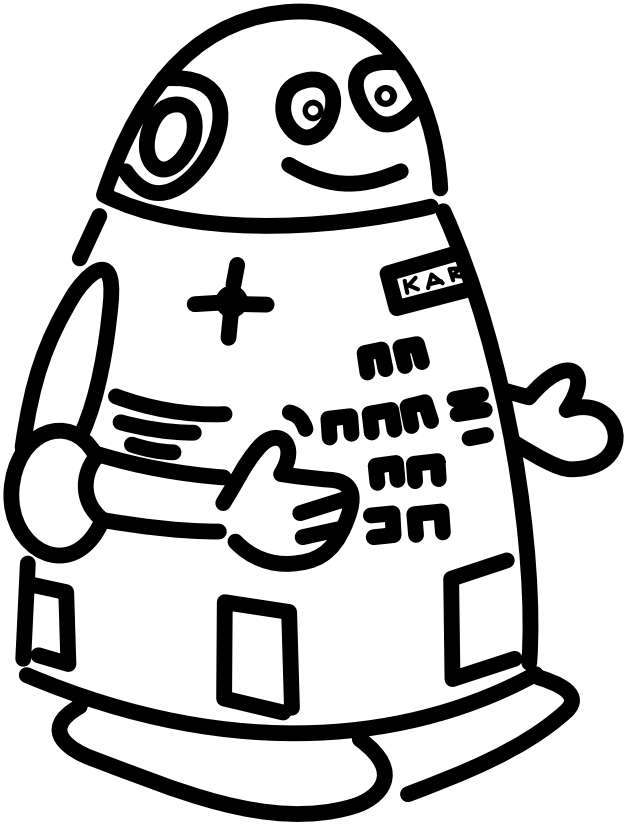
PROGRAMOVACÍ JAZYK



KAREL

1987, 2001

Programovací jayzk KAREL
Vejvoda Michal Ing., Rytíř Miroslav Ing.
programování, robot Karel

Pro Centrum pro mládež, vědu a techniku ÚV SSM připravila Stanice Mladých Techniků při ZRP Větřní v Českém Krumlově.

Metodický materiál pro kroužky programování s dětmi od druhé třídy ZŠ.

Autoři: Ing. Michal Vejvoda, Ing. Miroslav Rytíř.

Použito materiálů z VTM ing. Rudolfa Pecinovského, CSc, a materiálů CMVT ÚV SSM.

Sazba sytémem TEX.

Program PC-Karel, kterému je celý text přizpůsoben, si můžete zdarma stáhnout z internetové stránky www.pckarel.zde.cz.

Verze textu: 2001/10/06.

ISBN 99972-03-09-7

Úvodem

Ahoj mladí přátelé!

Dostáváte do rukou tuto knížku, abyste se mohli seznámit s počítači a jejich programováním. Programovací jazyk KAREL není určen k tomu, aby programy v něm sestavené za vás řešili *domácí úkoly*. Ale zato se naučíte to podstatné – *programovat*. A to dokonce moderně programovat. KAREL má tu výhodu, že veškeré příkazy vidíte na monitoru počítače, jak se provádějí. Tak snadno přijdete na případné chyby, kterých se při práci dopustíte.

V knížce rovněž není uvedeno vše, co KAREL dokáže. Je to jen stručná metodika pro výuku začátečníků.

Hodně zdaru při práci a trvalé kamarádství vám přejí

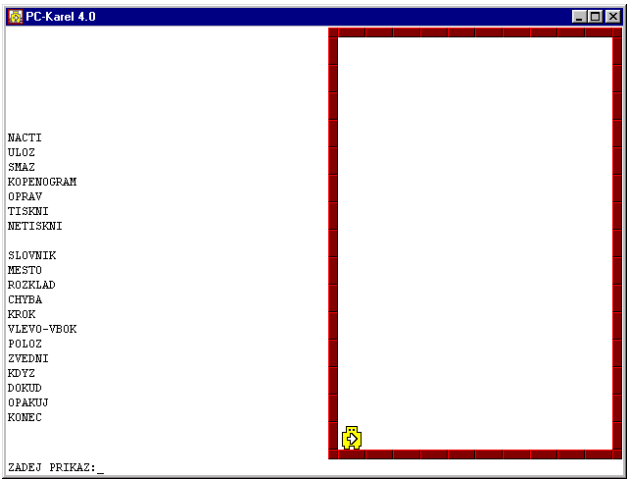
počítače, programátoři a robot KAREL.

1. KAREL se představuje

Ahoj kamaráde!

Vítám tě na naší první společné schůzce. Jmenuji se KAREL. Jsem robot a jméno KAREL mi dal učitel programování, autor prvního programu a knížky „Karel The Robot“ profesor Richard E. Pettis ze Stanfordské univerzity ve Spojených státech. Své jméno jsem získal na počest spisovatele Karla Čapka, autora divadelní hry R.U.R. – Rossum's Universal Robots.

V bývalém Československu jsem se poprvé objevil v Bratislavě zásluhou docenta Hvoreckého, CSc. Pro tebe, kamaráde, a pro československé mikropočítače realizovali své představy o robotu KARLOVI ing. Rudolf Pecinovský, CSc, a ing. Tomáš Bartovský, CSc. Na počítačích PC mohu pracovat i díky ing. Jiřímu Osobovi. V této brožuře vystupuji v představách výtvarníka Vladimíra Jiránka. Doufám, že se ti líbím, a že se nám spolu bude dobře spolupracovat.



Tak a tady – to je moje město! Vítám tě. A jako správný hostitel tě svým domovem provedu.

Při každém spuštění tě přivítám v levém dolním rohu města. To je můj domov. Jak se ti líbí?

Mé město – to je vlastně šachovnice, která má 10×15 políček. Jinam nemohu. Na začátku je vždy město prázdné. Uvnitř města jsem jenom já – robot KAREL. A záleží na tobě, jak útulné prostředí mi tu vybuduješ.

A co to tu máme? Spodní řádek ti bude sloužit k tomu, abys mi mohl zadávat příkazy. Jen nezapomeň, že o ničem, co jsi napsal, nevím, *dokud nestiskneš klávesu Enter* (↵). A piš mi, prosím, vždy jen jeden příkaz na řádku, abych se v tom vyznal.

V levé části ti zase já budu vypisovat různé věci. Teď je zde SLOVNIK. Příkazy, které jsou uvedeny ve slovníku, znám. Jsou to veledůležitá slova – moje příkazy. Dokonce si nechávají hrdě říkat – PRIMITIVA. Nyní se o nich dozvíš něco víc.

1.1. Primitiva

Milý žáčku poškoláčku (promiň, chtěl jsem říci chytráčku) – pekelně se soustřeď, protože teď tě seznámím s velice důležitými kamarády.

Jak už víš, říkáme jim PRIMITIVA. Jsou to příkazy, která já znám od narození. Rozdělil jsem si je do tří skupin.

V první skupině jsou příkazy ovládací. Jsou to: `NACTI`, `ULOZ`, `SMAZ`, `KOPENOGRAM`, `OPRAV`, `TISKNI`, `NETISKNI`, `SLOVNIK`, `MESTO`, `ROZKLAD`, `CHYBA`.

Do druhé skupiny patří příkazy výkonné: `KROK`, `VLEVO-VBOK`, `POLOZ`, `ZVEDNI`.

A konečně ve třetí skupině jsou příkazy podmiňující: `KDYZ`, `DOKUD`, `OPAKUJ`, `KONEC`.

Jací jsou to užiteční a chytří chlapíci se dozvíš o několik stránek dál. Teď ti dám jeden bleskový úkol. Poznáš, který ze základních příkazů – PRIMITIV – jsem na tabuli vynechal?

PRIMITIVA:

```
OPRAV  SLOVNIK  NACTI  POLOZ
MESTO  ROZKLAD  VLEVO-VBOK
OPAKUJ  SMAZ  KDYZ  ULOZ
ZVEDNI  DOKUD  KOPENOGRAM
KROK  NETISKNI  CHYBA
TISKNI
```

Kamarádi, ale včera, to jsem zkusil! Byl u mě Pepík, co se spolu taky učíme, a stěžoval si, že dostal ve škole pětku z diktátu. A přitom byl prý bez chybičky! Napsal jej úplně stejně, jak jsme se učili spolu.

Byli jsme z toho oba moc smutní. Já jsem Pepíkovi ale vysvětlil, že bez háčků a čárek můžeme psát jenom spolu. Paní učitelka měla pravdu pravdoucí. V češtině musíme háčky a čárky používat. To jenom já háčky a čárky neumím. Dokonce spolu nebudeme používat ani malá písmena. Proto i v knížce máte zápis příkazů tak, jak je píše počítač, velkými písmeny bez háčků a čárek.

Nezlobte se proto na mě! Já se to taky jednou naučím. Pak si spolu budeme psát tak, jak vás to učí ve škole.

Tak a teď už o mě víš vše, co pro začátek potřebuješ. Významy jednotlivých primitiv si postupně vysvětlíme. Ale, ale, jak ti koukám na nos, tak už sis je dokonce sám vyzkoušel! To ti chválím. Umíš už tedy tolik co já. Můžeš si alespoň ověřit, jaký jsem poslušný robot KAREL.

Zadávej mi postupně `KROK` a `VLEVO-VBOK`. Nezapomeň *každý příkaz odeslat klávesou Enter* (↵). Vidíš, jak se pohybuji po městě přesně podle tvých příkazů? Ovšem kdybys se mnou chtěl projít zdí, tak tě neposlechnu! Tam přece nesmím! Navíc si vždycky narazím nos, pardon, nosní elektronku, a oznámím ti to.

Dále znám ještě příkazy `POLOZ` a `ZVEDNI`. Příkaz `POLOZ` mi slouží k tomu, abych ze svého kouzelného batůžku, který mám na zádech, položil pod sebe při tvém příkazu značku. Těch značek mohu položit na jedno políčko až devět. A protože příroda ani město se nesmí znečišťovat, tak dokáži zase značky vysbírat pomocí příkazu `ZVEDNI`! Kdyby chtěl umístit na jedno políčko víc než devět značek, tak ti oznámím, že `Není kam položit!`. Se značkami mohu manipulovat pouze na políčku, kde právě stojím. To je velmi důležité, zapamatuj si to!

To to utíká! Další kus práce je za námi! A jak nám to už spolu hezky jde! Mám pravdu, kamaráde? Že se ti ještě nechce končit?

Víš co? Když ti doma zbyde chvilička, můžeš si zkusit programovat i bez počítače. Nakresli si na čtvrtku papíru moje město. Nebo můžeš použít i normální šachovnici a třeba figurku koně.

Hezkou zábavu ti přeje KAREL.

2. kapitola, ale 1. programátorská

Ano, správně, je to naše první kapitola, kdy si spolu budeme hrát. Doteď to bylo vlastně jen seznámení. Bylo to sice důležité, ale stejně tě slyším, jak si broukáš pod nos „…a to byla otrava, už abychom programovali…“

A teď máš příležitost. Otevřeme spolu ta tajemná dvířka programování a vydáme se vstříc neznámým koutům počítačového světa. Tak jdeme!

Posledně jsme si zkusili takovou malou procházku po městě. Sám sis ověřil už některá primitiva. Už víš, že když napíšeš na moji žádost `NAPIS PRIKAZ:` některý známý příkaz, tak jej provedu.

Teď ti ale prozradím, jak si svoji práci můžeš ulehčit. Zkus napsat třeba `VL.` – a já provedu `VLEVO-VBOK`. Zkratka musí být vždy ukončena tečkou! Pozor jenom na jednu věc: Já zkratky vybírám se svého sluvníku. A pokud mají některé příkazy stejnou zkratku, tak vyberu to, které je v mém slovníku umístěno blíž k začátku. Například když zadáš `K.`, tak vyberu `KROK` a nikoliv `KONEC`, jak bys třeba chtěl ty. Zrovna tak poslouchám jen ty příkazy, které jsou napsány správně. Jen si to zkus, vynechat pomlčku ve `VLEVO-VBOK`. Uvidíš, jak tě neposlechnu!

Už jsme si řekli, že s pomocí základních primitiv se spolu můžeme naučit další příkazy. Jak se ale takový nový příkaz dělá?

Je to celkem jednoduché. Když jsi mě nechal jít třeba z rohu do rohu, tak jsi musel na každý můj krok napsat jeden příkaz `KROK`. Napiš mi nyní příkaz, abych provedl dva kroky současně. Příkaz si nazveme třeba `DVOJ-KROK`. Napiš na klávesnici `DVOJ-KROK`. Po stisku `Enter` (↵) se na obrazovce objeví:

```
NOVY PRIKAZ DVOJ-KROK ZNAMENA
```

Tím se přiznávám, že tento příkaz dosud neznám. A teď napiš, co mám vykonat, abych udělal dva kroky. *Pozor!* Nemůžeš přece napsat `DVA KROKY`! To nejsou slova, která mám ve svém slovníku a proto je ani neznám. Dva kroky – to je přece `KROK` a `KROK`. Tak napíšeme `KROK` a stiskneme `Enter` (↵). Na obrazovce se objeví:

```
NOVY PRIKAZ DVOJ-KROK ZNAMENA
  KROK
```

A nyní napíšeme znovu `KROK`. Po stisku `Enter` (↵) je na obrazovce napsáno:

```
NOVY PRIKAZ DVOJ-KROK ZNAMENA
  KROK
  KROK
```

Tím ale ještě není příkaz hotov. Musíš mi říct, že už nemám příkaz dále vykonávat, že už mám končit. Proto napiš `KONEC`. Příkazem `KONEC` a odesláním pomocí `Enter` (↵) nový příkaz ukončím. Na obrazovce se objeví:

```
NOVY PRIKAZ DVOJ-KROK ZNAMENA
  KROK
  KROK
KONEC
```

A tím máme náš první nový příkaz – vlastně program – hotov. Ví to i počítač, protože nám na obrazovce rozsvítí nápis:

```
ZADEJ PRIKAZ:_
```

A teď můžeš zadat nově vytvořený příkaz `DVOJ-KROK`. Když jej odešleš (`Enter` ↵), tak udělám dva kroky.

Viď, že to nic nebylo? Důležité je jenom vědět, jak na to. A i ostatní spolu zvládnem stejně snadno jako `DVOJ-KROK`.

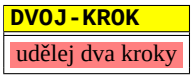
Abychom si spolu rozuměli, tak si musíme ještě dnes *domluvit několik pravidel.* Stanovíme si naši „počítačovou gramatiku“.

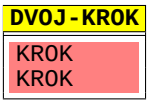
Viděl jsi sám, že neposlechnu, když mi v příkazu (tak, jak jej mám ve svém slovníku) vynecháš třeba jenom jednu čárku. Zápis programu, jak jej upravil počítač, budeme nazývat podle mě KARLOVSKÝ. Pro lidi je ale užitečnější a přehlednější zápis pomocí *kopenogramů*.

Kopenogramy jsou tou nejpřehlednější formou, jak program zapsat. A to nejenom pro mě, ale i pro použití v programovacím jazyku. Co to tedy kopenogram je?

Veškerou moji činnost můžeš rozdělit na dva druhy. Buď provádím nějaký příkaz, nebo se rozhoduji, co budu dělat dál. Každou činnost kopenogram znázorňuje obdélníkem. V něm je pak uveden název příkazu nebo je popsaný. A to buď slovy nebo rozepsáním na jednodušší příkazy. Příkaz, který se dál již nerozepisuje, je vhodné vybarvit. Jak se kopenogram tvoří a jak se jednotlivé barvy používají, máš souhrnně uvedeno v příloze 1. Protože ale i celý program (příkaz, který kopenogram definuje) je blok, je i on obdélníkem.

Program má ale oproti ostatním příkazům několik zvláštností. Jednou z nich je, že mi musíš říct, který program se má provést. To znamená, že se musí nějak jmenovat. V našem prvním příkladu jsme si pojmenovali příkaz `DVOJ-KROK`. Pak už mi oznámíš jenom jméno programu a já požadovaný příkaz – program – provedu. A já znám pouze ty příkazy, které mám uvedeny ve slovníku. Protože je jméno důležité, tak je i v kopenogramu vyznačíme zřetelně. Napíšeme jej nahoru a oddělíme dvojitou čárou. A dohodneme se, že jméno programu budeme značit vždy žlutou barvou.






```
DVOJ-KROK ZNAMENA
  KROK
  KROK
KONEC
```

A jdeme dál. Pod jménem programu DVOJ-KROK je uveden vlastní příkaz, tedy co to DVOJ-KROK znamená. Pro začátek jsem ti rozepsal kopenogram dvakrát. Jednou pomocí známých příkazů a jednou slovně. Každý takový příkaz je vlastně malým programem, který do mě vložil už můj táta. A protože jsou uváděny pod názvem nového programu DVOJ-KROK, říkáme jim taky někdy *podprogramy.*

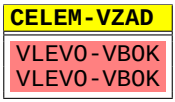
Teď se znovu podívej na náš rozepsaný kopenogram a na můj zápis. Našel jsi nějaký rozdíl? Správně. Kopenogram neobsahuje příkaz KONEC. To proto, že konec je tady znázorněn uzavřením do obdélníku.

Podobně si zkusíme další příkazy. Rozmysli si, jak bys mi přikázal, abych udělal třeba čelem vzad.

Správné řešení ti za chvilku prozradím. Ale žádné nahlížení do výsledků! Ani jedním očkem! Takové snadné vítězství by tě přece ani netěšilo. Jen ti malinko poradím. Zkus si to udělat sám – bez počítače.

Už jsi hotový? Tak nejdřív prakticky. Postav se a udělej čelem vzad. To znamená, že se napřed otočíš doleva – provedeš VLEVO-VBOK. Pak uděláš ještě jednou VLEVO-VBOK. Podobně i já.

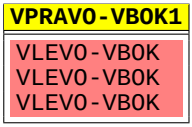
Příkaz CELEM-VZAD bude tedy vypadat takto:




```
CELEM-VZAD ZNAMENA
  VLEVO-VBOK
  VLEVO-VBOK
KONEC
```

Tak – a už máme druhý program. A dokonce je zapsaný jak v kopenogramu, tak i v karlovském zápisu. Teď jej zkus zapsat i do počítače.

