```
length([]) -> 0;
length([_ | Rest]) -> 1 + length(Rest).
```

První funkce vrací první prvek seznamu (usekne, uloží do proměnné `First`, zbytek odloží do „odpadní proměnné“ a vrátí tu první hodnotu). Druhá funkce vrací druhý prvek (obdobně). Třetí vrací seznam bez prvního prvku (tedy, o jeden ho zkrátí), čtvrtá dělá pravý opak, jeden prvek na začátek přidá. Pátá počítá délku seznamu – postupně odebírá, než nezbude nic, a přičítá jedničky. Na ní je také vidět, jak lze zařídit projití celého seznamu.

Všimněte si, že nelze pracovat s druhým koncem seznamu, máme k dispozici jen jeho začátek a k věcem dál se musíme prokousat (ukusovat hlavičky).

A ještě slíbená návštěva řetězců. Řetězce jsou seznamy jako každé jiné. Řetězec `"ahoj"` je totéž jako zápis `[97, 104, 111, 106]`, jen vypadá méně l33t (čti: je použitelný).

To bude pro tuto sérii stačit, v příští si rozebereme, jak pouštět více procesů, jak mezi nimi probíhá komunikace a jak k tomu použít více počítačů.

Aby vám čekání na další sérii nepřišlo tak dlouhé, je tu několik malých úložek:

1) Každý druhý: Napište funkci, která dostane seznam (libovolných hodnot) a vrátí seznam, ve kterém se bude vyskytovat každá druhá z tohoto seznamu. Znáte film „Nesmrtelná teta“, ne? *[3 body]*

2) Fibonacciho čísla: Funkce dostane číslo N a vrátí N-té Fibonacciho číslo v pořadí. Fibonacciho čísla jsou definována takto:

$$\mathrm{Fib}_1 = 0$$
$$\mathrm{Fib}_2 = 1$$
$$\mathrm{Fib}_n, n \geq 2 = \mathrm{Fib}_{n-1} + \mathrm{Fib}_{n-2}$$

Dejte si pozor na časovou složitost. *[4 body]*

3) Prvočísla: Funkce dostane číslo N a vrátí seznam prvních N prvočísel. *[5 bodů]*

Korespondenční Seminář

z Programování

Milí chlapci a děvčata, jako každý rok je tu opět **Korespondenční Seminář z Programování.** Že jste o něm ještě neslyšeli? V tom případě si zkuste odpovědět na následující kvíz:

- Zajímáš se o počítače?
- Rád soutěžíš?
- Chceš se dozvědět něco nového?
- Chceš poznat nové lidi?
- Chceš užitečně vyplnit volný čas?
- Hledáš výzvu pro svoji hlavu?

Odpověděl sis alespoň jednou „ano“? Pak hledáme právě Tebe. KSP hledá nové řešitele a zapojit se může každý. Máš-li chuť, otoč list …

Nyní určitě hledáš odpověď na otázku, která se Ti honí v hlavě od chvíle, kdy jsi spatřil tento leták. Ty znáš tu otázku. Bohužel Ti neumíme dost dobře popsat, co KSP je, ale můžeme Ti to ukázat. Pro bližší představu jsme připravili pár informací, které by Tě mohly zajímat:

- KSP je celostátní a celoroční soutěž v programování pro studenty středních i základních škol.
- Jeden ročník je rozdělen na 5 sérií.
- V každé sérii účastníci obdrží poštou zadání úloh.
- Úlohy vyřeší v teple domácího krbu a svá řešení nám zašlou zpět (opět poštou, případně přes webové rozhraní).
- My jim vrátíme opravené úlohy společně se vzorovými řešeními, zpravidla se zadáním další série.
- Na vyřešení jedné série je několik týdnů času.
- Série obsahuje šest programátorských úložek a každému řešiteli jsou započítány čtyři nejlépe vyřešené.
- Úlohy jsou čistě algoritmického rázu. Rychlejší a lépe popsané algoritmy mají přednost před programy hýřícími barvami.
- Každá úloha je bodována, body ze všech úloh ze všech sérií se sčítají a tvoří celkové hodnocení.
- Pro nejlepší řešitele pořádáme na začátku dalšího školního roku (obvykle v říjnu) **soustředění,** na kterém se nejen dozví užitečné věci z programování, ale také si protáhnou tělo i mysl při ryze neinformatických činnostech.
- Další informace a **přihlášku** nalezneš na *http://ksp.mff.cuni.cz/*, dotazy (ale ne řešení úloh) můžeš posílat na *ksp@mff.cuni.cz*.
- Hodně štěstí!


```
abs(N) -> if
        N < 0 -> (-1) * N;
        true -> N
        end.
```

Před každou větví se nachází podmínka: když bude pravdivá, vybere se tato větev, když nebude, pokračuje se ve zkoušení. Každá větev (může jich být libovolně mnoho) je ukončena středníkem – kromě té poslední. Celý `if` končí klíčovým slovem `end`. Je třeba si uvědomit, že `if` má výsledek – výsledek obsahu té větve, která se vybrala.

