Софийски университет „Св. Климент Охридски“
Факултет по математика и информатика

ЗАДАЧИ ЗА ДОМАШНО 2

курс Увод в програмирането
за специалности Информатика, Компютърни науки и Софтуерно инженерство
зимен семестър 2016/2017 г.

ОБЩИ ЗАБЕЛЕЖКИ

Съблюдавайте общите изисквания публикувани в Moodle.

Четете внимателно условията на задачите и проверявайте решенията си с примерните тестови данни. Ако смятате, че има грешка в тях, моля пишете на преподавателите.

Всякакви въпроси по условията също отправяйте към тях.

Всичко, което използвате, но не е преподавано на лекции и упражнения ще трябва да обясните на защитите. Например ако използвате функции от стандартната библиотека, ще се очаква, че можете да обясните как работят те, как сте стигнали до решението да ги използвате във вашия код и т.н.

Показатели

След като вече сте преминали през изпитанията на първото домашно, всеки от вас получава свой собствен Character Sheet, в който може да записва информацията за приключението си в света на програмирането и да следи своя напредък. Можете да го свалите от някой от дадените по-долу линкове:

Като MS Word (docx) https://1drv.ms/w/s!ArVhb7o_XqnAqKs92bqyNETD028xjg

Като PDF https://1drv.ms/b/s!ArVhb7o_XqnAqKs_6-Z5X7pWZSMcNA

Ще видите, че в това домашно, в началото на всяка задача, в скоби, са изредени нейни характеристики във формат *“+N показател”*. Например:

Задача 1. (+1 Math, +1 Program Control, +1 Looping, +1 Function Evocation, +1 Numbers)

Смисълът на показателите можете да откриете в Character Sheet (например Looping се отнася до това доколко добре сте овладели работата с цикли). Числата пред тях показват с колко можете да вдигнете съответния показател, ако успеете да решите задачата. Например ако решите задача 1, можете да увеличите Math, Program Control, Looping, Function Evocation и Numbers с 1 точка. Освен това, по тях можете да съдите за това какви умения ще се изискват за решаването на съответната задача и преди да започнете да я решавате, ще можете да се подготвите по съответната тема.

Задача 1. (+1 Math, +1 Program Control, +1 Looping, +1 Function Evocation, +1 Numbers)
Предвид разочароващото представяне на националния отбор по футбол през последните години, селекционерът на трикольорите се явил пред БФС с "иновативна" идея. Треньорът твърдял, че е измислил начин да предотврати скатаването и ленивото движение на играчите по терена. Планът му бил прост и разумен — да се заложи на технологията. От БФС бързо приели, най-вече защото били чули, че така се прави "на запад" от доста време насам. Решено било в бутонките на футболистите да бъдат монтирани специални чипове, които да отчитат представянето на играча, който ги носи.

Тежката задача по реализацията на проекта възложили на студентите от ФзФ и ФМИ. За физиците не било никакъв проблем да направят интегрална схема на чипа, съдържаща GPS модул, памет и микроконтролер. След кратка дискусия, на която се обсъдило как да бъдат извличани данните от чиповете, единодушно се решило, че данните няма да бъдат предавани по мрежата (всички знаем колко несигурен е света на IoT, а пък не искаме някои да краде информацията за това колко "всеотдайни" са нашите футболисти, нали?), а ще бъдат четени от паметта на чиповете след всяка тренировка / всеки мач.

На Вас се пада заключителната част на проекта — четенето на данните и преобразуването им в полезна информация. За целта напишете програма, която прочита от стандартния си вход: цяло положително число **n (1 ≤ n < 2000000000)** (броят на координатите записани в чипа), последвано от **n** реда, всеки от които с по две дробни числа с до пет значещи цифри (координатите, на които се е намирал играчът в дадения момент).

На стандартния си изход програмата извежда едно дробно число. При извеждането форматирайте изхода с три цифри след десетичната запетая. Числото представя общото разстояние, изминато от играча докато чипът е записвал.

Изминатото разстояние пресмятаме като сума от изминатите разстояния между всеки две последователни двойки координати. Тъй като играчите не се движат само по права линия, при пресмятане на разстояние между две точки трябва да отчетем и приблизителното им отклонение. За целта разглеждаме функция **f(x)**, която ни дава приблизителната дължина на разстоянието, изминато от играчите между две точки в равнината като приема за аргумент дължината на отсечката между тези две точки.

f(x) се пресмята със следната формула: $f(x) = \frac{\frac{\pi x}{2} + x}{2}$, където за стойност на константата π приемаме 3.14. Например, дължината на изминатия път между точките (0,3) и (4,0) е f(5) = 6.425.