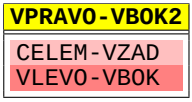
Jestli ses již podíval vzadu do slovníčku, tak jsi jistě zjistil, že tam máš uvedeny všechny příkazy. Teď ale bez dívání zkus napsat příkaz, který by mi řekl, jak mám udělat vpravo vbok. Nazveme jej VPRAVO-VBOK1. Proč je tam ta jednička? To proto, že si na tomhle příkazu ukážeme, jak je možno nadefinovat příkaz několika způsoby. Jedna z možných variant je třeba:




```
VPRAVO-VBOK1 ZNAMENA
  VLEVO-VBOK
  VLEVO-VBOK
  VLEVO-VBOK
KONEC
```

Zkus si jej naprogramovat do počítače. Když teď zadáš VPRAVO-VBOK1, tak poslechnu a udělám vpravo vbok.

Kdo se ale na příkaz podívá trochu pozorněji, tak vidí, že v příkazu je schován již jeden známý příkaz – CELEM-VZAD. A my jsme si přece řekli, že můžeš používat všechny známé příkazy, které mám ve slovníku. A proto si vytvoříme ještě jeden příkaz VPRAVO-VBOK2:




```
VPRAVO-VBOK2 ZNAMENA
   CELEM-VZAD
   VLEVO-VBOK
KONEC
```

A opět se přesvědčíme, jak jsme programovali. Vidíš, že i tento příkaz opět pracuje. Takže i tato definice je správná. Ale podle zápisu je VPRAVO-VBOK2 jednodušší a tedy i z hlediska programování správnější. Proto se dohodneme na tom, že pro všechny činnosti si budeš vytvářet pomocné programy – podprogramy – tam, kde to jen půjde. Jde nám o to, aby vlastní program byl co nejjednodušší. V takovém programu se i snáze hledají chyby.

Na tomto kopenogramu vidíš i další zvýraznění – zatímco výkonná primitiva, tj. příkazy, které mohu přímo vykonat, jsou vybarveny červeně, složené příkazy, u kterých se musím podívat jak jsou definovány, jsou vybarveny růžově.

Na závěr této lekce si ulož vytvořené programy pomocí příkazu ULOZ. Po napsání tohoto příkazu se počítač zeptá na jméno souboru. Zvol třeba ZAKLADY.

Ahoj příště. KAREL.

3. Ještě trochu turistiky

Tak jak se ti líbilo na naší minulé schůzce? Dnes budeme spolu pokračovat ještě trochu v turistice.

Napřed si – po spuštění KARLA – nahraj příkazy z minulé schůzky. Nahraješ je pomocí příkazu `NACTI`. A teď jak na to.

Po zadání (a odeslání – `Enter` ↵) příkazu `NACTI` ti nabídnu seznam slovníků, které jsi kdy uložil. No a ty můžeš vybrat – nejlépe pomocí myši – položku `zaklady.kar`. Nový obsah slovníku ti hned ukážu. To bylo rychlé, co? Minule to dalo daleko víc práce.

Než začneme dneska pracovat, tak si ještě spolu zkusíme tři ovládací příkazy.

První z nich je `SLOVNIK`. Po zadání tohoto příkazu prolistuji všechno co znám. Jak sám víš, když mi zadáš známý příkaz, tak jej hned vykonám. Když jej ale neznám, tak se zeptám – `NOVY PRIKAZ ZNAMENA` – a musíš jej nadefinovat.

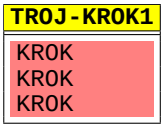
Při psaní se ale dopouštíš mnoha chyb. Proto, aby se daly opravit, mám ve svém slovníku zabudován příkaz `CHYBA`. Po napsání tohoto příkazu začnu rušit postupně jednotlivé příkazy z nově vytvářeného příkazu. Rušit začínám vždy od konce. Proto, když uděláš chybu na začátku, musíš zrušit i všechny příkazy, které následovaly. Když už je nově vytvářený příkaz zrušen, rušíš příkazem `CHYBA` postupně od konce i příkazy, které jsi mi dříve nadefinoval a už je mám uloženy ve slovníku. Základní příkazy – primitiva – ty ovšem zrušit nemůžeš. Proto, že ruším příkazy vždy od konce, je výhodné, když jsou krátké. Nemusíš toho tolik psát znovu.

Pro opravu můžeš použít i příkaz `OPRAV` – zeptám se tě na jméno příkazu, kterých chceš opravit a nabídnu ti jednoduchý textový editor pro opravu příkazu. Opravu ukončíš pomocí `Alt+F4` nebo myší – pak ještě stiskni nějakou klávesu, abych si mohl načíst to, co jsi mi napsal.

Tak – a teď už máme všechno zavedeno a umíme opravovat i chyby příkazem `CHYBA`. Na začátku jsme si slíbili ještě trochu turistiky. Aby se nám dobře chodilo, tak si na to vytvoříme nové příkazy.

Příkaz `KROK` znám od narození, `DVOJ-KROK` už máme, a tak si vytvoříme ještě `TROJ-KROK` a `CTYR-KROK`.

Opět můžeš tyto nové příkazy vytvořit několika způsoby. Ukážeme si to na `TROJ-KROK`u. pro rozlišení si vždy jednotlivé příkazy očíslujeme.

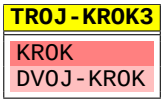



```
TROJ-KROK1 ZNAMENA
  KROK
  KROK
  KROK
KONEC
```

To je ten nejjednodušší způsob. Ty už ale víš, že i když je tohle správný příkaz, můžeš jej napsat jednodušeji:


```
TROJ-KROK2 ZNAMENA
  DVOJ-KROK
  KROK
KONEC
```

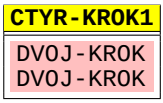
Stejně správný je i tento zápis:




```
TROJ-KROK3 ZNAMENA
  KROK
  DVOJ-KROK
KONEC
```

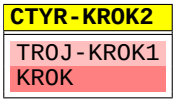
Jak sám vidíš, všechny zápisy jsou správné a přitom `TROJ-KROK2` a `TROJ-KROK3` jsou i z hlediska programování stejné. Mně je to jedno, jestli vykonám napřed `KROK` a potom `DVOJ-KROK` nebo naopak.

Podobným způsobem, ale ve více variantách je možno zapsat i příkaz pro vykonání čtyř kroků. My si uvedeme jenom dvě varianty, protože to bychom za chvíli popsali celou knížku jenom různými variantami jednoho příkazu. Těch je, jak sám vidíš, mnoho a se složitostí příkazu jich přibývá. Tak tedy `CTYR-KROK` můžeme definovat třeba takto:



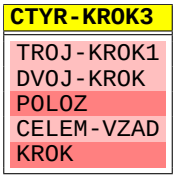

```
CTYR-KROK1 ZNAMENA
  DVOJ-KROK
  DVOJ-KROK
KONEC
```

Nebo takto:




```
CTYR-KROK2 ZNAMENA
  TROJ-KROK1
  KROK
KONEC
```

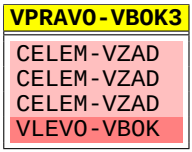
Zato ale následující definice je špatná:




```
CTYR-KROK3 ZNAMENA
  TROJ-KROK1
  DVOJ-KROK
  POLOZ
  CELEM-VZAD
  KROK
KONEC
```

Proč je špatná? Vždyť na tvůj příkaz vykonám požadované čtyři kroky dopředu? Ve skutečnosti sice udělám čtyři kroky dopředu, ale položím při tom i značku a to je úplně něco jiného, než jsme chtěli. A navíc mě necháš jít dopředu o krok víc a pak mě zbytečně vracíš zpátky, a nakonec zůstanu otočen jinak. Proto, když sis tuhle definici vyzkoušel, tak ji raději zruš.

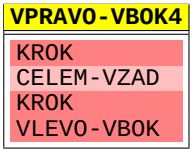
Přitom když někdy vykonávám některé pohyby navíc, nemusí to být na závadu. Zkusíme si to ještě na starém známém příkazu VPRAVO-VBOK, tentokrát s číslem 3:




```
VPRAVO-VBOK3 ZNAMENA
  CELEM-VZAD
  CELEM-VZAD
  CELEM-VZAD
  VLEVO-VBOK
KONEC
```

Tady provedu skutečně vpravo vbok. Ale navíc se ještě jednou otočím kolem dokola. Celkem nic se neděje, jenom mi to trvá trošku déle. To je ale někdy žádaná vlastnost. V některých případech totiž potřebuješ, aby program trval delší dobu a tak musíš použít zbytečné pohyby. Než je ale použiješ, tak si musíš být jistý, že ty pohyby navíc nebudou na závadu.

Ale co další příkaz?



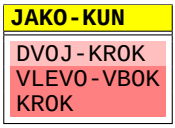

```
VPRAVO-VBOK4 ZNAMENA
  KROK
  CELEM-VZAD
  KROK
  VLEVO-VBOK
KONEC
```

Z hlediska toho, že tím splním svůj úkol, je i tahle definice správná. Ale pozor! Co když budu stát před zdí? No tak to přece pak úkol nesplním. Narazím a zastavím se. Proto definici, která nevyhovuje za všech okolností, nemůžeme považovat za správnou. Z toho všeho pro nás vyplývají dvě zásady:

1. Není důležité, jak je slovo nadefinováno. Důležité je, aby podle této definice za libovolných podmínek splnilo svoji úlohu.
2. Chyby v definici nemusí být na první pohled patrné. Proto když nějaké slovo nadefinuješ, musíš se přesvědčit, že je nadefinováno správně.

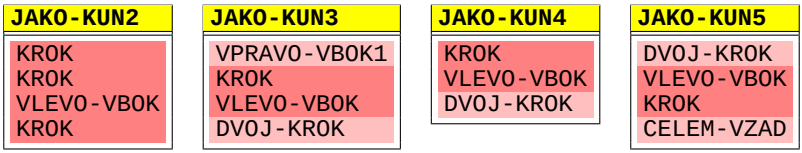
Nechodíme ale jenom rovně. Cesty mají i spoustu zatáček. Proto se teď nakonec spolu ještě naučíme chodit jako kůň. Ne sice ten doopravdový, ale šachový.

Víš jak chodí takový kůň při hře v šachy? Ne? Tak on vždy udělá dva kroky dopředu a krok stranou. Je jedno na kterou stranu. To by zase bylo možností, kdybychom si je chtěli vypsat. Můžeš se o to pokusit doma.




```
JAKO-KUN ZNAMENA
  DVOJ-KROK
  VLEVO-VBOK
  KROK
KONEC
```

Protože těch možností je skutečně hodně, tak si tady uvedeme kopenogramy jen některých.



A to je jen několik příkladů, jak se taková definice může lišit. A všechny jsou správné. Ve všech případech mě postavíš na políčko, na které se může dostat opravdový šachový kůň. My jsme si totiž spolu neřekli, kam se mám dostat. A proto je možné definovat nejen několik řešení s jedním správným výsledkem, ale i několik řešení s několika správnými výsledky.

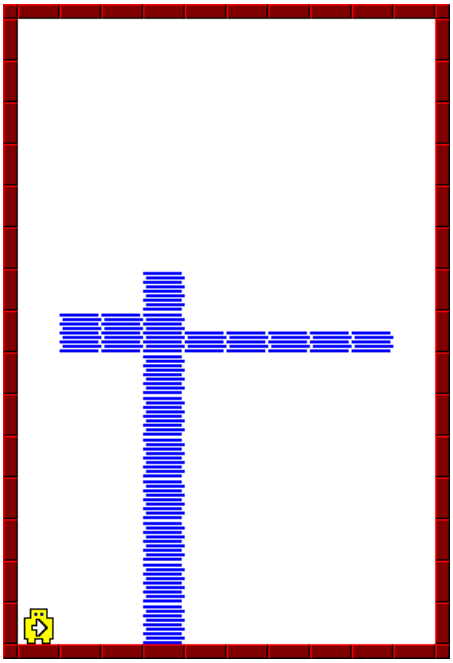
Až jde z toho hlava kolem. Ale vidíš, že to je celkem snadné. A na závěr ti prozradím ještě dva ovládací příkazy. Jedním z nich je `ROZKLAD`. V průběhu dne jsme si nadefinovali několik příkazů. A už jsi jistě zapomněl, jak je utvořen třeba `TROK-KROK2`. Abys to mohl zjistit, k tomu slouží právě `ROZKLAD`. Po jeho zadání ti počítač vypíše dotaz `JAKY PRIKAZ?` Ty teď napiš, jaký příkaz chceš rozložit. A na levou stranu obrazovky se vypíše, jak je příkaz utvořen. `KOPENOGRAM` funguje stejně, jenom místo karlovského zápisu dostaneš kopenogram.

Na závěr si smaž dnešní příkazy a nadefinuj si znovu jenom `TROJ-KROK`, `CTYR-KROK` a `JAKO-KUN`. Kterou z dnešních správných definic si vybereš, to už záleží jenom na tobě. Nakonec všechny příkazy ulož pomocí příkazu `ULOZ` s názvem `turista`. Nezapomeň, že se uloží pouze ty příkazy, které máš právě ve slovníku, a že slovníky nelze později spojovat. Takže pokud chceš mít uloženy všechny příkazy od začátku, tak je nutné je mít všechny zařazeny ve slovníku, který ukládáš.

4. Učíme se pracovat: KAREL montérem

Toulat se po světě, to už nám spolu jde. Ale – jak říkají staří mudrci – „Prací živ je člověk“. Tak se spolu taky naučíme pracovat. A když máme pracovat, tak aby to za něco stálo. Proto si postavíme v následujících kapitolách dům. Pochopitelně jenom na monitoru. Začneme tím nejjednodušším. Budeme napřed spolu vyrábět panely. Ale protože s holýma rukama se těžko něco dělá, tak si vyrobíme jeřáb.

Jak stavební jeřáb vypadá, to víš. Třeba jako ten na obrázku:

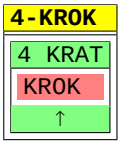


Výkres jeřábu tady máš proto, že než začneme něco dělat, tak si to nakreslíme. Vždyť i opravdový stavař pracuje podle výkresu. Tím, že svoji představu máš před očima na papíře, se ti bude lépe dělat.

Jistě už tě napadlo, jak by se takový jeřáb dal ze značek na obrazovce postavit. My ale podle hesla „nechte stroje, ať to dělají za nás“ použijeme nové primitivum. Je to `OPAKUJ`. Tohle slovo nám velmi usnadní práci.

Kdy jej budeš používat? Vždy tehdy, když budeš některý příkaz chtít opakovat a budeš přesně vědět kolikrát. `OPAKUJ` má totiž tu nevýhodu, že kdybys je použil neuváženě, tak by mohlo

dojít k havárii. Proč? To si ukážeme na příkladu. Zkusíme si pomocí OPAKUJ definovat znovu příkaz CTYR-KROK. Abychom si je odlišili, tak použijeme název 4-KROK:




```
4-KROK ZNAMENA
  OPAKUJ 4 KRAT
    KROK
  KONEC
KONEC
```

Vidíš, že zápis je jednoduchý. Jak při něm postupujeme? Když mi zadáš OPAKUJ, zeptám se tě OPAKUJ KOLIKRAT? Nyní musíš napsat číslo, kolikrát chceš příkaz opakovat. V našem případě 4. No a dále postupuješ, jak už jsi zvyklý. Použití OPAKUJ má ještě jeden problém – špatně se testují podmínky. Hrozí třeba nebezpečí, že při vykonávání narazím do zdi, nebo z jiných důvodů nemohu potřebný počet opakování provést. Jak na tento problém, to si ale vysvětlíme později. Teď si zkus nadefinovat ještě jednou třeba 5-KROK. Řešení máš uvedeno ve slovníku.

Jak vidíš, pomocí OPAKUJ vytváříme takový kruh. Ukazuje to šipka v kopenogramu. Ten kruh – říkáme mu *cyklus* – ale skončí. Netočí se pořád kolem dokola. Ty dokonce přesně víš kdy. V tomto našem případě po čtyřech krocích. Největší takový cyklus, který dovedu, je 255 opakování. To je největší číslo, které jsem schopen si zapamatovat. Ty ale jistě umíš i vyšší čísla a počítač samozřejmě taky.

Takovémuto cyklu říkáme *cyklus se známým počtem opakování*. To proto, že máme i cykly s neznámým počtem opakování. Ale o těch až jindy.

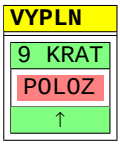
Teď ale zpátky k našemu jeřábu, který budeme pro stavbu potřebovat. Jak sám vidíš na obrázku, jeřáb se skládá z věže a ramena. Napřed si tedy uděláme programy pro VEZ a pro RAMENO. Věž je jednodušší, proto s ní začneme.

Věž je vlastně devět políček nad sebou vyplněných značkami. S pomocí OPAKUJ je to hračka. Vždy vyplníme jedno políčko a pak přejdeme na sousední a to taky vyplníme. To celé opakujeme devětkrát.

Než začneme stavět, tak si musíme celé město vypucovat. Při práci musí být přece pořádek. To sám dobře víš. Vysbírat značky můžeš pomocí příkazu ZVEDNI. To je ale při větším množství zdlouhavé. Já ti poradím kratší způsob.

Poslouží ti k tomu příkaz MESTO. Po jeho zadání zmizím z obrazovky a na její levé straně se ukáží názvy povelů s jejichž pomocí můžeš po městě malovat. Můžeš malovat různé překážky jako zdi, stavět bludiště, pokládat značky. Příkaz Nové město naopak všechno krásně vymaže a pracoviště budeš mít hezky připravené. Stiskem klávesy Esc (označené jako Konec) se opět na obrazovce objevím a můžeme začít.

Podle toho, co už víš, snadno vymyslíš příkaz pro zaplnění celého políčka značkami. Jde jen o to položit pod sebe devět značek. Víc se jich přece na jedno políčko nevejde. To už jsme si řekli. Takový příkaz si nazveme třeba VYPLN.

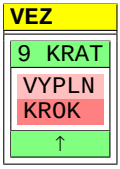



```
VYPLN ZNAMENA
  OPAKUJ 9 KRAT
    POLOZ
  KONEC
KONEC
```

No a máme jedno políčko vyplněné. A postavit VEZ bude znamenat, že musíme VYPLN devětkrát opakovat. Ale to snad na jednom políčku nejde? Správně! Proto mi vždy před každým opakováním dáš příkaz abych popošel o jeden KROK.

