Informácie a pravidlá

Pre koho je súťaž určená?

Do **kategórie B** sa smú zapojiť len tí žiaci základných a stredných škôl, ktorí ešte ani v tomto, ani v nasledujúcom školskom roku nebudú končiť strednú školu.
Do **kategórie A** sa môžu zapojiť všetci žiaci (základných aj) stredných škôl.

Odovzdávanie riešení domáceho kola

Riešitelia domáceho kola odovzdávajú riešenia sami, v elektronickej podobe, a to priamo na stránke olympiády: http://oi.sk/. Odovzdávanie riešení bude spustené niekedy v septembri.

Riešenia kategórie A je potrebné odovzdať najneskôr **15. novembra 2017**.
Riešenia kategórie B je potrebné odovzdať najneskôr **1. decembra 2017**.

Priebeh súťaže

Za každú úlohu domáceho kola sa dá získať od 0 do 10 bodov. Na základe bodov domáceho kola stanoví Slovenská komisia OI (SK OI) pre každú kategóriu bodovú hranicu potrebnú na postup do **krajského kola**. Očakávame, že táto hranica bude približne rovná **tretine maximálneho počtu bodov**.

V krajskom kole riešitelia riešia štyri teoretické úlohy, ktoré môžu tematicky nadväzovať na úlohy domáceho kola. V kategórii B súťaž týmto kolom končí.

V kategórii A je približne najlepších 30 riešiteľov krajského kola (podľa počtu bodov, bez ohľadu na kraj, v ktorom súťažili) pozvaných do **celoštátneho kola**. V celoštátnom kole účastníci prvý deň riešia teoretické a druhý deň praktické úlohy. Najlepší riešitelia sú vyhlásení za víťazov. Približne desať najlepších riešiteľov následne SK OI pozve na týždňové výberové sústredenie. Podľa jeho výsledkov SK OI vyberie družstvá pre Medzinárodnú olympiádu v informatike (IOI) a Stredoeurópsku olympiádu v informatike (CEOI).

Ako majú vyzerať riešenia úloh?

V praktických úlohách je vašou úlohou vytvoriť program, ktorý bude riešiť zadanú úlohu. Program musí byť v prvom rade korektný a funkčný, v druhom rade sa snažte aby bol čo najefektívnejší.

V kategórii B môžete použiť ľubovoľný programovací jazyk.

V kategórii A musíte riešenia praktických úloh písať v jednom z podporovaných jazykov (napr. C++, Pascal alebo Java). Odovzdaný program bude automaticky otestovaný na viacerých vopred pripravených testovacích vstupoch. Podľa toho, na koľko z nich dá správnu odpoveď, vám budú pridelené body. Výsledok testovania sa dozviete krátko po odovzdaní. Ak váš program nezíska plný počet bodov, budete ho môcť vylepšiť a odovzdať znova, až do uplynutia termínu na odovzdávanie.

Presný popis, ako majú vyzerať riešenia praktických úloh (napr. realizáciu vstupu a výstupu), nájdete na webstránke, kde ich budete odovzdávať.

Ak nie je v zadaní povedané ináč, riešenia teoretických úloh musia v prvom rade obsahovať **podrobný slovný popis použitého algoritmu, zdôvodnenie jeho správnosti** a diskusiu o efektivite zvoleného riešenia (t. j. posúdenie časových a pamäťových nárokov programu). Na záver riešenia uveďte program. Ak používate v programe netriviálne algoritmy alebo dátové štruktúry (napr. rôzne súčasti STL v C++), súčasťou popisu algoritmu musí byť dostatočný popis ich implementácie.

