current = t.root
        return 0
    }
    var current *TNode = t.current
    var count = 0
    if current.Parent != nil && current.Parent.Left == current {
        count++
    }
    t.current = current.Left
    count += t.GetLeftCount()
    t.current = current.Right
    count += t.GetLeftCount()
    return count
}

func (t Tree) GetLeafCount() int {
    if t.current == nil {
        t.current = t.root
        return 0
    }
    var current *TNode = t.current
    var count = 0
    if current.Left == nil && current.Right == nil {
        count++
    }
    t.current = current.Left
    count += t.GetLeafCount()
    t.current = current.Right
    count += t.GetLeafCount()
    return count
}
```

```
func (t Tree) GetLeafSum() int {
    if t.current == nil {
        t.current = t.root
        return 0
    }
    var current *TNode = t.current
    var sum = 0
    if current.Left == nil && current.Right == nil {
        sum += current.Data
    }
    t.current = current.Left
    sum += t.GetLeafSum()
    t.current = current.Right
    sum += t.GetLeafSum()
    return sum
}

func (t Tree) GetRightLeafCount() int {
    if t.current == nil {
        t.current = t.root
        return 0
    }
    var current *TNode = t.current
    var count int = 0
    if current.Left == nil && current.Right == nil &&
        current.Parent != nil && current.Parent.Right == current {
            count++
    }
    t.current = current.Left
    count += t.GetRightLeafCount()
    t.current = current.Right
    count += t.GetRightLeafCount()
    return count
}

func (t Tree) LevelToArray(array []int, level int, toCount bool) {
    if t.current == nil {
        t.current = t.root
        return
    }
    var current *TNode = t.current
    if toCount {
        array[level]++
    } else {
        array[level] += current.Data
    }
    t.current = current.Left
    t.LevelToArray(array, level+1, toCount)
    t.current = current.Right
    t.LevelToArray(array, level+1, toCount)
}

func (t Tree) Infix() {
    if t.current == nil {
        t.current = t.root
        return
    }
    var current *TNode = t.current
    t.current = current.Left
    t.Infix()
    fmt.Printf("%d\t", current.Data)
    t.current = current.Right
```

```
    t.Infix()
}

func (t Tree) Prefix() {
    if t.current == nil {
        t.current = t.root
        return
    }
    var current *TNode = t.current
    fmt.Printf("%d\t", current.Data)
    t.current = current.Left
    t.Prefix()
    t.current = current.Right
    t.Prefix()
}

func (t Tree) Postfix() {
    if t.current == nil {
        t.current = t.root
        return
    }
    var current *TNode = t.current
    t.current = current.Left
    t.Postfix()
    t.current = current.Right
    t.Postfix()
    fmt.Printf("%d\t", current.Data)
}

func (t Tree) InfixToN(index *int, n int) {
    if t.current == nil {
        t.current = t.root
        return
    }
    var current *TNode = t.current
    t.current = current.Left
    t.InfixToN(index, n)
    if *index < n {
        fmt.Printf("%d\t", current.Data)
        (*index)++
    } else {
        t.current = t.root
        return
    }
    t.current = current.Right
    t.InfixToN(index, n)
}

func (t Tree) PostfixFromN(index *int, n int) {
    if t.current == nil {
        t.current = t.root
        return
    }
    var current *TNode = t.current
    t.current = current.Left
    t.PostfixFromN(index, n)
    t.current = current.Right
    t.PostfixFromN(index, n)
    (*index)++
    if *index >= n {
        fmt.Printf("%d\t", current.Data)
    }
```

```
}

func (t Tree) PrefixBetween(index *int, n1 int, n2 int) {
    if t.current == nil {
        t.current = t.root
        return
    }
    (*index)++
    if *index >= n1 && *index <= n2 {
        fmt.Printf("%d\t", t.current.Data)
    } else if *index > n2 {
        t.current = t.root
        return
    }
    var current *TNode = t.current
    t.current = current.Left
    t.PrefixBetween(index, n1, n2)
    t.current = current.Right
    t.PrefixBetween(index, n1, n2)
}

func (t Tree) PrintLevel(index int, level int) int {
    if t.current == nil {
        t.current = t.root
        return 0
    }
    var count = 0
    if index == level {
        count++
        fmt.Printf("%d\t", t.current.Data)
    }
    var current *TNode = t.current
    t.current = current.Left
    count += t.PrintLevel(index+1, level)
    t.current = current.Right
    count += t.PrintLevel(index+1, level)
    return count
}

func (t Tree) GetMaxData() int {
    if t.current == nil {
        t.current = t.root
        return 0
    }
    var current *TNode = t.current
    var maximal int = current.Data
    if current.Left != nil {
        t.current = current.Left
        data := t.GetMaxData()
        if data > maximal {
            maximal = data
        }
    }
    if current.Right != nil {
        t.current = current.Right
        data := t.GetMaxData()
        if data > maximal {
            maximal = data
        }
    }
    t.current = t.root
    return maximal
```