Pozor! Řekli jsme si ale, že musíš zabránit tomu, abych narazil. Proto než začnu stavět VEZ, tak mě musíš natočit tak, aby přede mnou byl dostatek volného místa. O to se musíš postarat zatím sám.

Když už máš vše připraveno, tak začneme stavět VEZ. Vyplnit políčko už umíš, krok taky. Takže je jenom spojíme. A VEZ je tady.



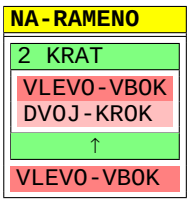

```
VEZ ZNAMENA
  OPAKUJ 9 KRAT
    VYPLN
    KROK
  KONEC
KONEC
```

A kus je hotovo. RAMENO – to je obdobné, jen musíš vyplňovat políčka vedle sebe. Protože ale začínáš tam, kde končí VEZ, to je *nad* vrcholkem, musíš ještě přejít. Podle plánu vidíš, že aby se dostal na výchozí pozici pro stavbu ramena, tak musíš udělat VLEVO-VBOK, DVOJ-KROK, VLEVO-VBOK, DVOJ-KROK, VLEVO-VBOK.

Když se na tenhle popis podíváš, tak vidíš, že se tam opakuje VLEVO-VBOK, DVOJ-KROK. Ejhle! A máme zase použití pro OPAKUJ. Potom nám ale zbyde jeden VLEVO-VBOK. Ten musíme dodat za OPAKUJ.

Teď už nám dojít na určené místo nedělá žádný problém. Máme tak nový příkaz, kde je návaznost přímá, ale kromě toho i jeden cyklus se známým počtem opakování. Nazveme si jej NA-RAMENO.

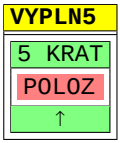
Název příkazu zadávej vždy tak, aby i v tom jednoduchém a krátkém názvu bylo ukryto pokud možno co nejpřesněji označení činnosti. Když zvolíš třeba název NA-MISTO, tak se ti bude za chvíli plést kam máš dojít.




```
NA-RAMENO ZNAMENA
  OPAKUJ 2 KRAT
    VLEVO-VBOK
    DVOJ-KROK
  KONEC
  VLEVO-VBOK
KONEC
```

A koukej, jak je zápis kopenogramem přehlednější a jednodušší. Jak už sis všimnul, aby se nám kopenogramy líbili, tak si je děláme barevné. Zvlášť si označujeme hezky barvou každý druh činnosti. Jak vidíš, tak cyklus s určitým počtem opakování označujeme zeleně. To podle trávníku, na kterém se můžeme hezky proběhnout.

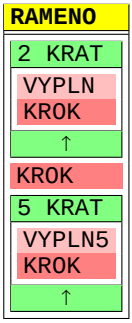
Ale aby ses nezaběhnul! Ještě musíme setrojit `RAMENO`. To už je složitější. Podle výkresu vidíš, že je tenčí. Je tam jenom pět značek. Proto máš příkaz `VYPLN5`:




```
VYPLN5 ZNAMENA
  OPAKUJ 5 KRAT
    POLOZ
  KONEC
KONEC
```

Na konci ramene je závaží. To je vlastně devět značek. Ale přes věž, ve spoji, už značky pokládat nemůžeš. To bych se ozval, že `Není kam položit`. Proto když dojdeš k věži, tak musíš udělat další krok. Protože se spolu ještě neumíme chovat podle okolností, tak je to zase na tobě. Napiš proto příkaz, kde vyplním dvě políčka devíti značkami, udělám krok navíc a pak vyplním pět políček třemi značkami.

A hele, vždyť je to lehké! A naráz mám v jednom příkazu spojené dva cykly za sebou. Takovému spojení se taky říká *postupné*. To, že ti to zdůzarňuji, znamená, že cykly můžeme spojovat taky jinak. Jak, to až za chviličku.




```
RAMENO ZNAMENA
  OPAKUJ 2 KRAT
    VYPLN
    KROK
  KONEC
  KROK
  OPAKUJ 5 KRAT
    VYPLN5
    KROK
  KONEC
KONEC
```

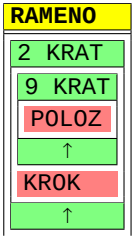
Na první pohled vidíš, že tento příkaz jsou vlastně dva příkazy spojené do jednoho. Ale je tu i další spojení cyklů, nejenom postupné. Zkus si to vedle na papíře rozkreslit.

Nedívej se ale dál. Zkus si to sám. Příště se k tomu vrátíme.

5. OPAKUJ – budeš chytřejší

Tak jak dopadlo rozkreslení ramene? Pokud jsi to náhodou nezvládl, tak použij následující postup:

Místo VYPLN použij jenom primitiv. Začátek příkazu RAMENO by pak vypadal takto:

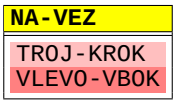



```
RAMENO ZNAMENA
  OPAKUJ 2 KRAT
    OPAKUJ 9 KRAT
      POLOZ
    KONEC
    KROK
  KONEC
```

Takovému spojení cyklů říkáme *cyklus vnořený.* A protože jsi žáček pozorný, tak už sis jistě všimnul dvou věcí. Jednak tento příkaz není úplný. Na počítači by nefungoval. Proč? Protože počítač by stále čekal na ukončení. V kopenogramu chybí ukončující čára a v karlovském zápisu musí začínat poslední příkaz pod začátkem hlavy programu. Je to jako v matematice se závorkami. Tady ZNAMENA a OPAKUJ znamenají totéž co levé závorky, KONEC co pravé závorky. A ty už víš, že jich musí být stejný počet.

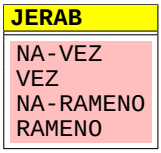
A je ti taky doufám jasné, jak program běží. Pro jistotu to slovně popíšu. Jednotlivé řádky programu si můžeš představit jako obálky, kde je vložen popis příkazu. Podle programu budu pracovat tak, že otevřu první obálku. Tu si otevřu v okamžiku, kdy stiskneš Enter. Je na ní napsáno RAMENO je v ní napsáno, že první příkaz je v obálce 2. V obálce 2 je napsáno, že mám udělat dvakrát příkazy z obálky 3. Do svého zápisníku si proto napíšu poznámku – obálka 3 a udělám dvě čárky a dvojku v obálce škrtnu. Po otevření obálky 3 se ale dozvím, že mám udělat devětkrát příkazy z obálky 4. Proto si poznamenám do zápisníku k obálce 3 devět čárek a devítku v obálce škrtnu. Otevřu obálku 4 a přečtu si příkaz POLOZ. Tomu rozumím a tak položím značku. Pak si otevřu obálku číslo 5. Tam mám lísteček, že se mám vrátit k obálce 3. Podívám se do zápisníku, kde jsem obálku 3 položil a vidím u ní pět čárek. Jednu si hned umažu a podívám se do obálky 3. Tam mám napsáno, abych vykonal příkaz z obálky 4. Otevřu tedy obálku 4 … A to tak dlouho, celkem devětkrát, až mám všechny čárky, které jsem si napsal k obálce 3, přeškrtlé. Pak otevřu obálku 5. Tam mám napsáno KROK. Udělám krok a otevřu obálku 7, kde je napsáno, abych se podíval do obálky 2. Proto si jednu čárku v zápisníku u obálky 2 škrtnu a jdu se podívat do obálky 2. Tam je napsáno, abych vykonal příkaz z obálky 3. Otevřu obálku 3, do zápisníku poznamenám devět čárek a jdu dál … Tak se zase dostanu k obálce 7, přeškrtnu si druhou čárku. A teď nevím co dál, protože už nemám další obálku, která by mi řekla JSI HOTOV – KONEC. Příkaz není úplný, jak jsme si řekli.

Teď už se ale musíme pustit do toho naplánovaného jeřábu. Umíme VEZ, NA-RAMENO a RAMENO. Ještě nám zbývá NA-VEZ – abych se dostal z domečku na vhodné místo pro stavu jeřábu. Je třeba udělat tři kroky (TROJ-KROK) a otočit se vlevo:




```
NA-VEZ ZNAMENA
   TROJ-KROK
   VLEVO-VBOK
KONEC
```

Tak – a máme vše potřebné pro JERAB: teď to jen správně seřadit:




```
JERAB ZNAMENA
   NA-VEZ
   VEZ
   NA-RAMENO
   RAMENO
KONEC
```

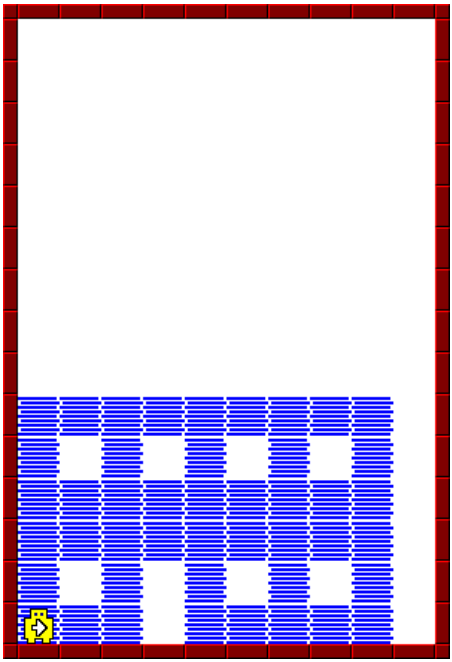
6. Panelárna

Tak co, nebolí tě ruce, jak jsme stavěli jeřáb? To byla už nějaká práce! Jen si vzpomeň, jak se ti to nejprve zdálo složité. A nakonec jsi to zvládnul na jedničku. Je to tím, že už toho spolu hodně dovedeme. Umíš už používat základní primitiva jako je `KROK`, `VLEVO-VBOK`, `ZVEDNI`, `POLOZ` a `KONEC`. A co je rovněž důležité, umíš si problém rozložit, čili – jak odborně říkáme – provést *dekompozici problému*.

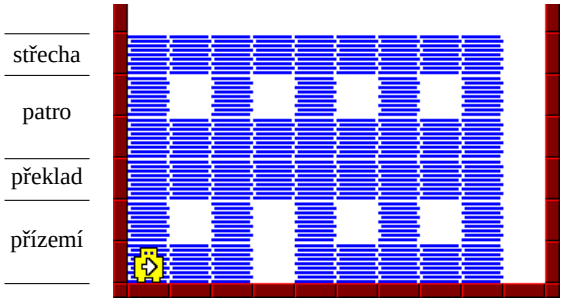
A teď s chutí do další práce. Pustíme se do stavění domu. Nejprve se pustíme do jeho přípravy. Začneme nejdůležitější věcí – takzvanou dekompozicí problému.

Náš postup, i když směřuje k jednomu cíli a má jeden výsledek, se totiž zásadně liší. Ty při řešení nějakého problému musíš postupovat od nejsložitější věci směrem k nejjednodušším, které už umíš zvládnout. Já ale musím každou úlohu řešit směrem od nejjednodušších věcí k nejsložitější – k výsledku úlohy. Jinými slovy, když řešíš úlohu ty, tak provedeš rozklad zadání až na primitiva. Já pak zase z jednotlivých primitiv složím dohromady celkové řešení.

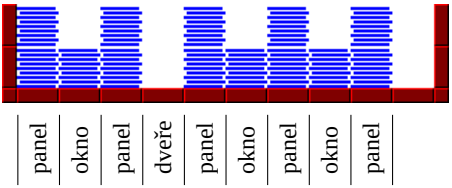
Ale abych ti tu nevykládal jen samé nudné definice. Ukážeme si to na příkladu. Chceme postavit dům. Vybereme si pro začátek nějaký jednoduchý. Například tento:



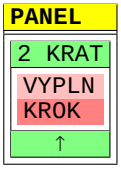
Vidíš, že se jedná o skutečně velmi jednoduchý domek. Pochopitelně bych mohl zvolit i složitější, ale pro začátek nám stačí tento. Postavit dům – to je náš úkol. Jenže jak začít? Postavit celý domek najednou, to neumíme. Proto si dům rozdělíme na jednotlivé díly. Vidíš, že se skládá z přízemí, překladu, patra a střechy:



Ale i tyto části jsou zatím pro tebe příliš složité, abys je rovnou naprogramoval. Zkusíme to jinak. Vzpomeň si jakým způsobem se staví velké domy. Už jsi na to přišel? Správně! Rozloží jednotlivé části – přízemí, překlad, patro a střechu – na jednotlivé prvky – panely. Když se podíváš na obrázek domu, tak vidíš, že nejsložitější částí je přízemí. V přízemí jsou jak panely plné, tak i okna a dveře. V ostatních dílech domu se nevyskytuje žádný prvek, který není použit v přízemí. Stačí, když si spolu rozložíme přízemí.

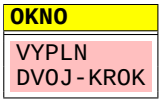


No a takové příkazy už přece dovedeš naprogramovat. Vzpomeň si na JERAB. Napřed ale spolu dokončíme rozklad až do konce. Začneme s panelem. Panel se skládá ze dvou políček nad sebou vyplněných značkami. Opět pro jednoduchost budeme předpokládat, že stavíme na čistém staveništi, že není na stavbě žádný nepořádek. Panel – to je vlastně taková věž, která je vysoká jenom dvě políčka. tady je:




```
PANEL ZNAMENA
  OPAKUJ 2 KRAT
    VYPLN
    KROK
  KONEC
KONEC
```

Okno je pro tebe složitější. Vidíš, že se skládá z jednoho políčka plného a jednoho prázdného. Takže tady nemůžeš použít OPAKUJ. Tedy vlastně můžeš, ale nemělo by to smysl. Mohl bys opakovat KROK, ale to je zbytečné, protože už máš nadefinovaný příkaz DVOJ-KROK.




```
OKNO ZNAMENA
  VYPLN
  DVOJ-KROK
KONEC
```

Nic zde neopakujeme. Jedná se jenom o následné spojení dvou již známých příkazů. Můžeš se ptát, proč vždy vycházím až nad příslušný panel? Vždyť by tady stačilo, abych udělal jenom jeden krok. Ale vzpomeň, jak jsme si řekli, že se budeme snažit o co největší jednotnost a jednoduchost. Zatím nezáleží na době trvání příkazu a tak nám ten jeden KROK navíc nevadí. A taky nezapomeň, že vždy po postavení panelu musíš přejít na místo, odkud budeš stavět nový panel. Proto je pro nás výhodné, když bude co největší jednotnost.

Posledním prvkem – panelem, který nám schází, jsou dveře. Dveře jsou dvě prázdná políčka. My bychom je tedy mohli jednoduše vynechat. Ale protože jsme si řekli o jednotnosti, tak je nadefinujeme jako DVOJ-KROK:


```
DVERE ZNAMENA
  DVOJ-KROK
KONEC
```

Je to vlastně jeden a ten samý příkaz, ale má jiné jméno.

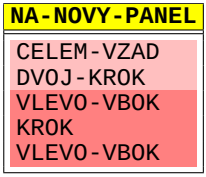
7. Stavíme dům

Ano a za chvíli budeme slavnostně vztyčovat glajchu. Už máme spolu dům rozložený na jednotlivé panely, dokonce jsme si je posledně stačili i vyrobit. Tím jsme provedli dekompozici problému.

Ale ještě nám kus zbývá. Vždyť my máme zatím vyrobeny vlastně jenom takové kostky! A zkus dát svojí dvouleté sestřičce kostky ať z nich postaví dům! Nepostaví ho. Neví jak. Zrovna tak ani já ještě nevím, jak bych postavil dům. Vím jak dům vypadá. Umím si i vyrobit potřebné kostky – panely. Jenže jak dál? Vzpomeň si na jeřáb. Tam jsme taky museli spolu cestovat na montážní místo. Proto si i teď musíme napřed nadefinovat přechod na nové místo.

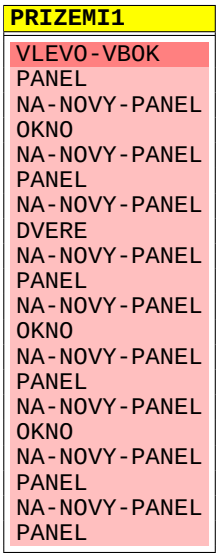
Zůstaneme zatím ještě u přízemí. Když začínáme stavět, tak já stojím v novém městě ve výchozí pozici. To znamená, že město je čisté a uklizené a já jsem otočen podle spodní zdi. Napřed se tedy musím otočit nahoru. To je snadné – to provedu jenom `VLEVO-VBOK`.

Pak postavím `PANEL`. Ale my chceme postavit přízemí. Proto musím sejít dolů, udělat krok stranou a opět se otočit nahoru. Když skončím `PANEL`, provedu `CELEM-VZAD` a pak mě musíš nechat udělat `DVOJ-KROK`. Tím se dostanu na místo, odkud jsem začal. Ale ty potřebuješ, abych se dostal do nové výchozí pozice! To znamená, že musím stát na vedlejším políčku otočený opět nahoru! To tedy udělám `VLEVO-VBOK`, `KROK`, `VLEVO-VBOK` – a je to.




```
NA-NOVY-PANEL ZNAMENA
  CELEM-VZAD
  DVOJ-KROK
  VLEVO-VBOK
  KROK
  VLEVO-VBOK
KONEC
```

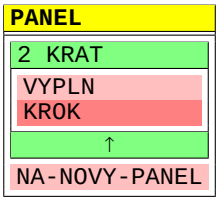
Teď můžeme už spolu postavit přízemí. Takhle bys to jistě svedl i bez nápovědy:

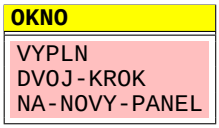


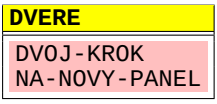

```
PRIZEMI1 ZNAMENA
  VLEVO-VBOK
  PANEL
  NA-NOVY-PANEL
  OKNO
  NA-NOVY-PANEL
  PANEL
  NA-NOVY-PANEL
  DVERE
  NA-NOVY-PANEL
  PANEL
  NA-NOVY-PANEL
  OKNO
  NA-NOVY-PANEL
  PANEL
  NA-NOVY-PANEL
  OKNO
  NA-NOVY-PANEL
  PANEL
KONEC
```

No ale takový příkaz? Vždyť má 19 řádků! To se pomalu ani nevejde na obrazovku! Když se na příkaz podíváš, tak vidíš, že nejvíce se tam vyskytuje příkaz `NA-NOVY-PANEL`. Nejjednodušeji se ho zbavíme, když si znovu nadefinujeme příkazy pro jednotlivé panely a zabudujeme do nich i příkaz pro přechod na nové pracoviště.