Všimněte si „podmínky“ `true`, která platí vždy, tedy se používá místo obvyklého `else` – jako poslední možnost, kam spadne vše, co nespadlo nikam dřív.

Obdobně funguje `case`, ale v něm probíhá test na „napasování“ do proměnných, podobně jako uvnitř závorek funkce. Ukážeme si s ním upravený faktoriál:

```
fac(N) -> case N of
        0 -> 1;
        _ -> N * fac(N - 1)
        end.
```

Toto vezme ono `N` a zkusí ho uložit (podobně jako v parametrech) do té věci nalevo od `->`. Pokud se to povede, provede se vnitřek (a proměnné použité nalevo se tím případně naplní).

Další věc, která stojí za zmínku, je „proměnná“ `_`. Aby nám `case` fungoval, musíme mu dát proměnnou (nebo výraz), do které obsah uloží. Ale pokud její hodnotu nechceme použít, překladač by si stěžoval. Proměnná `_` je speciální v tom, že se používá jako odpadní a znamená, že hodnotu tady chceme pouze zahodit, nikoliv ukládat. Lze ji samozřejmě použít i na jiných místech – na levé straně přiřazení, uvnitř parametrů a podobně. Do této „proměnné“ se smí uložit vícekrát, ale zase se z ní nesmí nikdy číst.

U všech podmínek (i těch přes parametry) je potřeba, aby některou z voleb program vybral. Pokud se výpočet dostane až „za konec“ možností, program spadne s chybou, protože neví, jaký je výsledek daného výrazu.

Co se týče jazykových konstrukcí, v jedné funkci smí být více „příkazů“, které se provedou za sebou. Lze tak například rozložit složitější výpočet či vložit ladící výpis.

```
kvadratic(x, a, b, c) -> kvad = a * x * x,
        linear = b * x,
        kvad + linear + c.
```

V takovém případě je výsledkem funkce poslední výraz.

Nejen jazykovými konstrukcemi živ je program, ještě jsou potřeba data. Zatím jsme používali jen čísla. Co se jich týče, tak se s nimi smí dělat obvyklé věci, jako sčítat, násobit, dělit a podobně. `/` dělí neceločíselně, k celočíselnému dělení se používají `div`, resp. `rem` pro zbytek.

Celá čísla mohou být libovolné délky, tedy nám nehrozí, že by číslo přeteklo, jako to může nastat u jiných jazyků.

Čísla jsou užitečná, ale co kdybychom chtěli rozlišovat mezi několika druhy informace? Potom použijeme atomy. Atom je nějaké slovo, které začíná na malé písmeno, dá se použít přímo jako hodnota proměnné. Mezi atomy nefunguje žádná aritmetika a jedinou věc, kterou s nimi (kromě ukládání) smíme dělat, je porovnávat.

Například, pokud chceme rozlišovat pohlaví našich řešitelů, tak budeme mít hodnoty `girl` (slečny mají přednost, přestože jsou v KSP v menšině) a `boy`.

Dalším typem, i když trochu falešným, jsou pravdivostní hodnoty. Pravdivostní hodnotu dostaneme například porovnáním dvou hodnot. Reprezentovány jsou atomy `true` a `false`. Mezi nimi fungují obvyklé spojky `not`, `and` a `or`. U `and` a `or` nefunguje zkrácené vyhodnocování výrazů – pokud potřebujeme vyhodnocovat zkráceně, použijeme `andalso` a `orelse`.

Data je možné seskupovat do n-tic. Je to jakási obdoba recordu v Pascalu, jen jsou položky nepojmenované a jejich význam je stanoven pořadím. Každá n-tice se uzavírá do složených závorek a dá se k ní přistupovat jako k jedné proměnné, nebo ji rozložit na části (v parametrech, v přiřazení, v `case`). Mohli bychom například mít n-tici, která by obsahovala jméno člověka, věk a jeho pohlaví. Pak by jeden záznam mohl vypadat třeba takto:

```
{"Tomáš Marný", 14, boy}
```

Řetězec, použitý v tomto případě, zajisté nikoho nezmate, k němu se dostaneme později.