Областта, в която ще следим за движението на футболистите ще бъде ограничена от игралния терен, зададен чрез правоъгълника **R** с върхове (-5,-2) и (5,2). За всички прочетени точки, които не попадат в тази област, ще спазваме следните правила:

1. Ако предходната точка **A** е вътре в терена, а текущата **B** е извън него, то при пресмятане на разстоянието вместо точка **B** използваме точката **B'**, която е пресечната точка на отсечката **AB** с правоъгълника **R** (т.е. не отчитаме разстоянието изминато извън терена).
   Пример: Нека предходната точка **A** = (0,0), а текущата точка **B** = (0,3) (очевидно извън областта). Пресечната точка на отсечката **AB** и **R** е **B'** = (0,2). Изминатото разстояние между **A** и **B'** е f(2) = 2.570.

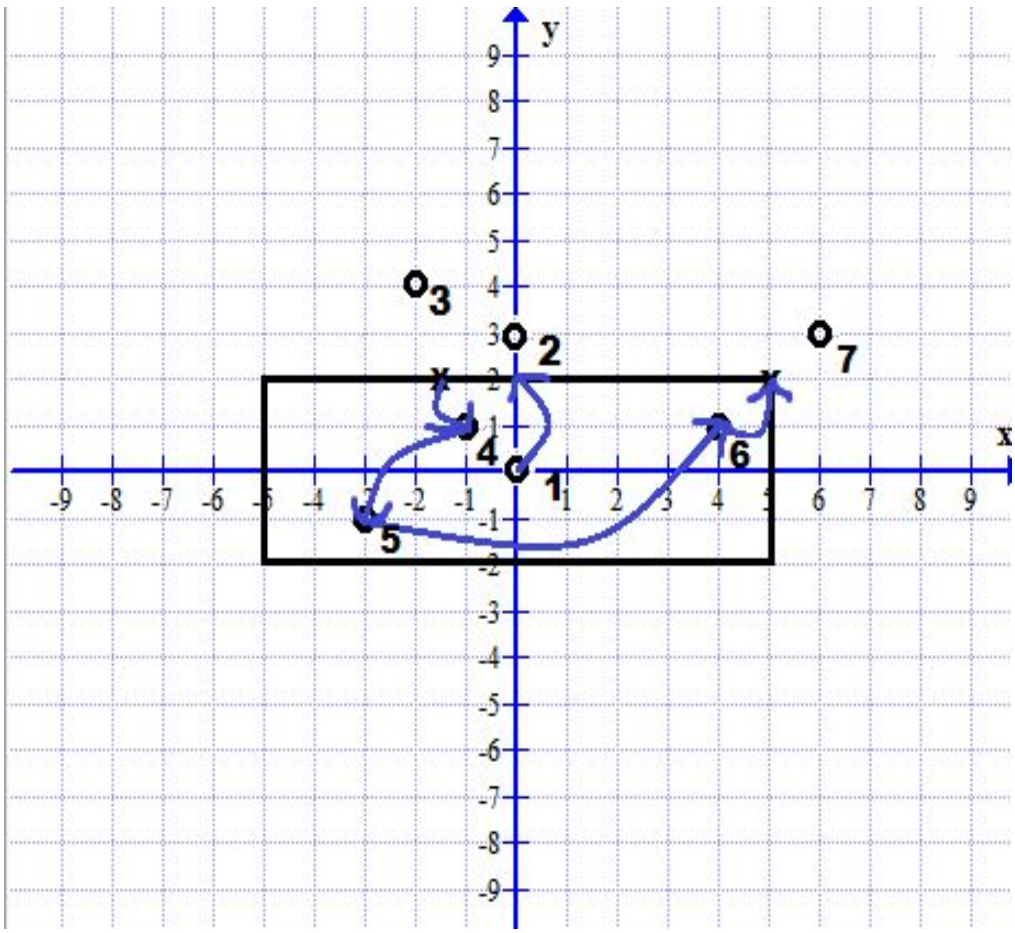
2. Ако и предишната, и текущата точка са извън терена, то разстоянието между тях не се прибавя към общото изминато разстояние. Тук приемаме, че дори отсечката между тях да пресича два пъти границите на игрището, то прекараното време на терена е незначително и не бива да калкулираме изминатото разстояние.
3. Ако предходната точка **A** е извън терена, а текущата **B** е в него, вместо точка **A** използваме точката **A'**, която е пресечната точка на отсечката **AB** с правоъгълника **R**.
   Пример: Нека **A** = (7,4) е извън терена, а **B** = (3,0) е вътре в терена. Пресечната точка на **AB** и **R** е **A'** = (5,2). Пресмятаме пътя между (3,0) и (5,2) по формулата f(√8) = 3.635.
4. Точки лежащи на контура на игрището приемаме за вътрешни. Използвайте за точност на сравненията само две цифри след десетичната точка.