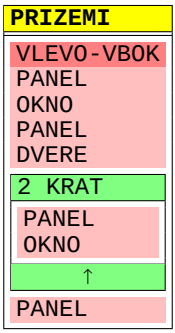
Proto pomocí příkazu `CHYBA` vymaž `NA-NOVY-PANEL`, `DVERE`, `OKNO` a `PANEL` a znovu si je nadefinuj. Ale pozor na pořadí: protože v jednotlivých panelech chceme použít i `NA-NOVY-PANEL`, musíme jej nadefinovat nejdříve.







Další možné zjednodušení je, že se za dveřmi dvakrát za sebou vyskytuje `PANEL` a `OKNO`. proto můžeme tyto dva příkazy nechat opakovat. Přízemí pak vypadá takto:




```
PRIZEMI ZNAMENA
  VLEVO-VBOK
  PANEL
  OKNO
  PANEL
  DVERE
  OPAKUJ 2 KRAT
    PANEL
    OKNO
  KONEC
  PANEL
KONEC
```

No, sice ani toto není moc hezky napsaný příkaz, ale my jej zatím líp neumíme. Ostatní příkazy nám už půjdou líp.

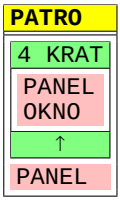
Nad přízemím je překlad. To je celá řada značkami vyplněných políček. Vidíš, že jich je celkem devět vedle sebe. My už vlastně takový příkaz máme a dokonce i s přechodem na nové pracoviště! Je to `OKNO`!

Tím, že uděláme krok, nakreslím sice prázdné políčko, ale to nevadí. Vždyť i do takového políčka já mohu značky pokládat. A protože teď ze začátku nám nezáleží na rychlosti, tak použijeme `OKNO` na sestrojení překladu. Nevadí, že je to jiný příkaz. Nám vyhovuje. Je správný, protože s ním nakreslíme překlad. A to tak, že mě necháš postavit devět oken vedle sebe. To je přece krásný příklad pro `OPAKUJ`:


```
PREKLAD ZNAMENA
  OPAKUJ 9 KRAT
    OKNO
  KONEC
KONEC
```

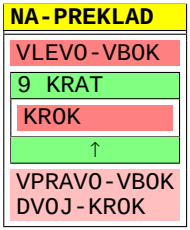
Ale tímto jedním příkazem máme zároveň nadefinovanou i střechu! Vždyť střecha není nic jiného než překlad nad patrem. Proto nám zbývá už jenom vytvořit definici pro `PATRO`.

Na patře, tam mám čtyři okna, která sousedí s panely. Proto použijeme nám již dobře známé `OPAKUJ`. Ale ten jeden panel pak ještě musíme přidat! Na to nezapomeň!




```
PATRO ZNAMENA
  OPAKUJ 4 KRAT
    PANEL
    OKNO
  KONEC
  PANEL
KONEC
```

A jednotlivé díly domu jsou hotovy. Teď už je zbývá jenom pospojovat. Nezapomeň, že já skončím vždy na konci a teď se potřebuju dostat na další pozici. My si pro jednoduchost nadefinujeme dva příkazy: `NA-PREKLAD` a `NA-PATRO`. Jejich konstrukce už ti je jistě tak jasná, že je nemusím ani komentovat.

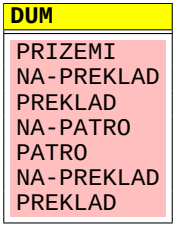



```
NA-PREKLAD ZNAMENA
  VLEVO-VBOK
  OPAKUJ 9 KRAT
    KROK
  KONEC
  VPRAVO-VBOK
  DVOJ-KROK
KONEC
```

A `NA-PATRO` je příkaz obdobný:


```
NA-PATRO ZNAMENA
  VLEVO-VBOK
  OPAKUJ 9 KRAT
    KROK
  KONEC
  VPRAVO-VBOK
  KROK
KONEC
```

Takže teď už znáš všechno potřebné pro stavbu domu. Náš první dům vypadá takto:




```
DUM ZNAMENA
  PRIZEMI
  NA-PREKLAD
  PREKLAD
  NA-PATRO
  PATRO
  NA-PREKLAD
  PREKLAD
KONEC
```

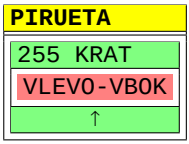
To byla ale dřina! To proto, že ještě některé věci neznáme. Ty se spolu naučíme v následujících kapitolách. Až dojdeme na konec knížky, tak si zkus dům postavit znovu. Uvidíš, že to bude snadnější. Ale i takto jsme zvládli kus práce a základy programování.

8. Karlova přítelkyně

Nepochybuji, že už sis mě nahrál do počítače, a tak dovol, abych ti dnes představil svoji nejmilejší přítelkyni. Je to slečna *rekurze*.

Na nose ti vidím i z obrazovky, že jsi na ni zvědavý. Máš být na co. Rekurze je jedním z nejzajímavějších jevů ve výpočetní technice. Co to vlastně rekurze je? Když ti to prozradím učeně, tak je to definování příkazu pomocí sebe samotného. Teď si to vysvětlíme jednodušeji. Dosud jsme spolu definovali nové příkazy pomocí známých příkazů. Například VPRAVO-VBOK jsme si nadefinovali pomocí VLEVO-VBOK. U rekurze použijeme ale nové, neznámé slovo hned v jeho definici. Je to asi tak, jako když řekneš, že školní rok je období mezi začátkem a koncem školního roku. Když tě uslyší nějaký logik, tak se hrozně vyděsí a do žákovské knížky ti napíše velikou pětku několikrát podtrženou. Poučí tě, že ses dopustil chyby definice v kruhu. Nám ale o to jde! Potřebujeme něco, co by mě dostalo do nějakého začarovaného kruhu, kde bych se točil pořád dokola jako na kolotoči.

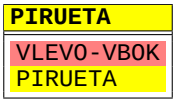
Ukážeme si ale raději příklad. Představ si, že mě chceš roztočit jako krasobruslaře při piruetě. Budu se tedy otáčet pořád doleva. Zatím bys to uměl zapsat třeba pomocí příkazu OPAKUJ takto:




```
PIRUETA ZNAMENA
  OPAKUJ 255 KRAT
    VLEVO-VBOK
  KONEC
KONEC
```

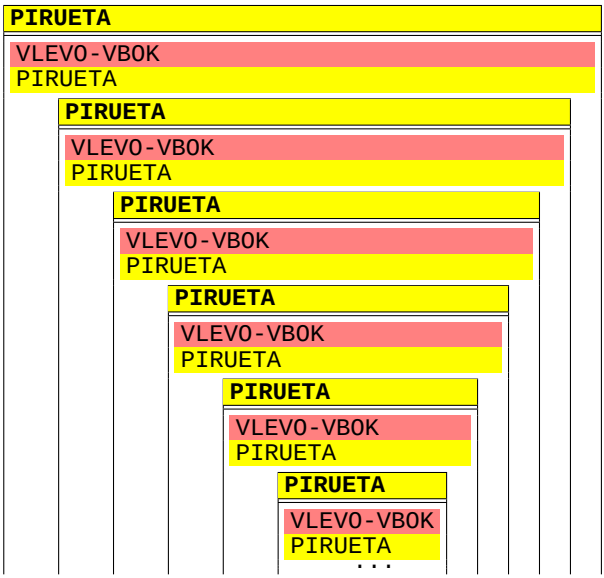
Tady bych ale vykonávání příkazu po chvíli přerušil. Když chceš, abych se točil déle, tak musíš použít místo VLEVO-VBOK příkaz VPRAVO-VBOK. To zní divně, viď, ale jen si vzpomeň, že VPRAVO-VBOK máme definován pomocí tří VLEVO-VBOKů. Ale zase se za chvíli zastavím. Jak mě roztočit trvale? Nejlépe to jde pomocí rekurze.

Zápis pomocí rekurze by vypadal takto:




```
PIRUETA ZNAMENA
  VLEVO-VBOK
  PIRUETA
KONEC
```

Vypadá to divně, co? Ale vzpomeň si na cyklus se známým počtem opakování. Nebo se podívej, jak jsme si rozepsali moji činnost při OPAKUJ. A tady to je něco podobného. Ještě lépe to pochopíš, když ti kopenogram rozepíši.



Všimni si jedné věci. V kopenogramu vybarvujeme rekurzi jako hlavu. To znamená žlutou barvou. Proč? No přeci proto, že rekurze je vlastně hlava nového podprogramu. Tady je správně analogie s `OPAKUJ`.

Jak tedy vlastně při rekurzi pracuji? To je jednoduché. Zadáš mi příkaz `PIRUETA`. Já se při každém příkazu, který není primitivem, podívám do slovníku, co vlastně znamená, co mám provádět. Tam si přečtu, že při příkazu `PIRUETA` mám provést `VLEVO-VBOK`. Provedu `VLEVO-VBOK`. Jako další příkaz mám provést `PIRUETA`. Ale to nevím, co znamená. Tak se musím znovu podívat do slovníku. Pochopitelně víš, že před každým podíváním – odskočením, si musím udělat čárku, poznamenat do svého zápisníku adresu – abych věděl, kam se mám vrátit. A tak to probíhá donekonečna, dokud se nevyčerpám.

Řekli jsme si, že je určitá podobnost mezi rekurzí a opakováním. Z našeho výkladu je vidět. Ale ve skutečnosti se podstatně od sebe liší! Rekurzi můžeme přirovnat k tvému obrazu jak se vidíš, když si stoupneš mezi dvě zrcadla. A opakování, to je zkrátka opakování. Paní učitelka tě nechá taky opakovat za domácí úkol násobilku, když ji neumíš. Opakování násobilky ale ukončíš v okamžiku, když ji umíš. Nebo jestli jsi měl opakovat písemně, tak když ji napíšeš třeba desetkrát. Ale svoje obrázky v zrcadlech vidíš stále. Ty nekončí. Je jich nekonečně mnoho.

Proto abys mě alespoň na obrazovce mohl zastavit, tak při provádění každého příkazu napiši nápis `Zastavím se když stiskneš Esc.` Pokud mě sám nezastavíš, tak jsou jenom dvě možnosti. Buď se program zhroutí, anebo budu provádět `PIRUET`u donekonečna. Jestli jsi dával pozor, tak už to určitě víš. Jestli ne a chceš to vysvětlit, tak dočti tuto kapitolu až do konce. Pokud ne, začni číst novou.

Proč bych se měl při provádění rekurze zastavit? Příčina je jednoduchá: Zaplním si pamět návratovými adresami. Rekurze v našem případě totiž není nekonečná, ale pouze jakoby nekonečná – čili odborně kvazinekonečná.

Na začátku jsme si řekli, že součástí počítače jsou obvody, které si pamatují tvoje příkazy. To znamená, že je to něco jako sešit. Při zapnutí je čistý, bílý. Pak si ale do tohoto sešitu namaluješ mě – Karla. Tím už ten sešit tak čistý není, něco jsi popsal. Ale ještě máš nějaké čisté stránky – volnou paměť. Já teď dostanu příkaz, abych prováděl rekurzi. Například `PIRUETA`. To slovo já ale neznám. Proto si musím nalistovat ve svém programu, co mám provádět při neznámém příkazu. Ale abych věděl, kam se mám vrátit, tak si musím poznamenat stránku a řádek, kde jsem právě byl. Je to stejné, jako když si čteš knížku ty a maminka ti řekne umyj nádobí. Vezmeš si papírek a poznamenáš, kam jsi knížku dočetl. Já tedy ve slovníku najdu příkaz `PIRUETA`. Přečtu si, že mám vykonat `VLEVO-VBOK`. Když jej ale vykonám, tak narazím opět na `PIRUETA`. Co teď? Opět se musím podívat do slovníku. Já už to zapomněl. Proto si poznamenám na volné místo, kam jdu. A opět se jdu podívat, co mám dělat. No a tebe kdyby maminka stále někam posílala, tak si za chvíli papírek popíšeš a nebudeš vědět, kde jsi přestal číst. Já vím, že ty si sice vezmeš nový papír. Ale já nemohu. A vygumovat poznámky si také nemohu. Vždyť to bych se pak nedostal zpátky. Neuměl bych se vrátit. To je jako ty kdyby sis poznámky vygumoval, tak bys nevěděl, na které stránce tě maminka poprvé někam poslala. Já proto přestanu pracovat. Na mém programu záleží, jestli je uzpůsobený tak, že tě na tuto okolnost upozorní tím, že oznámí `Už mě to nebaví`, nebo jestli se zhroutí – přestane pracovat program.

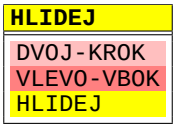
Někdy jsem ale naprogramován tak, že když zjistím rekurzi *na konci příkazu*, tak to znamená, že už se nemusím nikam vracet – že si nemusím zapisovat, kam se mám vrátit, protože jsem už všechnu práci na daném příkazu vykonal – pak už je tam jenom `KONEC`. A tak se mohu točit donekonečna.

Tak, a to je ta celá věda. Podstatné přitom je, že já si dovedu podle svých poznámek, které přitom sesbírám – vygumuju – najít místo, odkud jsem vyšel, vrátit se. Proto se mým poznámkám taky říká návratové adresy.

9. Seskakujeme z kolotoče

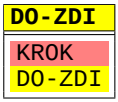
Tak co, nezatočila se ti hlava z toho věčného otáčení při příkazu `PIRUETA`? Já se svojí kamarádkou dokáži i jiné věci. Než se naučíme, jak se rekurzivní kolotoč opouští, tak si ještě ukážeme některé příkazy s rekurzí.

Dáš mi za úkol třeba hlídat nějaký předmět na pomyslném dvorku uvnitř města. Abych jej uhlídal, musím obcházet kolem dokola. Procházet se i zahýbat už umíme. Ale teď máme i rekurzi, takže můžeme i hlídat.




```
HLIDEJ ZNAMENA
  DVOJ-KROK
  VLEVO-VBOK
  HLIDEJ
KONEC
```

Při provádění příkazů musíš dát pozor, abych nenarazil do zdi. To bych vykonávání příkazu přerušil a oznámil ti – `Narazil jsem! Au!` – protože mě to bolí. Nevěříš? Tak si to zkus na příkladu, kdy dojdu až ke zdi a zastavím se tím, že do ni narazím.

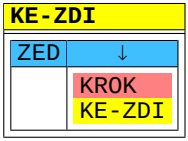



```
DO-ZDI ZNAMENA
  KROK
  DO-ZDI
KONEC
```

K tomu, abych nenarážel, slouží jiné příkazy. Při nich se už musím rozhodovat. Pokusme se spolu takový příkaz vytvořit.

Vytvoříme příkaz, kdy dojdu až ke zdi a před ní se zastavím. Nazveme jej `KE-ZDI`. Vždy před každým krokem zeptám, jestli je přede mnou na dalším políčku zeď. To mi moje robotí čidla umožní.

Tak a teď se musím jen rozhodnout. K tomu slouží třetí skupina primitiv. Říkáme jim *podmiňující*. Jedno z nich – `OPAKUJ` – už znáš. Jak tedy nadefinovat `KE-ZDI`? Třeba takto:




```
KE-ZDI ZNAMENA
  KDYZ JE ZED
  KONEC JINAK
    KROK
    KE-ZDI
  KONEC
KONEC
```

Vidíš, jak je zápis v kopenogramu přehlednější? Ale napřed si jej ještě vysvětlíme. Když odešleš do počítače (pomocí klávesy `Enter` ↵) příkaz `KDYZ`, tak se tě zeptám, zda `KDYZ JE` nebo `KDYZ NENI`. Po tvé odpovědi `JE` (nebo `NENI`, ale v našem případě `JE`) se tě zeptám na podmínku, kterou mám testovat. Můžeš napsat `ZED` nebo jenom `ZE`.

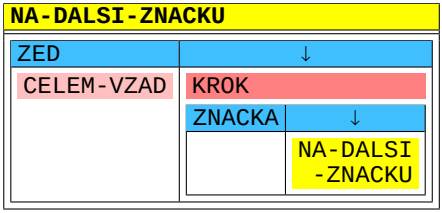
Já pak při provádění příkazu narazím na příkaz KDYZ. Zastavím se a zkoumám, jestli je následující podmínka – v našem příkladě JE ZED – splněna. Na základě tohoto zjištění se rozhodnu. Pokud je splněna, vykonám levou část kopenogramu – v tomto případě neudělám nic. Pokud ale podmínka splněna není, vyberu si možnost JINAK (v kopenogramu je vyznačena šipkou dolů). Důležité je, že ať se rozhodnu jakkoliv, obě cesty se vždy nakonec spojí. Ale podle toho, kterou cestou si podle pravdivosti nebo nepravdivosti podmínky vyberu, se liší činnost, kterou budu vykonávat.

Protože rozhodování je činnost velmi důležitá, vyznačíme ji výrazně i v kopenogramu. Rozhodovací záhlaví vybarvujeme výrazně – zvolíme si proto modrou barvu. Ta nám vyniká.

Do záhlaví vždy vypíšeme testovanou podmínku. V našem případě ZED. Je-li tato podmínka splněna, pokračujeme tou částí programu, která je umístěna v kopenogramu pod podmínkou. V našem případě tam není nic. Jdeme tedy na konec činnosti. Když podmínka splněna není, přesuneme se vpravo, kde šipka ukazuje, kudy máme pokračovat. Konec bloku odpovídá místu, kde se obě cesty sbíhají.

