PRASK
I. ročník, 2014/15
Katedra základov a vyučovania informatiky FMFI UK,
Mlynská Dolina, 842 48 Bratislava

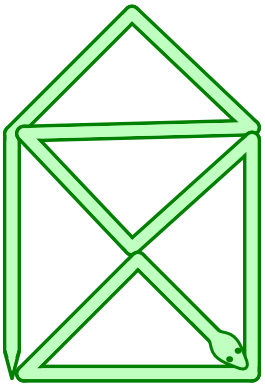
Vzorové riešenia 1. kola letnej časti

1. Pytón s umeleckými sklonmi

vzorák napísal Jano
(max. 15 b za riešenie)

Podúloha a)

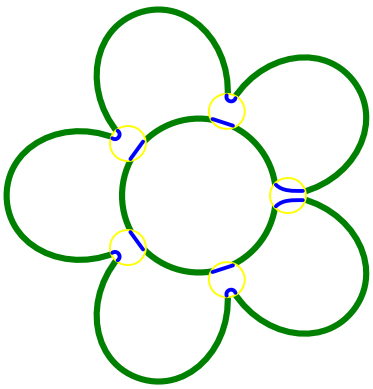
Existuje niekoľko správnych riešení tejto úlohy, jedno z nich vyzerá napríklad takto:

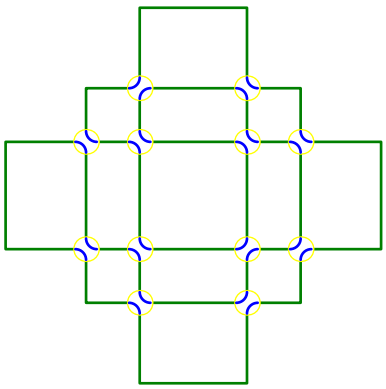


Stačilo si uvedomiť, že had môže zahnúť aj v strede domčeka a potom sa to už riešilo samo.

Podúloha b)

Prvé dva obrázky sa dali nakresliť a to nasledovne:





Druhý obrázok je pre lepšiu zrozumiteľnosť nakreslený hranatý. Na nájdenie riešenia sa stačilo trochu pohrať s obrázkami.

Tretí obrázok sa nielenže nedal nakresliť tak, aby sa had neprekrýval, ale dokonca sa ani nedal nakresliť jednou čiarou.

Čo sa deje, ak obrázok kreslíme jedným ťahom? Zoberme si takúto čiaru a prejdime po nej prstom. Vždy, keď prstom vojdeme do nejakej križovatky (krúžku), poznačíme si ku križovatke modrú bodku a vždy keď vychádzame z krúžku, poznačíme si červenú bodku. Pokiaľ je čiara uzavretá (had sa zakusol do svojho chvosta), tak vždy, keď do nejakej križovatky vojdeme, tak z nej aj musíme výjsť. S výnimkou krúžku, kde začíname. Tam najskôr výjdeme von, ale na konci sa tam zase vrátime. Tým pádom, pri každej križovatke máme rovnako veľa modrých a červených bodiek. Celkový počet bodiek pri každej križovatke je preto párny.

Uvedomme si, že pri každej križovatke máme po nakreslení cesty pre hada presne toľko bodiek, koľko z nej trčí čiar. Preto na to, aby sme vedeli nakresliť trasu hada, z každej križovatky musí trčať párny počet čiar. Na našom obrázku sú 4 križovatky, z ktorých trčia 3 čiary, a preto sa obrázok nedá nakresliť jednou uzavretou čiarou.

Ešte dodám, že v prípade, že by čiara nebola uzavretá, teda by mohla začínať na inom mieste ako končiť, tak by sme mohli mať **dve križovatky**, z ktorých by trčal nepárny počet čiar (keď v križovatke začíname, tak máme o 1 červenú bodku viac, keď v križovatke končíme, tak je tam o 1 modrú bodku viac). Náš obrázok má 4 križovatky s nepárnym počtom čiar, takže sa nedá nakresliť ani neuzavretou čiarou.

Podúloha c)

Chceme ukázať, že keď dostaneme hocijaký obrázok nakreslený jednou čiarou, tak ho vieme prekresliť tak, aby sa čiara nikde nepretínala.

Ukážeme si dve rôzne riešenia tejto úlohy. Prvé je jednoduchšie a predpokladá, že v každej križovatke sa stretávajú práve štyri čiary. V druhom riešení môžu do križovatiek vchádzať akékoľvek množstvo čiar, ale toto riešenie je trocha komplikovanejšie.

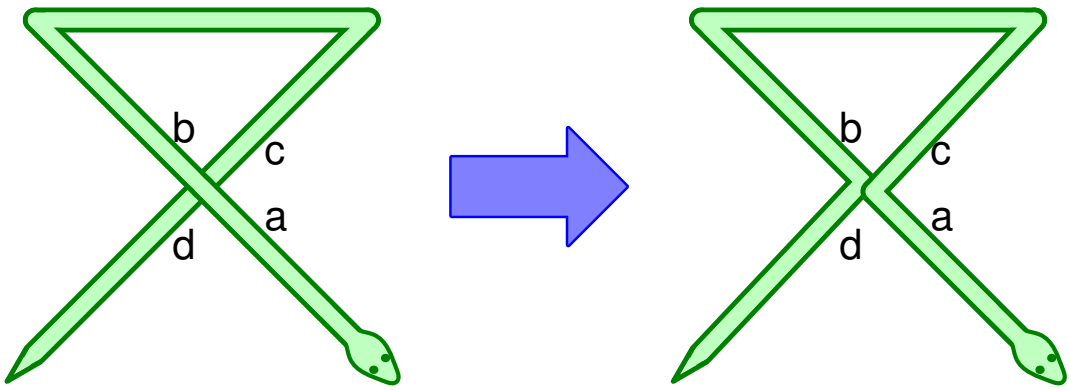
Prvé riešenie

Na začiatku máme obrázok nakreslený jednou čiarou. Pokiaľ sa táto čiara nikde nepretína, tak máme to čo sme chceli a môžeme skončiť. Pokiaľ má čiara nejaký priesečník (križovatku kde sa had kríži cez seba), môžeme ho odstrániť nasledovným spôsobom:

Pozrieme sa na celú čiaru ako na hada a prejdime prstom od hlavy až ku chvostu. Čiaru, ktorou sme prvýkrat vošli do priesečníka označíme a, čiarou ktorou sme následne vyšli von označíme b, čiaru ktorou sme druhýkrat vošli do tejto križovatky označíme c a čiaru, ktorou sme nakoniec vyšli von označíme d. (Keďže má križovatka stupeň 4, musíme dvakrát vojsť a dvakrát vyjsť von).

Jediné, čo potrebujeme spraviť je hada prekresliť tak, aby nešiel z a do b a z c do d, ale namiesto toho z a do c a z b do d. Takto dostaneme rovnaký obrázok, ktorý bude naďalej tvorený jednou čiarou, ale bude mať o jeden priesečník menej.

Prekreslenie jedného priesečníka je znázornené na obrázku.

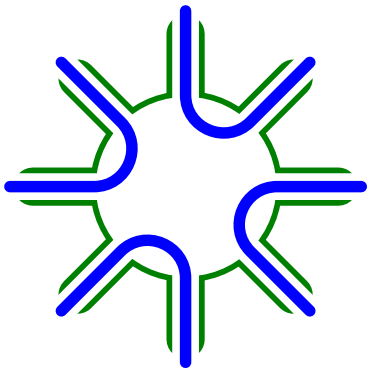


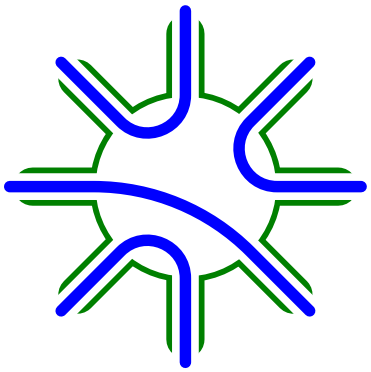
Pokiaľa má celá čiara viacero priesečníkov, jednoducho ich postupne po jednom odstránime práve predvedeným spôsobom. Napokon dostaneme jednu nepretínajúcu sa čiaru.

Druhé riešenie

Obrázok vyrobíme v dvoch krokoch.