Забележка: Забележете, че в дадените по-горе примери стойностите, върнати от f са закръглени до третата цифра след десетичната точка — тоест те са приблизителни. От вас се очаква да закръглите само крайния резултат, но НЕ и резултатите от междинните изчисления, понеже това би довело до допълнителна загуба на точност.

Пример:

Вход	Изход
7 0 0 0 3 -2 4 -1 1 -3 -1 4 1 6 3	18.731

Приблизителна схема на движението на играча:



Задача 2. (+1 Looping, +1 Program Control, +1 Function Evocation)
Простите числа **p** и **q** наричаме числа близнаци, ако **q = p + 2**. Например **5** и **7** са числа близнаци. Да се на пише програма, която прочита от стандартния вход цяло положително число **n** (n < 100000) и извежда на стандартния изход първите **n** двойки числа близнаци — всяка двойка на отделен ред, самите числа разделени с интервал. Припомняме, че 1 не е просто число.

Пример:

Вход	**Изход**
3	3 5 5 7 11 13

Задача 3. (+1 Looping, +1 Function Evocation, +1 Resourcefulness)
Да се напише програма, която прочита от стандартния вход две естествени числа **a** и **b** (a и b са по-малки от 4000000000) и извежда на екрана броя на срещанията на **a** в десетичния запис на **b**. Например, **a** = 84 се съдържа 3 пъти в **b** = 6845848456, **a** = 70 се съдържа 1 път в **b** = 8574704 и **a** = 55 се съдържа 3 пъти в **b** = 5555. Обърнете внимание, че срещанията **могат** да се припокриват, както е показано в последния пример.

Примери:

Вход	**Изход**
4 98454	2
123 949512	0
43 43056543	2
77 877677747	3

Задача 4. (+1 Looping)
Да се напише програма, която прочита две цели числа, побиращи се в тип `int`, и проверява дали множествата от цифрите им съвпадат. Програмата да извежда “`Yes`” при съвпадение и “No” в противен случай. Числата **могат** да бъдат и отрицателни. В такъв случаи минусът се “игнорира”, т.е едно отрицателно число има същото множество от цифри като абсолютната си стойност.

Примери:

Вход	**Изход**
123 312	Yes
12 1523	No
-43 340	No

-675678 8765	Yes

Задача 5. (+1 Looping, +1 Function Evocation, +1 Math, +1 Numbers)

Министерството на вселенските технологии в Замунда спечелило международен проект за финансиране на изследвания в областта на подводното бананово дело. За съжаление в Замунда нямало подводници, но всички били чували, че най-добрите инженери идват от България. Затова те решили да прехвърлят на нас задачата по изграждането на подводница и нейното управление, а те запазили за себе си най-съществената част от проекта — яденето на банани и хвърлянето на развалените от тях в океана.

Изграждането на подводницата вървяало правилно и логически, докато инженерите не открили един малък пропуск. Подводницата няма колела, а оттам и няма за какво да се закачи километражът. Същевременно по международния проект трябвало да се отчете колко метра са навъртяни по време на изследванията, защото срещу това се давало финансирането. Създавал се и проблем за екипажа, защото нямало как да се определи какво разстояние е изминато в дадена посока, освен ако някой не надничал през илюминаторите на подводницата и не преценявал “на око”.

За щастие, инженерите били опитни и веднага се сетили, че за тази цел може да се използва акселерометър. Накратко, това е устройство, което измерва ускорението на движещи се обекти. То може да се сложи на борда на подводницата и когато тя се движи, да връща информация за текущата посока на движението и ускорението.

Тъй като инженерите са притиснати от кратките срокове за работата по замундския проект, те са възложили на вас да напишете софтуер, който по измерванията на акселерометъра ще може да пресметне колко метра е изминала подводницата. Между другото, никой от тях не разбира защо от Замунда са поискали той да може да се сваля и са попитали дали може да се завърже на някоя маймуна, която подскачайки по палмите да навърта километраж по време на проекта.