Podívej se ale, co všechno se dá s KDYZ dokázat. Zkusíme příkaz NA-DALSI-ZNACKU. Máš za úkol nadefinovat mi příkaz, když stojím na značce, abych šel směrem, kam jsem natočen. A to do doby, než dojdu na další značku. Tam se mám zastavit. V případě, že dojdu dřív ke zdi, udělám čelem vzad a zastavím se taky. Brrrr. To už je pěkně složité.

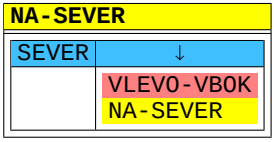
Tady budu muset vykonávat nějakou činnost, ať půjdu kteroukoliv cestou. A přitom testovat dvě podmínky. Za prvé, je-li přede mnou zeď, to už znám. Ale za druhé, je-li pode mnou značka. Ale tuto podmínku mohu testovat, až udělám první krok! Vždyť na značce stojím. Podívej se, jak tedy takový příkaz vypadá.




```
NA-DALSI-ZNACKU ZNAMENA
  KDYZ JE ZED
    CELEM-VZAD
  KONEC JINAK
    KROK
    KDYZ JE ZNACKA
    KONEC JINAK
      NA-DALSI-ZNACKU
    KONEC
  KONEC
KONEC
```

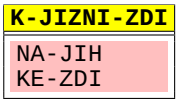
Jak příkazem – a tady už skutečným programem – procházím, je ti teď podle kopenogramu jasné.

Další podmínky, které umím testovat, jsou světové strany. Na obrazovce, stejně jako na mapě, je SEVER nahoře, ZAPAD vlevo, JIH je dole a VYCHOD vpravo. NA-SEVER mě pak můžeš otočit tak, že mě necháš provádět VLEVO-VBOK pomocí rekurze a zastavíš mě, když bude SEVER:



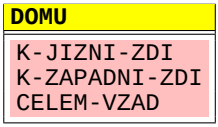

```
NA-SEVER ZNAMENA
  KDYZ JE SEVER
  KONEC JINAK
    VLEVO-VBOK
    NA-SEVER
  KONEC
KONEC
```

A natočit mě na jakoukoliv jinou světovou stranu je pro tebe už hračka. Kdybys to náhodou nevěděl, tak příkazy jsou ve slovníku.




```
K-JIZNI-ZDI ZNAMENA
  NA-JIH
  KE-ZDI
KONEC
```

Kdybys chtěl takový příkaz nadefinovat rovnou, jó pane, to by byl problém. Zrovna tak je lehké si vytvořit příkazy kterými bych došel k ostatním zdem. A teď už nemůžeme zabloudit! Jak to? No přece už spolu vždy trefíme domů. Protože když půjdeme třeba k jižní zdi a až k ní dojdeme, dáme se k západní zdi, tak jsme doma. A tento příkaz musí fungovat, ať budeme kdekoliv ve městě. Ovšem pokud mi nepostavíš schválně nějakou zeď do cesty. Vzpomínáš, jak jsme spolu stavěli dům? Tam jsme to potřebovali. A slíbili jsme si později lepší řešení. Tak tady je.

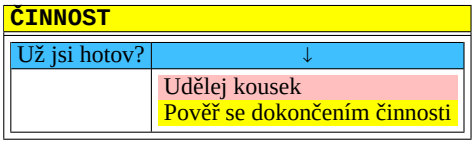



```
DOMU ZNAMENA
  K-JIZNI-ZDI
  K-ZAPADNI-ZDI
  CELEM-VZAD
KONEC
```

A jsem doma. Ale jak se dívám, ty ještě ne. Teď ti zrovna vrtá hlavou, jaký je rozdíl mezi rekurzí a `KDYZ`. Co to vlastně to naše seskočení z kolotoče je?

To si ukážeme na následujícím kopenogramu. Nadefinujeme si něco, co nebudu nikdy provádět. Nadefinujeme si jakoukoliv činnost. Když ji budeme provádět jen pomocí rekurze, tak nám činnost skončí buď v kvazinekonečnu, nebo na chybě. Vzpomeň si na příkaz `DO-ZDI`. Tam jsi nejpozději v desátem (nebo patnáctém – podle rozměru města) se mnou třískl do zdi, až se mi v nosní elektronce zajiskřilo.

Oproti tomu když testuji vstupní podmínku, tak nevím, jak dlouho budu činnost provádět. To je rozdíl proti cyklu se známým počtem opakování (`OPAKUJ`). Testuji podmínku do té doby, než zjistím, že je splněna. Pak musím jít nazpět. A hele, jak se mi teď hodí, že jsem si do notýsku psal návratové adresy. Teď si je tedy hezky posbírám, přitom si notýsek vygumuju a můžu pokračovat dalším příkazem. V kopenogramu je to hezky znázorněné.



Tento kopenogram si můžeš vedle rozkreslit podobně, jako jsme si spolu rozkreslili rekurzi. Ještě lépe tak pochopíš, jak vlastně pracuji.

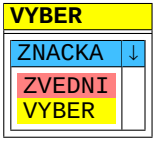
10. Stáváme se sběrateli

Tak už jsme se spolu prošli i po světě, respektive po městě. Ale sám víš ze svých výletů, že jenom koukat se nám moc nechce. Rád si vždycky něco dovezeš na památku z výletu. Někdo vozí pohledy, jiný fotografie a někdo rád zase něco jiného. Se mnou je to trochu složitější. A nebo vlastně jednodušší. Já mohu sbírat do svého kouzelného batůžku jenom značky.

Dnes se spolu naučíme posbírat značky, ať jsou kdekoliv. Jedná se o úkol na první pohled možná jednoduchý, ale uvidíš, že přináší různé problémy. Nejsem přeci tak dokonalý jako ty. Když tobě maminka řekne, abys uklidil, tak se kolem sebe jenom rozhlédneš a už vidíš, kde je nepořádek. Já to mám ale trochu složitější. Značku vidím jen když na ní stojím. Proto musím úsek, který mám uklidit, projít vždy celý. A přitom si pochopitelně dávat pozor, abych nenarazil do zdi. Proto můj přikaz obsahuje i povel abych – když dojdu ke zdi – provedl buď `VLEVO-VBOK`, nebo abych provádění ukončil. Takový příkaz je už přece jenom delší. Musíme si pořád pamatovat několik věcí:

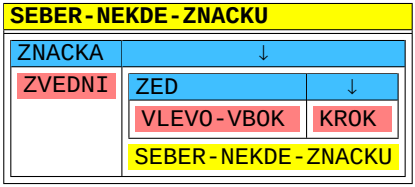
1. Když přijdu na značku → zvednu ji a uklidím do batůžku.
   POZOR – značka nemusí být jen jedna!
2. Když přijdu ke zdi → uhnu
   → dokončím činnost.
3. Když si pod sebou uklidím → pokračuji v pohybu.

Vidíš sám, že to tak moc jednoduché není. Ale my to spolu zvládneme. Ke zdi už spolu dojít umíme. Sebrat všechny značky pod sebou – to také není taková práce. Nadefinujeme si pro to příkaz `VYBER`.




```
VYBER ZNAMENA
  KDYZ JE ZNACKA
    ZVEDNI
    VYBER
  KONEC JINAK
  KONEC
KONEC
```

Za pomoci `VYBER` už to žádný problém nebude. Pro naši definici bychom sice nový příkaz `VYBER` mohli použít, ale vytvoříme si příkazy jiné. Třeba `SEBER-NEKDE-ZNACKU`.

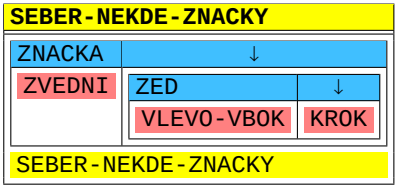



```
SEBER-NEKDE-ZNACKU ZNAMENA
  KDYZ JE ZNACKA
    ZVEDNI
  KONEC JINAK
    KDYZ JE ZED
      VLEVO-VBOK
    KONEC JINAK
      KROK
    KONEC
    SEBER-NEKDE-ZNACKU
  KONEC
KONEC
```

Vidíš sám, že vytvořený příkaz je dlouhý jako had. A teď už nám znovu vyniká i jednoduchost a přehlednost kopenogramu proti karlovskému zápisu. Jenže když si příkaz spustíš, tak přijdeš i na malou vadu na kráse. Zastavím se totiž na místě, kde zvednu značku. Schválně si to zkus.

Po nadefinování příkazu zadej počítači příkaz `MESTO`. A do města naskládej na libovolná místa různé počty značek. Asi tě nemusím upozorňovat, že tentokrát budu chodit jenom podél zdí. Pak teprve spusť příkaz `SEBER-NEKDE-ZNACKU`. Výsledek bude ten, že budu chodit okolo zdí tak dlouho, dokud nenarazím na značku. Když narazím na značku, tak ji seberu a skončím. To je způsobeno umístěním rekurze. Když totiž vykonám příkaz `ZVEDNI`, pokračuji následující úrovní vykonávání. No a tam už nic není – jenom příkazy `KONEC`. Takže rekurzi minu a skončím.

Proto si spolu sestrojíme obdobný příkaz `SEBER-NEKDE-ZNACKY`. Ten nás nechá sesbírat všechny značky po obvodu města.

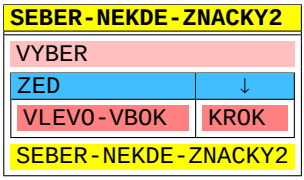



```
SEBER-NEKDE-ZNACKY ZNAMENA
  KDYZ JE ZNACKA
    ZVEDNI
  KONEC JINAK
    KDYZ JE ZED
      VLEVO-VBOK
    KONEC JINAK
      KROK
    KONEC
  KONEC
  SEBER-NEKDE-ZNACKY
KONEC
```

Rozdíl proti minulému příkazu `SEBER-NEKDE-ZNACKU` není skoro žádný, a přece se chovám zcela jinak. Je to způsobeno jiným umístěním rekurze – je až na konci příkazu a budu tedy pracovat pořád – ať seberu značku nebo ne.

Protože je druhá podmínka `KDYZ JE ZED` v jedné větvi podmínky `KDYZ JE ZNACKA`, říkáme taky, že jde o podmínky *vnořené*.

A teď ti ukážu jak lze takový příkaz zjednodušit. Napřed použijeme již vytvořený příkaz VYBER:



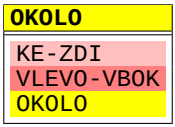

```
SEBER-NEKDE-ZNACKY2 ZNAMENA
  VYBER
  KDYZ JE ZED
    VLEVO-VBOK
  KONEC JINAK
    KROK
  KONEC
  SEBER-NEKDE-ZNACKY2
KONEC
```

Ale i tento příkaz je ještě příliš dlouhý. Další zjednodušení si můžeš provést sám. Podívej se pozorně na příkaz SEBER-NEKDE-ZNACKY2. Vidíš, že nejvíce místa zabírá příkaz otočení u zdi. Tak si to můžeme zjednodušit tím, že mi nadefinuješ pomocný příkaz OTOC-U-ZDI. To už jistě svedeš sám.

Ale to stále sbíráme značky jen podél zdí. A ty chceš přece vysbírat značky po celém městě! Ale o tom až příště.

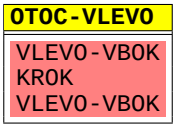
11. Kolo kolo mlýnské

Posledně jsme se učili uklízet. A přitom jsme používali i příkaz, abych chodil okolo zdí. Když teď budeš chtít, abych chodil kolem zdí, bude nejjednodušší, když mi dáš v čistém městě příkaz SEBER-NEKDE-ZNACKU. Ale to přeci není programátorské! Za prvé by tam nějaká značka mohla být a já bych se zastavil, takže bys musel použít SEBER-NEKDE-ZNACKY. Ale co když ty značky nebudeme chtít sbírat? Proto si spolu vytvoříme příkaz OKOLO. Je to náramně jednoduché.




```
OKOLO ZNAMENA
  KE-ZDI
  VLEVO-VBOK
  OKOLO
KONEC
```

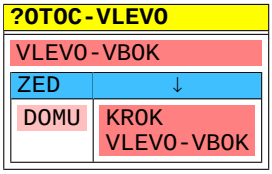
Ale my se přece chceme dostat i jinam! Nebudeme se pořád držet zdi jako nějaká nemluvňata. Jednou jsme velcí a tak se odvážně pustíme i do neznámých krajů. Jde nám o to, jak projít celé město. Pro jednoduchost vždy předpokládáme, že vycházíme z domu. Tam už se umím dostat pomocí příkazu DOMU. Dokážeme jít i ke zdi. Teď nám už jenom zbývá, abychom prošli všechna políčka. Je třeba, abychom se dostali vždy o jednu řadu políček výš, když se nám podaří dojít ke zdi. Proto si nadefinujeme ještě dva pomocné příkazy: OTOC-VLEVO a OTOC-VPRAVO:




```
OTOC-VLEVO ZNAMENA
  VLEVO-VBOK
  KROK
  VLEVO-VBOK
KONEC
```

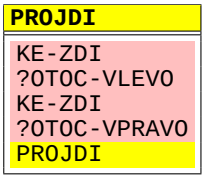
Příkaz pro otáčku vpravo ti už tady psát nebudu. To by byla ostuda! Jestli přesto nevíš, jak na to, tak ti pomůže slovník na konci.

Důležité je ošetřit příkaz proti narážení do zdi. Když bych totiž stál již na poslední řadě a otočil se, udělám krok rovnou do zdi. Proto si příkaz OTOC-VLEVO (a ty sám i OTOC-VPRAVO) nadefinujeme jinak – s podmínku.




```
?OTOC-VLEVO ZNAMENA
  VLEVO-VBOK
  KDYZ JE ZED
    DOMU
  KONEC JINAK
    KROK
    VLEVO-VBOK
  KONEC
KONEC
```

A teď už pro nás není žádný problém projít celé město. A po procházce se vždy vrátíme hezky domů.




```
PROJDI ZNAMENA
  KE-ZDI
  ?OTOC-VLEVO
  KE-ZDI
  ?OTOC-VPRAVO
  PROJDI
KONEC
```

Tak, teď už umíme projít celé město – pochopitelně za předpokladu, že vycházíme z domu.

Nadefinovat příkaz pomocí vstupní podmínky a cyklu už umíš. Já ale rozumím i dalším příkazům, které jsme si už dříve slíbili vysvětlit. Při tvé definici můžeš použít třeba i primitivum `DOKUD`.

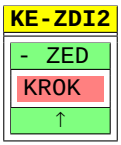
Čím se liší od `KDYZ`? Než si to ukážeme na příkladu, zopakujeme si pravidla pro používání `KDYZ`.

Pokud použijeme `KDYZ`, pak vždy následují dvě cesty, z nichž se provede pouze jedna: buď ta pro splněnou podmínku, nebo ta pro nesplněnou podmínku. Na konci se obě cesty spojí a pokračuje se dalším příkazem.

Pokud ale použiješ `DOKUD`, pak je-li podmínka splněna, vykonají se příkazy následující za příkazem `DOKUD` až po příslušný `KONEC`. Ten ale vrátí vykonávání zpět na úvodní podmínku `DOKUD`. A to seprovádí až do té doby, dokud je podmínka splněna.

V příkazu `KDYZ` musíme projít jednou z jeho větví. U `DOKUD` při nesplnění podmínky vnitřkem cyklu neprojdeme vůbec a budeme pokračovat za příslušným příkazem `KONEC`. Vstupní podmínka může být i typu `NENI podmínka`. Tělo cyklu se pak provádí tehdy, pokud `NENI podmínka`. Cyklus se přeskakuje, pokud `NENI NENI podmínka`, takže vlastně pokud `JE podmínka`.

Protože to vypadá složitě, ukážeme si to na následujícím příkladu. Půjdeme zde opět ke zdi. To už umíme s pomocí `KDYZ` a rekurze. Nyní si to ukážeme s pomocí `DOKUD`. Schválně, co bude lepší?

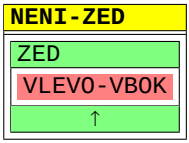



```
KE-ZDI2 ZNAMENA
  DOKUD NENI ZED
    KROK
  KONEC
KONEC
```

Já při vykonávání tohoto příkazu postupuji následovně: Testuji je-li splněno, že přede mnou `NENI ZED` (v kopenogramu se zapisuje symbolicky „`- ZED`“). Pokud je to pravda (takže přede mnou není zeď), vykonám vnitřek cyklu (`KROK`) a vrátím se opět na test vstupní podmínky. Pokud přede mnou zeď je (a podmínka „`NENI ZED`“ není splněna), tělo cyklu nevykonávám

a v tomto případě skončím. Šipka v kopenogramu naznačuje, že se má skočit nahoru před začátek vstupní podmínky (stejně jako u OPAKUJ). A ještě jedna informace pro tebe: Tento cyklus, kterému programátoři odborně říkají *cyklus se vstupní podmínkou a s degenerovanou výstupní podmínkou*, si vybarvuj zeleně – tedy pouze jeho řídící část: řádek „- ZED“ a řádek „ ↑ “.

A aby sis tento cyklus procvičil, nadefinujeme si příkaz, který zabezpečí, že se budu vždy natáčet do toho směru, kde přede mnou není zeď. To znamená, že po provedení tohoto příkazu budu schopný udělat KROK aniž bych při něm narazil. Pokud mě ovšem schválně nezavřeš do dvorku o jednom políčku! To potom takový východ nenajdu a budu se točit donekonečna. To ale není dvorek, to je vězení! Příkaz si nazveme třeba NENI-ZED:




```
NENI-ZED ZNAMENA
  DOKUD JE ZED
    VLEVO-VBOK
  KONEC
KONEC
```

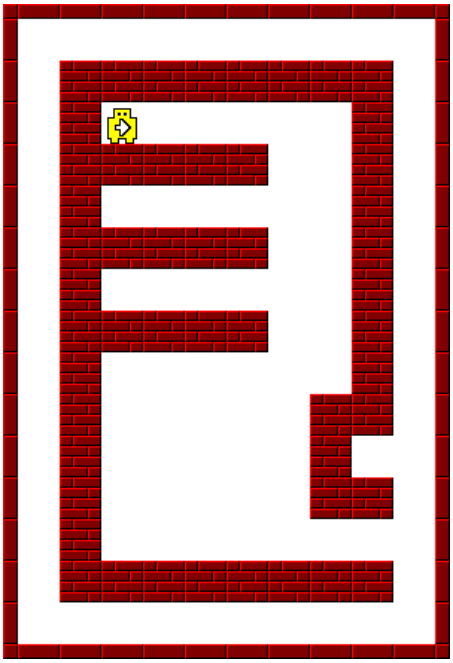
Tady vidíš, že při příkazu DOKUD můžeš vyžadovat jak splnění podmínky (JE ZED), tak její nesplnění (NENI ZED). Sám si teď zkus všechny příkazy, kde jsme použili KDYZ a rekurzi, napsat pomocí DOKUD.