Usporiadateľ súťaže

Olympiádu v informatike (OI) vyhlasuje *Ministerstvo školstva SR* v spolupráci so *Slovenskou informatickou spoločnosťou* (odborným garantom súťaže) a *Slovenskou komisiou Olympiády v informatike*. Súťaž organizuje *Slovenská komisia OI* a v jednotlivých krajoch ju riadia *krajské komisie OI*. Na jednotlivých školách ju zaisťujú učitelia informatiky. Celoštátne kolo OI, tlač materiálov a ich distribúciu po organizačnej stránke zabezpečuje IUVENTA v tesnej súčinnosti so Slovenskou komisiou OI.

B-I-1 Babičkina špajza

Počas prázdnin u babky sa Peti podarilo objaviť tajnú špajzu – zdroj skvelých koláčov, zaváranín, kompótov a hlavne úžasného babičkinho jahodového džemu. Na dlhočiznej poličke stoja pred ňou v rade poukladané rôzne veľké poháre džemu a Peťa by si rada nejaké z nich zobrala.
Naraz však neunesie viac ako tri poháre. Naviac, ak by si zobrala poháre, ktoré sú príliš blízko seba, babka by si to veľmi rýchlo všimla. Chcela by preto, aby medzi každými dvoma pohármi, ktoré zoberie, ostalo aspoň k ďalších pohárov.
Peťa teraz rozmýšľa, ktoré tri poháre si má vybrať, aby odišla s čo najviac džemom. Pomôžte jej a ona sa oň s vami určite podelí. A ak nie, aspoň my vám dáme nejaké body do súťaže.

Súťažná úloha

Petina babka má v špajzi n pohárov džemu postavených v rade jeden za druhým. Na vstupe dostanete n celých čísel popisujúcich tento rad: i-te z týchto čísel určuje objem džemu v i-tom pohári v rade. Vašou úlohou je vybrať tri poháre, ktoré majú dokopy čo najväčší objem džemu a pre ktoré platí, že každé dva z týchto pohárov majú medzi sebou aspoň k iných, nevybratých pohárov. Môžete predpokladať, že vždy bude možné vybrať aspoň jednu takúto trojicu.

Formát vstupu a výstupu

Dostanete od nás 5 vstupných súborov, označených `1.txt` až `5.txt`. Každý z nich má nasledujúci formát:
Na prvom riadku sú dve medzerou oddelené čísla n a k. Číslo n udáva počet pohárov a číslo k hovorí, aspoň koľko pohárov treba vynechať medzi vybranými.
Na druhom riadku je n medzerou oddelených čísel $v_1, v_2, \ldots, v_n$. Hodnota v_i je objem džemu v i-tom pohári zľava.
Do výstupného súboru vypíšte jeden riadok a v ňom jedno číslo – najväčší objem džemu, ktorý vie Peťa zo špajze odniesť, ak splní všetky podmienky popísané v zadaní.

Veľkosti vstupov

V jednotlivých vstupoch má premenná n postupne hodnoty 100, 2 000, 50 000, 700 000 a 2 000 000. Pre hodnotu k platí, že $2 \cdot k + 3 \leq n$. Naviac, pre všetky hodnoty v_i platí, že sú väčšie ako 0 a menšie alebo rovné ako 300 000 000.

Odovzdávanie riešení

Toto je praktická úloha. Napíšte **v ľubovoľnom programovacom jazyku** program, ktorý ju rieši.
Zo stránky `http://oi.sk/` stiahnite ZIP archív obsahujúci 5 testovacích vstupov, nazvaných `1.txt` až `5.txt`.
Vyrobte k čo najviac vstupom správne výstupy a uložte ich do súborov `sol1.txt` až `sol5.txt`.
Odovzdajte ZIP archív obsahujúci **zdrojový kód vášho programu** a tieto výstupné súbory.
Za každý správny výstupný súbor získate 2 body.