Funkce, která vybere věk „gentlemanským“ způsobem (ženám ho snižuje), by vypadala takto:

```
age({_, Age, boy}) -> Age;
age({_, Age, girl}) -> if
        Age > 1 -> Age - 2;
        true -> 0
        end.
```

Všimněme si, že n-tici rozebereme hned v parametrech (nemuseli bychom), jméno zahodíme a pomocí atomů rozlišujeme, o koho se jedná. Taktéž, pokud by nám někdo chtěl do funkce propašovat jedince jiného nějakého třetího pohlaví, tak program spadne (což je správné řešení – pokud funkce něco neumí, tak je lepší spadnout, než to dělat špatně).

Problém s n-ticemi je, že obsahují pevný počet hodnot. Většinou se používají, když patří několik informací různých druhů k sobě (proto také přirovnání k recordu v Pascalu). Ale často je potřeba uchovávat víc hodnot stejného druhu, ale v době psaní programu se neví, kolik (technicky vzato, jazyk nevynucuje, aby měly stejný typ, ale bývá zvykem uchovávat pohromadě jen věci, které k sobě patří). Na toto se používají seznamy (obdoba pole v Pascalu).

Seznamy mají závorky hranaté, hodnoty se opět oddělují čárkami. Prázdný seznam jsou prostě prázdné hranaté závorky, tedy `[]`.

Není nutné pracovat vždy s celým seznamem. Pomocí svislítka lze „odseknout“ přední část a pracovat se začátkem a zbytkem. (Často se používají termíny hlava a ocas, představme si hada, který, když mu uříznete hlavu, tak ten zbytek je zase had, kterému lze uříznout hlavu, … až nám zbude jen hlava a zbylý had bude mít nulovou délku. Ne, možná si to radši nepředstavujte, nebudeme ubližovat nevinným hadům.) Tímto způsobem lze na začátek přidávat a ze začátku odebírat hodnoty. Tedy seznam `[1, 2, 3]` je totéž jako `| [1 | [2 | [3 | []]]]`. Lépe zhlédnout na příkladu:

```
first([First | _]) -> First.

second([_, Second | _]) -> Second.

rest([_ | Rest]) -> Rest.

add(List, Item) -> [Item | List].
```

22-1-6 Náhradní — 10 bodů

Vymyslete algoritmus, který když na vstupu dostane velký objem textu a několik klíčových slov, vyplivne nejkratší úsek textu, který obsahuje všechna zadaná klíčová slova, a nebo zahlásí, že se v něm nějaké z nich vůbec nevyskytuje.

Dostane-li kupříkladu algoritmus na vstupu text předchozího odstavce a dotaz na slova „klíč“ a „text“, vrátí „textu a několik klíč“. Dostane-li stejná klíčová slova k prohledání v tomto odstavci, najde řetězec „klíč” a ”text“.

Dostane-li v tomto odstavci vyhledat po slovech nasekanou první větu z Alenky v říši divů, nenajde slovo, které tu z taktických důvodů neuvedeme, ale jehož znění algoritmus vypsat má.

BC: Dobře. Tady máš ten diktafon ... Hodně štěstí při sepisování!

MS: Děkuju pěkně. Hezký zbytek zkouškového!

* * *

Byl jednou jeden vlak. 5. července 2009, kolem jedenácté hodiny dopolední, v Brně, právě vyjížděl z nádraží. Oplýval elektrickou lokomotivou a šesti vagónky.

Takový vlak jenom jeden určitě nebyl. Ve skutečnosti jich byla docela spousta a navzájem se lišily verzí papíru v třetím kupé druhého vagónu.

22-1-7 Když telefony pekly jazyk — 12 bodů

„Ale né, už zase!“ – tak takto bude reagovat nebohý agent, který tupě zírá do mobilu, protože spadl server stroje času.

„Ale né, už zase!“ – tak takto bude reagovat nebohý programátor, který je nucený mistrem Motivátorem napsat software na server tak, aby mohl běžet na více počítačích, a pokud některý z nich odnese povodeň, ostatní ho zvládly zastoupit.

V této a příští sérii se podíváme na zoubek jazyku, který takovým programátorům dokáže ulehčit život – na Erlang, který je dělaný na snadné spouštění mnoha vláken na více počítačích. Na zvýšení výkonu dokoupením dalších počítačů sice není nejlepší (je drobátko líný), ale na zvýšení spolehlivosti je (jako) dělaný.

Napřed si zkusíme jednoduchý příklad. Nainstalujte si Erlang, který je na adrese *http://erlang.org/download.html* a spusťte. Měla by se objevit jeho příkazová řádka. Pokud po napsání `2*21.` odpoví odpovědí na otázku života, vesmíru a vůbec, tak je vše v pořádku. Všimněte si té tečky na konci, ta ukončuje výraz a říká, že bychom rádi získali odpověď.