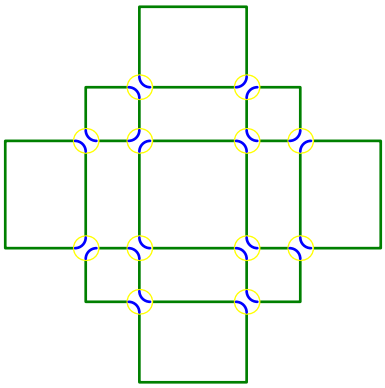
V prvom kroku nakreslíme obrázok tak, aby sa nepretínal, ale dovolíme, aby bol tvorený viacerými čiarami (čiže môžeme mať viacero hadov, ktoré dokopy tvoria daný obrázok). Všetky čiary budú cyklické – nebudú mať začiatok ani koniec (ako keby sme nakreslili viac hadov, ktorí si zahryzli do chvosta). Takýto obrázok vytvoríme veľmi jednoducho. V každej križovatke sa pozrieme na všetky čiary, ktoré z križovatky trčia a tieto trčiace čiary pospájame ľubovoľným spôsobom, tak aby sa nepretínali. Napríklad môžeme zoradiť čiary do kruhu v smere hodinových ručičiek a spojiť prvú s druhou, tretiu so štvrtou atď.





Na obrázku môžeme vidieť zväčšenú križovatku, z ktorej trčí osem čiar. Modrou farbou je znázornené, ktoré trčiace čiary sú spojené. Vyobrazené sú dve možnosti, ako môže byť celá križovatka pospájaná bez toho, aby sa čiary pretínali. Samozrejme existujú aj ďalšie možnosti. V prvej fáze kreslenia je jedno, akú možnosť si vyberieme, zvolíme si ľubovoľnú.

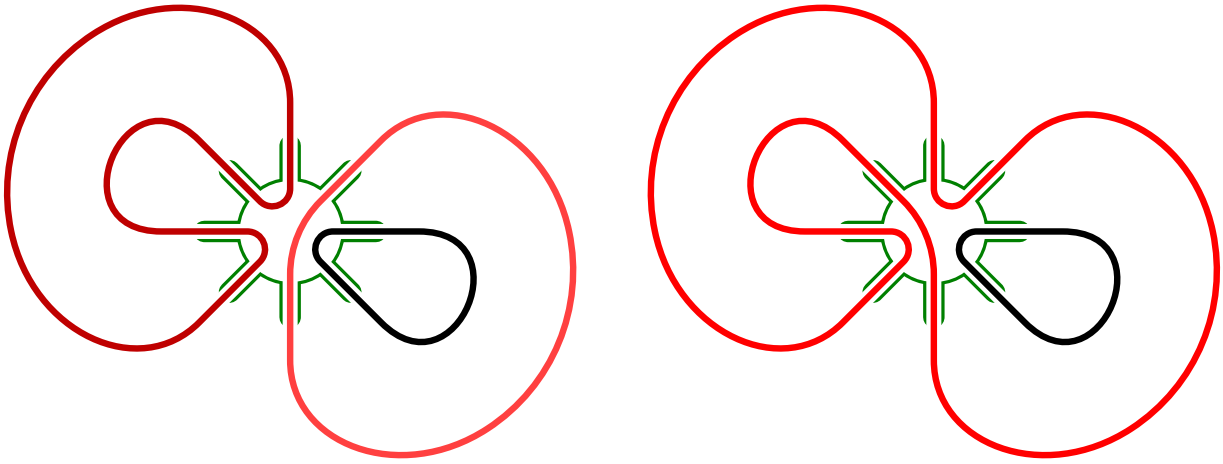
Takto dostaneme niekoľko nepretínajúcich sa čiar, ktoré dokopy tvoria celý obrázok. Pre tretí príklad z podúlohy b) by to mohlo vyzerať napríklad tak, ako na nasledujúcom obrázku:



V druhom kroku celého postupu sa budeme snažiť z veľa cyklických čiar spraviť jednu. Tieto cyklické čiary budeme volať cykly. Ak už náhodou je celý obrázok tvorený len jedným cyklom, sme hotoví a máme riešenie úlohy.

V opačnom prípade zoberieme ľubovoľné dva cykly, ktoré sa *dotýkajú*. (V našich obrázkoch kreslíme tak, že sa len takmer dotýkajú, pre lepšiu prehľadnosť). Cykly sa *dotýkajú* vtedy, keď majú nejakú spoločnú križovatku a zároveň neexistuje žiaden iný cyklus, ktorý by ich oddeľoval (t.j. vieme sa dostať od jedného cyklu k druhému bez toho aby sme museli prejsť cez nejaký iný cyklus). Keď z každej križovatky trčia 4 čiary, tak sa všetky cykly, ktoré majú spoločnú križovatku dotýkajú. Úloha je v takomto prípade trocha jednoduchšia (potom nám stačí riešiť predošlú, jednoduchšiu úlohu).

Dobre, tak teda zoberieme nejaké dva cykly, ktoré sa dotýkajú (vždy také musia existovať, zamyslite sa prečo) a v ich spoločnej križovatke ich prepojíme. Na obrázku je znázornené, ako by sme mohli prepojiť dve červené čiary, ktoré majú spoločnú zelenú križovatku.



Križovatky, z ktorých trčia 4 čiary sa prepájajú veľmi jednoducho, keďže tam sú len dva spôsoby ako môže križovatka vyzerať, jednoducho zmeníme jeden spôsob za druhý. Pre väčšie križovatky sa dá tiež ľahko vymyslieť spôsob ako ich prepojiť.

Vieme teda zobrať dotýkajúce sa cykly a spojiť ich do jedného (bez toho aby sa ostatné cykly zmenili), čím zmenšíme počet cyklov o jedna. Po niekoľkých takýchto úpravách nám teda zostane len jediný cyklus, ktorý je riešením našej úlohy.

To čo sme si práve predviedli je postupnosť krokov (algoritmus), ktorý nájde pre ľubovoľný obrázok nakresliteľný jednou čiarou taký nákres, ktorý sa nikde nepretína.

Pozrime sa ešte na to, čo sa nám podarilo dokázať. V časti b) sme ukázali, že obrázok sa dá nakresliť jednou nepretínajúcou sa čiarou, iba ak z každej križovatky vychádza párny počet čiar. V časti c) sme si potom ukázali postup, ako sa dá ľubovoľný takýto obrázok nakresliť. Uvedomme si, že tieto dva výsledky nám popisujú všetky obrázky, ktoré sa dajú nakresliť jednou čiarou – stačí ak z každej križovatky vedie párny počet čiar. Je to teda nutná a zároveň postačujúca podmienka.

2. Prskavkový problém

vzorák napísal Syseľ
(max. 15 b za riešenie)

Podúloha a)

Vieme, že Miško vyhrá. Ako teda bude vyzerať posledný ťah? Janko zoberie posledné prskavky z krabice. Spraví to však iba v prípade, že nebude mať na výber, čiže vtedy, ak v krabici zostane už len jedna prskavka. Ťah pred ním mohol mať Miško v krabici 2, 3, 4 alebo dokonca aj 5 prskaviek. Z tohoto počtu totiž vždy dokázal zobrať toľko, aby Jankovi zostala posledná prskavka. Janko vedel, že ak Miškovi v krabici zostanú 2, 3, 4 alebo 5 prskaviek, prehrá. Takže ak by mohol, snažil by sa týmto štyrom situáciam vyhnúť. Lenže on sa im nevyhol. To znamená, že keď sa dostal na ťah, musel mať v krabici práve 6 prskaviek. Pri takom počte prskaviek totiž nemá na výber. Nech spraví ľubovoľný ťah, Miško sa ocitne v situácii, z ktorej vie vyhrať.

Presnejšie, môžu nastať tieto štyri prípady:

- Janko zoberie 1 prskavku. Potom Miško zoberie 4, Jankovi zostane posledná a prehrá.
- Janko zoberie 2 prskavky. Potom Miško zoberie 3, Jankovi zostane posledná a prehrá.
- Janko zoberie 3 prskavky. Potom Miško zoberie 2, Jankovi zostane posledná a prehrá.
- Janko zoberie 4 prskavky. Potom Miško zoberie 1, Jankovi zostane posledná a prehrá.

