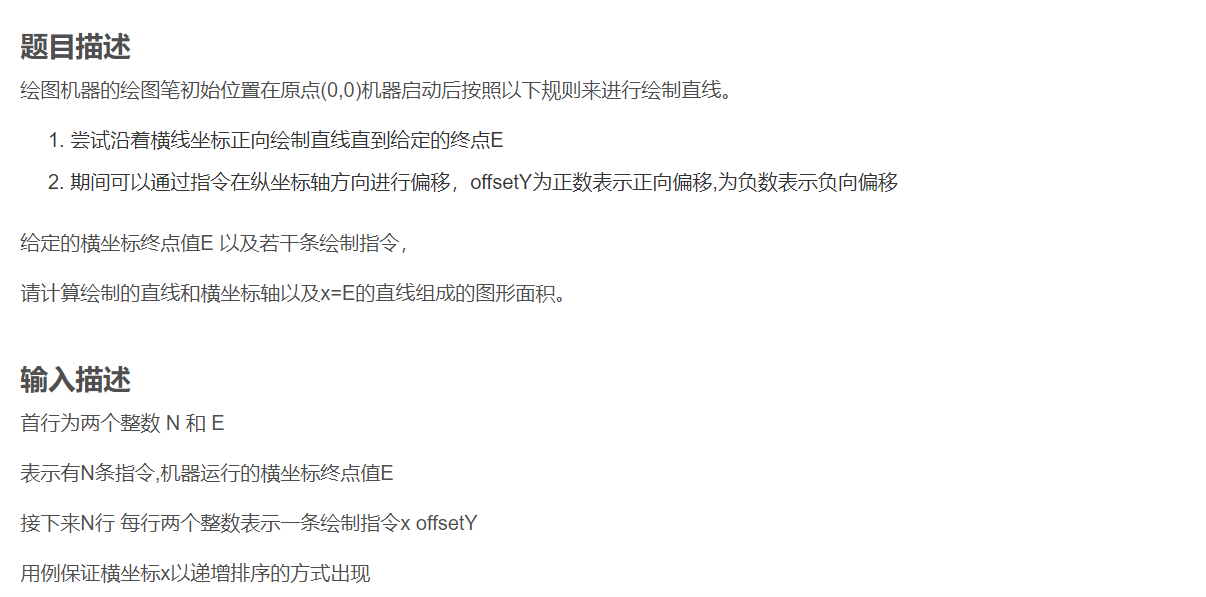
# **E卷-计算面积[100分]（ Java | Python3 | C++ | C语言 | JsNode | Go）**





4 10

1 1

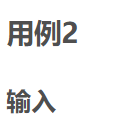
2 1

3 1

4 -2



12



2 4

0 1

2 -2



4



import java.util.Scanner;

public class Main{

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// 读取第一个输入行的N和E

int n = scanner.nextInt();

int e = scanner.nextInt();

// 初始化存储绘图指令的数组，并添加起点(0,0)

int[][] arr = new int[n + 2][2];

arr[0][0] = 0;

arr[0][1] = 0;

// 读取绘图指令

for (int i = 1; i <= n; i++) {

int x = scanner.nextInt();

int y = scanner.nextInt();

arr[i][0] = x;

arr[i][1] = y;

}

// 添加终点(E, 0)

arr[n + 1][0] = e;

arr[n + 1][1] = 0;

// 计算面积

int m = arr.length;

int y = 0; // 当前纵坐标偏移

long res = 0; // 结果面积

for (int i = 1; i < m; i++) {

int dx = arr[i][0];

int dy = arr[i][1];

// 计算面积

res += (long)Math.abs(y) \* (dx - arr[i - 1][0]);

// 更新y坐标

y += dy;

}

// 输出结果

System.out.println(res);

}

}



# 读取第一个输入行，包含两个整数N和E，分别表示有N条指令和机器绘图的横坐标终点值E

n, e = map(int, input().split())

# 初始化指令列表，初始点为原点(0,0)

arr = [(0, 0)]

# 读取N条指令，每条指令包含一个横坐标x和一个纵坐标偏移量offsetY

for \_ in range(n):

x, y = map(int, input().split())

arr.append((x, y))

# 将终点(E, 0)加入指令列表，表示在E点结束绘图，并回到横坐标轴上

arr.append((e, 0))

# 计算指令总数

m = len(arr)

# 初始化当前纵坐标y为0，表示在原点开始

y = 0

# 初始化面积结果变量res为0

res = 0

# 循环从第1个指令到最后一个指令

for i in range(1, m):

# 获取当前指令的横坐标dx和纵坐标偏移量dy

dx, dy = arr[i]

# 通过绝对值求当前图形的面积累加到总面积，多边形面积计算公式

res += abs(y) \* (dx - arr[i - 1][0])

# 更新当前纵坐标y

y += dy

# 输出计算得到的总面积

print(res)



#include <iostream>

#include <vector>

#include <cmath>

using namespace std;

int main() {

int n, e;

cin >> n >> e;

