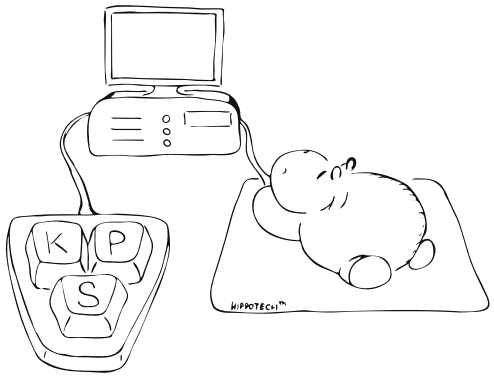
Korespondenční Seminář



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?
- Nevíš co s volným časem?
- 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 4–5 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!

li udaná dvojice má vztah prarodič–vnuk. Jak na to? Uvědomíme si, co vlastně znamená být prarodičem. Prarodičem jste, pokud máte dítě a toto dítě má zase dítě. Zapsáno v Prologu:

```
prarodic(Pra,Vnuk) :- rodic(Pra,Rod),
    rodic(Rod,Vnuk).
```

Napsat babičku a dědečka by pro vás jistě bylo jednoduché.

Rekurze

Chtěli bychom napsat predikát `predek(Pred,Pot)`, který bude zjišťovat, zdali je `Pred` předkem potomka `Pot`, čili jeho rodič, prarodič, praprarodič atd. V příkladu s prarodičem jsme dopředu věděli, že hledáme vztah přes jednu generaci a také jsme tak daný predikát napsali. Jenže teď nemáme ani tušení, kolik generací může mezi Pred a Pot být. Pomůže nám rekurze:

```
predek(Pred,Pot) :- rodic(Pred,Pot).
predek(Pred,Pot) :- rodic(Pred,X),predek(X,Pot).
```

A zeptáme se:

```
?-predek(anna,kvetos).
```

Jak už víme, Prolog se nejdřív podívá na první řádek a zjistí, jestli náhodou `anna` není přímo rodičem `kvetose`. Rodič je přeci také předek. Pokud `anna` opravdu je rodičem `kvetose`, problém je vyřešen a končíme s `yes`.

Dobře, ale co když zjistíme, že `anna` není přímým rodičem `kvetose`? Pak se musíme zamyslet nad tím, co znamená být předkem: `anna` je předkem `kvetose`, pokud má `anna` nejaké dítě `X`, které je předkem `kvetose`. Prolog najde nějaké dítě `anny`, třeba `pavla`. Pak vezme `pavla` a `kvetose` a zkoumá predikát `predek(pavel,kvetos)`. Opět najde třeba nějakého `cyrila`, který je dítětem `pavla` a měl by být předkem `kvetose`. Takto pokračuje dál a dál, až najde celý řetězec dětí a pradětí `anny`, začínající `pavel`, `cyril`, ... a poslední dítě je přímým rodičem `kvetose`.

Samozřejmě v každé generaci může mít Prolog na výběr spoustu dětí, ale on je všechny vyzkouší (to už víme, postupně unifikuje proměnné), a tak prohledá celý generační strom a najde cestu od `anny` ke `kvetosovi`. (Samozřejmě pouze pokud nějaká existuje.)

Kvíz

Vyzkoušejte si, co si vám utkvělo v paměti. Správné výsledky s vysvětlením jsou na *http://ksp.mff.cuni.cz/prolog/*.

⋆ Jakým písmenem může začínat proměnná
1. velkým písmenem
2. malým písmenem
3. podtržítkem

⋆ Označte řádek, na kterém je právě jedna klauzule
1. `pes(alik). pes(brok).`
2. `pes(hafistek).`
3. `savec(X) :- pes(X).`

⋆ Jaký je vztah pravidla a faktu
1. Fakt je pravidlo, které nemá žádnou pravou stranu.
2. Fakt je pravidlo, které vždy uspěje.
3. Mezi faktem a pravidlem není žádný vztah.