```
}

func (t Tree) GetMinData() int {
    if t.current == nil {
        t.current = t.root
        return 0
    }
    var current *TNode = t.current
    var minimal int = current.Data
    if current.Left != nil {
        t.current = current.Left
        data := t.GetMinData()
        if data < minimal {
            minimal = data
        }
    }
    if current.Right != nil {
        t.current = current.Right
        data := t.GetMinData()
        if data < minimal {
            minimal = data
        }
    }
    t.current = t.root
    return minimal
}

func (t Tree) GetLeafDataCount(data int) int {
    if t.current == nil {
        t.current = t.root
        return 0
    }
    var current *TNode = t.current
    var count = 0
    if current.Data == data && current.Left == nil && current.Right == nil {
        count++
    }
    t.current = current.Left
    count += t.GetLeafDataCount(data)
    t.current = current.Right
    count += t.GetLeafDataCount(data)
    return count
}
```

Tree 1

Дан адрес P_1 записи типа TNode с полями Data (целого типа), Left и Right (типа PNode — указателя на TNode). Эта запись (*корень дерева*) связана полями Left и Right с записями того же типа (левой и правой *дочерней вершиной*). Вывести значения полей Data корня, его левой и правой

дочерних вершин, а также адреса левой и правой дочерних вершин в указанном порядке.

An address P_1 of a record of TNode type is given. The record consists of the *Data* field (of integer type) and the *Left* and *Right* fields (of PNode type). The given record (*a tree root*) is linked by its *Left* and *Right* fields with records of the same type (named the left and right *child nodes* respectively). Output the Data fields of the tree root and its left and right children. Also output the addresses of left and right child nodes.

```
package main

import "./tree"
import "fmt"

func main() {
    var P1 *tree.TNode = new(tree.TNode)
    fmt.Print("P1.Data = ")
    fmt.Scan(&(P1.Data))

    P1.Left = new(tree.TNode)
    fmt.Print("P1.Left.Data = ")
    fmt.Scan(&(P1.Left.Data))

    P1.Right = new(tree.TNode)
    fmt.Print("P1.Right.Data = ")
    fmt.Scan(&(P1.Right.Data))

    fmt.Printf("P1.Data = %d\n", P1.Data)
    fmt.Printf("P1.Left.Data = %d\n", P1.Left.Data)
    fmt.Printf("P1.Right.Data = %d\n", P1.Right.Data)
    fmt.Printf("P2 = %p\n", P1.Left)
    fmt.Printf("P3 = %p\n", P1.Right)
}
```

Tree 2

Дан адрес P_1 записи типа TNode — корня дерева. Эта запись связана полями Left и Right с другими записями того же типа (дочерними вершинами), они, в свою очередь, — со своими дочерними вершинами, и так далее до записей, поля Left и Right которых равны NULL (у некоторых вершин может быть равно NULL одно из полей — Left или Right). Вывести количество вершин дерева.

An address P_1 of a record of TNode type (a tree root) is given. This record is linked by its *Left* and *Right* fields with records of the same type (child nodes); they, in turn, are linked with their own child nodes and so on, until records whose *Left* and *Right* fields are equal to NULL. Some of the nodes may have one field (*Left* or *Right*) equals NULL. Output the amount of tree nodes.

```
package main

import (
    "fmt"
    "math/rand"
    "time"
    "./tree"
)

func main() {
    rand.Seed(time.Now().UnixNano())
    var tr tree.Tree
    tr.AutoMake(rand.Intn(15)+1)
    tr.Display()
    fmt.Printf("\n%d\n", tr.GetNodeCount())
}
```

Tree 3

Дан указатель P_1 на корень непустого дерева и число K. Вывести количество вершин дерева, значение которых равно K.

A pointer P_1 to the root of a nonempty tree and an integer K are given. Output the amount of nodes whose value equals K.

```
package main

import (
    "fmt"
    "math/rand"
    "time"
    "./tree"
)

func main() {
    rand.Seed(time.Now().UnixNano())
    var tr tree.Tree
    tr.AutoMake(rand.Intn(15)+1)
    tr.Display()
    var k int
    fmt.Print("K = ")
    fmt.Scan(&k)
    fmt.Printf("\n%d\n", tr.GetDataCount(k))
```

```
}
```

Tree 4

Дан указатель P_1 на корень непустого дерева. Вывести сумму значений всех вершин данного дерева.

A pointer P_1 to the root of a nonempty tree is given. Output the sum of values of all tree nodes.