Môžeme si všimnúť, čo majú tieto ťahy spoločné. V každom prípade zoberú chlapci dokopy 5 prskaviek. A to je základom Miškovej stratégie. Nech Janko urobí čokoľvek, Miško vie jeho ťah doplniť tak, aby spolu zobrali 5 prskaviek. Ak by napríklad pred Jankovým ťahom bolo v krabici 16 prskaviek, po ťahu oboch chlapcov by tam Miško nechal 11, potom 6 a nakoniec 1. Nech by Janko spravil akýkoľvek ťah, Miško by svoj cieľ vždy dosiahol. A v krabici by Jankovi zostala posledná prskavka.

Teraz už určite vieme, akým spôsobom Miško vyhrá. V prvom ťahu musí zobrať jednu prskavku a nechať Jankovi v krabici 41 prskaviek. A potom zakaždým doplniť Jankov ťah tak, aby dokopy zobrali práve 5 prskaviek. Takto bude počet prskaviek v krabici na začiatku Jankovho ťahu postupne 41, 36, 31, 26, 21, 16, 11, 6 a nakoniec 1.

Uvedomme si, že to, čo sme zistili, platí aj všeobecne. Nech si zoberieme krabicu s ľubovoľným počtom prskaviek, vieme zistiť, ktorý hráč vyhrá, ak bude môcť brať iba 1, 2, 3 alebo 4 prskavky. Môžeme si všimnúť, že Miško sa snažil dostať Janka do situácie, v ktorej počet prskaviek dával po delení číslom 5 zvyšok 1. Miško sa preto bude vo svojom úplne prvom ťahu snažiť znížiť počet prskaviek v krabici tak, aby táto vlastnosť platila. Potom totiž môže dopĺňať Jankov ťah na 5 prskaviek a vyhrať. Jediný prípad, kedy sa mu to nepodarí spraviť je, ak je v krabici na začiatku $5k + 1$ prskaviek pre ľubovoľné k. V tom prípade je vo výhode Janko, lebo môže použiť Miškovu stratégiu proti Miškovi a dopĺňať každý Miškov ťah tak, aby dokopy zobrali 5 prskaviek. Takýmto postupom bude Miško ten, komu zostane posledná prskavka a Janko vyhrá.

Podúloha b)

V predchádzajúcej časti sme postupovali od konca a zisťovali, ktorý z chlapcov vyhrá. V tejto časti nás k podobnému postupu zadanie dokonca navádza (chlapci otvárali krabice postupne). Poďme sa teda na to pozrieť lepšie.

Nebudeme však zisťovať, či vyhrá Miško alebo Janko, ale či hráč, ktorý je na ťahu vyhrá alebo prehrá. Inými slovami, keď si zoberieme krabicu, chceme zistiť, či pre hráča, ktorý spraví prvý ťah, existuje víťazná stratégia alebo prehrá bez ohľadu na to čo spraví.

- Ak máme jednu prskavku, hráč na ťahu ju musí zobrať a prehrať.
- Ak máme dve prskavky, hráč na ťahu zoberie jednu a druhú nechá protihráčovi, ktorý prehrá.
- Pri troch prskavkách vie prvý hráč buď zobrať všetky alebo zobrať jednu, nechať protihráčovi druhú a sám potom zobrať tretiu. V oboch prípadoch prehrá.
- Pri štyroch prskavkách môže hráč na ťahu zobrať jednu a nechá súpera v situácii troch prskaviek, v ktorej, ako sme už zistili, protihráč nemá šancu. Začínajúci hráč teda vyhrá.
- Pri piatich prskavkách môže hráč na ťahu zobrať štyri a nechať súperovi jednu.
- Pri šiestich prskavkách vieme zobrať tri a znova nechať prehrávajúce tri súperovi.
- Podobne to je aj pri siedmich prskavkách, ak zoberieme štyri.
- Pri ôsmich prskavkách môžeme zobrať päť a znova nechať súperovi tri.
- Pri deviatich prskavkách však nemáme šancu. Nech zoberieme ľubovoľný povolený počet z nich, dostaneme súpera do situácie, z ktorej sa mu (podľa toho, čo sme už zistili), podarí vyhrať.

Čo vieme zistiť z týchto úvah? Vidíme, že pri nejakom počte prskaviek v krabici (1, 3 a 9) hráč, ktorý je na ťahu nemá šancu vyhrať, nech spraví čokoľvek. A pri zvyšných krabiciach existuje pre začínajúceho hráča ťah, ktorý mu zaručí výhru.

Uvedomme si ale jednu dôležitú vec. Ak sme začali napríklad s krabicou, ktorá obsahovala 19 prskaviek, potom obaja hráči odobrali nejaké prskavky a Miško je zrazu v situácii, že je na ťahu a v krabici zostáva 9 prskaviek. V tomto prípade teda určite prehrá. Je jedno, čo sa stalo predtým a aké ťahy chlapci robili. Jediné, na čom záleží je, koľko prskaviek je v krabici v tomto momente.

Ak teda chceme vyrátať, kto vyhrá, ak bude v krabici 10 prskaviek, vieme to zistiť pomocou predchádzajúcich menších riešení (ktoré máme hore spísané). Postupne sa pozrieme, aké ťahy vieme spraviť, ak začíname s 10 prskavkami. Ak nás aspoň jeden ťah vedie do situácie, kde je zaručené, že hráč na ťahu prehrá, chceme ho spraviť. Lebo na ťahu bude náš súper a teda prehrá. Ak však všetky ťahy vedú do situácií, kde hráč na ťahu vyhrá, znamená to, že prehráme my bez ohľadu na to, čo spravíme.

Všetky situácie môžeme označiť ako vyhrávajúce (V) a prehrávajúce (P). Pomocou týchto dvoch pravidiel a znalosti všetkých situácií s menším počtom prskaviek sme schopní vyplniť nasledujúcu tabuľku:

Počet prskaviek	1	2	3	4	5	6	7	8	9	10
Prvý hráč	P	V	P	V	V	V	V	V	P	V

Počet prskaviek	11	12	13	14	15	16	17	18	19	20
Prvý hráč	P	V	V	V	V	V	P	V	P	V

Keďže prvý hráč je vždy Miško, vieme pre každý počet prskaviek povedať, kto vyhrá.

Opäť sa tento postup dá uplatniť všeobecne a pri rôznych hrách môžeme zaviesť prehrávajúce a vyhrávajúce pozície (samozrejme, od konkrétnej hry závisí, ako presne budú vyzerať). Obe pravidlá však zostávajú platné

tak, ako sú:

- Pozícia je vyhrávajúca, ak existuje ťah, ktorý nás dostane do prehrávajúcej pozície.
- Pozícia je prehrávajúca, ak všetky ťahy vedú do vyhrávajúcich pozícií.

Podúloha c)

2015 prskaviek je trochu veľa, skúsme sa však pozrieť aspoň na niektoré menšie počty prskaviek v krabici za rovnakých podmienok ťahania. Podobne ako v predchádzajúcej časti, aj tu si vieme vytvoriť tabuľku:

Počet prskaviek	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Prvý hráč	P	V	P	V	V	V	V	P	V	P	V	V	V	V

Môžeme si všimnúť, že tabuľka začína vzorom $PVPVVVV$, ktorý sa po 7 prskavkách opakuje, pričom vzor vždy končí násobkom sedmičky. Keďže platí $2016 = 7 \cdot 288$, tabulka pre veľké počty prskaviek bude vyzerať takto:

Počet prskaviek	2010	2011	2012	2013	2014	2015	2016
Prvý hráč	P	V	P	V	V	V	V

Víťazom bude opäť Miško.

Tento postup si však vyžaduje predpoklad, že vzor $PVPVVVV$ sa opakuje naprieč všetkými možnými veľkosťami krabíc. Môžeme si však uvedomiť, že keď zisťujeme výsledok pre nejaký počet prskaviek, pozeráme sa len na počty o jedno, tri alebo štyri menšie. To znamená, že výsledok závisí od najviac štyroch predchádzajúcich výsledkov. Ak sa teda zopakujú prvé štyri výsledky $PVPV$, tak sa náš vzor začne opakovať automaticky. A tieto prvé štyri výsledky sa naozaj zopakujú pri počtoch 8 až 11.

3. Príjemné nič

vzorák napísal Mário
(max. 15 b za riešenie)

Čo je to webová stránka[1]?

Všetci vieme, ako taká stránka vyzerá. Je na nej text, obrázky, okienka alebo vôbec nič. Ale čo to je?