⋆ Jakého rodinného příslušníka hledá `prislusnik(X)`?

```
manzele(X,Y).
rodic(X,Y).
prislusnik(X) :- manzele(A,B), rodic(A,Y),
    rodic(B,Z), Y=Z, X=Y.
```

1. každého rodiče, který je v manželském svazku
2. všechny manželské děti
3. všechny manžele, kteří mají vnuka

⋆ Uspěje dotaz `pred(_)`, pokud mame program:

```
pred(a).
```

1. Uspěje.
2. Neuspěje.

Soutěžní úlohy

1. **Tchyně (2 body)** Napište predikát `tchyne(Tch,X)`. Vysvětlení pro ty, kdo neví, co je tchyně: Pokud je někdo ženatý/vdaná, tak tchyně je matka jeho/jejího partnera/partnerky.

2. **Oprava (3 body)** Popište, proč tento program nefunguje, a zkuste jej opravit, aby fungoval tak, jak nejspíš zamýšlel autor:

```
predek(Pred,Pot) :- predek(MlPred,Pot),
    rodic(Pred,MlPred).
predek(Rod,Pot) :- rodic(Rod,Pot).
```

3. **Evoluce (7 bodů)** Biologové vás požádali o řešení následujícího problému.

Existuje databáze rostlin, ve které jsou uloženy informace o tom, která rostlina se vyvinula z které. Máte tedy predikát (fakt) `mutace(X,Y)`, který popisuje, že rostlina `Y` se vyvinula z rostliny `X` mutací. Biologové vědí, že některé rostliny jsou nejpůvodnější, takže nemají žádného evolučního předka. Lze rozpoznat, které rostliny to jsou, takže máte k dispozici predikát `je_puvodni_druh(X)`, který uspěje, pokud byla rostlina `X` na začátku evoluce. Dále máte k dispozici predikát `je_odvozeny_druh(X)`, který uspěje, pokud je rostlina odvozená od nějaké původní (jinými slovy není původní). Je dokázáno, že každá rostlina se vyvinula mutací z právě jedné rostliny, tudíž neexistuje křížení. Každá rostlina tedy odvozuje svůj původ od jedné z evolučně původních rostlin. Biologa by zajímalo, jestli daná dvojice rostlin odvozuje svůj původ od stejné původní rostliny.

Napište predikát `stejny_druh(X,Y)`, který uspěje, pokud rostliny `X` a `Y` odvozují svůj původ od stejného druhu od počátku evoluce.

Posílejte i nefunkční a částečná řešení, bodové odměny budou i za ně!

Rozloučení

Děkujeme vám za pozornost a doufáme, že se na nás brzy obrátíte se svými dotazy. Nakonec se s vámi rozloučíme příkazem pro ukončení Prologu:

```
?-halt.
```

vybere si tedy `cecilku`. Jinými slovy unifikuje `X=cecilka`. Pak hledá, jestli je `cecilka` mužem, jenže zjistí, že `cecilka` není mužem, proto `cecilka` nebyla ta správná volba. A teď přichází kouzlo Prologu: Prolog *odunifikuje* `X=cecilka`, tedy uvolní proměnnou `X` a zkusí novou volbu, tedy znovu unifikuje `X=bonifac`, zkusí to s `bonifac`em a tentokrát uspěje, neboť `bonifac` je rodič i muž. V tomto okamžiku samozřejmě vypíše `X=bonifac`.

Když se tedy Prolog snaží splnit nějaký predikát a má nějakou volnou, nesvázanou proměnnou `X`, zkusí za ni „dosadit“, unifikovat nějakou hodnotu. Nejprve vybere tu, která je v programu na nejbližším řádku od začátku programu. Když má proměnnou `X` svázanou s nějakou hodnotou, zkusí splnit všechny predikáty, které mu přikazuje pravá strana pravidla, a používá přitom tuto zvolenou hodnotu `X`. Když se to podaří, skončí s úspěchem a vypíše tuto zvolenou hodnotu proměnné `X` jako správnou. Když Prolog někde narazí na nesplnitelný predikát (prostě `cecilka` holt není muž), musí se vrátit a proměnnou `X` odunifikovat. `X` je zase volná. Pokud máme na výběr ještě nějaké jiné hodnoty, zkusíme proměnnou `X` znovu zunifikovat s jinou hodnotou (další v pořadí v programu) a vyhodnotit predikáty znovu. Pokud takto vyčerpáme všechny možnosti, musíme bohužel skončit neúspěšně a vypsat `no`.