Podívejme se na to, jak se programy v Erlangu zapisují. Erlang pochází z rodiny funkcionálních jazyků, základním stavebním kamenem jsou tedy funkce, nikoliv příkazy, jako tomu je například u Pascalu. A protože bychom raději trvanlivější programy, které není potřeba pokaždé přímo zadávat, budeme je psát do souboru.

Zatímco klasická ukázka procedurálních jazyků je program „Hello world“, funkcionální klasikou je faktoriál. Uložme tento kód do souboru `factorial.erl`. (Bez tečky na konci, ta ukončuje větu.)

```
-module(factorial).
-export([fac/1]).
fac(0) -> 1; % Konec cyklu
fac(N) -> N * fac(N - 1). % Ještě jednou, prosím
```

Na začátku vidíme direktivu `module`, která dává jméno aktuálnímu modulu (takové knihovničky funkcí), a direktivu `export`, která říká, které funkce mají být vidět zvenčí. Ona `1` za lomítkem udává, kolika parametrovou verzi dané funkce máme na mysli (funkce se rozlišuje podle jména a počtu parametrů).

Část s vlastní funkcí je již zajímavější. První věc, která stojí za povšimnutí – jména proměnných začínají velkým písmenem. Tímto způsobem jazyk rozlišuje proměnné od ostatních věcí, jako jsou funkce, atomy (k nim později) a klíčová slova.

Proměnné v Erlangu jsou trochu jiné, než v „běžných“ programovacích jazycích. Nepředstavují ani tak pojmenované místo v paměti, jsou to spíš matematické definice. To znamená, že do jedné proměnné se za dobu jejího života může přiřadit jen jednou (v době jejího vzniku).

Druhou zajímavou věcí je to, že funkce má dvě „verze“ – jednu pro případ, kdy je parametrem nula, druhou, když je to cokoliv jiného. Při zavolání funkce Erlang vezme první verzi, zkusí jestli do ní parametry „pasují“, pokud ano, zavolá tu (a tím skončí), pokud ne, zkouší dále. Každá verze je ukončená středníkem, poslední tečkou.

Třetí věcí jsou komentáře. Cokoli je za znakem `%` až do konce řádku, je ignorováno. Stejně tak jsou ignorovány všechny bílé znaky (mezery, nové řádky, tabulátory), ty jen od sebe oddělují tokeny.

A poslední věc, která stojí za zmínku, funkce volá sama sebe. To je povolené a v Erlangu je to jediný způsob, jak akce opakovat, Erlang nemá cykly jako procedurální jazyky. Pro zkušenější, Erlang dělá takzvanou ocasní optimalizaci[3], takže pokud jako poslední věc ve funkci je volána jiná funkce, aktuální se ze zásobníku odstraní. Toto umožní napsat nekonečný „cyklus“ bez strachu o přetečení zásobníku.

Nyní zkusíme tento modul použít, opět z Erlangovské příkazové řádky.

```
1> c(factorial).
{ok,factorial}
2> factorial:fac(3).
6
```

Funkce `c` zkompiluje a zpřístupní modul daného jména. Pokud nastane chyba (například chybí někde tečka), tak vypíše hlášku, která by měla pomoci s jejím nalezením.

Pokud voláme funkci ze stejného modulu, stačí napsat jméno funkce. Pokud ale voláme nějakou „cizí“ funkci, je potřeba před dvojtečku zadat i modul původu.

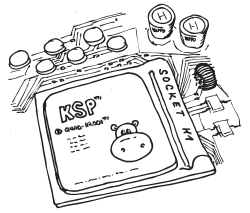
Další věc, která se hodí, jsou nějaké podmínky. První způsob jsme již probrali – uvnitř parametrů funkce. Pokud toto nestačí, můžeme vhodnost funkce upřesnit pomocí konstrukce `when`, například takto by mohla vypadat funkce počítající absolutní hodnotu:

```
abs(N) when N < 0 -> - N;
abs(N) -> N.
```

Další možností je použít `if`. Ten funguje podobně jako podmínky za `when`, ale je „uvnitř“ funkce. Absolutní hodnota by s ním vypadala takto:

[3] Překlad „tail optimalization“ :-)

Milí řešitelé a řešitelky!