12. Opět na cestách

Tak to vidíš! Ve městě už jsme jako doma. Doposud jsme se ale pohybovali tak, že jsme vždy vybíral všechny možnosti, které mohli nastat.

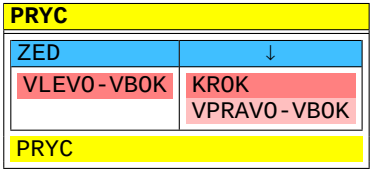
To je sice jedna z možností řešení, tady nám spolu funguje, ale dobré to není. My jsme zatím značky vyhledávali tak, že jsme procházeli město políčko za políčkem. Sám vidíš, jak to dlouho trvá. Je to více méně jen takové chaotické chození sem a tam. Tak se přece nedá pracovat. Naše programy doposud pracují jen za předpokladu, že ve městě nic není. Ale sám víš, že ve městě jsou domy a spousta jiných překážek. Tam bychom tak chodit nemohli! Ale já ti prozradím, jak vyzrajeme na všechny případné překážky.

Zkusíme vyjít z bludiště. To si nakresli na obrazovku pomocí příkazu `MESTO`. A teď běž! Zkus to napřed podle uvedeného vzoru.



Jak vidíš, bludiště má jenom jeden východ. A když si vezmeš tužku a půjdeš podle libovolné zdi, tak zjistíš, že vždy dojdeš k východu.

Na tomhle principu je založen i následující příkaz, který nás z bludiště vyvede.




```
PRYC ZNAMENA
  KDYZ JE ZED
    VLEVO-VBOK
  KONEC JINAK
    KROK
    VPRAVO-VBOK
  KONEC
  PRYC
KONEC
```

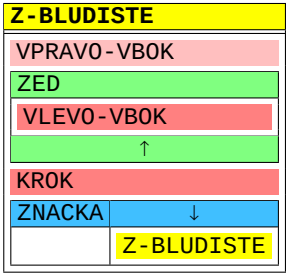
Schválně si jeho pravdivost ověř na jiném bludišti. Co vlastně děláme celou cestu? Jdeme podél pravé zdi. Před každým krokem se zeptáme, je-li před námi zeď. Pokud ano, tak se pouze otočíme. Pokud tam zeď není, uděláme `KROK` a `VPRAVO-VBOK`, abychom se drželi zdi. Před tímto příkazem musím být pravým bokem u zdi. A nebo udělat příkaz dvouúrovňový – nejprve udělat `KE-ZDI` a `VLEVO-VBOK`, čímž se dostanu pravým bokem ke zdi, a pak vlastní `PRYC`.

Po spuštění ti dokáži, že sice najdu východ z bludiště, ale nezastavím se ani venku. Chodím pořád dokola. Je to tím, že v příkazu je použita rekurze. A zdi, které ohraničují město, jsou pro mě jen pokračováním bludiště. Kdyby zdi bludiště sahaly až ke zdi města, tak by se mi podařilo do bludiště opět vejít. Vyjdu sice ven, ale stále budu chodit sem a tam. Nejjednodušší ukončení příkazu bude, když mi u východu připravíš značku a příkaz nadefinuješ tak, že se na značce zastavím.

Určitě jsi taky zpozoroval, že chodím sem a tam. Jak taky ne, když si nepamatuji, kde už jsem všude byl! Abych si to dokázal zapamatovat, k tomu mi opět slouží značky. Já si je budu na cestu pokládat a pak vím, kudy jsem už prošel. Tento způsob má ale jeden háček. Představ si, že se dostanu na dvorek, který má vchod široký jenom jedno políčko. To znamená, že budeme muset značky počítat, protože jinak bych se nemusel dostat ven. Ale ještě než si to spolu zkusíme, podívej se, jak se chovám v jiném bludišti. Jeho obrázek jsem ti nakreslil na další stránce.

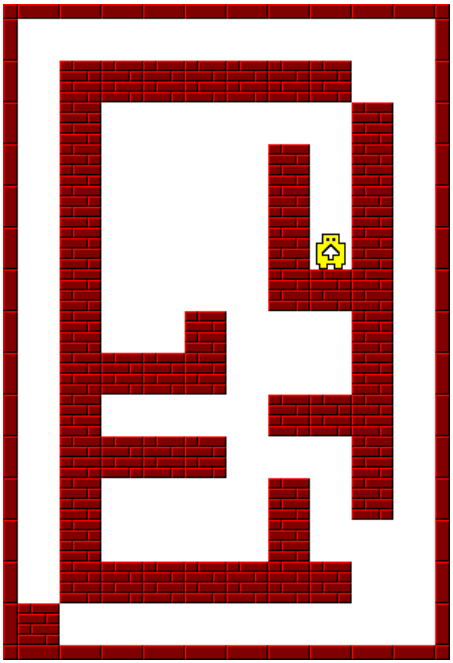
Bludiště je tady spojeno s venkovní zdí. Když se na obrázek pozorně podíváš, tak je to vlastně jedno bludiště, které nemá žádný východ. Je tedy logické, že v takovém bludišti budu pořád chodit a zároveň hledat východ. Když neuspěji ani u venkovní zdi, pustím se zpátky dovnitř bludiště a budu hledat i tam. Co když jsem poprvé v bludišti ten správný východ přehlédl!

A tak bych tady mohl chodit donekonečna. Sám uvidíš, že po čase začnu vykonávat stále stejný cyklus. Může být pěkně dlouhý. Schválně mě nech běhat se stisknutým tlačítkem `Shift`. Jak tomu bloudění zabránit? Něco jsme si už naznačili. Polož mi někam značku a dej mi příkaz, abych se na značce zastavil. Příkaz si nazveme třeba `Z-BLUDISTE`.




```
Z-BLUDISTE ZNAMENA
  VPRAVO-VBOK
  DOKUD JE ZED
    VLEVO-VBOK
  KONEC
  KROK
  KDYZ JE ZNACKA
  KONEC JINAK
    Z-BLUDISTE
  KONEC
KONEC
```

Ať už mi nakreslíš sebesložitější bludiště, východ v něm vždy najdu. Nejdu sice nejkratší cestou, ale zato spolehlivě. Zrovna tak je jasné, že dokážu najít značku kdekoliv v bludišti. Ale! To jen za podmínky, že je značka umístěna u zdi. Vždyť víš, že jsem krátkozraký. Proto chodím bludištěm jen podle zdí. A protože značku vidím jen když si na ni stoupnu, tak nebude-li vedle zdi, neuvidím ji a budu chodit stále dokola.



13. A co takhle trochu matematiky?

Nelekej se! Nechci tě připravit o hodiny matematiky, které patří mezi tvoje nejoblíbenější ve škole. To není mým cílem. Naučíme se jen o něco víc, abychom mohli později počítač využívat naplno. I když ty už vlastně trochu počítat umíš. Dovedeš položit stanovený počet značek, udělat příslušný počet kroků. To vše pomocí `OPAKUJ`.

Ale co když nemáš nic přesně určeno? I v takovéhle situaci si už umíš poradit. Vždyť dokážeš jít třeba ke zdi a přitom nevíš, jak je zeď daleko. Ale co takhle dojít třeba jenom na polovinu vzdálenosti ke zdi? Jestli budeš vědět, že je zeď vzdálena čtyři kroky, tak to umíš. Uděláš jenom dva. Ale to jsi spočítal za mě ty a jen pro tento jeden výjimečný případ! Já to chci umět sám a vždy. Tedy i tehdy, když nevím, jak jsem daleko od zdi. To je oříšek, co?

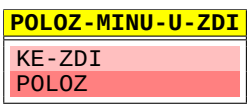
Jak je ta zeď daleko? Víme, že je od nás vzdálena dvě poloviny vzdálenosti. Připadá ti tahle jednoduchost jako nesmysl? Že nám to nic nepomůže? Ale naopak. Trochu odskočíme. Zalistuj si teď zpátky a podívej se, jak jsme definovali příkaz `CTYR-KROK3` na stránce 12.

Tehdy jsme si řekli, že takováhle definice je z programátorského hlediska nesprávná. Ano, ale jenom proto, že jsem měl udělat čtyři kroky dopředu … a tady chodím zbytečně sem tam. Jsou však případy, kdy se nám i zdánlivá zbytečnost může hodit. Jak vypadá třeba krok z tohoto pohledu? Můžeš jej nadefinovat jako dva kroky dopředu a jeden zpátky. Zajímavé, viď? Ale tímto krokem ujdu polovinu vzdálenosti, jako dvěma kroky. Takže do poloviček vzdálenosti dojdu tehdy, když na každé dva kroky dopředu udělám jeden krok zpátky. Pochopitelně musím napřed dělat dvojkroky dopředu a pak tolik kroků zpátky, jako jsem udělal dvojkroků dopředu. Teď už nám zbývá jenom určit počet dvojkroků dopředu, abych mohl udělat stejný počet kroků vzad. To je každému pochopitelné.

Ale stále ještě nevíš, jakým způsobem spočítáme dvojkroky. Přemýšlíš, hlavu namáháš a stejně nakonec nic zjistíš, že žádná z přímých cest, ani s použitím podmínky nebo cyklu, k cíli nevede. Ale to nic. Vzpomeň si na moji přítelkyni rekurzi. Ta nám pomůže. A ani nepotřebuješ znát počet dvojkroků. Úplně stačí, když vím, že udělám tolik kroků jako dvojkroků. A jestli to jsou tři nebo sedm, to mě vůbec nemusí zajímat.

Ještě ti to není úplně jasné? Tak si to spolu zopakujeme podrobněji. Teď totiž začínám přímo napínavou kapitolu a tak napni svoji pozornost, aby ti nic, ale vůbec nic neuniklo!

Na začátku kapitoly jsi mi přikývnul, že se mnou umíš dojít opatrně `KE-ZDI`, aniž bych naboural. Ale co když mám za úkol dojít ke zdi a položit u ní značku? Zahrajeme si tedy na vojáky. Já – KAREL – mám za úkol položit u zdi nepřítelova opevnění minu – značku. Jak se s tím vypořádám? No to je přece jednoduché. Dojdu ke zdi a položím značku.



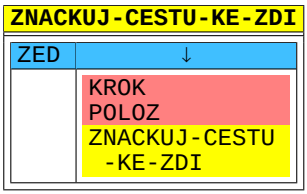

```
POLOZ-MINU-U-ZDI ZNAMENA
  KE-ZDI
  POLOZ
KONEC
```

„Poslušně hlásím, že zadaný úkol jsem podle rozkazu splnil!“ Ale co teď se mnou? Vždyť žádná armáda si nemůže dovolit ztrácet tak dobré vojáky, jako jsem já. Dej mi rychle příkaz,

abych se vrátil zpět. Jak? Možností je několik. Jako dobrý voják mám vědět, že se budu muset vrátit zpátky. Proto, abych nezabloudil, si mohu cestu značkovat. A po svých značkách se mohu vrátit zpět. Při zpáteční cestě je pochopitelně posbírám, aby nepřítel neobjevil, že jsem k němu pronikl.

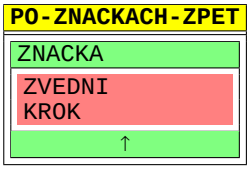
A jak jsem to udělal? To už určitě víš sám. Všimnul sis, že jsme tu zrovna nakousli dekompozici? Trochu z jiného konce, ale zase jsme museli rozebrat celek – v tom jsem ti trochu pomohl – a opět jej složit dohromady.

Ale abych pořád neodbíhal. Tady je příkaz, abych položil minu a opět se vrátil na své původní stanoviště. Napřed si nadefinuji příkaz `ZNACKUJ-CESTU-KE-ZDI`:




```
ZNACKUJ-CESTU-KE-ZDI ZNAMENA
  KDYZ JE ZED
  KONEC JINAK
    KROK
    POLOZ
    ZNACKUJ-CESTU-KE-ZDI
  KONEC
KONEC
```

No a u zdi pak jenom položím značku, udělám čelem vzad a půjdu po značkách zpátky. Při zpáteční cestě si značky sesbírám. Ale pozor! Minu sebrat nesmím! Proto ještě jeden příkaz – `PO-ZNACKACH-ZPET`:




```
PO-ZNACKACH-ZPET ZNAMENA
  DOKUD JE ZNACKA
    ZVEDNI
    KROK
  KONEC
KONEC
```

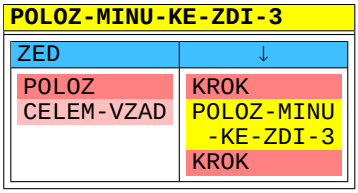
A teď je příkaz `POLOZ-MINU-KE-ZDI-2` úplně jasný.


```
POLOZ-MINU-KE-ZDI-2 ZNAMENA
  ZNACKUJ-CESTU-KE-ZDI
  POLOZ
  CELEM-VZAD
  PO-ZNACKACH-ZPET
KONEC
```

Možná tě napadlo, proč je řešení tak složité? Ty už přece dokážeš takto zadaný příkaz zjednodušit. Stačí pouze zadat, abych na místě, kde stojím, položil značku, došel ke zdi položit minu, otočil se a šel zpět, dokud nedojdu na značku.

To je pravda, našemu zadání by i takové jednodušší řešení plně vyhovělo. Ale co kdybych se musel po cestě vyhýbat překážkám? Pak takový příkaz stačí jenom trochu doplnit a máme hned

obecnější řešení. A tím i lepší. Takto jsme to spolu ale schválně rozebrali, aby další výklad byl srozumitelnější. Ale ještě než začneme doopravdy počítat, rozkreslíme si do kopenogramu předcházející řešení. Jak bys napsal příkaz s použitím jednodušších příkazů? Asi takto:

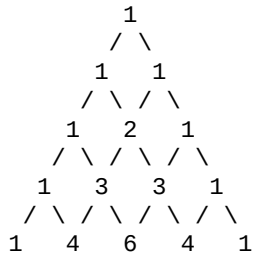



```
POLOZ-MINU-KE-ZDI-3 ZNAMENA
  KDYZ JE ZED
    POLOZ
    CELEM-VZAD
  KONEC JINAK
    KROK
    POLOZ-MINU-KE-ZDI-3
    KROK
  KONEC
KONEC
```

To jsou zajímavé věci,co? Neboj! Příště nás čekají ještě zajímavější. Vydáme se spolu do dávných věků pyramid.

14. Stavíme pyramidu

Ale neboj se. Nepostavíme ji z kamenných kvádrů, ale jenom tak na papíru. Ty nevíš, co to je pyramida? Jak vypadá? Já teď tady nemyslím ty honosné hrobky staroegyptských faraónů, ale takovou docela malinkatou pyramidu. A aby se nám sem vešla, tak si ji namalujeme jako trojúhelník. Vypadá třeba takto:

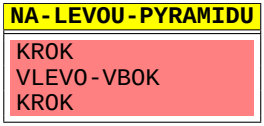


V tomto trojúhelníku – pyramidě – je hodnota každého čísla dána součtem jeho pravého a levého horního souseda. Někdy se mu také říká *Pascalův trojúhelník* – podle francouzského matematika Blaise Pascala.

Já začínám stavět na vršku směrem k dolní zdi. Na místě jednotlivých čísel si představ moje bratříčky. Já jim rozdám značky, ale jenom svým dvěma spodním bratrům. A teď každý z nich dá tolik značek, kolik dostal, svým dvěma spodním bratrům. Ti ale dostanou každý ještě další značky od svého dalšího horního bratra z druhé strany. A co s tím má společného naše kamarádka rekurze? Moji bratříčci – to budu vlastně já – pověřím sám sebe nějakou prací a až ji dokončím, budu pracovat znovu.

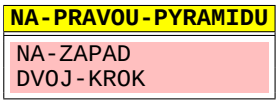
A teď trochu počítačově: Já na každém políčku – uzlu – pyramidy položím značku. Pak pokud nejsem u zdi (což znamená konec pyramidy), rozběhnu se vlevo dolů (z mého pohledu, protože stojím otočen směrem dolů; z tvého pohledu je to vpravo). Tam postavím pyramidu (první bratříček – rekurze). Pak se posunu o dva kroky na západ a tam postavím další pyramidu (druhý bratříček – rekurze). Nakonec se vrátím zpět na místo, odkud jsem vyšel.

Každou pyramidu – tedy i ty v nižší úrovni – stavím ze severu na jih. Přechod na `NA-LEVOU-PYRAMIDU` je tedy poměrně jednoduchý:



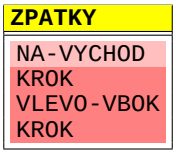

```
NA-LEVOU-PYRAMIDU ZNAMENA
  KROK
  VLEVO-VBOK
  KROK
KONEC
```

Přechod `NA-PRAVOU-PYRAMIDU` je ještě jednodušší: stačí udělat dva kroky směrem na západ:




```
NA-PRAVOU-PYRAMIDU ZNAMENA
  NA-ZAPAD
  DVOJ-KROK
KONEC
```

Návrat o úroveň výš nám taky nedá žádnou práci:

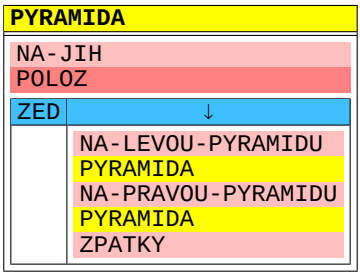



```
ZPATKY ZNAMENA
  NA-VYCHOD
  KROK
  VLEVO-VBOK
  KROK
KONEC
```

No a vlastní pyramida je jednoduchá: Otočím se `NA-JIH` – tím směrem přece budu stavět pyramidu. Pak položím značku. Nyní mám dvě možnosti:

1. Jsem-li u zdi, už není co stavět, protože není kam. Nebudu tedy dělat nic.
2. Nejsem-li u zdi, pak se přesunu `NA-LEVOU-PYRAMIDU`, postavím `PYRAMIDU`, přesunu se `NA-PRAVOU-PYRAMIDU`, postavím `PYRAMIDU` a vrátím se `ZPATKY`. „Elementary, my dear Watson“ jak říká slavný anglický detektiv Sherlock Holmes.