```
package main

import (
    "fmt"
    "math/rand"
    "time"
    "./tree"
)

func main() {
    rand.Seed(time.Now().UnixNano())
    var tr tree.Tree
    tr.AutoMake(rand.Intn(15)+1)
    tr.Display()
    fmt.Printf("\n%d\n", tr.GetDataSum())
}
```

Tree 5

Дан указатель P_1 на корень непустого дерева. Вывести количество вершин дерева, являющихся левыми дочерними вершинами (корень дерева не учитывать).

A pointer P_1 to the root of a nonempty tree is given. Output the amount of left child nodes (the tree root should not be counted).

```
package main

import (
    "fmt"
    "math/rand"
    "time"
    "./tree"
)

func main() {
    rand.Seed(time.Now().UnixNano())
    var tr tree.Tree
    tr.AutoMake(rand.Intn(15)+1)
    tr.Display()
    fmt.Printf("\n%d\n", tr.GetLeftCount())
```

```
}
```

Tree 6

Дан указатель P_1 на корень непустого дерева. *Листом дерева* называется его вершина, не имеющая дочерних вершин. Вывести количество листьев для данного дерева.

A pointer P_1 to the root of a nonempty tree is given. Nodes without children are called *the external nodes* or *the leaf nodes* (*the leaves*). Output the amount of leaf nodes.

```
package main

import (
    "fmt"
    "math/rand"
    "time"
    "./tree"
)

func main() {
    rand.Seed(time.Now().UnixNano())
    var tr tree.Tree
    tr.AutoMake(rand.Intn(15)+1)
    tr.Display()
    fmt.Printf("\n%d\n", tr.GetLeafCount())
}
```

Tree 7

Дан указатель P_1 на корень непустого дерева. Вывести сумму значений всех листьев данного дерева.

A pointer P_1 to the root of a nonempty tree is given. Output the sum of values of all tree leaves.

```
package main

import (
    "fmt"
    "math/rand"
    "time"
    "./tree"
)

func main() {
    rand.Seed(time.Now().UnixNano())
    var tr tree.Tree
    tr.AutoMake(rand.Intn(15)+1)
    tr.Display()
```

```
    fmt.Printf("\n%d\n", tr.GetLeafSum())
}
```

Tree 8

Дан указатель P_1 на корень дерева, содержащего по крайней мере две вершины. Вывести количество листьев дерева, являющихся правыми дочерними вершинами.

A pointer P_1 to the root of a tree is given, the tree contains at least two nodes. Output the amount of tree leaves that are the right child nodes.

```
package main

import (
    "fmt"
    "math/rand"
    "time"
    "./tree"
)

func main() {
    rand.Seed(time.Now().UnixNano())
    var tr tree.Tree
    tr.AutoMake(rand.Intn(15)+1)
    tr.Display()
    fmt.Printf("\n%d\n", tr.GetRightLeafCount())
}
```

Tree 9

Дан указатель P_1 на корень непустого дерева. Считается, что корень дерева находится на *нулевом уровне*, его дочерние вершины — на *первом уровне* и т. д. Вывести *глубину дерева*, т. е. значение его максимального уровня (например, глубина дерева, состоящего только из корня, равна 0).

A pointer P_1 to the root of a nonempty tree is given. The root node is said to be on the *zero level*, its child nodes — on the *first level*, and so on. Output the *depth* of the tree (that is, the maximal level of tree nodes). For example, the depth of a tree containing only a root node is equal to 0.

```
package main

import (
    "fmt"
    "math/rand"
    "time"
    "./tree"
)

func main() {
    rand.Seed(time.Now().UnixNano())
    var tr tree.Tree
    tr.AutoMake(rand.Intn(15)+1)
    tr.Display()
    fmt.Printf("\n%d\n", tr.GetLevel())
}
```

Tree 10

Дан указатель P_1 на корень непустого дерева. Для каждого из уровней данного дерева, начиная с нулевого, вывести количество вершин, находящихся на этом уровне. Считать, что глубина дерева не превосходит 10.

A pointer P_1 to the root of a nonempty tree is given. For each tree level (including the zero one) output the amount of its nodes. The tree depth is assumed to be not greater than 10.