Pokud čtete tyto řádky, pak poštovní dromedár v poušti nezabloudil a donesl vám první ratolest 22. ročníku Korespondenčního semináře z programování. Nelelkujte, do úložek se pusťte a vyřešené nám je do 9. října 2009 odevzdávejte. Řešení nám pošlete buď přes trubky digitální (*http://ksp.mff.cuni.cz/submit/*), nebo trubky analogové:

Korespondenční seminář z programování
KSVI MFF UK
Malostranské náměstí 25
118 00 Praha 1

První série dvacátého druhého ročníku KSP

Díky neuvěřitelnému pozitivnímu ohlasu na kratší příběhy (přesné číslo z bezpečnostních důvodů neuvádíme, pouze napovíme, že bylo nezáporné a zároveň nekladné) jsme se rozhodli v nich pokračovat. Dnešní porci úložek s hranolky vám s úsměvem na tváři servíruje Lukáš Lánský. Seriál vyšel z digitálního pera Michala Vanera.

Byl jednou jeden vlak. 5. července 2009, kolem jedenácté hodiny dopolední, v Brně, právě vyjížděl z nádraží. Oplýval elektrickou lokomotivou a šesti vagónky.

Takto určený vlak byl nejspíš jenom jeden: jel z Brna do Pardubic a na české poměry rychle. Po celý zbytek vyprávění nás na něm bude zajímat toliko třetí kupé v druhém vagónku odpředu.

Naše motivace ke sledování konkrétně tohohle objektu bude taková, že si tam mezi sebou lidé začali povídat (což už je samo o sobě zajímavé) a že si tam začali povídat o něčem důležitém: tomto textu. Tedy, „tomto“ ...

* * *

Pavel dostudoval první ročník čtyřletého gymnázia a nudil se. To tam bylo prvotní okouzlení novým prostředím a do jeho chování a postojů se během posledního půlroku znovu vkradla apatie: vždyť proplouvat školou je tak snadné!

Nebylo tomu tak vždy. V osmé třídě základky zavadil o matematickou olympiádu: dostal diplom a poukázku do knihkupectví, vůbec mu to přišlo jako příjemné setkání, které by si byl rád zopakoval, kdyby nebyl tak strašně líný dělat příští rok domácí kolo. Hm, dobrá, možná tomu tak vždy bylo: Pavlovi se však nedal upřít talent.

Do kupé přišel bez většího zavazadla, jel k babičce. Vlastně měl s sebou jenom jednu knížku.

„Tak ta je hustá,“ osloví ho dívka sedící u okénka. Kouká po knížce. „Sice trošku mainstreamová, ale hustá.“

Pavel se chtěl nejdřív nevrle ohradit, že Gaiman není žádný mainstream, ale po pečlivém prostudování dívčina vzezření zvolil vlídnější „Ahoj, jmenuju se Pavel!“

Vypadala totiž takto: Středně vysoká, snad osmnáctiletá, dlouhé hnědé vlasy.

Odpověděla: „Já jsem Alča.“

V kupé nebyli sami, ale o tom až někdy dál. Prozatím jen: Třetí člověk, zhruba dvacetiletý, seděl u dveří a vypadal, že je hluboce pohroužen do knížky o Haskellu. Našim dvěma hlavním aktérům to po celou dobu cesty nepřišlo zajímavé, ani jeden z nich neprogramoval v něčem, čeho název by nezačínal písmenem C.

Ta slečna že programuje? Pavel na to přišel záhy, když poznamenala:

A: Co ti z ní čouhá, je KSPéčko?

P: Těžko, vždyť ani nevím, co to „KSPéčko“ je.

A: Ale je to KSPéčko! Poznám to podle obrázků.

P: Hroši? Divný. Aha, dali nám to ve škole, použil jsem to jako záložku.

A: Tohle zadání je divný, ale ne protože by na něm byli hroši. To je normální a v pořádku a vůbec opovaž se říct cokoliv proti hrochům!

Vypadala naštvaně!

P: Promiň! Hm, můžeš...

A: No?

P: Můžeš mi říct, co to KSP je?

A: Můžu ti to ukázat. Ale jak říkám, tohle zadání je divné ... Podívej: KSP znamená Korespondenční seminář z programování. Zhruba to funguje tak, že když vyřešíš, co je na tomhle papíře za programátorské příklady, dostaneš body a budeš se cítit chytře.

P: Žádný peníze?

A: Ne. Je to prostě něco jako škola, má tě to vzdělávat.

P (skepticky): Aha. Hm, takže Gaiman...

A: Gaiman je proti tomuhle nudnej. Podívej, obyčejně ta zadání vypadají tak, že tam je nějaký příběh, v rámci kterého jsou ty úlohy.

P: Aby se mi v tom hůř hledalo zadání?

A: Ne, to aby se orgové vyblbli. S řešiteli to nesouvisí ... Nekoukej tak nedůvěřivě, tak to je!

P: Tady na tom papíře příběh není. Jen pár příkladů.