Poznámka: Vidíme, že neexistuje žádná jiná možnost, jak změnit hodnotu proměnné, než že se v průběhu vyhodnocování odunifikuje při návratu z neúspěšné větve a zunifikuje při vstupu do nové větve. Do jednou zunifikované proměnné neumíme už „přiřadit“ novou hodnotu.

Ve skutečnosti je tento popis dost nepřesný, Prolog neunifikuje zvlášť proměnné, ale celý predikát s hlavou klauzule. (Že to zní ďábelsky :–)

Vyhodnocení dotazu, predikáty

Stejným způsobem, jako Prolog postupně zkouší unifikovat volné proměnné, pracuje i s výběrem predikátů. Ukážeme si příklad. Máme program:

```
kocka(micka).
pes(alik).
zelva(matylda).
je_savec(X) :- kocka(X).
je_savec(X) :- pes(X).
```

Zeptejme se:

```
?-je_savec(alik).
```

Prolog nejprve zkusí klauzuli `je_savec(X) :- kocka(X)` s alíkem a samozřejmě zjistí, že alík není kočka. Ale nevzdá se tak snadno, protože má na výběr ještě jednu variantu predikátu `je_savec(X) :- pes(X)` a tam s alíkem uspěje.

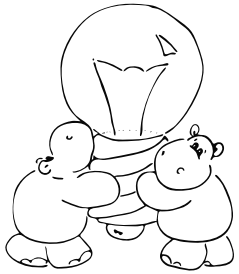
Pro Prolog predikáty se stejným názvem a stejným počtem argumentů jaksi „patří k sobě“ a postupně je vyzkouší všechny.

Pozor, predikát `je_savec(X,Y)` je jiný predikát, Prolog tedy rozlišuje predikáty stejného názvu a rozdílného počtu argumentů.

Poznámka: Místo rozepsání na dva řádky můžeme napsat také:

```
je_savec(X) :- kocka(X) ; pes(X).
```

Středník má tedy při splňování pravé strany pravidla význam *nebo*.



```
?-Kolik programátorů v Prologu
  je potřeba na výměnu žárovky?
  no.
```

Anonymní proměnná

Vzpomeňme si na predikát `je_otec`:

```
je_otec(X) :- rodic(X,Y), muz(X).
```

V predikátu je_otec nám šlo o to, zda je někdo otcem, ale nezajímalo nás, čí otec to je. Je nám popravdě úplně jedno, jak se ono dítko jmenuje, hlavně, že nějaké je. Proměnná `Y` je tady vlastně docela zbytečná a může v ní být cokoliv. To můžeme vyjádřit tzv. *anonymní proměnnou*:

```
je_otec(X) :- rodic(X,_), muz(X).
```

Anonymní proměnná se značí _ (podtržítkem). Pokud máme v klauzuli více anonymních proměnných, tak spolu nemají vůbec žádný vztah, i když jsou všechny značeny podtržítkem. Prostě je to hromada proměnných, z nichž každá může mít jakoukoli hodnotu a ta nás nezajímá.

Porovnávání

Zjistit, jestli se dvě proměnné rovnají, není v Prologu tak jednoduché. Záleží na tom, jestli už jsou proměnné zunifikované a co v nich vlastně je. Pro naše účely zatím stačí vědět, jak porovnáváme obsahy dvou již zunifikovaných proměnných, které obsahují atomy (například jména).

```
jsou_stejne(X,Y) :- X=Y.    % Prolog greenhorn
```

Predikát `jsou_stejne` uspěje, pokud je `X` rovno `Y` v tom smyslu, jak bychom čekali, tedy pokud atomy v nich jsou stejné. Namísto pravidla můžeme takovéto porovnání vlastně provést ještě jednodušeji:

```
jsou_stejne(X,X).           % Prolog guru
```

Tento predikát uspěje, pokud dostane dvě proměnné svázané se stejnými hodnotami atomů.

A co by se stalo, kdybychom do predikátu `jsou_stejne` pustili proměnnou `X` zunifikovanou třeba `X=pavel` a proměnnou `Y` zatím nezunifikovanou? Odpověď se nabízí – `Y` by se zunifikovala na `Y=pavel` a predikát by uspěl.