Príklad

vstup

```
8 2
4 2 3 8 1 5 4 2
```

výstup

```
16
```

Optimálne je, aby si Peťa zobrala poháre na pozíciách 1, 4 a 7. Celkový objem odneseného džemu bude $4+8+4 = 16$. *Všimnite si, že ak zoberie pohár z pozíciie 4, nemôže zobrať pohár na pozícii 6, lebo ich oddeľuje iba jeden iný pohár, ale k je rovné 2.*

vstup

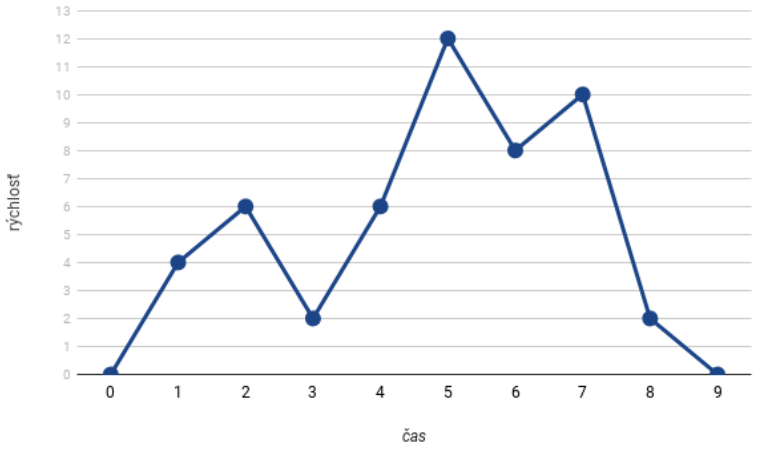
```
10 1
4 1 7 8 3 6 10 10 7 3
```

výstup

```
25
```

B-I-2 Bicyklový výlet

Janko nedávno oslavoval narodeniny. Od rodičov dostal ako darček multifunkčný monitorovací systém na bicykel. Znie to komplikovane, ale je to iba malá elektronická škatuľka, ktorú si pripne na bicykel a ktorá mu meria rýchlosť, prevýšenie, spálené kalórie a ešte veľa iných údajov.
Samozrejme, Janko chcel novú hračku hneď vyskúšať a tak išiel na celodenný bicyklový výlet. Keď sa vrátil, zapojil škatuľku do počítača a začal si prezerať všetky namerané údaje. Najviac ho zaujal graf rýchlosti. Na x-ovej osi tohto grafu bol zaznačený čas a na y-ovej rýchlosť. Graf bol tvorený lomenou čiarou, ktorá znázorňovala, akú mal Janko rýchlosť v danom momente cesty. Vykreslený graf mohol vyzerať napríklad takto:



Janka by zaujímalo, akou rýchlosťou išiel najčastejšie. Monitorovací systém mu však túto hodnotu nevie povedať. Vedeli by ste mu pomôcť a vyčítať ju z grafu jeho rýchlosti?

Súťažná úloha

Jankov výlet trval n minút. Vždy po celej minúte sa prístroj na jeho bicykli zapol a zaznamenal aktuálnu rýchlosť. Na začiatku aj na konci výletu (teda v čase 0 minút aj v čase n minút) bola Jankova rýchlosť rovná nule. Je zaručené, že medzi každými dvoma meraniami Janko buď rovnomerne zrýchľoval alebo rovnomerne spomaľoval. Inými slovami, môžete predpokladať, že graf Jankovej rýchlosti je lomená čiara s vrcholmi v nameraných bodoch.
Aby ste nemuseli v programe ošetrovať rôzne špeciálne prípady, je zaručené, že všetky namerané rýchlosti sú **párne celé čísla** a že žiadne dve po sebe nasledujúce merania neboli rovnaké.
Vašou úlohou je nájsť takú **nepárnu** rýchlosť, ktorou išiel Janko počas svojho výletu najčastejšie. Ak je takýchto nepárnych rýchlostí viac, môžete nájsť ľubovoľnú z nich.

Príklad: Na obrázku vyššie by správna odpoveď mohla byť 3, 5 alebo 9. Všetky tri rýchlosti totiž Janko dosiahol počas svojho výletu štyrikrát. Rýchlosti 1, 7 a 11 dosiahol iba dvakrát.