Dáta. Všetko, čo sa nachádza v počítači alebo na internete sú dáta – súbory. Obrázky, videá, hudba, programy sú všetky uložené v počítači ako postupnosti núl a jednotiek a tieto postupnosti nazývame súbory. Rovnako aj webová stránka je len súbor – textový dokument, podľa ktorého vie webový prehliadač/browser čo, kde a ako má zobraziť. Obsah tohto súboru voláme **zdrojový kód** stránky, lebo v jeho texte je zakódované všetko čo môžeme vidieť na stránke.

Zdrojový kód býva najčastejšie napísaný v **jazyku HTML** (Hypertext markup language), ktorý používa špeciálne značky na formátovanie textu, tabuliek, zoznamov atď. Napríklad všetko, čo sa napíše medzi značky/tagy `<b>` a `</b>` sa v prehliadači zobrazí **hrubým/bold** fontom. Tag `<img src=„obrazok.png" />` vloží na stránku obrázok, ktorý je v jednom priečinku so zdrojovým súborom a volá sa `obrazok.png`.

Takýto textový dokument viete vytvoriť aj vy, napríklad písaním do notepadu. Stačí, keď ho uložíte s príponou `.html` a potom ho viete otvoriť v prehliadači.

Keď do prehliadača naťukáme webovú adresu alebo keď klikneme na hypertextový odkaz, počítač pošle prosbu serveru[2]. Server potom vášmu počítaču pošle súbory, o ktoré ste požiadali – zdrojový kód stránky a ďalšie potrebné súbory ako napr. obrázky, súbory popisujúce grafický výzor stránky, miniprogramy/skripty, čo sa na stránke spúšťajú...

Tieto súbory sa dočasne uložia v pracovnej pamäti vášho počítača a stránka sa zobrazí v prehliadači.

Nájdenie neviditeľného

Prehliadač na stránke ksp.sk/~prask/specialne/1/1/3/ nič nezobrazuje. Nedá sa tu na nič klikať ani nič označovať. Vieme však, že webová stránka je akýsi textový dokument so zdrojovým kódom, a preto sa pozrieme do tohto dokumentu.

[1] Jeden web/website (napr. prask.ksp.sk, ufo.fks.sk) obsahuje viacero webstránok/webpage (napr. prask.ksp.sk/ulohy/, prask.ksp.sk/vysledky/).

[2] Server je program, ktorý beží na ľubovoľnom počítači a odpovedá na rôzne prosby (napr. na HTTP requesty).

V menu alebo v popup menu (pri kliknutí pravým tlačítkom myši na stránke) vášho prehliadača nájdite možnosť **zobraziť zdrojový kód/view source code/view page source**. V Google Chrome a Mozille Firefox môžete použiť skratku `Ctrl+U`. Ak ste bezradní, vyhľadajte si na internete spôsob pre zobrazenie zdrojového kódu – určite to vie aj váš prehliadač.

Čítanie neviditeľného

Ak sme úspešne našli zdrojový kód stránky, mali by sme sa poriadne pozrieť, čo sa v ňom nachádza. Niečo tam predsa musí byť. A veruže je. A oveľa viac akoby ste čakali. Na úplnom vrchu vidíme tag <`title`>, vnútri ktorého je napísané *Nic*. Asi nie je ťažké si tipnúť, že to bude zodpovedať textu, ktorý vidíme ako názov stránky v priehliadači.

Nasleduje tag <`style`>. Na ten sa pozrieme neskôr, zatiaľ len môžeme usúdiť, že mení *štýl* stránky, nech to už znamená čokoľvek.

Ďalší zo zaujímavých tagov je <`table`>, ktorý bude zodpovedať nejakej tabuľke, ktorú síce nevidíme, ale je niekde na čiernej stránke skrytá. No a potom nasleduje veľké množstvo už spomínaných tagov, ktoré zobrazujú rôzne obrázky. Pri každom obrázku je naviac napísaná presná adresa toho, kde sa obrázok nachádza, takže si môžeme tieto obrázky pozerať tak, že ich adresu nakopírujeme do prehliadača, kde sa nám zobrazia samostatne. A s prekvapením zisťujeme, že obrázky sú vlastne nakreslené písmená.

Kopírovanie adries je však pomalé, mohli ste si preto všimnúť, že adresa každého obrázka písmenka sa začína predponou `http://people.ksp.sk/~prask/specialne/1/1/3/obr/`. Ak si otvoríte tento odkaz v prehliadači, zobrazí sa vám obsah celého priečinku s obrázkami. Ďalej stačí klikať, čo je oveľa jednoduchšie.

Ani tento postup však nie je dostatočne rýchly, predsa len je tých obrázkov 95. Po zobrazení prvého písmenka ale viete, že text je napísaný čiernou farbou. Otázka je, prečo ho teda nevidíme na stránke. Lebo na našej stránke je čierne všetko, aj pozadie. A čierny obrázok na čiernom pozadí nevidieť.

Chceli by sme preto zmeniť pozadie stránky na inú farbu, aby sme videli, čo ukrýva. A v tomto prípade sa dostávame späť k tagu <`style`>. Keď si pozrieme, čo je vo vnútri, väčšina vecí nedáva zmysel, ale nachádza sa tam jedna veľmi podozrivá časť `background-color: #000000`, čo je po preložení do slovenčiny `farba pozadia`. A nie je ťažké zistiť, že `#000000` je špeciálny kód pre čiernu farbu.

Ako teda zmeniť túto farbu, napríklad na bielu? Jedna možnosť je skopírovať si zdrojový kód do počítača[3] a upraviť vlastnosť `background-color`[4]. Buď zistíte, ako sa kóduje biela farba (je to `#FFFFFF`) a nahradíte ňou tú čiernu alebo stačí, ak zmažete celé `background-color:#000000;`, lebo automatická farba pozadia je biela.

V Google Chrom-e a aj v iných prehliadačoch je navyše možnosť použiť nástroje pre vývojárov/developer tools. Buď sa k nim preklikáme alebo použijeme skratku `Ctrl+Shift+I` a otvorí sa nám okienko, v ktorom môžeme upravovať zdrojový kód stránky priamo v prehliadači. Tam môžeme spraviť podobnú úpravu s rovnakým výsledkom.

Objaví sa text: *VYBORNE! CEZ BEZPECNOSTNU PRASKLINU V STRANKE SEM PRENIKLO 'SVETLO' MOZES UZ ZHASNUT. DOBRU NOC.*

Odpoveďou je teda slovo **SVETLO**.

Študijný text: O podprogramoch a rekurzii

V úlohách 4 a 5, kde ste riešili problémy robota Zerga, bolo treba používať aj komplikovanejšie koncepty ako je napríklad rekurzia. Tento študijný text slúži na bližšie vysvetlenie tohoto pojmu a toho, čo všetko sa s ňou dá robiť. Odporúčame si ho prečítať pre hlbšie pochopenie zvyšných dvoch vzorákov a takisto pojmu rekurzie.

Čo je to podprogram?

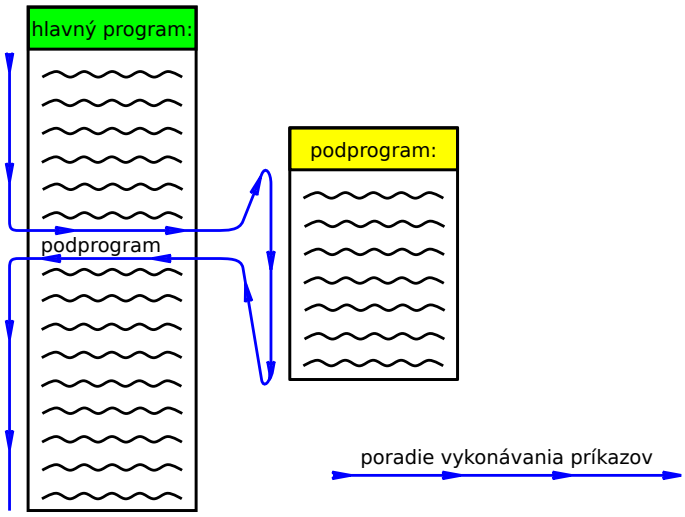
Koncept podprogramu je jednoduchý: je to skupina príkazov, ktorej dáme nejaké meno a neskôr ju podľa tohto mena voláme.