Rozsah platnosti proměnných

Platnost proměnné se omezuje na jednu klauzuli. Tedy proměnná `X` je platná (čili jedna a tatáž) v klauzuli

```
je_otec(X) :- rodic(X,_), muz(X).
```

Ale máme-li klauzule s predikáty

```
je_otec(X) :- ...
je_matka(X) :- ...
```

`X` v jedné a druhé klauzuli jsou různá.

Představme si, že bychom chtěli naprogramovat predikát `prarodic(X,Y)`, který by dokázal zjišťovat, kdo je čím prarodičem, resp. kdo je čím vnukem/vnučkou, a případně zda-

Svá řešení nám zasílejte do 16. října 2006 buďto elektronicky na *http://ksp.mff.cuni.cz/submit/*, nebo klasickou poštou na adresu:

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

Předmluva

Je trochu malý zázrak, že teď čtete tyto řádky. Řádky, na nichž vás očekává příběh o mně, detektivu Přesprstovi. A tak Ti ještě jednou děkuji, mé milé KSP. Nejen že jsi opravilo chyby v mém rukopisu, ale dokonce jsi bylo tak laskavo a jako jediné jsi mé dílko vydalo.

19-1-0 Vzorová úloha

Poznámka KSP: To si zaslouží vysvětlit. Rukopisy pro náš speciální tiskařský lis je třeba psát v následující formě:

```
name jméno_kapitoly
        text kapitoly
        name jméno_podkapitoly
                text kapitoly
        end
        text kapitoly
        name jméno_další_podkapitoly
                text kapitoly
                name jméno_vnořené_podkapitoly
                        text kapitoly
                end
        end
        text kapitoly
end
```

Každá kapitola tedy začíná řádkem, jehož prvním slovem je `name`, a končí řádkem obsahujícím jediné slovo `end`. V každé kapitole pak může být libovolný počet podkapitol s totožnou syntaxí. V čem je problém? Přesprst totiž občas nedodržoval tuto syntaxi a slova `name` a `end` psal víceméně nahodile. Buď začal novou kapitolu, aniž by ukončil předchozí, nebo ukončil kapitolu, kterou ani nezačal. A chudák mistr tiskař, který nevěděl, co si s rukopisem počít, ze samého zoufalství počal jíst barvu. Je jasné, že mistrů tiskařů zase nemáme tolik, a tak bychom rádi poznali, že rukopis je chybný, abychom ho mohli Přesprstovi vrátit.

19-1-0 Vzorové řešení vzorové úlohy – Jak na to?

Nejprve se zamyslíme, co se vlastně může pokazit. Můžeme neukončit existující kapitolu a ukončit neexistující kapitolu. Dále by se mohlo stát, že se nějaké kapitoly „překříží“, čili že ukončíme nějakou kapitolu dříve než její podkapitolu. Ale počkat – jednotlivé `end` nejsou pojmenované a každý patří k „nejbližšímu“ `name` nad ním, takže nemůžeme ukončit kapitolu předtím, než ukončíme její podkapitolu. Jména kapitol nejsou tím pádem vůbec důležitá, roli hraje pouze jejich vnořený počet.

Když jsme si to uvědomili, úlohu již vyřešíme jednoduše. Stačí nám si pamatovat, kolik kapitol zatím začalo a nebylo ještě ukončeno. Každý `name` na vstupu zvýší tento počet o jedna, každý `end` ho o jedna sníží (pokud může). Správný text pak poznáme tak, že každý `end` snižuje nenulový počet začatých kapitol a navíc je na konci tento počet nulový. Řešení v pseudopascalu může vypadat například takto:

```
počet_kapitol := 0;
while not konec_textu do begin
  if řádek_začíná_na_name then inc(počet_kapitol);
  if řádek_začíná_na_end then
    if počet_kapitol=0 then begin
      write('Ukončena neexistující kapitola!'); halt;
    end else dec(pocet_name);
end

if počet_kapitol<>0 then write('Neukončená kapitola!')
else write('Rukopis je v pořádku.');
```

Nyní určíme časovou a paměťovou náročnost našeho řešení. Řešení načítá řádky textu, každý právě jednou, a s každým řádkem provede jednoduchou operaci (zjistí, zda začíná slovem `name` či `end` a případně příslušně upraví proměnnou `počet_kapitol`) v čase úměrném délce tohoto řádku. Celkový čas našeho řešení je tedy lineární vzhledem ke vstupnímu textu, což je jistě asymptoticky nejlepší možné, protože v menším než lineárním čase bychom ani nedokázali načíst celý vstup. Paměti potřebujeme jenom konstatní množství (proměnná `počet_kapitol` a čtyři první nemezerové znaky načítaného řádku pro porovnání s `name` a `end`), což také můžeme těžko zlepšit. Naše řešení je tedy v určitém smyslu „nejlepší možné“. Hurá!