Formát vstupu a výstupu

Dostanete od nás 5 vstupných súborov, označených `1.txt` až `5.txt`. Každý z nich má nasledujúci formát:
Na prvom riadku je číslo n – počet minút Jankovho výletu.
Druhý riadok obsahuje $n+1$ medzerou oddelených čísel: $v_0, v_1, \ldots, v_n$. Tieto čísla sú merania Jankovej rýchlosti: v_i je rýchlosť, ktorú mal po i minútach výletu. Prvá aj posledná z týchto rýchlostí sú nuly. Všetky rýchlosti sú párne a žiadne dve po sebe idúce rýchlosti nie sú rovnaké.
Na výstup vypíšte jeden riadok a v ňom jediné číslo – nepárnu rýchlosť, ktorou sa Janko počas výletu pohyboval najčastejšie. Ak je takýchto rýchlostí viac, vypíšte ľubovoľnú z nich.

Veľkosti vstupov

V jednotlivých vstupoch má hodnota n postupne hodnoty 1 000, 999, 200 000, 197 936 a 199 765.
Pre hodnoty v_i naviac platia nasledovné obmedzenia: vo vstupoch `1.txt` a `2.txt` je $0 \leq v_i \leq 3\,000$, vo vstupoch `3.txt` a `4.txt` je $0 \leq v_i \leq 10^6$ a vo vstupe `5.txt` je $0 \leq v_i \leq 2 \cdot 10^9$.

Odovzdávanie riešení

Toto je praktická úloha. Napíšte **v ľubovoľnom programovacom jazyku** program, ktorý ju rieši.
Zo stránky `http://oi.sk/` stiahnite ZIP archív obsahujúci 5 testovacích vstupov, nazvaných `1.txt` až `5.txt`.
Vyrobte k čo najviac vstupom správne výstupy a uložte ich do súborov `sol1.txt` až `sol5.txt`.
Odovzdajte ZIP archív obsahujúci **zdrojový kód vášho programu** a tieto výstupné súbory.
Za každý správny výstupný súbor získate 2 body.

Príklad

vstup

```
9
0 4 6 2 6 12 8 10 2 0
```

výstup

```
5
```

Vstup zodpovedá obrázku zo zadania. Rovnako správna by bola aj odpoveď 3 alebo 9.

B-I-3 Dvanásťminútovka

Učiteľ telesnej Peter vyšiel so svojimi žiakmi von na bežecký okruh, aby si zabehli každoročný dvanásťminútový beh. Žiakov postavil na štart, zapískal na píšťalke a kým oni utekajú, Peter rozmýšľa, kedy tú dvanásťminútovku vlastne ukončiť.
Možno vám to príde ako podozrivo ľahká otázka, nie je tomu však tak. Peter je totiž lenivý a chcel by beh ukončiť v momente, keď budú všetci žiaci opäť na štarte. Vďaka tomu každý žiak ubehne niekoľko celých kolečiek a jemu sa to bude ľahšie počítať, keď nebude musieť merať, do akej časti kruhu každý žiak zabehol. A keďže žiaci pri sebe nemajú hodinky, to či bežali osem alebo dvadsať minút aj tak nerozlíšia.
Peter svojich žiakov veľmi dobre pozná, a preto o každom z nich vie, koľko minút mu trvá obehnúť jeden celý okruh. Navyše vie, že každý žiak si svoje tempo udrží počas celej „dvanásť“minútovky. Teraz by ho zaujímal najkratší čas od začiatku, po ktorom sa všetci žiaci stretnú opäť na štarte. A kvôli byrokratickým políčkam v triednej knihe tiež potrebuje vedieť, koľko kolečiek pri tom dokopy zabehli.
Pomôžte mu, inak vás nechá drepovať.

Súťažná úloha

Peter má n žiakov. Všetci žiaci vyštartovali v čase 0. Pre každého z nich Peter pozná čas t_i, za ktorý tento žiak zabehne jeden okruh. Všetky t_i sú kladné celé čísla.