Celé to teda vyzerá tak, že niekde definujeme podprogram a v inej časti programu ho potom zavoláme. Keď sa program dostane na toto miesto, vykoná všetky príkazy podprogramu a potom pokračuje z miesta, kde bol podprogram zavolaný. Počítač si teda (aspoň pri väčšine programovacích jazykov) vždy uchováva informáciu o tom, odkiaľ bol podprogram zavolaný, aby sa tam po jeho dokončení mohol vrátiť (pozri Obr. 1).

Toto vyzerá ako úplná zbytočnosť – radšej sme mohli príkazy z podprogramu napísať na miesto, odkiaľ sme ho chceli volať. Výhodou podprogramu ale je, že ho môžeme zavolať aj viackrát, z rôznych miest programu. Ak

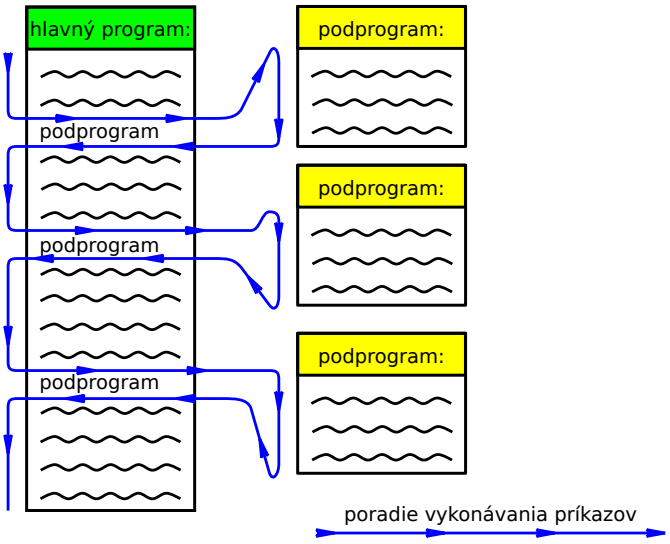
[3] Napríklad si vytvoríte nový súbor s príponou .html v notepade.

[4] Štýl býva väčšinou oddelený (v samostatnom súbore, alebo aspoň tagmi <`style`>) od obsahu stránky – textu, obrázkov... Na jeho popis sa používa jazyk CSS.



Obr. 1: Volanie podprogramu

vieme, že nejaký zložitejší úkon budeme musieť robiť na rôznych miestach programu, môžeme si na to napísať podprogram, čím si ušetríme kopec písania a znížime tým aj šancu, že sa pri písaní pomýlime (pozri Obr. 2).



Obr. 2: Viacnásobné volanie podprogramu

Jazykové okienko

Existujú rôzne slová označujúce podprogramy, každé má trochu iný význam:

1. ***procedúra*** je najjednoduchší druh podprogramu: Je to jednoducho zhluk príkazov, ktoré sa vykonajú (dali by sa nakopírovať rovno do programu namiesto volania podprogramu a nič by sa nezmenilo).

2. ***funkcia*** je termín prebraný z matematiky. Ide o podprogram, ktorý má na základe nejakých vstupných údajov vypočítať nejaký výstup. Tento vypočítaný výsledok potom odovzdá programu, ktorý ho zavolal. V niektorých programovacích jazykoch (napr. Haskell) sa trvá na tom, aby funkcia nerobila nič iné, aby nemala žiadne vedľajšie účinky (teda sa naozaj správala ako funkcia v matematike). Iné programovacie jazyky (napr. C++) sú v tomto ohľade menej striktné a povoľujú, aby funkcia robila aj veci ovplyvňujúce zvyšok programu. V takýchto jazykoch potom nie je potrebné mať špeciálny zápis na procedúry (procedúry

sú funkcie, ktoré nič nevracajú). Stačí ak funkcia vráti ľubovoľnú hodnotu (napr. 0), ktorá sa potom zvyškom programu ignoruje.

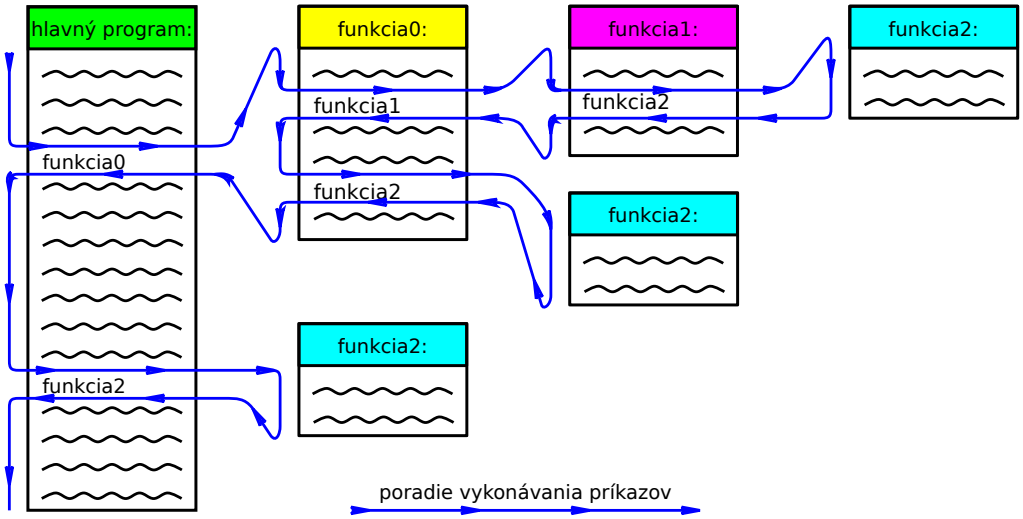
Preto sa slovom funkcia občas myslí aj podprogram všeobecne.

3. Niekedy sa možno stretnete aj s termínom ***metóda***. Toto slovo sa používa v objektovo orientovanom programovaní a neznamená nič iné ako funkciu alebo procedúru, ktorá existuje v kontexte nejakého objektu (ak ste tomuto vôbec nerozumeli, nevadí, zatiaľ to ani nepotrebujete).

V tomto texte budem ďalej namiesto slova ***podprogram*** používať slovo ***funkcia***. A postupnosti príkazov, ktoré tvoria funkciu budeme hovoriť ***telo*** funkcie.

Volanie funkcie vo funkcii

Volanie funkcie nie je obmedzené iba na hlavný program. Vnútri tela funkcie môžeme volať iné funkcie. Vnútri iných funkcií potom môžeme volať ďalšie atď. Vyzerá to asi ako na obrázku 3.



Obr. 3: Volanie funkcie vo funkcii

Pri každom volaní funkcie si počítač zapamätá, odkiaľ bola zavolaná. Túto informáciu si uchováva, až kým sa nevykoná celé telo danej funkcie. Vtedy program túto informáciu využije (aby vedel, kde má pokračovať) a následne ju už môže zabudnúť. To znamená, že keď vnútri jednej funkcie zavoláme druhú funkciu, počítač si musí zároveň pamätať, odkiaľ bola zavolaná prvá funkcia, aj odkiaľ bola zavolaná druhá funkcia. Pre príklad na obrázku to znamená, že keď sa `funkcia2` vykonávala prvýkrát, počítač si musel pamätať miesta, z ktorých boli zavolané `funkcia2`, `funkcia1`, aj `funkcia0`. Keď bola `funkcia2` volaná druhý raz, už si nemusel pamätať, odkiaľ bola volaná `funkcia1`, keďže táto funkcia medzičasom skončila. Keď bola `funkcia2` zavolaná tretí raz, už si nemusel pamätať ani to, odkiaľ bola volaná `funkcia0`.