Kapitola 1

Můj příběh se odehrává v dobách, kdy dané slovo mělo větší váhu než tisíc smluv a přijít o čest bylo horší než přijít o život. A v mnoha ohledech i těžší. Být soukromým očkem v divočině mafiánských rodin bylo pořádně nebezpečnou hrou a člověk si musel dávat sakra, ale sakra pozor, aby nešlápl na něčí kuří oko, jestli mi rozumíte. Jenže občas stačí jen chvilka nepozornosti a…

Ten den jsem seděl v kanceláři. Stejně jako den předtím. A i den předtím. Vlastně už si ani nevzpomínám, kdy jsem měl trochu víc případů a hlavně teda peněz.

19-1-1 Zlaté časy — 8 bodů

Pojďme Přesprstovi pomoci zjistit, kdy byly jeho zlaté časy, aby si mohl zavzpomínat, a nahlédněme do jeho knihy příjmů. V té si pečlivě vedl své příjmy (kladná čísla) a výdaje (záporná čísla). No a zlaté časy je přirozeně takové souvislé období, kdy dosáhl nejvyššího součtu příjmů a výdajů.

Vstupem bude posloupnost příjmů a výdajů a na výstupu by měl váš program vypsat číslo záznamu, kdy zlaté časy začaly, a číslo záznamu, kdy zlaté časy skončily.

Příklad: Pro posloupnost $1, 6, -13, 12, 3, -2, 1, 5, 6, -4$ se Přesprstovi nejlépe vedlo mezi 4 a 9 (tehdy byl součet 25).

Ještě že mi zbylo alespoň na mou oblíbenou whisky, doutníky a čokoládu. Čokoládu… Kde jen ji mám. Začal jsem se přehrabovat v šuplíku. Konečně jsem ji našel a třesoucíma se rukama ji začal lámat.

19-1-2 Čokoláda — 6 bodů

Třesoucíma se rukama se ale láme špatně, a tak by Přesprsta zajímalo, kolikrát bude muset lámat. Představte si tabulku čokolády a dvě ruce. Ruce uchopí čokoládu a rozlomí ji (zlom je úsečka vedená zúženým místem mezi dílky) na dvě ne nutně stejně velké části. Poté ruce uchopí jednu z částí a opět ji rozlomí na dva kusy. A Přesprsta by zajímalo, kolikrát je třeba lámat v nejlepším a v nejhorším případě, aby získal jednotlivé dílečky tabulky čokolády.

Pomůžete mu? Na vstupu dostanete dvě čísla N a M, což jsou rozměry tabulky čokolády v dílcích. Na výstupu by měla být rovněž dvě čísla: minimální a maximální počet

zlomů nutných k dosažení cíle. A protože Přesprst už nikomu nedůvěřuje, měli byste mu i dokázat, že na menší a větší počet zlomů to nejde.

Příklad: Má-li Přesprst čokoládu o rozměrech 1×3, musí v každém případě lámat dvakrát, protože jedním zlomením jednotlivé dílky čokolády nedostane, a třikrát lámat nelze.

A je to. Vložil jsem hladově jeden dílek do úst. Pak jsem si nalil whisky, sedl si a čokoládu zapil. Jak mě whisky tak hladila po jazyce, hledal jsem východisko ze své situace a asi bych se prohledal až ke dnu lahve, kdyby někdo nezaklepal.
„Dále," promluvil jsem překvapeně ke dveřím.
„Dobrý den, mohu se posadit?"
Mlčky jsem kývl a zavřel pusu.