Podúloha A (7 bodov).
Napíšte program, ktorý zo štandardného vstupu načíta číslo n a následne čísla $t_1, \ldots, t_n$. Z týchto čísel následne vypočíta a na štandardný výstup vypíše dve čísla t a k s nasledovným významom: t je najmenšie kladné číslo také, že v čase t budú všetci žiaci naraz prebiehať štartom, k je počet kolečiek, ktoré pri tom všetci žiaci dokopy prebehli.

Podúloha B (3 body).
Zadanie podúlohy B je rovnaké ako zadanie podúlohy A, avšak s jedným obmedzením navyše: V tejto podúlohe však musí vaše riešenie použiť iba **konštantne veľa pamäte**.

To znamená, že nemôžete používať žiadne polia (ani len na prečítanie a uloženie si celého vstupu) a ani rekurziu. Môžete použiť iba konštantný počet celočíselných premenných. Inými slovami, množstvo pamäte použité vaším programom nesmie závisieť od čísla n.
Pri písaní programu môžete predpokladať, že čísla rádovo také veľké ako správne výstupné hodnoty t a k sa vám zmestia do celočíselných premenných.

Funkcia `gcd()`

Vo vašom riešení môžete používať (bez toho aby ste ju implementovali) funkciu `gcd()`, ktorá vypočíta najväčšieho spoločného deliteľa[1] **dvoch** zadaných čísel. Napr. `gcd(4,2)=2`, `gcd(7,5)=1`, `gcd(12, 42) = 6`, atď.

Predpokladajte, že táto funkcia je implementovaná pomocou Euklidovho algoritmu[2]. Volanie funkcie `gcd(a,b)` používa konštantne veľa pamäte a logaritmicky veľa času v závislosti od väčšieho z čísel a a b. To znamená, že ak v programe x-krát nájdete `gcd` dvoch čísel, ktorých veľkosť je nanajvýš y, bude vás to dokopy stáť $O(x \log y)$ krokov výpočtu.

Hodnotenie

V každej podúlohe počet získaných bodov závisí od časovej zložitosti vášho riešenia. Aj pomalšie korektné riešenia teda dostanú nejaké body. Funkciu `gcd` nie je nutné využiť. Ak viete úlohu riešiť iným spôsobom bez jej použitia, aj takéto riešenie môžete odovzdať.

Odovzdávanie riešení

Toto je teoretická úloha. Spísané riešenie vo formáte PDF odovzdajte pomocou webového rozhrania.

Príklad

vstup	výstup
2 4 2	4 3

Máme dvoch žiakov. Prvý obehne kolečko za čas $t_1 = 4$, druhý za čas $t_2 = 2$. Prvýkrát sa zjavne obaja žiaci na štarte stretnú v čase $t = 4$. Dovtedy prvý žiak ubehol jedno kolečko a druhý dve, čo je dokopy $k = 1 + 2 = 3$.

vstup	výstup
5 1 3 2 4 2	12 31

B-I-4 Reálne čísla

V tejto súťažnej úlohe sa zoznámime s tým, ako si počítače poradia s reálnymi číslami. Presnejšie, budeme sa zaoberať najbežnejším formátom používaným na uloženie reálnych čísel v počítači: formát *s plávajúcou bodkou* (floating-point representation). Tieto divne znejúce slová znamenajú to, že si počítač samostatne pamätá cifry čísla a samostatne miesto, kde je hranica medzi celou a necelou časťou.[3]
Celé to teda bude fungovať podobne, ako keď číslo 1 234 567 zapíšeme v tvare 1.234567×10^6. Samozrejme, počítače nepoužívajú desiatkovú sústavu ale dvojkovú – teda jediné dve cifry sú 0 a 1. V nasledujúcom texte preto budeme občas aj my používať dvojkovú sústavu.[4]