A: Jo. Podívej, jak tady vypadá první úloha: „Buď G orientovaný graf, jehož hrany jsou ohodnoceny právě jedním prvkem z množiny $\{+,-\}$. Je dán počáteční vrchol S a cílový vrchol C. Nechť P je cesta z S do C: najděte takové P, že se po ní co nejméněkrát střídají znaménka na hranách. Kupříkladu pokud existuje cesta se znaménky `+++--------+`, vybereme spíš ji než takovou se znaménky `+-+-+`, přestože je co do počtu použitých hran delší.“

P: To je strašný! Co je to graf? Co je to hrana, vrchol?

A: Kus matematiky. Když si na něj zvykneš, budou se ti takovýhle úlohy řešit snáz, dobrou knížku o tom napsal třeba Deml[1] nebo Matoušek s Nešetřilem[2]. Podstatný je, že takhle se tam podobné věci nezadávají! Nesmí z toho tak trčet matika, na tu si musíš přijít sám, v tom je kus legrace!

P: Jak by to teda mělo vypadat?

A: Jejda, to záleží. Vlastně na tom nezáleží, jen ... Já nevím, třeba takhle:

22-1-1 Alčina interpretace — 10 bodů

Máme velký dům se spoustou pokojů, mezi některými z nich vedou schodiště: z jednoho pokoje do druhého se buď stoupá, nebo klesá. Z nějakého důvodu hledáme cestu z jednoho pokoje do druhého tak, abychom co nejméněkrát musili přestat vycházet schody a začít scházet, nebo naopak přestat

[1] viz *http://ksp.mff.cuni.cz/study/paperbooks.html*

[2] tamtéž

scházet a začít vycházet.

A: Rozumíš?

P: Hm, jo. Ale můžu být upřímný? Proč kolem toho tak jančíš? Tak změnili styl. Nová krev, mladí lidé s názory radikálně odlišnými od svých předchůdců ... Nic zvláštního, nic, nad čím stojí za to mluvit, když slunko svítí, prázdniny sotva začaly a my oba jsme tak báječně mladí. (Mrk.)

A: (Jemně zdvižené obočí, jinak nic.) Ale ty další úlohy jsou ještě horší! Vždyť podívej na tu dvojku ... Na Catalanova čísla a nic jiného. Kdo je zná, má to hned hotové!

P: Život není peříčko: vždyť tak to bude vždycky! Každou úlohu někdo zná.

A: Ale Catalanova čísla jsou mainstream.

P: Jako Gaiman, viď ...

A: Jo!

P: Tak vymysli něco lepšího! Vymysli krásnou novou úlohu a pošli jim ji spolu se svým řešením, vždyť jistě budou rádi.

A: Vidíš ten sad, kolem kterého projíždíme?

P: Ty vzory jsou zajímavé. Jak jsou stromy vysázené do čtvercové mřížky, ustupují nám z výhledu souhlasně, a stejně tak do něj vstupují. Je to pěkné a dělá to zajímavé obrázky.

A: (Zamyšleně.) Představ si, že stojíš v počátku kartézské soustavy souřadnic a sad jsou všechny body se souřadnicemi (a, b), kde a, b jsou nezáporná celá čísla, jejichž součet je menší než nějaké N.

```
. . . . . . . .
. x . . . . . .
. x x . . . . .
. x x x . . . .
. x x x x . . .
. P x x x x . .
```

A: Které stromy v takové zjednodušené situaci vidíš?

P: Hmpf. :-(

A: Zkus si to nakreslit takhle!

```
P    1/1  2/2  3/3 ...
0/1  1/2  2/3  ...
0/2  1/3  ...
0/3  ...
...
```

P: Aha, takže je to vlastně úloha na hledání zlomků v základním tvaru! Protože ty stromy, které „nejsou v základním tvaru“, jsou takovými určitě zakryté! A ty, které jsou, nejsou!

A: Příznačně řečeno.

22-1-2 Sad — 9 bodů

Váš program dostane číslo N a vy máte za úkol vypsat podle velikosti seřazené všechny zlomky v základním tvaru, které jsou větší nebo rovny nule a menší nebo rovny jedničce a ve jmenovateli mají méně nebo rovno N.

Pro $N = 5$ nechť vypíše $0/1, 1/5, 1/4, 1/3, 2/5, 1/2, 3/5, 2/3, 3/4, 4/5, 1/1$.

Nebo, chcete-li: dostanete velikost strany trojúhelníka z Alčina obrázku a máte vypsat seznam stromů, které jsou vidět z bodu $0/0$.

P: Uznávám, i když jsi evidentně dobrá (mrk), je divné, že vymýšlíš z ničeho úlohy lepší, než jaké zadávají v KSP.

A: Matematika je všude kolem. Koukej se kolem sebe trošku a uvidíš to taky.