Nebudu vás otravovat popisem té dámy, která zabloudila do mé kanceláře, byla prostě kus.
„Co máte na srdci?" prohodil jsem nedbale a nenápadně jsem si ji prohlížel v místech, kde jsem srdce tušil.
To, co mi ta chudinka říkala, mě ani nepřekvapilo. Jenom jsem netušil, proč mi to všechno vykládá. Byla ženou jednoho z místních mafiánských kmotrů, Carla Assassina. Vyprávěla o tom, jak její manžel obchoduje s alkoholem a zbraněmi, pere špinavé a tiskne falešné peníze a pořádá večírky pro podsvětní smetánku.

19-1-3 Tiskárna — 10 bodů

Během vyprávění se Carlova žena rozpovídala o tom, jak se ony falešné peníze tiskly. Tiskárna měla vstup a výstup. Na vstup se daly bankovky a na výstupu se objevily bankovky původní a ještě jedna kopie každé z nich. Tedy přesně dvojnásobný počet. Když se všechny bankovky okopírovaly, dal se celý balík z výstupu zpět na vstup a na vrch se přidala jedna bankovka s novým sériovým číslem. Carlově ženě se podařilo získat všechny bankovky ze vstupu ještě před tím, než tiskárna začala kopírovat, a Přesprstovi je donesla. Přesprsta by teď zajímalo, která bankovka se přidala naposledy na vrch, tedy ta bankovka, která má v celém balíku unikátní sériové číslo. Bohužel se během cesty promíchalo pořadí bankovek.

Na vstupu dostane program seznam sériových čísel bankovek přinesený Carlovou ženou a na výstupu by měl program vypsat sériové číslo bankovky, která byla přidána jako poslední, tedy takové, která je v seznamu právě jednou, zatímco ostatní lze spárovat do dvojic. Sériové číslo je řetězec kratší než 100 znaků obsahující pouze číslice a velká písmena anglické abecedy.

Příklad: Pro vstup `9G873W`, `Z8D43`, `9G873W`, `9G873W`, `A456C`, `Z8D43` a `9G873W` by měl program napsat `A456C`.

Všechno trpělivě snášela a snažila se do jeho věcí nemíchat, ale to, že si našel jinou ženu, pro ni byla asi poslední kapka k tomu, aby se do jeho věcí míchat začala. Život je boj, pomyslel jsem si, a vstal jsem, abych ji utěšil, neboť vypadala, že se každou chvilkou zhroutí...

Kapitola 2

Nevím, co přesně se v tu chvíli stalo, vím jen, že se rozrazily dveře a já se probudil svázaný v temné místnosti. Nejprve jsem si jazykem přepočítal zuby a pak jsem si překontroloval všechny kosti. Zdálo se, že až na bolesti hlavy se mi nic nestalo. Začal jsem přemýšlet, komu jsem stál za únos. Ve městě byla spousta mafiánských rodin. Popravdě řečeno, nikdo nevěděl přesně kolik. A přitom bylo tak snadné je spočítat.

19-1-4 Mafiánské rodiny — 10 bodů

Policie si vedla o každém mafiánovi záznam o jeho přímých nadřízených a podřízených. Každý mafián má nejvýše jednoho přímého nadřízeného, pokud nadřízeného nemá, je to kmotr. Každá rodina má právě jednoho kmotra. No, zas tak snadné to nebylo, protože na popud rodiny Farmot byly spisy poničeny vandaly a nebylo možné vyčíst, jestli je záznam ve tvaru šéf – podřízený nebo naopak.

Na vstupu dostane váš program seznam policejních záznamů a na výstupu by měl program vypsat počet rodin ve městě.

Příklad: pro záznamy: $1-2, 2-1, 3-6, 4-6, 5-6, 6-3, 6-4, 6-5, 5-7, 7-5$ lze zjistit, že ve městě jsou dvě rodiny.

Nestihl jsem si ani v duchu přeříkat jména rodin, jimž bych se nechtěl znelíbit, když tu mě z uvažování vytrhly hlasy.
„Já bych navrhoval nejdžíve thání nechtů, pak vlasů, pak usekat pfsty a..."
„Neblbni má to přežít. Aspoň napoprvé."
„Aha, na to šem šapomněl, tak co tšeba vyřžazit žubi a žlámat nohi?"
„Ne, to už jsme dělali minule."
„Ale žubi byšme mohli, né? Mě to proště baví a š těma novejma kleštiškama, co nám pořídil šéfík, to jde jako po mášle."
„Ach jo. Si jak malej. Tak jo, ale jen tak, aby mohl mluvit."
„Hulá! Mušeníško, to je moje potěšeníško."
Snažil jsem se je moc nevnímat, což stejně moc nešlo, protože jsem se všemožně snažil uniknout ze svých pout.