```
package main

import (
    "fmt"
    "math/rand"
    "time"
    "./tree"
)

func main() {
    rand.Seed(time.Now().UnixNano())
    var tr tree.Tree
    tr.AutoMake(rand.Intn(15)+1)
    tr.Display()
    fmt.Println()
    var array []int = make([]int, tr.GetLevel()+1)
    tr.LevelToArray(array, 0, true)
    fmt.Println(array)
}
```

Tree 11

Дан указатель P_1 на корень непустого дерева. Для каждого из уровней данного дерева, начиная с нулевого, вывести

сумму значений вершин, находящихся на этом уровне. Считать, что глубина дерева не превосходит 10.

A pointer P_1 to the root of a nonempty tree is given. For each tree level (including the zero one) output the sum of values of its nodes. The tree depth is assumed to be not greater than 10.

```
package main

import (
    "fmt"
    "math/rand"
    "time"
    "./tree"
)

func main() {
    rand.Seed(time.Now().UnixNano())
    var tr tree.Tree
    tr.AutoMake(rand.Intn(15)+1)
    tr.Display()
    fmt.Println()
    var array []int = make([]int, tr.GetLevel()+1)
    tr.LevelToArray(array, 0, false)
    fmt.Println(array)
}
```

Tree 12

Дан указатель P_1 на корень непустого дерева. Вывести значения всех вершин дерева в *инфиксном порядке* (вначале выводится содержимое левого поддерева в инфиксном порядке, затем выводится значение корня, затем — содержимое правого поддерева в инфиксном порядке).

A pointer P_1 to the root of a nonempty tree is given. Using the recursive algorithm named *inorder tree walk* output the values of all tree nodes as follows: output the left subtree (using inorder tree walk), then output the root node, then output the right subtree (using inorder tree walk too).

```
package main

import (
    "fmt"
    "math/rand"
    "time"
    "./tree"
```

```
)

func main() {
    rand.Seed(time.Now().UnixNano())
    var tr tree.Tree
    tr.AutoMake(rand.Intn(15)+1)
    tr.Display()
    fmt.Println()
    tr.Infix()
}
```

Tree 13

Дан указатель P_1 на корень непустого дерева. Вывести значения всех вершин дерева в *префиксном порядке* (вначале выводится значение корня, затем содержимое левого поддерева в префиксном порядке, затем — содержимое правого поддерева в префиксном порядке).

A pointer P_1 to the root of a nonempty tree is given. Using the recursive algorithm named *preorder tree walk* output the values of all tree nodes as follows: output the root node, then output the left subtree (using preorder tree walk), then output the right subtree (using preorder tree walk too).

```
package main

import (
    "fmt"
    "math/rand"
    "time"
    "./tree"
)

func main() {
    rand.Seed(time.Now().UnixNano())
    var tr tree.Tree
    tr.AutoMake(rand.Intn(15)+1)
    tr.Display()
    fmt.Println()
    tr.Prefix()
}
```

Tree 14

Дан указатель P_1 на корень непустого дерева. Вывести значения всех вершин дерева в *постфиксном порядке* (вначале выводится содержимое левого поддерева в

постфиксном порядке, затем — содержимое правого поддерева в постфиксном порядке, затем — значение корня).

A pointer P_1 to the root of a nonempty tree is given. Using the recursive algorithm named *postorder tree walk* output the values of all tree nodes as follows: output the left subtree (using postorder tree walk), then output the right subtree (using postorder tree walk too), then output the root node.

```
package main

import (
    "fmt"
    "math/rand"
    "time"
    "./tree"
)

func main() {
    rand.Seed(time.Now().UnixNano())
    var tr tree.Tree
    tr.AutoMake(rand.Intn(15)+1)
    tr.Display()
    fmt.Println()
    tr.Postfix()
}
```

Tree 15

Дан указатель P_1 на корень непустого дерева и число N (> 0), не превосходящее количество вершин в исходном дереве. Нумеруя вершины в инфиксном порядке (см. задание Tree12, нумерация ведется от 1), вывести значения всех вершин с порядковыми номерами от 1 до N.

A pointer P_1 to the root of a nonempty tree and an integer N (> 0) are given. The value of N is not greater than the amount of tree nodes. Output the values of tree nodes whose order numbers are not greater than N (the tree nodes are numbered from 1 using inorder tree walk — see Tree12).

```
package main

import (
```

```
    "fmt"
    "time"
    "math/rand"
    "./tree"
)

func main() {
    rand.Seed(time.Now().UnixNano())
    var tr tree.Tree
    tr.AutoMake(rand.Intn(15)+1)
    tr.Display()
    var n, index int
    fmt.Print("N = ")
    fmt.Scan(&n)
    fmt.Println()
    tr.InfixToN(&index, n)
}
```

Tree 16

Дан указатель P_1 на корень непустого дерева и число N (> 0), не превосходящее количество вершин в исходном дереве. Нумеруя вершины в постфиксном порядке (см. задание Tree14, нумерация ведется от 1), вывести значения всех вершин с порядковыми номерами от N до максимального номера.

A pointer P_1 to the root of a nonempty tree and an integer N (> 0) are given. The value of N is not greater than the amount of tree nodes. Output the values of tree nodes whose order numbers are N or greater (the tree nodes are numbered from 1 using postorder tree walk — see Tree14).