// 初始化存储绘图指令的数组，并添加起点(0,0)

vector<pair<int, int>> arr;

arr.push\_back({0, 0});

// 读取绘图指令

for (int i = 0; i < n; i++) {

int x, y;

cin >> x >> y;

arr.push\_back({x, y});

}

// 添加终点(E, 0)

arr.push\_back({e, 0});

// 计算面积

long long y = 0; // 当前纵坐标偏移

long long res = 0; // 结果面积

for (size\_t i = 1; i < arr.size(); i++) {

long long dx = arr[i].first;

long long dy = arr[i].second;

// 计算面积

res += abs(y) \* (dx - arr[i - 1].first);

// 更新y坐标

y += dy;

}

// 输出结果

cout << res << endl;

return 0;

}



#include <stdio.h>

#include <stdlib.h>

// 定义一个结构体来存储绘图指令

typedef struct {

int x;

int y;

} Point;

int main() {

int n, e;

scanf("%d %d", &n, &e);

// 初始化存储绘图指令的数组，并添加起点(0,0)

Point\* arr = (Point\*)malloc((n + 2) \* sizeof(Point));

arr[0].x = 0;

arr[0].y = 0;

// 读取绘图指令

for (int i = 1; i <= n; i++) {

int x, y;

scanf("%d %d", &x, &y);

arr[i].x = x;

arr[i].y = y;

}

// 添加终点(E, 0)

arr[n + 1].x = e;

arr[n + 1].y = 0;

// 计算面积

int y = 0; // 当前纵坐标偏移

long long res = 0; // 结果面积

for (int i = 1; i <= n + 1; i++) {

int dx = arr[i].x;

int dy = arr[i].y;

// 计算面积

res += abs(y) \* (dx - arr[i - 1].x);

// 更新y坐标

y += dy;

}

// 输出结果

printf("%d\n", res);

// 释放内存

free(arr);

return 0;

}



const readline = require("readline");

const rl = readline.createInterface({

input: process.stdin,

output: process.stdout,

});

// 获取输入数据

const input = [];

rl.on("line", (line) => {

input.push(line);

if (input.length === parseInt(input[0].split(" ")[0], 10) + 1) {

rl.close();

}

});

rl.on("close", () => {

const [n, e] = input[0].split(" ").map(Number);

// 初始化存储绘图指令的数组，并添加起点(0,0)

const arr = [[0, 0]];

// 读取绘图指令

for (let i = 1; i <= n; i++) {

const [x, y] = input[i].split(" ").map(Number);

arr.push([x, y]);

}

// 添加终点(E, 0)

arr.push([e, 0]);

// 计算面积

let y = 0; // 当前纵坐标偏移

let res = 0; // 结果面积

for (let i = 1; i < arr.length; i++) {

const [dx, dy] = arr[i];

// 计算面积

res += Math.abs(y) \* (dx - arr[i - 1][0]);

// 更新y坐标

y += dy;

}

// 输出结果

console.log(res);

});



package main

import (

"bufio"

"fmt"

"os"

"strconv"

"strings"

)

// Point 结构表示绘图机器在横坐标和纵坐标上的一个位置

type P struct {

x, y int

}

func main() {

// 使用bufio来读取标准输入

reader := bufio.NewReader(os.Stdin)

// 读取第一个输入行，包含两个整数N和E，表示有N条指令和机器绘图的横坐标终点值E

line, \_ := reader.ReadString('\n')

line = strings.TrimSpace(line)

values := strings.Split(line, " ")

N, \_ := strconv.Atoi(values[0])

E, \_ := strconv.Atoi(values[1])

// 初始化指令列表，起始点为原点(0, 0)

points := make([]P, 0)

points = append(points, P{0, 0})

// 读取N条指令，每条指令包含一个横坐标x和一个纵坐标偏移量offsetY

for i := 0; i < N; i++ {

line, \_ := reader.ReadString('\n')

line = strings.TrimSpace(line)

values := strings.Split(line, " ")

x, \_ := strconv.Atoi(values[0])

y, \_ := strconv.Atoi(values[1])

points = append(points, P{x, y})

}

// 将终点(E, 0)加入指令列表，表示在E点结束绘图，并回到横坐标轴上

points = append(points, P{E, 0})

// 计算指令总数

m := len(points)

// 初始化当前纵坐标y为0，表示在原点开始

y := 0

// 初始化面积结果变量res为0

res := 0

// 循环从第1个指令到最后一个指令

for i := 1; i < m; i++ {

// 获取当前指令的横坐标dx和纵坐标偏移量dy

dx := points[i].x - points[i-1].x

dy := points[i].y

// 通过绝对值求当前图形的面积累加到总面积，多边形面积计算公式

res += abs(y) \* dx

// 更新当前纵坐标y

y += dy

}

// 输出计算得到的总面积

fmt.Println(res)

}

// abs 函数计算整数的绝对值

func abs(a int) int {

if a < 0 {

return -a

}

return a

}