[1] Názov gcd pochádza z anglického „greatest common divisor“.
[2] O Euklidovom algoritme si môžete prečítať napríklad na Wikipédii https://sk.wikipedia.org/wiki/Euklidov_algoritmus
[3] V slovenčine v desiatkovej sústave toto miesto označujeme desatinnou čiarkou. V anglicky hovoriacom svete a preto aj v programovacích jazykoch sa však používa bodka – to je ten „point“ z názvu.
[4] Ak ešte dvojkovú sústavu dostatočne dobre nepoznáte, silne odporúčame zoznámiť sa s ňou skôr, ako budete ďalej čítať tento študijný text. Začať môžete na https://sk.wikipedia.org/wiki/Dvojkov%C3%A1_%C4%8D%C3%ADseln%C3%A1_s%C3%BAstava.

Na miestach, kde nebude jasné, ktoré čísla sú v dvojkovej a ktoré v desiatkovej sústave, použijeme dolné indexy. Napr. 11_{10} je jedenásť, zatiaľ čo 11_2 je tri.

To, že počítače používajú dvojkovú sústavu namiesto desiatkovej, ich nijak neobmedzuje. Totiž podobne ako v desiatkovej sústave, aj v dvojkovej vieme zapisovať aj necelé čísla: 0.1_2 je jedna polovica, 0.01_2 je jedna štvrtina, a tak ďalej. Teda napríklad $0.1001_2 = 0.5625_{10}$.
Samozrejme, niektoré reálne čísla majú konečný zápis, zatiaľ čo iné nie. Všetky iracionálne čísla, ako napríklad π, e, alebo $\sqrt{2}$, majú aj v dvojkovej sústave nekonečný zápis, ktorý navyše nie je periodický. Ako sme už videli vyššie, niektoré racionálne čísla majú v dvojkovej sústave konečný zápis. Ale nie všetky. Napríklad jedna tretina v dvojkovej sústave je $0.\overline{01} = 0.01010101\ldots$
Zaujímavé je tiež, že každé racionálne číslo, ktoré má konečný zápis v dvojkovej sústave, má konečný zápis aj v desiatkovej sústave. (Viete, prečo je to tak? Skúste si to dokázať.) Naopak to však neplatí. Napríklad jedna pätina je v desiatkovej sústave 0.2, zatiaľ čo v dvojkovej je to $0.\overline{0011}$.

V desiatkovej sústave niekedy používame tzv. vedecký zápis reálnych čísel. Pri tomto zápise posunieme desatinnú bodku za najvýznamnejšiu cifru čísla a zvlášť uvedieme, akou mocninou 10 treba toto číslo vynásobiť. Vyššie sme si už spomínali jeden príklad: číslo 1 234 567 zapíšeme ako 1.234567×10^6. Číslo 0.0047 vieme zase zapísať ako 4.7×10^{-3}.
Toto isté ideme teraz spraviť v dvojkovej sústave. Tam je to ešte o čosi jednoduchšie – totiž v každom čísle okrem nuly má jeho najvýznamnejšia cifra hodnotu 1. Každé číslo okrem nuly teda vieme v dvojkovej sústave zapísať v nasledovnom tvare: $\pm 1.m \times 2^e$.
V tomto zápise m (nazývané *mantisa*) predstavuje ľubovoľnú, konečnú alebo aj nekonečnú, postupnosť cifier, zatiaľ čo e (nazývané *exponent*) je nejaké celé, možno aj záporné číslo.
Napríklad uvažujme číslo $x = 5.125_{10}$. Jeho klasický zápis v dvojkovej sústave je $x = 101.001_2$. Toto isté číslo ale vieme zapísať aj nasledovne: $x = 1.01001 \times 2^2$. Pre toto x teda máme mantisu 01001 a exponent 2.
Číslo $y = -0.125_{10} = -0.001_2$ v tomto tvare zapíšeme nasledovne: $y = -1. \times 2^{-3}$.