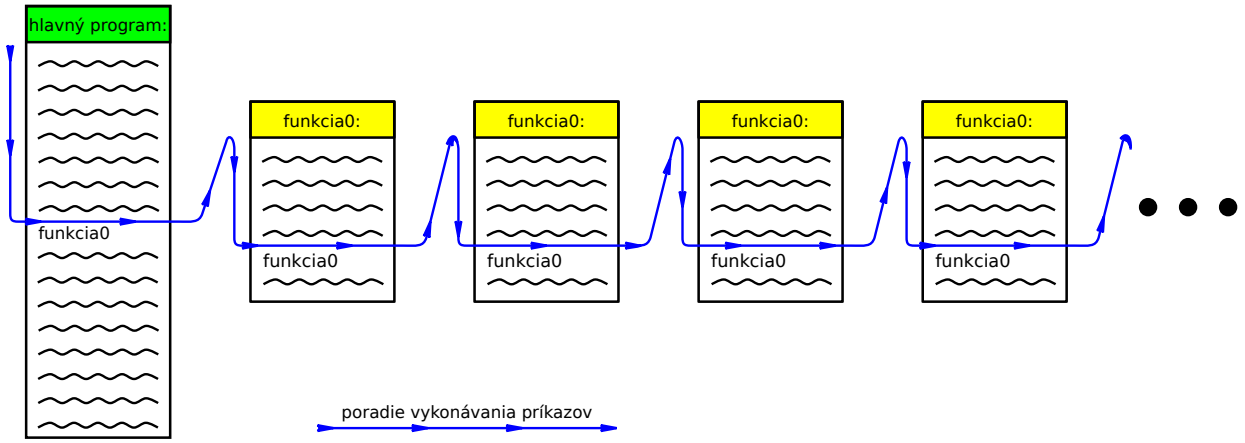
Nekonečná rekurzia

Čo by sa stalo, keby sme vnútri tela funkcie zavolali tú istú funkciu? Po prvom zavolaní funkcie v hlavnom programe by sa začali vykonávať jej príkazy, až pokým by funkcia nezavolala sama seba. V tom okamihu by sa znovu od začiatku začali vykonávať príkazy danej funkcie, až kým by znovu nezavolala sama seba. A takto by to išlo ďalej a ďalej (pozri Obr. 4).

Technika, kde funkcia volá sama seba, sa nazýva ***rekurzia***. V praktickom programovaní vznikajú pri nekonečnej rekurzii isté technické problémy. Konkrétne, pri každom zavolaní funkcie si počítač musí zapamätať, odkiaľ bola zavolaná, pričom túto informáciu môže zabudnúť, až keď sa funkcia úplne vykoná (čo nenastane nikdy). To znamená, že po nejakom počte zavolaní sa mu zaplní celá časť pamäte, ktorú mal na tento účel vyhradenú. Preto je v mnohých jazykoch nepoužiteľná.

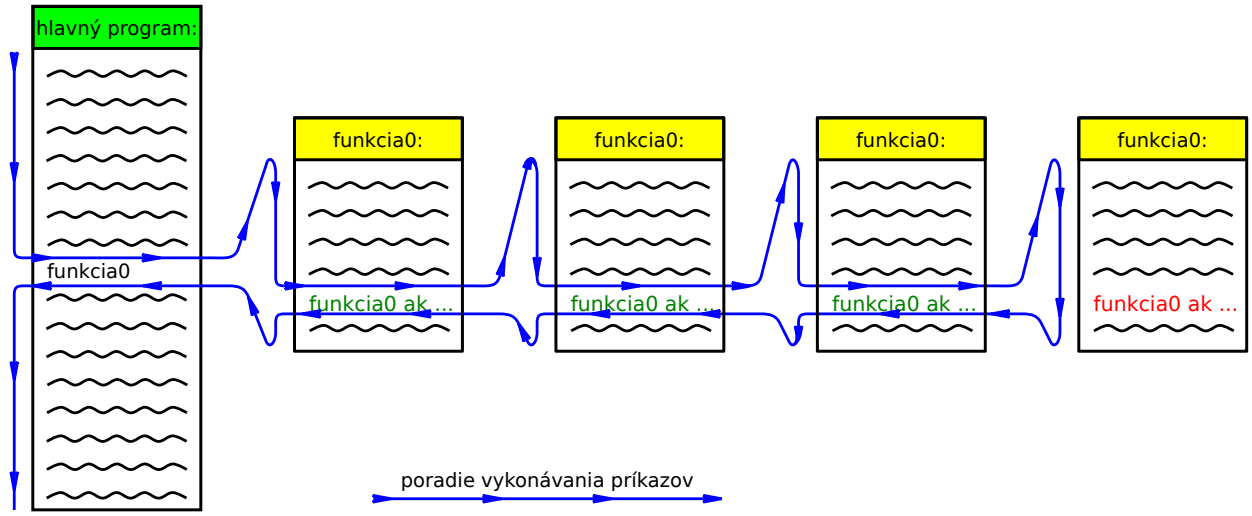
Chvostová rekurzia

Rekurzia však nemusí byť nutne nekonečná. Ak funkcia vo svojom tele volá sama seba **len ak sú splnené určité podmienky**, pričom tieto podmienky po nejakom počte zavolaní prestanú platiť, funkcia sa zavolá



Obr. 4: Obr. 4: Nekonečná rekurzia

iba konečný počet krát. Po poslednom zavolaní funkcie sa funkcia vykoná (bez toho, aby sa zavolala znovu), následne sa dokončí vykonávanie predošlého volania (vykonajú sa príkazy nachádzajúce sa za volaním funkcie), potom sa dokončí predošlé volanie atď., až kým sa nedostaneme k miestu, kde bola funkcia zavolaná na začiatku (pozri Obr. 5).



Obr. 5: Obr. 5: Chvostová rekurzia

Všeobecná rekurzia

Nie sme obmedzení na to, aby sa funkcia vo svojom tele volala iba raz, pokojne sa môže zavolať aj viackrát. Alebo môže volať nejakú inú funkciu, ktorá potom naspäť zavolá ju. Kreativite sa medze nekladú.

4. Zerg Bot 1

vzorák napísal Baklažán
(max. 0 b za riešenie)

1 - Komnata

Tu stačilo robota odnavigovať na správne tlačidlo, prepnúť ho a odnavigovať do cieľa. To, ktoré tlačidlo je správne, sa dalo zistiť napríklad tak, že ste postupne vyskúšali všetky. Správne bolo tretie tlačidlo zdola.

Kód:

```
doprava
krok
krok
doprava
krok
prepni
doprava
krok
krok
doprava
krok
krok
krok
```

2 - Učko

V tomto leveli je na prvý pohľad jasné, čo má robot urobiť. Problém ale je, že nemá dosť pamäte na všetky príkazy. Môžeme si však všimnúť, že U-čko sa skladá z troch rovnakých častí. Napíšeme si teda funkciu, ktorá nám prejde jednu časť U-čka a túto funkciu potom zavoláme trikrát za sebou. Treba si však dať pozor, aby bol robot po dokončení jedného ramena U-čka správne otočený, na konci funkcie ho preto otočíme doprava.

Kód:

```
funkcia0
funkcia0
funkcia0

aha funkcia0:
krok
prepni
krok
krok
doprava
```

3 - Schody

Potrebujeme prejsť veľa rovnakých schodov, preto sa nám oplatí napísať si funkciu na prejdenie jedného schodu:

```
aha funkcia0:
krok
doprava
krok
dolava
```

Teraz však narazíme na problém: môžeme použiť už iba dva príkazy a schodov je oveľa viac, než dva.

Použijeme preto jeden zaujímavý trik: napíšeme funkciu, ktorá prejde jeden schod a potom *zavolá sama seba*. Keď sa táto funkcia začne vykonávať, prejde jeden schod a sama sa zavolá znovu. Opäť prejde jeden schod a znovu sa zavolá. A takto bude náš program pokračovať až do nekonečna, resp. kým robotovi nedôjde batéria alebo nepríde do cieľa.