Celý příkaz `PYRAMIDA` jsme si popsali v minulém odstavci:



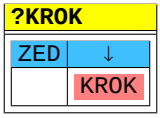

```
PYRAMIDA ZNAMENA
  NA-JIH
  POLOZ
  KDYZ JE ZED
  KONEC JINAK
    NA-LEVOU-PYRAMIDU
    PYRAMIDA
    NA-PRAVOU-PYRAMIDU
    PYRAMIDA
    ZPATKY
  KONEC
KONEC
```

A stejně funguje rekurze při zjišťování počtu dvojkroků, když jdeme ke zdi.

Jak to tedy vypadá tam? Začnu od samého začátku, od zdi. Kolik mohu udělat udělat dvojkroků? No přece žádný. Ale teď nestojím před zdí. Vím, že mohu udělat ke zdi alespoň krok. Ale taky vím, že o dvě políčka dál je brácha. A já pomocí rekurze předám řízení jemu. Ale o je naprosto stejný jako já. Podívá se, je-li před ním zeď. To by uměl splnit – neudělal by nic, jen mi vrátí řízení. Ale když tam zeď není, tak to přihraje dalšímu bratrovi … já na bráchu, brácha na mě … a tak si přehazujeme úkol do té doby, než jsme schopni jej splnit. Ale tím jsme teprve u zdi. My ještě musíme každému bráchovi, který nám pomáhal, říci, aby mě poslal o jeden krok zpět. Vždyť chceme dojít do poloviny vzdálenosti. A když to udělají, je úkol splněn.

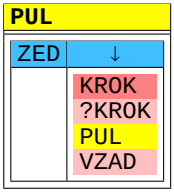
Nejprve několik pomocných příkazů: VZAD – to je vracení se o krok zpátky. A ?KROK je opatrný krok – protože testujeme počet dvojkroků, které můžeme udělat, tak se může stát, že bych stál lichý počet kroků od zdi. No a protože jdu vždy po dvou krocích, tak bych mohl narazit.


```
VZAD ZNAMENA
  CELEM-VZAD
  KROK
  CELEM-VZAD
KONEC
```




```
?KROK ZNAMENA
  KDYZ JE ZED
  KONEC JINAK
    KROK
  KONEC
KONEC
```

No a samotný příkaz PUL už je jednoduchý: není-li přede mnou zeď, udělám KROK a ?KROK (opatrný krok, abych ani zde nenarazil do zdi), zavolám svého bráchu (PUL) a na dva kroky, které jsem udělal (KROK a ?KROK) udělám jeden VZAD.



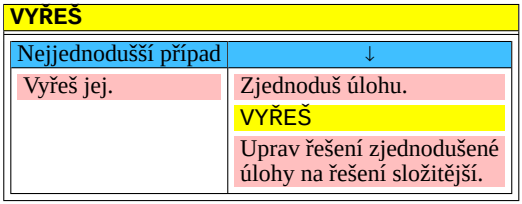

```
PUL ZNAMENA
  KDYZ JE ZED
  KONEC JINAK
    KROK
    ?KROK
    PUL
    VZAD
  KONEC
KONEC
```

Tímto způsobem můžeš definovat pak již cokoliv. To je šikovná kamarádka rekurze, co? Bez ní bychom si neporadili. Musíme jen zachovávat několik zásad.

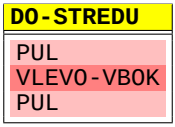
Ať nám dá někdo hlavolam jaký chce, musíme najít takové podmínky, za kterých jej dokážeme vyřešit. V našich případech to jsou situace, kdy stojíme u zdi. Pak teprve začneme hledat jádro úlohy, které bude mít čtyři vlastnosti:

1. Umíme v něm přejít od složitějšího k jednoduššímu – zjednodušit úlohu. U nás jsou to třeba počáteční dva kroky.
2. Umíme řešení jednodušší úlohy upravit na řešení složitější – třeba tím, že uděláme krok zpět.
3. Musíme mít jistotu, že toto postupné zjednodušování jednou skončí. To znamená, že např. dojdeme ke zdi.
4. Musíme mít jistotu, že dřív, než dospějeme ke krachu, najdeme situaci, kterou umíme vyřešit bez dalšího zjednodušování.

Ale to, co jsme si teď pověděli, jde vyjádřit i obrázkem. Nevěříš? No přece kopenogramem. Bude to kopenogram *obecného algoritmu řešení jednoho typu úloh rekurzí*. Brrrr, to je věda.



A teď – ale už jenom rekreačně – spolu dojdeme do středu dvorku:




```
DO-STREDU ZNAMENA
  PUL
  VLEVO-VBOK
  PUL
KONEC
```

A stejně jako `PUL` můžeš definovat příkaz `TRETINA`. Kdybys náhodou nevěděl jak, je výsledek uveden ve slovníčku v příloze 2.

Tak, a to je poslední příkaz, kterým jsme se tady spolu zabývali. Já ale věřím, že mi zůstaneš věrný a že se spolu ještě mnohokrát setkáme. V každém případě, když jsi vydržel až sem, je z tebe programátor a vše, co jsme se spolu naučili, ty uplatníš.

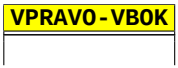
Hodně zdaru, krásných programů a hodně bitů paměti

přeje na rozloučenou

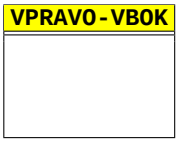
Tvůj KAREL.

Příloha 1: Jak dělat kopenogramy

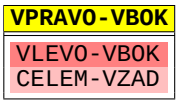
Na stránce 8 jsme se poprvé seznámili s kopenogramy. Abychom s nimi mohli úspěšně pracovat, je nejlepší, když jsou barevné. Ne vždy máš k dispozici barevnou knížku. Taky jako programátor potřebuješ dělat kopenogramy svoje. Proto si tady stručně shrneme několik hlavních zásad, jak kopenogramy dělat.



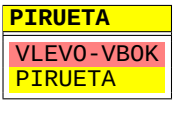
– *Hlava* kopenogramu na *žlutém* pozadí.
Hlavu oddělujeme dvojitou čárou. Někdy se můžeš setkat taky s tím, že hlava je na bílém pozadí. Ale zůstaňme spolu u žluté. Je to výraznější, když si sám maluješ do sešitu.



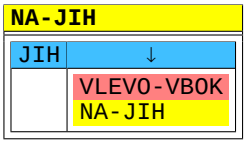
– Hlava kopenogramu
– *Tělo* kopenogramu necháme bílé. Je to vždy čtverec, nebo obdélník.



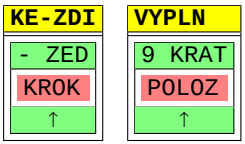
– Hlava kopenogramu
– *Blok příkazů* udává, které příkazy a v jakém pořadí se budou provádět. Primitiva jsou vybarvena *červeně*, složené příkazy *růžově*.



– Hlava kopenogramu
– Blok příkazů (červeně primitiva, růžově složené příkazy)
– *Žluté* pole rekurzivního volání – je to vlastně hlava programu, proto je žlutá.



– Hlava kopenogramu
– *Podmíněný blok – modře.* Šipka nám naznačuje, kam se vydáme při nesplnění podmínky.



– Hlava kopenogramu
– *Cyklus se vstupní podmínkou – zeleně.* Znak „ - “ = „NENI“.
– Blok opakovaných příkazů
– *Degenerovaná výstupní podmínka – zeleně.*

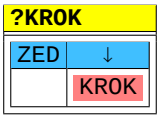
To jsou jen stručná pravidla. Když si budeš malovat kopenogramy do sešitu, můžeš použít sice různá zjednodušení, ale vždy zachovávej barevné označení. Aby měly kopenogramy význam, dodržuj stanovené barvy!

Výrazné barvy přeje KAREL.

Příloha 2: Karlův slovník

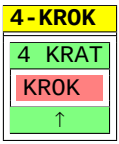
Ve slovníku jsou příkazy seřazeny podle abecedy. Najdeš tu všechny kromě duplicitních. Jsou zde uvedeny i příkazy, na které je v textu pouze odkaz.

Vpravo je uvedeno číslo stránky, kde najdeš, jak jsme příkaz vytvořili.



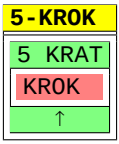

```
?KROK ZNAMENA
  KDYZ JE ZED
  KONEC JINAK
 KROK
  KONEC
KONEC
```

[49]



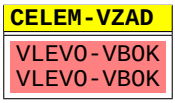

```
4-KROK ZNAMENA
  OPAKUJ 4 KRAT
 KROK
  KONEC
KONEC
```

[16]



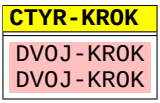

```
5-KROK ZNAMENA
  OPAKUJ 5 KRAT
 KROK
  KONEC
KONEC
```

[16]



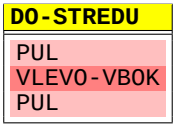

```
CELEM-VZAD ZNAMENA
  VLEVO-VBOK
  VLEVO-VBOK
KONEC
```

[9]




```
CTYR-KROK ZNAMENA
  DVOJ-KROK
  DVOJ-KROK
KONEC
```

[12]

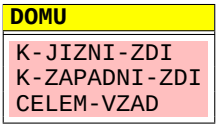



```
DO-STREDU ZNAMENA
  PUL
  VLEVO-VBOK
  PUL
KONEC
```

[50]

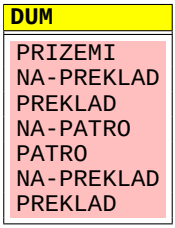

```
DO-ZDI ZNAMENA
  KROK
  DO-ZDI
KONEC
```

[31]



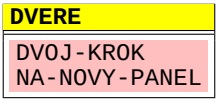

```
DOMU ZNAMENA
  K-JIZNI-ZDI
  K-ZAPADNI-ZDI
  CELEM-VZAD
KONEC
```

[33]



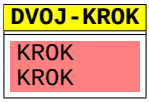

```
DUM ZNAMENA
  PRIZEMI
  NA-PREKLAD
  PREKLAD
  NA-PATRO
  PATRO
  NA-PREKLAD
  PREKLAD
KONEC
```

[27]



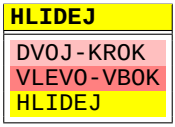

```
DVERE ZNAMENA
  DVOJ-KROK
  NA-NOVY-PANEL
KONEC
```

[25]



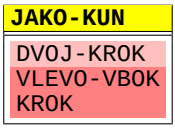

```
DVOJ-KROK ZNAMENA
  KROK
  KROK
KONEC
```

[9]



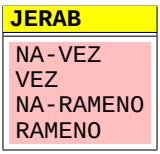

```
HLIDEJ ZNAMENA
  DVOJ-KROK
  VLEVO-VBOK
  HLIDEJ
KONEC
```

[31]



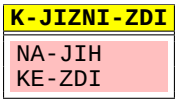

```
JAKO-KUN ZNAMENA
  DVOJ-KROK
  VLEVO-VBOK
  KROK
KONEC
```

[13]



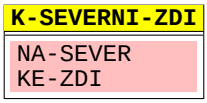

```
JERAB ZNAMENA
  NA-VEZ
  VEZ
  NA-RAMENO
  RAMENO
KONEC
```

[20]



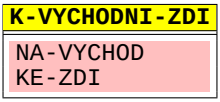

```
K-JIZNI-ZDI ZNAMENA
  NA-JIH
  KE-ZDI
KONEC
```

[33]



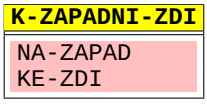

```
K-SEVERNI-ZDI ZNAMENA
  NA-SEVER
  KE-ZDI
KONEC
```

[33]



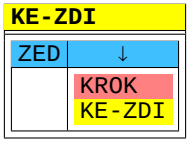

```
K-VYCHODNI-ZDI ZNAMENA
  NA-VYCHOD
  KE-ZDI
KONEC
```

[33]



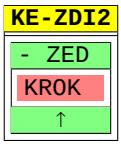

```
K-ZAPADNI-ZDI ZNAMENA
  NA-ZAPAD
  KE-ZDI
KONEC
```

[33]



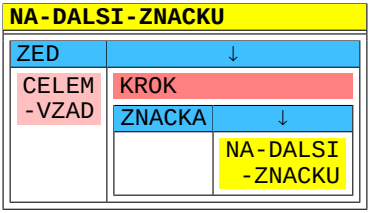

```
KE-ZDI ZNAMENA
  KDYZ JE ZED
  KONEC JINAK
 KROK
 KE-ZDI
  KONEC
KONEC
```

[31]



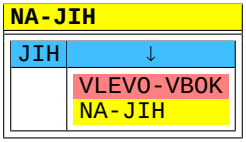

```
KE-ZDI2 ZNAMENA
  DOKUD NENI ZED
 KROK
  KONEC
KONEC
```

[39]



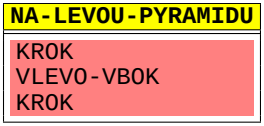

```
NA-DALSI-ZNACKU ZNAMENA
  KDYZ JE ZED
 CELEM-VZAD
  KONEC JINAK
 KROK
 KDYZ JE ZNACKA
 KONEC JINAK
NA-DALSI-ZNACKU
 KONEC
  KONEC
KONEC
```

[32]



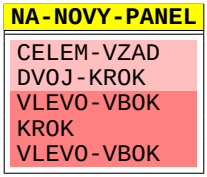

```
NA-JIH ZNAMENA
  KDYZ JE JIH
  KONEC JINAK
 VLEVO-VBOK
 NA-JIH
  KONEC
KONEC
```

[33]

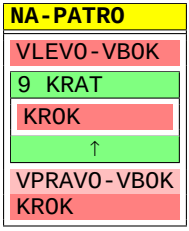



```
NA-LEVOU-PYRAMIDU ZNAMENA
  KROK
  VLEVO-VBOK
  KROK
KONEC
```

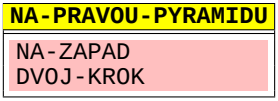
[47]



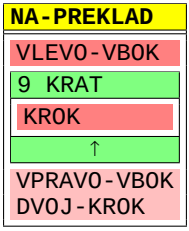

```
NA-NOVY-PANEL ZNAMENA                    [24]
  CELEM-VZAD
  DVOJ-KROK
  VLEVO-VBOK
  KROK
  VLEVO-VBOK
KONEC
```



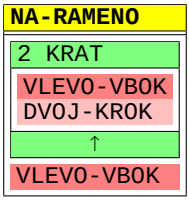

```
NA-PATRO ZNAMENA                         [27]
  VLEVO-VBOK
  OPAKUJ 9 KRAT
 KROK
  KONEC
  VPRAVO-VBOK
  KROK
KONEC
```



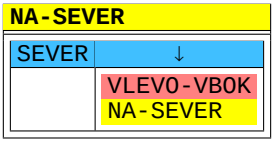

```
NA-PRAVOU-PYRAMIDU ZNAMENA               [48]
  NA-ZAPAD
  DVOJ-KROK
KONEC
```




```
NA-PREKLAD ZNAMENA                       [27]
  VLEVO-VBOK
  OPAKUJ 9 KRAT
 KROK
  KONEC
  VPRAVO-VBOK
  DVOJ-KROK
KONEC
```

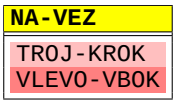



```
NA-RAMENO ZNAMENA                        [17]
  OPAKUJ 2 KRAT
 VLEVO-VBOK
 DVOJ-KROK
  KONEC
  VLEVO-VBOK
KONEC
```



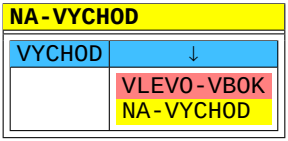

```
NA-SEVER ZNAMENA
  KDYZ JE SEVER
  KONEC JINAK
 VLEVO-VBOK
 NA-SEVER
  KONEC
KONEC
```

[33]



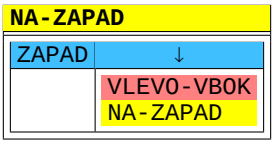

```
NA-VEZ ZNAMENA
  TROJ-KROK
  VLEVO-VBOK
KONEC
```

[20]



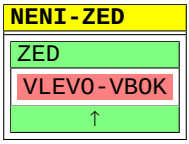

```
NA-VYCHOD ZNAMENA
  KDYZ JE VYCHOD
  KONEC JINAK
 VLEVO-VBOK
 NA-VYCHOD
  KONEC
KONEC
```

[33]



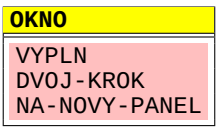

```
NA-ZAPAD ZNAMENA
  KDYZ JE ZAPAD
  KONEC JINAK
 VLEVO-VBOK
 NA-ZAPAD
  KONEC
KONEC
```

[33]



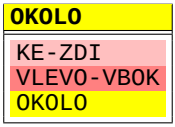

```
NENI-ZED ZNAMENA
  DOKUD JE ZED
 VLEVO-VBOK
  KONEC
KONEC
```

[40]



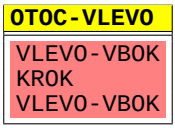

```
OKNO ZNAMENA
  VYPLN
  DVOJ-KROK
  NA-NOVY-PANEL
KONEC
```

[25]



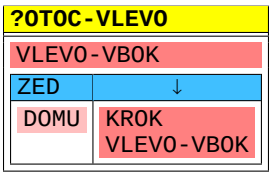

```
OKOLO ZNAMENA
  KE-ZDI
  VLEVO-VBOK
  OKOLO
KONEC
```

[38]



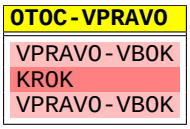

```
OTOC-VLEVO ZNAMENA
  VLEVO-VBOK
  KROK
  VLEVO-VBOK
KONEC
```

[38]