Keď sa reálne čísla ukladajú do pamäte počítača vo formáte s plávajúcou bodkou, používame presne tento formát, s jedinou zmenou: pamäť počítača je bohužiaľ konečná. Preto v tomto formáte nevieme reprezentovať ani zďaleka všetky reálne čísla. Presne vieme uložiť len tie, ktoré majú v dvojkovej sústave konečný a dostatočne krátky zápis (t.j. krátku mantisu) a zároveň dostatočne malý exponent. Ak chceme uložiť nejaké iné číslo, máme smolu. Najlepšie, čo vieme spraviť (a čo sa aj v skutočnosti v počítači stane) je, že nastane *zaokrúhľovacia chyba* a namiesto skutočnej hodnoty si počítač zapamätá najbližšiu hodnotu, ktorú skutočne uložiť vie. Inými slovami, zapamätá si z mantisy len toľko *najvýznamnejších bitov*, koľko sa mu zmestí do pamäte.[5]
Ak si teda v takomto formáte uložíme hodnotu $4/3 = 1.\overline{01}$, nebude v pamäti počítača uložená presná hodnota štyroch tretín, ale nejaké číslo, ktoré sa od nich o maličkú trošku líši. Ak by sme napríklad v pamäti mali miesto len na sedem bitov mantisy, uložili by sme si namiesto presných štyroch tretín číslo 1.0101011×2^0. Všimnite si, že uložená mantisa (0101011) končí cifrou 1, keďže toto číslo je ku štyrom tretinám bližšie ako 1.0101010×2^0.

Dva najbežnejšie používané verzie formátu s plávajúcou bodkou sa volajú *single precision* a *double precision* (stručne len *single* a *double*, voľne preložené: obyčajná a dvojnásobná presnosť). Číslo uložené vo formáte single zaberá 4 bajty pamäte, a to nasledovne: 1 bit je znamienko[6], potom 8 bitov exponent, a na záver 23 bitov tvorí mantisa. Číslo vo formáte double zaberá dvakrát toľko pamäte, teda 8 bajtov. Opäť je tam jeden bit na znamienko, potom máme 11-bitový exponent a na záver 52-bitovú mantisu.
Bežné formáty „reálnych“ čísel v programovacích jazykoch zvyknú zodpovedať týmto dvom verziám: single precision je napríklad `float` v Jave aj C++ a `single` vo FreePascale, double precision je napríklad `double` v Jave, C++, aj FreePascale a `float` v Pythone.
Všimnite si, že pri ukladaní čísla v tvare $(\pm 1.m \times 2^e)$ ukladáme ako mantisu len m. Jednotku pred ňou ukladať netreba, keďže tú majú všetky čísla rovnakú. Ak teda máme 23-bitovú mantisu, znamená to, že si do pamäte

[5] Drobný technický detail: Čo sa stane, ak sa presná ukladaná hodnota nachádza presne v strede medzi dvoma hodnotami, ktoré si vieme v pamäti počítača reprezentovať? My budeme v tomto zadaní v súlade so štandardom predpokladať, že takúto hodnotu vždy zaokrúhlime na „okrúhlejšie“ číslo – inými slovami, vyberie sa tá možnosť, pri ktorej uložená mantisa končí nulou.
[6] Nula je plus, jednotka je mínus.

uložíme *24 najvýznamnejších bitov* ukladaného čísla. A keďže $2^{24} > 10^7$, neformálne to znamená, že v desiatkovej sústave by sa nám aj napriek zaokrúhľovacím chybám malo zachovať aspoň sedem najvýznamnejších cifier.
Príklad: číslo $z = 1.234567890_{10}$ má v dvojkovej sústave rozvoj $z = 1.00111100000011001010010000101\ldots_2$. Ak by sme z uložili do premennej typu single, počítač by si zapamätal len prvých 24 bitov tohto čísla. Skutočná hodnota uložená v pamäti by teda nebola presne z. (Pre zaujímavosť, bola by to presne hodnota $z' = 1.2345678806304931640625$. Všimnite si, o koľko sa z a z' líšia.)
Pre formát double je 2^{53} o čosi menej ako 10^{16}. Ak teda reálne číslo reprezentujeme vo formáte double, chyba sa pohybuje na úrovni zhruba sedemnástej cifry v jeho desiatkovom zápise.