Kód:

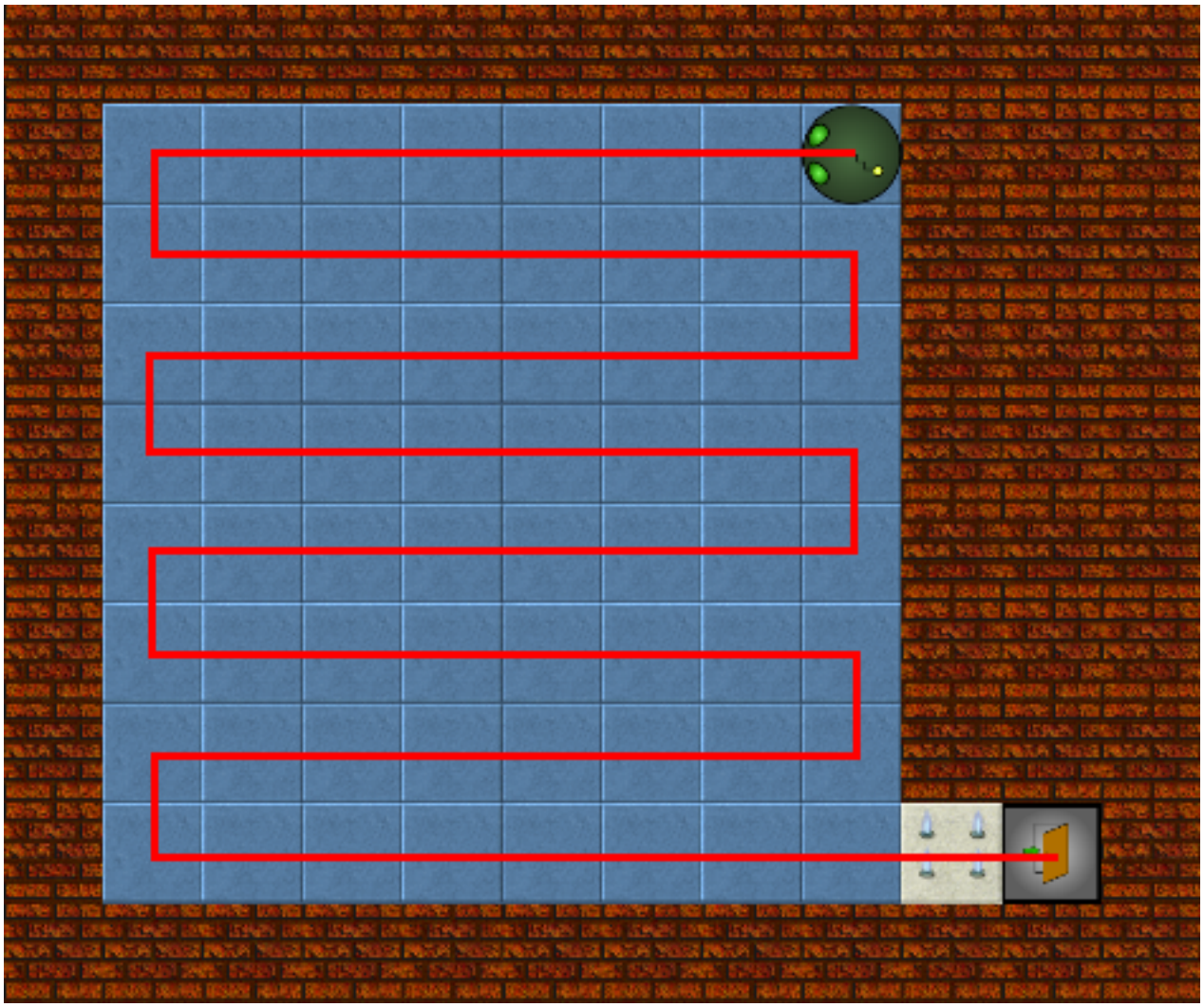
```
funkcia0

aha funkcia0:
krok
doprava
krok
dolava
funkcia0
```

Ako už viete zo študíjneho textu, tento princíp sa volá *rekurzia*. V našom prípade ide o nekonečnú rekurziu, čo pekne vystihuje Obr. 4. Na našu `funkcia0` sa môžeme pozerať tak, že je to funkcia, ktorá prejde nekonečne veľa schodov. Urobí to tak, že najprv prejde jeden schod (prvé 4 príkazy) a potom ešte nekonečne veľa ďalších (posledný príkaz).

4 - Šachovnica

Najjednoduchšie asi bude ísť po riadkoch a cestou prepnúť správne políčka.



Obr. 6: Zamýšľaná trajektória

Existuje viacero spôsobov, ako prejsť takúto cestu. Jednou z možností je napísať si štyri funkcie, ktoré sa navzájom volajú v nekonečnej rekurzii. My si ale vystačíme aj s niečím jednoduchším.

Budeme potrebovať, aby robot prepol prepínač na každom druhom políčku, cez ktoré prejde. Napíšme si teda funkciu, ktorá prejde dve políčka, pričom to, na ktoré stúpi skôr, prepne:

```
aha funkcia0:
krok
prepni
krok
```

S jej pomocou si môžeme napísať funkciu, ktorá prejde jeden riadok šachovnice a prepne pri tom potrebné políčka – za predpokladu, že štartujeme na tom konci riadka, ktorý nemá byť zapnutý:

```
aha funkcia1:
funkcia0
```

```
funkcia0
funkcia0
funkcia0
```

Všimnime si, že táto funkcia sa po zapnutí posledného prepínača v riadku bude snažiť urobiť ešte jeden krok. To nám ale neprekáža, robot sa iba pokúsi urobiť krok do steny (alebo na JetTorch, ktorý už v tom čase bude vypnutý).

Aj na prejdenie ľavotočivej a pravotočivej zákruty si môžeme napísať funkcie:

```
aha funkcia2:
dolava
krok
dolava

aha funkcia3:
doprava
krok
doprava
```

Na ceste, po ktorej má robot prejsť, sa striedajú rovné úseky a zákruty, pričom zákruty sú striedavo ľavotočivé a pravotočivé. Napíšme si teda funkciu, ktorá nám prejde dva rovné úseky a zákruty nasledujúce po nich:

```
aha funkcia4:
funkcia1
funkcia2
funkcia1
funkcia3
```

Keď túto funkciu zavoláme štyrikrát po sebe, robot by mal prejsť celú šachovnicu. Problém ale je, že po prejdení posledného riadka sa bude snažiť prejsť ešte jednu pravotočivú zákrutu. Preto ju radšej zavoláme iba trikrát a posledné dva rovné úseky (aj so zákrutou medzi nimi) rozpíšeme. Robot by potom skončil na políčku s JetTorchom (nezabúdajte, že `funkcia1` sa po prejdení riadka snaží urobiť ešte jeden krok), preto na koniec pridáme ešte jeden krok.

Hlavný kód:

```
funkcia4
funkcia4
funkcia4
funkcia1
funkcia2
funkcia1
krok
```

5 - Zeleno-modrá cesta

Kľúčovým pozorovaním v tomto leveli je to, že po zapnutom tlačidle treba ísť najbližšie dva kroky smerom na juh a po vypnutom prepínači treba ísť najbližšie dva kroky smerom na východ. Po urobení dvoch krokov správnym smerom sa bude robot nachádzať na nasledujúcom prepínači.

Tiež si môžeme všimnúť, že máme povolených iba veľmi málo príkazov. Preto opäť využijeme nekonečnú rekurziu: napíšeme si funkciu, ktorá (za predpokladu, že robot stojí na prepínači) urobí dva kroky správnym smerom, a potom sa zavolá znovu.

Tu ale narazíme na jeden problém: robot nevie zistiť, ktorým smerom je práve otočený, teda nebude vedieť, ktorým smerom je východ a ktorým smerom je juh. To môžeme vyriešiť tak, že vždy potom, čo robot prejde na ďalší vypínač, sa otočí na východ (teda ak išiel na juh, otočí sa doľava a ak išiel na východ, ostane tak, ako je). Na to si ale bude musieť pamätať, či bolo predošlé tlačidlo stlačené alebo nie. To docielime tak, že namiesto toho, aby sa robot na tlačidle iba otočil alebo neotočil (podľa toho, či je zapnuté), a potom urobil dva kroky, sa na tlačidle zavolá buď funkcia, ktorá urobí dva kroky smerom na východ, alebo funkcia, ktorá urobí dva kroky smerom na juh a potom sa otočí na východ.