P: Kolik ti je let?

A: Dvacet.

P: Vypadáš na osmnáct!

A: Děkuju. :-)

P: Mně je šestnáct...

A: Vypadáš na osmnáct!

P: Děkuju. :-)

A: Hlavně to znamená, že není nic ztraceno: máš spoustu času na sobě pracovat, pozorně číst a pečlivě řešit a jednou se třeba sám postaráš o to, aby se už víckrát nestal tenhle trapas ... Já sama jsem začala řešit taky ve druháku.

P: Co je tam dál?

A: Codexová úloha. Abys věděl, ostatní úlohy vlastně ani nemusíš programovat nebo dokonce ladit. Stačí na kus papíru dobře popsat a odůvodnit vymyšlený algoritmus. V každé sérii je ale jedna praktická, tu napíšeš buď v Pascalu, Céčku, C++ nebo C#, odladíš ji a pak ji necháš automaticky vyhodnotit přes webový rozhraní.

P: Aha. No, to není tak zlé. Nemusím nic popisovat ani dokazovat...

A: Má to svoje jemnosti. Musíš si dávat pozor na implementační detaily, aby sis dobrý algoritmus nezpomalil nějakou špatnou prací se vstupem nebo nevhodnou reprezentací dat. Musíš si taky dobře uvědomit, jak vypadají všechny korektní vstupy, abys nezapomněl ošetřit různé speciální případy. Už jenom to, že se tvůj program kompiluje a spouští někde daleko, je trošku nepříjemný, protože musíš programovat podle toho, co tam mají. Oni se teda snaží jít podle norem: myslím, že v případě C++ to znamená, že tam STL mají, zatímco Crt v Pascalu ne.

P: A proč se tomu říká Codexová úloha?

A: Tak se jmenuje to vyhodnocovadlo, CodEx, Code Examiner. Je to na webu KSP, hlásíš se tam stejně jako do submitovátka.

P: Submitovátka?

A: Tím se elektronicky odevzdávají řešení.

P: Jo. Jasně. Tak co tam třeba teď mají za Codexovou úlohu?

22-1-3 Sazba — 10 bodů

Na vstupu dostanete text a číslo N. Vaším úkolem je zarovnat ho do bloku tak, aby byl co nejhezčí. Protože krása je věc názoru, zadefinujeme si pro naše účely vhodné objektivní měřítko: pro každou posloupnost mezer mezi slovy délky k vezmeme číslo `(k-1)^2` a tato čísla sečteme přes všechny posloupnosti mezer ve vysázeném textu. No a sazba je nejhezčí, pokud je tento součet nejmenší.

Slova nelze dělit mezi řádky a máte zaručeno, že se v textu neobjeví slovo delší než `N`. Na výstup od vás nechceme vypisovat vytvořené zarovnání, ale pouze minimální výše popsaný součet pro daný text, tedy číslo.

Například pro text „This is the example you are actually considering.“ a N=28 má program vypsat 12, protože optimální zarovnání je

```
This  is  the  example   you
are  actually   considering.
```

a ohodnocení $1+1+1+4+1+4 = 12$.

P: To zní rozumně!

A: To je mé převyprávění. Ušetři se sám prosím toho trápení zkoumat, jak hloupě to tam popisují oni.

P: (zklamaně) Áách ... A já jim tolik věřil!

A: Další úloha je taková zvláštní ... Jiná, nová. Tohle tam minulý rok taky nebylo.

P: (čte) „Tato úloha je určena pro celoroční přípravu řešitelů programátorských soutěží, jako je MO-P či IOI.“ To mi přijde docela fajn.

A: Ale jo, proč ne. Jak to mají zadané?

22-1-4 Bludiště — 15 bodů

Váš program na vstupu dostane bludiště $M \times N$, každé jeho políčko se skládá buď ze stěny, nebo volného místa. Do tohoto bludiště je vložen hráč, objekt a je také určená koncová pozice, do které má hráč objekt dostat.

Jak se v bludišti pohybuje? Hráč může volně popocházet na sousední volná políčka. Pokud před ním stojí bedna, za kterou je volné místo, může objekt odsunout na toto volné místo.

Vaším úkolem je najít nejmenší počet posunutí bedny, na který ho lze dostat na koncové místo, a nebo -1, pokud to nejde. Nezáleží tedy na počtu tahů hráče!

Formát vstupu: Na prvním řádku jsou čísla M a N udávající velikost bludiště, následuje M řádků, každý s N znaky. Ty mohou být:

- `#` – stěna
- `.` – volné políčko
- `H` – počáteční pozice hráče
- `O` – počáteční pozice objektu
- `P` – koncová pozice objektu

Znaky `H`, `O`, `P` se na vstupu vyskytují právě jednou.