Програмата ви ще получи като вход положително цяло число N — брой измервания, които е направил датчикът. След това число ще следва положително дробно число D — това е интервалът във времето, с който датчикът събира показания. Например, ако D = 0,1, това означава, че датчикът прави едно измерване на всеки 0,1 секунди. Ако D = 10, той прави това на всеки 10 секунди и т.н. След това следват N броя дробни числа. Това са измерванията на датчика.

За целите на задачата считаме, че датчикът се включва веднага след тръгване и се изключва непосредствено преди спиране. Т.е. скоростта на движение преди първото измерване и веднага след последното е равна на нула.

Вашата програма трябва да изведе на екрана неотрицателно число с плаваща запетая — дължината на изминатия път в метри. Възможно е той да е нула, например когато датчикът получи като вход N = 0 или когато N ≠ 0, но всички измервания на датчика са нули. Числото да се изведе с точно пет цифри след десетичната запетая. След него **не трябва** да се слагат никакви означения (например символът за метър “m”).

За целите на задачата ще считаме, че посоката на движение няма значение. Тоест независимо дали подводницата върви нагоре, надолу, напред и т.н., винаги ще получаваме само стойност за ускорението. Тъй като се интересуваме само от абсолютната стойност на скоростта, стойностите с които работим ще бъдат неотрицателни.

Обърнете внимание, че при рязко спиране на подводницата може да получите много голяма по модул отрицателна стойност за ускорението. Ако сляпо приложим формулата за пресмятане на скоростта, може да получим отрицателно число. Това, обаче, съвсем не означава, че подводницата започна да се движи в обратна посока[1]! Затова трябва да се грижите винаги да "отрязвате" получената стойност за скоростта до 0, т.е. да напишете решението си така, че ако се получи отрицателна стойност за скоростта (например -10), вместо нея да се използва стойност нула.

Упътване: Пътят, изминат от подводницата, може условно да се разбие на N части, като всяка отговаря на времето между две последователни измервания на акселерометъра. Когато измерванията са достатъчно често (т.е. D е достатъчно малко), можем да считаме, че между две измервания ускорението на подводницата не се променя, т.е. движението е "почти" равноускорително или равнозакъснително[2]. Дължините на тези части ще обозначим със $S_1, S_2, ..., S_N$. За да се пресметне всяка от тях може да се използват формулите за равноускорително и/или равнозакъснително движение:

$$S = v_0 t + \frac{1}{2} a t^2$$
$$v = v_0 + at$$

По-конкретно:

$$S_n = v_{n-1} D + \frac{1}{2} a_n D^2$$
$$v_0 = 0$$
$$v_n = v_{n-1} + a_n D$$

След това целият изминат път ще се смята по формулата:

$$S = \sum_{i=1}^{N} S_i$$

Примери:

Вход	**Изход**
`5 0.1 1 2 -4 3 1`	`0.085`
`0 0.1`	`0`
`2 0.1 0 0`	`0`

[1] Ако беше така, на автомобилите нямаше да слагат задна скорост, а само спирачка.

[2] Като аналогия на това допускане можете да се замислите как можете да нарисувате крива линия като последователност от малки отсечки, или да сметнете лицето под тази крива линия като сума на лицата от малки трапеци — напомня ли ви на нещо?

Задача 6. (+1 Looping, +1 Math, +1 Function Evocation)

Биномен коефициент на естествените числа **k** и **n** е броят на всички възможни **k**-елементни подмножества на дадено **n**-елементно множество и може да бъде пресметнат чрез формулата:

$$\binom{n}{k} = \frac{n!}{k!\,(n-k)!} \quad \text{for } 0 \le k \le n,$$

Теоремата за биномните коефициенти описва алгебричното разлагане на двучлен, повдигнат на степен, а именно:

$$(x+y)^n = \binom{n}{0}x^n y^0 + \binom{n}{1}x^{n-1}y^1 + \binom{n}{2}x^{n-2}y^2 + \cdots + \binom{n}{n-1}x^1 y^{n-1} + \binom{n}{n}x^0 y^n,$$

Или записано накратко:

$$(x+y)^n = \sum_{k=0}^{n}\binom{n}{k}x^{n-k}y^k = \sum_{k=0}^{n}\binom{n}{k}x^k y^{n-k}.$$

Да се напише програма, която прочита от стандартния вход цяло положително число **n** и извежда на екрана, коефициентите пред различните степени в разлагането на двучлена $(x+y)^n$. Коефициентите да са на един ред, разделени с интервал. (След последния няма интервал.) При реализацията на програмата се опитайте да поддържате безпроблемна работа с възможно най-големи стойности на **n**, при ограничението, че всички биномни коефициенти, които програмата ще извежда, ще се побират в тип unsigned.