Kód:

```
funkcia0

aha funkcia0:
funkcia1 ak svieti
funkcia2 ak nesvieti
funkcia0

aha funkcia1:
doprava
krok
krok
dolava

aha funkcia2:
krok
krok
```

Počas behu programu si môžeme všimnúť, že niekedy sa v jednom volaní `funkcia0` zavolá aj `funkcia1` aj `funkcia2` (nastane to vtedy, keď je robot na zapnutom tlačidle, ale po vykonaní `funkcia1` sa ocitne na vypnutom tlačidle). To nám v našom prípade neškodí, ale keby sme chceli, aby sa v jednom volaní `funkcia0` zavolala vždy iba jedna z funkcií 1 a 2, mohli by sme to docieliť napríklad tak, že by sme `funkcia0` zavolali aj na konci `funkcia1`. Teda ak by tlačidlo bolo zapnuté, netestovalo by sa, či sa má spustiť `funkcia2`. [text o rekurzii]: TO_DO

vzorák napísal Baklažán
(max. 0 b za riešenie)

5. Zerg Bot 2

1 - Riedke bludisko

Jeden z možných spôsobov, ako dôjsť do cieľa, je ísť vždy čo najdlhšie rovno a keď prídeme ku stene, otočiť sa doprava. To vieme s pomocou nekonečnej rekurzie (Obr. 4 v študijnom texte o rekurzii) zapísať napríklad takto:

```
funkcia0

aha funkcia0:
doprava ak je stena vpredu
krok
funkcia0
```

2 - Hrebeň

Keďže tlačidlá sú v rôznych zuboch hrebeňa rôzne ďaleko, budeme potrebovať funkciu, ktorá robotom pohne po najbližšie zapnuté tlačidlo. Tá môže vyzerať takto:

```
aha funkcia0:
krok
funkcia0 ak nesvieti
```

Táto funkcia sa teda bude volať a hýbať robotom, až kým nestúpi na tlačidlo. V tom okamihu nebude platiť podmienka `ak nesvieti`, teda funkcia sa už znovu nezavolá a robot zastane. Takýto druh rekurzie sa nazýva *chvostová rekurzia*. Podobne si môžeme napísať funkciu, ktorá robotom pohne po najbližšiu stenu:

```
aha funkcia1:
krok
funkcia1 ak nie je stena vpredu
```

S pomocou týchto dvoch funkcií už ľahko napíšeme funkciu, ktorá vstúpi do zubu, nájde tlačidlo, stlačí ho, otočí sa a presunie sa k spodnej stene, kde sa rekurzívne zavolá. Musíme si však dať pozor, aby bol robot pri rekurzívnom volaní v takej istej situácii ako na začiatku, čo bude znamenať, že musí skončiť postavený na (teraz už vypnutom) JetTorch-i otočený doprava.

```
aha funkcia2:
krok
dolava
funkcia0
prepni
dolava
dolava
funkcia1
dolava
krok
funkcia2
```

V hlavnom programe nám stačí zavolať `funkcia2`.

3 - Zeleno-modrá cesta 2

Pred čítaním tohto riešenia vám odporúčam prečítať si vzorové riešenie levelu Zeleno-modrá cesta (posledný level v predošlej úlohe).

Podobne ako pri prvej zeleno-modrej ceste platí, že stav prepínača určuje, ktorým smerom z neho máme ísť:

- Z vypnutého tlačidla treba ísť dva kroky na sever.
- Zo zapnutého tlačidla treba ísť dva kroky smerom na východ.
- Z ukradnutého tlačidla (miesto, kde by malo byť tlačidlo, ale nie je) treba ísť dva kroky smerom na juh.

Podobne ako v zeleno-modrej ceste si snapíšeme funkciu, ktorá urobí dva kroky správnym smerom, otočí sa na východ a potom zavolá sama seba.

Tu ale narazíme na problém: ako vieme rozlíšiť ukradnuté tlačidlo od vypnutého, keď pre obe platí, že nesvietia? Jednoducho tak, že sa ho pokúsime prepnúť. Ak je po prepnutí zapnuté, znamená to, že bolo iba vypnuté. Ak stále nie je zapnuté, znamená to, že je ukradnuté. Kód by teda vyzeral takto:

```
funkcia0

aha funkcia0:
funkcia1 ak svieti
prepni
funkcia2 ak svieti
funkcia3 ak nesvieti
funkcia0

aha funkcia1:
krok
krok

aha funkcia2:
dolava
krok
krok
doprava

aha funkcia3:
doprava
krok
krok
dolava
```

Tu ale príde ďalší problém: môže sa stať, že sa v jednom volaní `funkcia0` zavolá `funkcia2` a potom aj `funkcia3` (ak sa vykonaním `funkcia2` robot dostane na nesvietiace políčko). Pri volaní `funkcia3` pritom už nemusí platiť, že robot je na ukradnutom vypínači, keďže môže byť aj na vypnutom. A naozaj sa to aj stane, už pri treťom vypínači. Preto zaručíme, aby sa v jednom volaní `funkcia0` zavolala vždy iba jedna z funkcií 1,2,3.

To urobíme tak, že aj na konci každej z funkcií 1,2,3 zavoláme `funkcia0` (volanie `funkcia0` na konci `funkcia0` sa tým pádom stane zbytočným).

4 - Bludisko

Na vyriešenie tohto levelu použijeme algoritmus, ktorý sa nazýva *pravidlo pravej ruky* a slúži na prehľadávanie bludísk. Je veľmi jednoduchý: na začiatku sa pravou rukou chytíme múru a celý čas ideme pozdĺž neho bez toho, aby sme sa ho pustili. Takýmto spôsobom prejdeme po celom obvode bludiska a buď sa vrátime späť tam, kde sme začali, alebo cestou nájdeme nejaký iný východ z bludiska (čo sa v našom prípade nestane, keďže bludisko iný východ nemá). Všimnime si, že všetky políčka nášho bludiska sa nachádzajú po jeho obvode, teda pravidlom pravej ruky prejdeme cez všetky prepínače.

Pre nášho robota to bude znamenať, že vždy, keď vojde na nové políčko, sa bude snažiť ísť čo najviac doprava. To znamená, že ak vpravo nie je stena, tak pôjde doprava, ak je vpravo stena, tak pôjde rovno, ak je stena aj vpredu, tak pôjde doľava a ak je stena aj vľavo, tak sa vráti, odkiaľ prišiel.

Kód:

```
funkcia0

aha funkcia0:
krok
prepni ak nesvieti
doprava ak nie je stena vpravo
dolava ak je stena vpredu
dolava ak je stena vpredu
funkcia0
```

Po tom, čo robot prepne všetky vypínače, pôjde ďalej pozdĺž múru, až kým príde do cieľa.

5 - JetTorchové peklo

Cesta obsahuje štyri rovné úseky, pričom prvý a tretí sú "bezpečné", keďže sú zakončené stenou, kým druhý a štvrtý sú "nebezpečné", keďže za nimi nasleduje JetTorch. Kľúčovým pozorovaním tohto levelu je, že každý nebezpečný úsek je rovnako dlhý ako bezpečný úsek pred ním.

Napíšme si teda funkciu, ktorá pôjde po najbližšiu stenu, potom sa otočí doľava a urobí rovnako veľa krokov, ako urobila cestou k stene. Ako začiatok si môžeme zobrať funkciu z levelu 2, ktorá ide po najbližšiu stenu:

```
aha funkcia0:
krok
funkcia0 ak nie je stena vpredu
```

Po tom, čo robot príde k stene, sa má otočiť doľava, preto doplníme jeden príkaz:

```
aha funkcia0:
krok
funkcia0 ak nie je stena vpredu
dolava ak je stena vpredu
```

Všimnime si, že táto funkcia sa zavolá presne toľkokrát, koľko krokov urobí robot cestou k stene. A to je presne toľko, koľko má urobiť cestou od steny. Tiež si môžeme všimnúť, že žiadne volanie funkcie sa neskončí, kým robot nepríde k stene. To znamená, že ak by robot v každom zavolaní funkcie urobil ešte jeden krok po tom, ako sa otočí pri stene, urobil by presne to, čo potrebujeme. Odporúčam si pozrieť obrázok 5 zo študijného textu. Ak si nakreslíte, ako sa budú vykonávať príkazy, bude vám to oveľa jasnejšie.

Funkcia teda bude vyzerať takto:

```
aha funkcia0:
krok
funkcia0 ak nie je stena vpredu
dolava ak je stena vpredu
krok
```

Ak sa vám toto zdalo nezrozumiteľné, môžeme sa na našu funkciu pozrieť aj ako na vysvetľovanie významu slova pomocou toho istého slova. Príkaz "choď rovno po najbližšiu stenu, potom sa otoč doľava a choď rovnako dlho rovno" vieme totiž popísať nasledovne:

1. Najprv urob krok
2. - Ak je už pred tebou stena, otoč sa doľava a urob ešte jeden krok (kvôli kroku, ktorý si urobil v bode 1).
   - V opačnom prípade choď rovno po najbližšiu stenu, potom sa otoč doľava a choď rovno rovnako dlho. Nakoniec urob ešte jeden krok (za krok, ktorý si urobil v bode 1).

A presne to naša funkcia robí.
S touto funkciou bude hlavný kód jednoduchý:

```
funkcia0
dolava
funkcia0
dolava
krok
```