19-1-5 Zámek — 13 bodů

Přesprst měl štěstí, únosci použili pouta s číselným zámkem, který on velmi dobře znal. Věděl, že tento typ zámku lze vždy odemknout jednou z kombinací, které splňují podmínku, že po nějaké číslici následuje pouze číslice z určité množiny. Dokonce si ty seznamy pro jednotlivé číslice dobře pamatoval a teď by ho zajímalo, kolik různých kombinací musí prozkoumat.

Na vstupu dostane program K, počet cifer čísla na zámku, a seznam číslic, které mohou následovat za jednotlivými číslicemi. Na výstup by měl program vypsat, kolik všech možných čísel je na zámku možno nastavit.

Příklad: Pro tříciferné číslo ($K = 3$) a seznam

cifra	*možné následující cifry*
1	1, 3
2	3
3	1

je počet různých kombinací 6 (111, 113, 131, 231, 311, 313).

Už jsem měl odzkoušeno několik desítek kombinací, když najednou hlasy utichly a otevřely se dveře...

To be continued...

19-1-6 Prolog — 12 bodů

Milí řešitelé,

v letošním seriálu se budeme zabývat poněkud zvláštním, ale velmi zajímavým programovacím jazykem *Prolog*. Většina programovacích jazyků, které znáte (Pascal, C) patří do skupiny tzv. *procedurálních jazyků*. Programátor píšící kód v procedurálním jazyce přesně popíše, jakým způsobem se má daná úloha vyřešit. V Prologu budeme programovat jinak, logicky. Nejprve nějakým způsobem popíšeme nám známý svět a poté se Prologu zeptáme na řešení daného problému. Nepřikazujeme tedy, jak se má Prolog dobrat výsledku, pouze říkáme, co chceme vyřešit, ale ne přesně, jakým způsobem. Zapomeňme tedy na chvíli na klasické proměnné coby „škatulky", na přiřazovací příkaz a na alokování paměti. Programujeme logicky – *PROgramming in LOGic*.

Jak na Prolog

Aby se vám s Prologem seznamovalo co nejlépe a nejradostněji, připravili jsme pro vás kromě klasického „papírového" seriálu také internetovou prologovskou poradnu. Na adrese *http://ksp.mff.cuni.cz/prolog/* najdete fórum, kam můžete posílat své otázky a my vám s vaším problémem poradíme. Nestyďte se ptát, rádi vám pomůžeme. Kromě poradny najdete na uvedené adrese také zadání úloh, učební texty, užitečné odkazy a také on-line interpreter Prologu, kde si můžete zkoušet své prográmky.

Poznámka: Tento symbol označuje obtížnou část, která ale není nutná pro pochopení dalšího textu.

Instalace Prologu

Rady k instalaci a odkazy na Prolog jak pro Windows, tak pro Linux najdete také na *http://ksp.mff.cuni.cz/prolog/*.

Program v Prologu, predikát, fakt, klauzule, proměnná

Program v Prologu je možné napsat buď přímo v prostředí Prologu, pokud to vaše prostředí umožňuje, nebo v libovolném textovém editoru (vi, emacs, ...). Ukážeme si příklad jednoduchého prográmku.

```
muz(antoch).
muz(bonifac).
zena(cecilka).
rodic(cecilka,antoch).
rodic(bonifac,antoch).
manzele(bonifac,cecilka).
```

Prolog popisuje situaci pomocí *predikátů*. Predikát `muz(X)` říká „`X` je muž". V našem prográmku máme čtyři predikáty: *unární* predikáty `muz(X)` a `zena(X)` a *binární* predikáty `manzele(X,Y)` a `rodic(X,Y)`.

Proměnné se v Prologu značí velkým písmenem na začátku, např. `X`, `Y`, zatímco konkrétní hodnoty – *atomy* (můžete si je představovat trošku jako stringy) začínají malým písmenem, např. `antoch`, `bonifac`.

Takže jakmile jsme v programu použili predikát `muz` a „dosadili" do něj `antoch`, dali jsme světu najevo, že `antoch` je muž.