Pokud tedy program dostane něco takového:

```
12 10
#####...##
#####.#.##
###P....##
#####.####
#.......##
#....O#H##
#.###.####
#.###.####
#.###.####
#.###.####
#.....####
##########
```

Má vypsat: `7`

Tato úloha má i své vlastní bodování: Pokud bude vaše řešení rychlé pro $M, N \leq 1000$, dostanete plných 15 bodů. Napadne-li vás program, který úlohu bude řešit rychle jen pro $M, N \leq 50$, budete ohodnoceni 7 body. Za chyby se bodíky strhávají jako obvykle.

P: Takže Sokoban!

A: Ano. Inu, proč ne. Hádám, že tím uvedeným obodováním se chtějí přiblížit podmínkám na soutěžích.

P: Asi jo. Bylas někdy na nějaké?

A: Ne. Víš, já mám matiku a programování ráda, ale vždycky jsem měla radši zvířátka a evoluční genetiku vůbec. Od září budu studovat na Přírodovědě.

P: Myslel jsem, že jsi úplně jasná matfyzačka!

A: Hm, jak to myslíš?

P: Jaké tam jsou další úlohy?

A: Ošklivé, nezajímavé. Jaks to myslel s tou matfyzačkou?

P: Jako že jsi chytrá. Hele, povídej mi třeba o nějakých minulých zadáních ...

A: Najdi si je na webu. Chytré nejsou jenom matfyzačky, ale třeba i historičky. Tys to myslel jinak, viď?

P: Dobře, tak neznáš nějaké pěkné úlohy, které nebyly v KSPéčku? Hrozně rád bych je slyšel, jsem strašně žhavý do řešení programátorských problémů!

A: Tuhle jsem viděla jednu pěknou ...

P: Ufff ...

22-1-5 Šachovnice na pneumatice — 10 bodů

Existuje docela známá úloha naskládat N královen na šachovnici $N \times N$ tak, aby se podle normálních šachových pravidel vzájemně neohrožovaly.

Co když takové $N \times N$ veliké šachovnici slepíme levý okraj s pravým a horní s dolním, takže bude najednou jedna dáma ohrožovat jednak políčka, která viděla původně, ale i taková, která nyní vidí přes tyto slepené okraje?

Pro jaká N jde za takových podmínek naskládat N dam do šachovnice $N \times N$ tak, aby se neohrožovaly?

Pro $N = 2$ to kupříkladu určitě nejde: položíme-li dámu na kterékoliv ze čtyř políček, bude ohrožovat všechny další. Pro $N = 5$ to můžeme udělat takto:

```
D____
__D__
____D
_D___
___D_
```

A: Pravda, není to ani tolik programátorská, jako spíš logická úložka. Řešením není algoritmus, ale vzoreček nebo několik vzorečků, prostě popis takových N. Ale i takové, logické úlohy, v KSP bývají. Nebo spíš bývávaly.

P: Zajímavé. Rozumím tomu názvu: když vezmeš papír a slepíš ho tak, jak se tam popisuje, vznikne pneumatika. Jestlipak by se ta úloha dala řešit na ještě nějakém jiném povrchu? Möbiově pásce, Kleinově lahvi ...

A: Nejspíš bys zjistil, že to o moc zajímavější není. Ale líbí se mi, jak nad tím přemýšlíš.

P: Už jsme v Pardubicích. Viď, že ještě někdy uvidíme?

A: Proč ne.

Vyměnili si sadu identifikátorů a rozešli se. Spolucestující zavřel knížku o Haskellu, a to na stejné straně, na jaké ji otevřel. Rozhlédl se, to aby věděl, že je vzduch skutečně čistý. Vytáhnul z kapsy mobil, nalistoval vhodnou adresu a zmizel.

Kdo říká, že stroj času nemůže být serverová aplikace?

Objevil se o měsíc dřív ve velmi podobném vlaku a rychle odešel, protože už se nemohl chlubit platným jízdním dokladem.

Byl 5. červen a spolucestující si to pelášil na smluvené, tajné místo, kde už na něj čekal mistr Sazeč.

MS: Ahoj R ... Ehm, bratře Cestovateli!

BC: Ahoj. Mám to. :-)

MS: Fajn. Co s tím provedli?

BC: Byla to hrozná patlanina, posunuli to hodně. Vždyť víš, že jsme nemohli doufat, že už po téhle iteraci to bude použitelné.

MS: Tak jo. Máš všechny úlohy?

BC: Odstřelili jich víc, než zavedli nových, jedna teď chybí.

MS: No tak dáme třeba takovou tu náhradní ... Vždyť víš ...