```
package main

import (
    "fmt"
    "math/rand"
    "time"
    "./tree"
)

func main() {
    rand.Seed(time.Now().UnixNano())
    var tr tree.Tree
    tr.AutoMake(rand.Intn(15)+1)
    tr.Display()
    var n, index int
    fmt.Print("N = ")
    fmt.Scan(&n)
    fmt.Println()
```

```
    tr.PostfixFromN(&index, n)
}
```

Tree 17

Дан указатель P_1 на корень непустого дерева и два числа N_1, N_2 ($0 < N_1 < N_2$), которые не превосходят количество вершин в исходном дереве. Нумеруя вершины в префиксном порядке (см. задание Tree13, нумерация ведется от 1), вывести значения всех вершин с порядковыми номерами от N_1 до N_2.

A pointer P_1 to the root of a nonempty tree and two integers N_1, N_2 ($0 < N_1 < N_2$) are given. The value of N_2 is not greater than the amount of tree nodes. Output the values of tree nodes whose order numbers are in the range N_1 to N_2 (the tree nodes are numbered from 1 using preorder tree walk — see Tree13).

```
package main

import (
    "fmt"
    "math/rand"
    "time"
    "./tree"
)

func main() {
    rand.Seed(time.Now().UnixNano())
    var tr tree.Tree
    tr.AutoMake(rand.Intn(15)+1)
    tr.Display()
    var n1, n2, index int
    fmt.Print("N1 = ")
    fmt.Scan(&n1)
    fmt.Print("N2 = ")
    fmt.Scan(&n2)
    fmt.Println()
    tr.PrefixBetween(&index, n1, n2)
}
```

Tree 18

Дан указатель P_1 на корень непустого дерева и неотрицательное число L. Используя любой из описанных в заданиях Tree12–Tree14 способов обхода дерева, вывести

значения всех вершин уровня L, а также их количество N (если дерево не содержит вершин уровня L, то вывести 0).

A pointer P_1 to the root of a nonempty tree and an integer L (≥ 0) are given. Using tree walk of any type (see Tree12−Tree14) output values of all nodes of the level L. Also output the amount N of these nodes. If the given tree does not contain nodes of level L then output 0.

```
package main

import (
    "fmt"
    "math/rand"
    "time"
    "./tree"
)

func main() {
    rand.Seed(time.Now().UnixNano())
    var tr tree.Tree
    tr.AutoMake(rand.Intn(15)+1)
    tr.Display()
    var level int
    fmt.Print("L = ")
    fmt.Scan(&level)
    fmt.Println()
    n := tr.PrintLevel(0, level)
    fmt.Printf("\nN = %d\n", n)
}
```

Tree 19

Дан указатель P_1 на корень непустого дерева. Вывести максимальное из значений его вершин и количество вершин, имеющих это максимальное значение.

A pointer P_1 to the root of a nonempty tree is given. Output the maximal value of the tree nodes and the amount of nodes with this value.

```
package main

import (
    "fmt"
    "math/rand"
    "time"
    "./tree"
)
```

```
func main() {
    rand.Seed(time.Now().UnixNano())
    var tr tree.Tree
    tr.AutoMake(rand.Intn(15)+1)
    tr.Display()
    fmt.Println()
    maximal := tr.GetMaxData()
    count := tr.GetDataCount(maximal)
    fmt.Printf("maximal = %d\t\tcount = %d\n", maximal, count)
}
```

Tree 20

Дан указатель P_1 на корень непустого дерева. Вывести минимальное из значений всех его вершин и количество листьев, имеющих это минимальное значение (данное количество может быть равно 0).

A pointer P_1 to the root of a nonempty tree is given. Output the minimal value of the tree nodes and the amount of leaves with this value (the amount may be equal to 0).

```
package main

import (
    "fmt"
    "math/rand"
    "time"
    "./tree"
)

func main() {
    rand.Seed(time.Now().UnixNano())
    var tr tree.Tree
    tr.AutoMake(rand.Intn(15)+1)
    tr.Display()
    fmt.Println()
    minimal := tr.GetMinData()
    count := tr.GetLeafDataCount(minimal)
    fmt.Printf("minimal = %d\t\tcount = %d\n", minimal, count)
}
```