Примери:

Вход	Изход
2	1 2 1
3	1 3 3 1
7	1 7 21 35 35 21 7 1
4	1 4 6 4 1

Задача 7. (+1 Looping, +1 Math, +1 Function Evocation)

Да се напише програма, която прочита от стандартния вход три цели числа **n**, **m** и **y**, (0 < **n** < 1000001, 1 < **m** < 100001, 0 < **y** < **m**) и извежда на стандартния изход всички естествени числа **x** в интервала [1, **m**-1], за които е в сила $\mathbf{x}^{\mathbf{n}} \bmod \mathbf{m} = \mathbf{y}$. Ако има повече от едно такова **x,** да се изведат всички такива числа на един ред, разделени с по един интервал в нарастващ ред. Обърнете внимание, че след последното число **не трябва** да има интервал. Ако такива числа няма, програмата да извежда ред съдържащ само числото -1.

Примери:

Вход	Изход
2 5 1	1 4
2 4 2	-1

Задача 8. (+1 Looping, +1 Function Evocation, +1 Numbers)
Направете суматор за числа, записани в различна бройна система от десетична. Суматорът да приема от стандартния вход три цели числа — **n**, **m** и **k**, където **k** е основата на бройната система, в която са записани **n** и **m**. На стандартния изход да се извежда сумата на двете числа, отново записана в **k**-ична бройна система. Ако поне едно от числата **m** или **n** не е валидно число в посочената система, да се изведе низът "`Bad input data!`".
Ограничения:

- 2 ≤ **k** ≤ 10
- 0 ≤ **m**, **n** ≤ 500000 (имат се предвид стойностите на числата m и n)

Примери:

Вход	**Изход**
`250 340 10`	`590`
`110 11 2`	`1001`
`54 453 8`	`527`
`421 340 5`	`1311`
`645 342 6`	`Bad input data!`

Задача 9. (+1 Looping, +1 Function Evocation)
Да се напише програма, която прочита от стандартния вход две числа от тип unsigned и проверява дали след задраскване на някоя цифра от първото, може да се получи второто. Ако е възможно, на стандартния изход да се изведе "`Yes`", в противен случай "`No`".

Примери:

Вход	**Изход**
`123 12`	`Yes`
`75648 7648`	`Yes`
`423 41`	`No`
`12 1`	`Yes`
`4 13`	`No`

Задача 10. (+1 Looping, +1 Program Control, +1 Math)
P-ична адитивна степен на цяло неотрицателно число ***n*** дефинираме, като най-голямата степен ***v***, такава че p^v е делител на числото и ***плюс безкрайност***, при ***n = 0*** и я бележим с $V_p(n)$, където ***p*** е просто число. (https://en.wikipedia.org/wiki/P-adic_order)

$$\nu_p(n) = \begin{cases} \max\{v \in \mathbb{N} : p^v \mid n\} & \text{if } n \neq 0 \\ \infty & \text{if } n = 0 \end{cases}$$

Да се напише програма, която по въведен брой заявки ***nq*** от вида:
p1 n p2
и самите заявки, за всяка заявка да връща $\boldsymbol{max(V_{p1}(n), V_{p2}(n))}$.

Ограничения:
0 < ***nq*** < 1 000 000 000; 1 < ***p1, p2*** ≤ 100 000 007; 0 ≤ ***n*** ≤ $100\,000\,007^2$

Вход:
На първият ред от стандартния вход е въведен броят заявки ***nq***. На всеки от следващите ***nq*** на брой реда стои по една заявка от гореописания вид (три числа, разделени с интервали).

Изход:
За всяка заявка от входа на един ред от изхода да се изведе търсеният отговор на заявката. Ако отговорът е плюс безкрайност, да се изведе "**infinity**" без кавички.

Примери:

Вход	**Изход**
3 2 209952 3 5 18 2 13 0 17	8 1 infinity
2 2 6 3 13 13 2	1 1
1 5 50 11	2