Základní jednotkou prologovského programu je *klauzule*. Klauzule vždy končí *tečkou*. V našem programu jsme zatím použili nejjednodušší typ klauzulí, *fakta*.

Jakmile jsme napsali náš první program v Prologu, můžeme si jej pustit v prologovském prostředí. Prolog je interaktivní jazyk, takže po spuštění se objeví výzva:

```
?-
```

Nejprve musíme Prologu sdělit, že si přejeme pracovat s naším programem. To uděláme pomocí

```
?-['nasprogram.pl'].
```

Teď začneme konečně náš program využívat. Zeptejme se, jestli je `antoch` muž.

```
?-muz(antoch).
```

Co nyní Prolog udělá? Projde námi dodaný program a podívá se, jestli v něm existuje predikát `muz(antoch)`. A odpoví nám:

```
?-muz(antoch).
    yes.
?-
```

Poznámka: `muz(antoch)` jsme napsali my. Nesmíme zapomenout na tečku na konci. Pak stiskneme Enter a výsledné `yes` napsal zase Prolog, pochopitelně. Nakonec se znovu vypíše otazník a čeká se na další dotaz. Kdyby Prolog nenašel v programu `muz(antoch)`, odpověděl by `no`.

Můžeme se ale zeptat jinak:

```
?-muz(X).              % kdo je muž?
    X=antoch.          % antoch je muž
```

Prolog opět projde celý program a pokusí se za `X` „dosadit", správně říkáme *unifikovat*, někoho, kdo je muž, a nabídne nám `antocha`. Kdyby žádného muže nenašel, odpoví nám `no`.

Pokud jsme s `antoch`em spokojení, dáme Enter a program odpoví `yes` a opět otazníkem čeká na další dotaz. Může se ale stát, že nechceme `antocha`, nýbrž `bonifac`e. Pak můžeme tuto odpověď odmítnout a vyzvat Prolog, aby našel jiného muže tím, že zmáčkneme středník:

```
?-muz(X).              % kdo je muž?
    X=antoch.    ;     % chceme dalšího muže
    X=bonifac.   ;     % ještě dalšího muže
    no.                % už žádný není
```

Jistě víte, co dělá dotaz

```
?-rodic(bonifac,X).
```

Vidíme, že dotazy činíme pomocí proměnných. Na začátku je proměnná `X` *volná*, tedy nevíme, kdo je dítě `bonifac`e. Proměnná `X` zatím *není svázaná*. Prolog zjistí, že dítětem `bonifac`e je `antoch` a *sváže* neboli unifikuje proměnnou `X` s `antoch`em.

```
    X=antoch.
```

Pravidla

Zatím by se mohlo zdát, že Prolog je jen šikovná databáze. Bylo by opravdu slabé, kdybychom k dispozici měli jen klauzule typu fakta a mohli se na ně jen ptát. Existují tedy ještě klauzule typu *pravidla*. Přidejme do programu řádku:

```
je_otec(X) :- rodic(X,Y), muz(X).
```

Tohle pravidlo říká „`X` je otec, pokud platí predikáty, že `X` je něčí rodič (vyskytuje se v predikátu rodič) a ještě k tomu `X` je muž". *Čárka* mezi predikáty má význam *a zároveň*. Tedy aby se splnilo, že je někdo otcem, musím splnit oba predikáty na pravé straně.

Poznámka: Je jasné, že za predikáty na pravé straně se mohou skrývat další pravidla.

Vyhodnocení dotazu, unifikace volných proměnných

Jak tedy Prolog vyhodnotí dotaz:

```
?-je_otec(cecilka).
```

Nejprve zjistí, že `cecilka` je skutečně rodičem (rodičem `antocha`, ale to nás moc nezajímá), a pak se snaží splnit také predikát, že `cecilka` je muž, což se mu nepovede, a proto skončí s `no`. Měli jsme splnit oba dva predikáty a to se nám nepovedlo.

A co dotaz:

```
?-je_otec(X).
```

To už je trošku zapeklitější. Prolog ví, že musí splnit nejprve predikát `rodic`, takže se podívá, kdo je rodičem. Nejprve mu „padne do oka" predikát `rodic(cecilka,antoch)`,