Pozorného čitateľa už isto nejakú dobu trápi otázka čo s nulou. Nulu totiž nevieme zapísať v tvare $(\pm 1.m \times 2^e)$, a teda ju je nutné ošetriť ako špeciálny prípad. Konkrétne riešenie otázky ako uložiť nulu je prekvapivo zložité. V skutočnosti totiž nula nie je jediným špeciálnym prípadom. Pri návrhu formátov single a double sa totiž ich autori rozhodli pridať aj ďalšie špeciálne hodnoty: plus aj mínus nekonečno, malé hodnoty veľmi blízke nule, aj špeciálne hodnoty predstavujúce, že výsledok operácie nie je číslo. (Napríklad pokus v reálnych číslach odmocniť -1 vedie k takejto hodnote.) Pre účely tejto úlohy nepotrebujete vedieť technické detaily, postačí poznamenať, že všetky tieto špeciálne prípady sa riešia tak, že sú pre ne rezervované dve konkrétne hodnoty exponentu.

Zaokrúhľovacie chyby vznikajú aj pri aritmetických operáciách. Aj číslo 10, aj číslo 3 vieme uložiť presne, ale keď vypočítame ich podiel, dostaneme číslo $10/3$, ktoré už presne uložiť nevieme. A aj ak máte doma hocijakú jednoduchú starú kalkulačku, môžete si vyskúšať, že keď 10 vydelíme a následne vynásobíme tromi, na displeji sa nám namiesto desiatky zjaví 9.9999999.
Iný príklad: predstavme si, že človek, ktorý si vie pamätať len sedem cifier z každého čísla, dostane za úlohu sčítať čísla 1 000 000 a 1 234.567. Čo dostane ako výsledok? Namiesto presného 1 001 234.567 si zvládne zapamätať iba zaokrúhlenú hodnotu 1 001 235. A presne to isté sa stane pri aritmetických operáciách v počítači: môže sa stať, že výsledok operácie nebudeme vedieť uložiť presne, a teda jeho zaokrúhlením vznikne ďalšia chyba.

Súťažná úloha

Ak ste sa úspešne dočítali až sem, iste sa už tešíte na súťažné úlohy. Sú hneď tri. Hodnotené sú každá zvlášť.

- **Podúloha A (2 body).** Nech a a b sú premenné typu double. Platí, že $a + b$ je vždy rovné $b + a$?
- **Podúloha B (4 body).** Nech a, b, c sú premenné typu double. Platí, že $(a+b)+c$ je vždy rovné $a+(b+c)$?
- **Podúloha C (4 body).** Uvažujme nasledujúci jednoduchý program: zoberieme premennú typu double, nastavíme ju na 0 a potom ju zvyšujeme o 0.1 a vypisujeme hviezdičky až kým nedostaneme hodnotu 1. Príklad implementácie v Pythone:

```
x = 0
while x != 1:
    x += 0.1
    print('*')
```

  Čo sa stane, keď tento program spustíme? Vysvetlite, prečo je to tak, a povedzte, aké poučenie z toho vyplýva pre programátorov.

Odovzdávanie riešení

Toto je teoretická úloha. Spísané riešenie vo formáte PDF odovzdajte pomocou webového rozhrania.

TRIDSIATY TRETÍ ROČNÍK OLYMPIÁDY V INFORMATIKE

Príprava úloh: Michal Anderle, Michal Forišek
Recenzia: Michal Forišek
Slovenská komisia Olympiády v informatike
Vydal: IUVENTA – Slovenský inštitút mládeže, Bratislava 2017