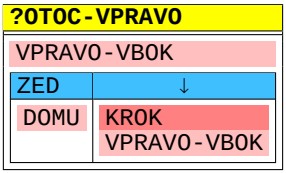

```
?OTOC-VLEVO ZNAMENA
  VLEVO-VBOK
  KDYZ JE ZED
 DOMU
  KONEC JINAK
 KROK
 VLEVO-VBOK
  KONEC
KONEC
```

[38]



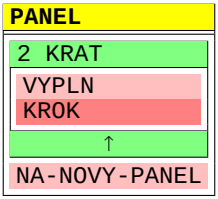

```
OTOC-VPRAVO ZNAMENA
  VPRAVO-VBOK
  KROK
  VPRAVO-VBOK
KONEC
```

[38]

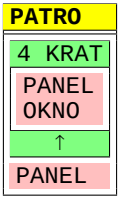



```
?OTOC-VPRAVO ZNAMENA
  VPRAVO-VBOK
  KDYZ JE ZED
 DOMU
  KONEC JINAK
 KROK
 VPRAVO-VBOK
  KONEC
KONEC
```

[38]

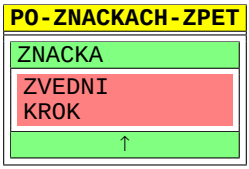



```
PANEL ZNAMENA                    [25]
  OPAKUJ 2 KRAT
 VYPLN
 KROK
  KONEC
  NA-NOVY-PANEL
KONEC
```

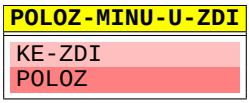



```
PATRO ZNAMENA                    [26]
  OPAKUJ 4 KRAT
 PANEL
 OKNO
  KONEC
  PANEL
KONEC
```


```
PIRUETA ZNAMENA                  [28]
  VLEVO-VBOK
  PIRUETA
KONEC
```

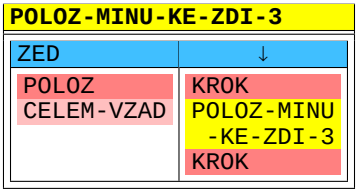



```
PO-ZNACKACH-ZPET ZNAMENA         [45]
  DOKUD JE ZNACKA
 ZVEDNI
 KROK
  KONEC
KONEC
```




```
POLOZ-MINU-U-ZDI ZNAMENA         [44]
  KE-ZDI
  POLOZ
KONEC
```


```
POLOZ-MINU-KE-ZDI-2 ZNAMENA      [45]
  ZNACKUJ-CESTU-KE-ZDI
  POLOZ
  CELEM-VZAD
  PO-ZNACKACH-ZPET
KONEC
```

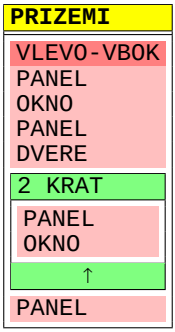



```
POLOZ-MINU-KE-ZDI-3 ZNAMENA
  KDYZ JE ZED
 POLOZ
 CELEM-VZAD
  KONEC JINAK
 KROK
 POLOZ-MINU-KE-ZDI-3
 KROK
  KONEC
KONEC
```

[46]

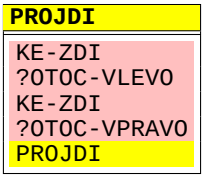

```
PREKLAD ZNAMENA
  OPAKUJ 9 KRAT
 OKNO
  KONEC
KONEC
```

[26]



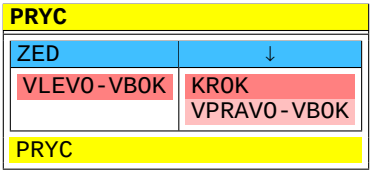

```
PRIZEMI ZNAMENA
  VLEVO-VBOK
  PANEL
  OKNO
  PANEL
  DVERE
  OPAKUJ 2 KRAT
 PANEL
 OKNO
  KONEC
  PANEL
KONEC
```

[26]

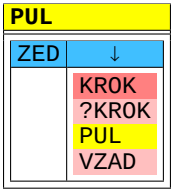



```
PROJDI ZNAMENA
  KE-ZDI
  ?OTOC-VLEVO
  KE-ZDI
  ?OTOC-VPRAVO
  PROJDI
KONEC
```

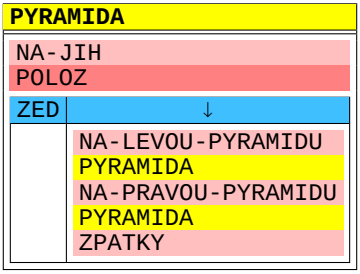
[39]



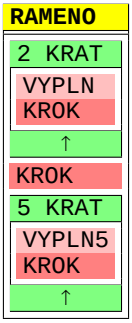

```
PRYC ZNAMENA                    [42]
  KDYZ JE ZED
 VLEVO-VBOK
  KONEC JINAK
 KROK
 VPRAVO-VBOK
  KONEC
  PRYC
KONEC
```



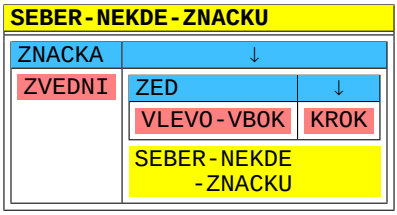

```
PUL ZNAMENA                     [49]
  KDYZ JE ZED
  KONEC JINAK
 KROK
 ?KROK
 PUL
 VZAD
  KONEC
KONEC
```



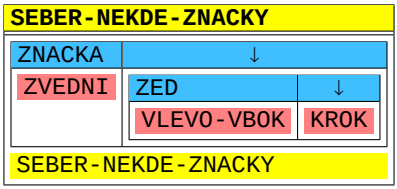

```
PYRAMIDA ZNAMENA                [48]
  NA-JIH
  POLOZ
  KDYZ JE ZED
  KONEC JINAK
 NA-LEVOU-PYRAMIDU
 PYRAMIDA
 NA-PRAVOU-PYRAMIDU
 PYRAMIDA
 ZPATKY
  KONEC
KONEC
```



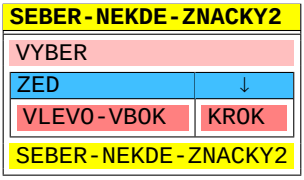

```
RAMENO ZNAMENA                    [18]
  OPAKUJ 2 KRAT
 VYPLN
 KROK
  KONEC
  KROK
  OPAKUJ 5 KRAT
 VYPLN5
 KROK
  KONEC
KONEC
```




```
SEBER-NEKDE-ZNACKU ZNAMENA        [36]
  KDYZ JE ZNACKA
 ZVEDNI
  KONEC JINAK
 KDYZ JE ZED
VLEVO-VBOK
 KONEC JINAK
KROK
 KONEC
 SEBER-NEKDE-ZNACKU
  KONEC
KONEC
```

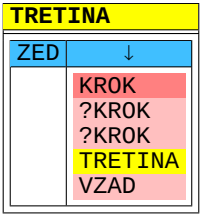



```
SEBER-NEKDE-ZNACKY ZNAMENA        [36]
  KDYZ JE ZNACKA
 ZVEDNI
  KONEC JINAK
 KDYZ JE ZED
VLEVO-VBOK
 KONEC JINAK
KROK
 KONEC
  KONEC
  SEBER-NEKDE-ZNACKY
KONEC
```



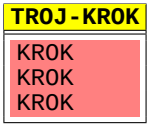

```
SEBER-NEKDE-ZNACKY2 ZNAMENA
  VYBER
  KDYZ JE ZED
 VLEVO-VBOK
  KONEC JINAK
 KROK
  KONEC
  SEBER-NEKDE-ZNACKY2
KONEC
```

[37]



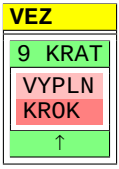

```
TRETINA ZNAMENA
  KDYZ JE ZED
  KONEC JINAK
 KROK
 ?KROK
 ?KROK
 TRETINA
 VZAD
  KONEC
KONEC
```

[50]



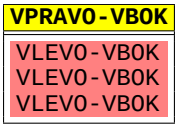

```
TROJ-KROK ZNAMENA
  KROK
  KROK
  KROK
KONEC
```

[11]



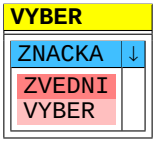

```
VEZ ZNAMENA
  OPAKUJ 9 KRAT
 VYPLN
 KROK
  KONEC
KONEC
```

[17]

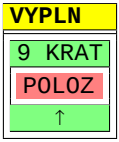



```
VPRAVO-VBOK ZNAMENA
  VLEVO-VBOK
  VLEVO-VBOK
  VLEVO-VBOK
KONEC
```

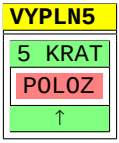
[9]




```
VYBER ZNAMENA                    [35]
  KDYZ JE ZNACKA
 ZVEDNI
 VYBER
  KONEC JINAK
  KONEC
KONEC
```

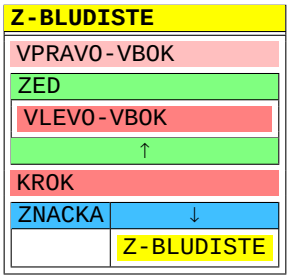



```
VYPLN ZNAMENA                    [17]
  OPAKUJ 9 KRAT
 POLOZ
  KONEC
KONEC
```

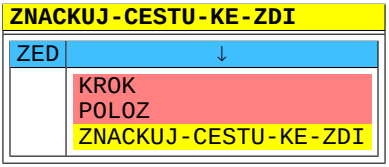



```
VYPLN5 ZNAMENA                   [18]
  OPAKUJ 5 KRAT
 POLOZ
  KONEC
KONEC
```


```
VZAD ZNAMENA                     [49]
  CELEM-VZAD
  KROK
  CELEM-VZAD
KONEC
```

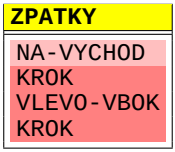



```
Z-BLUDISTE ZNAMENA               [43]
  VPRAVO-VBOK
  DOKUD JE ZED
 VLEVO-VBOK
  KONEC
  KROK
  KDYZ JE ZNACKA
  KONEC JINAK
 Z-BLUDISTE
  KONEC
KONEC
```




```
ZNACKUJ-CESTU-KE-ZDI ZNAMENA
  KDYZ JE ZED
  KONEC JINAK
 KROK
 POLOZ
 ZNACKUJ-CESTU-KE-ZDI
  KONEC
KONEC
```

[45]




```
ZPATKY ZNAMENA
  NA-VYCHOD
  KROK
  VLEVO-VBOK
  KROK
KONEC
```

[48]

Pokud používáte PDF variantu tohoto textu, pak dvojklikem na následující ikonu, vybráním celého textu a použitím příkazu `OPRAV` můžete celý slovník přenést do programu `PC-Karel`.

Příloha 3: Seznam Karlových příkazů

V seznamu jsou řazeny příkazy tak, jak po sobě v knížce následují. U každého příkazu je stručně vysvětlena jeho funkce. Zároveň je uvedeno, na které stránce knížky jej najdeš. Nejsou zde vysvětlena primitiva – to by přece bylo primitivní. Nenajdeš zde taky příkazy obdobné – jako jsou příkazy `NA-JIH` a `NA-SEVER`.

`DVOJ-KROK`	Karel udělá dva kroky ve směru, do kterého je natočen.	[9]
`CELEM-VZAD`	Karel provede čelem vzad.	[9]
`VPRAVO-VBOK1`	Karel provede vpravo vbok pomocí tří vlevo vboků.	[9]
`VPRAVO-VBOK2`	Karel provede vpravo vbok pomocí čelem vzad a vlevo vbok.	[10]
`TROJ-KROK1`	Karel provede tři kroky ve směru, do kterého je natočen, pomocí tří příkazů `KROK`.	[11]
`TROJ-KROK2`	Karel provede tři kroky ve směru, do kterého je natočen, pomocí příkazů `DVOJ-KROK` a `KROK`.	[12]
`CTYR-KROK1`	Karel udělá čtyři kroky ve směru, do kterého je natočen, pomocí dvou příkazů `DVOJ-KROK`.	[12]
`CTYR-KROK2`	Karel udělá čtyři kroky ve směru, do kterého je natočen, pomocí příkazů `TROJ-KROK` a `KROK`.	[12]
`JAKO-KUN`	Karel jde po městě jako šachový kůň po šachovnici.	[13]
`4-KROK`	Karel udělá čtyři kroky ve směru, do kterého je natočen, pomocí příkazu `OPAKUJ`.	[16]
`5-KROK`	Karel udělá pět kroků ve směru, do kterého je natočen, pomocí příkazu `OPAKUJ`.	[16]
`VYPLN`	Karel vyplní prázdné políčko, kde stojí, devíti značkami.	[17]
`VEZ`	Karel postaví ve směru, do kterého je natočen, věž vysokou 9 políček.	[17]
`NA-RAMENO`	Karel přejde na místo, odkud začne stavět rameno jeřábu.	[17]

VYPLN5	Karel položí do prázdného políčka, na kterém stojí, pět značek.	[18]
RAMENO	Karel postaví rameno jeřábu.	[18]
NA-VEZ	Karel přejde z domu na místo, kde bude stavět jeřáb.	[20]
JERAB	Karel sestrojí na obrazovce jeřáb z předem nadefinovaných příkazů.	[20]
NA-NOVY-PANEL	Karel přejde při stavbě domu na místo, odkud bude stavět nový panel.	[24]
PANEL	Karel sestrojí panel pro stavbu domu a přejde na místo pro stavbu nového panelu (NA-NOVY-PANEL).	[25]
OKNO	Karel sestrojí okno pro stavbu domu a přejde na místo pro stavbu nového panelu (NA-NOVY-PANEL).	[25]
DVERE	Karel sestrojí dveře pro stavbu domu a přejde na místo pro stavbu nového panelu (NA-NOVY-PANEL).	[25]
PRIZEMI	Karel z jednotlivých prvků smontuje přízemí domu.	[26]
PREKLAD	Karel sestaví překlad nad přízemím domu. Tento příkaz je zároveň příkazem pro sestrojení střechy.	[26]
NA-PREKLAD	Karel přejde při stavbě domu na místo, odkud začne montovat překlad.	[27]
NA-PATRO	Karel přejde při stavbě domu na místo, odkud začne montovat patro.	[27]
DUM	Karel postaví dům.	[27]
PIRUETA	Karel dělá piruetu pomocí rekurze.	[28]
HLIDEJ	Karel hlídá pmocí rekurze pomyslný dvorek.	[31]
DO-ZDI	Karel dojde pomocí rekurze k libovolně vzdálené zdi a narazí do ní.	[31]
KE-ZDI	Karel dojde pomocí KDYZ a rekurze ke zdi, k níž je natočen, a zůstane před ní stát aniž by do ni narazil.	[31]

NA-DALSI-ZNACKU	Karel dojde na značku, která je položena ve směru, do kterého je natočen. Na značce se zastaví.	[32]
NA-SEVER	Karel se natočí ve směru žádané světové strany (SEVER, JIH, VYCHOD, ZAPAD).	[33]
K-JIZNI-ZDI	Karel dojde k zvolené (JIZNI, SEVERNI, ZAPADNI, VYCHODNI) zdi.	[33]
DOMU	Karel se vrátí z libovolného místa v prázdném městě domů.	[33]
VYBER	Karel úplně vysbírá libovolný počet značek na místě, kde stojí.	[35]
SEBER-NEKDE-ZNACKU	Karel dojde na značku položenou u zdi, zvedne ji a zůstane stát.	[36]
SEBER-NEKDE-ZNACKY	Karel vysbírá všechny značky položené u zdí.	[36]
SEBER-NEKDE-ZNACKY2	Zjednodušený zápis SEBER-NEKDE-ZNACKY.	[37]
OKOLO	Karel obchází okolo celého města bez zdí.	[38]
OTOC-VLEVO	Karel se po příchodu ke zdi otočí a přejde na sousední řadu políček vlevo.	[38]
?OTOC-VLEVO	Karel po příchodu ke zdi přejde na sousední řadu vlevo jen pokud nenarazí do zdi.	[38]
PROJDI	Karel prochází všemi políčky města beze zdí.	[39]
KE-ZDI2	Karel dojde opatrně až ke zdi pomocí DOKUD.	[39]
NENI-ZED	Karel se otáčí tak dlouho, dokud je před ním zeď.	[40]
PRYC	Karel vyjde z jednoduchého bludiště.	[42]
Z-BLUDISTE	Karel vyjde z jednoduchého bludiště a zastaví se na značce.	[43]
POLOZ-MINU-U-ZDI	Karel dojde ke zdi, položí tam minu – značku – a zůstane na ní stát.	[44]

ZNACKUJ-CESTU-KE-ZDI	Karel jde ke zdi a cestou pokládá značky.	[45]
PO-ZNACKACH-ZPET	Karel jde po značkách zpět na výchozí místo a sbírá je po sobě.	[45]
POLOZ-MINU-KE-ZDI-2	Karel dojde ke zdi značkuje si cestu, položí tam minu – značku – a po značkách se vrátí zpět na výchozí místo.	[45]
POLOZ-MINU-KE-ZDI-3	Karel dojde ke zdi, položí tam minu – značku – a vrátí se zpět.	[46]
NA-LEVOU-PYRAMIDU	Karel se při stavbě pyramidy posune na místo stavby levé podpyramidy.	[47]
NA-PRAVOU-PYRAMIDU	Karel se při stavbě pyramidy posune na místo stavby pravé podpyramidy.	[48]
ZPATKY	Karel se při stavbě pyramidy posune zpět na výchozí místo.	[48]
PYRAMIDA	Karel postaví pyramidu – Pascalův trojúhelník – ze značek.	[48]
VZAD	Karel udělá jeden krok vzad.	[49]
?KROK	Karel udělá opatrný krok – krok jen pokud před ním není zeď.	[49]
PUL	Karel dojde do poloviny vzdálenosti mezi ním a zdí.	[49]
DO-STREDU	Karel dojde z domečku do středu města.	[50]
TRETINA	Karel dojde do třediny vzdálenosti mezi ním a zdí.	[50]

Obsah

ISBN 99972-03-09-7