Projekt OP VK „Inovace studijních oborů zajišťovaných katedrami PřF UHK“

Registrační číslo: CZ.1.07/2.2.00/28.0118

1 Programovací jazyk KAREL

1.1 Jak byl vytvořen a co získal do vínku robot KAREL

S myšlenkou použít k výuce algoritmického myšlení a nejjednodušších základů programování světa robota, přišel v roce 1981 profesor **Richard E. Pattis** ve své knize *Karel the Robot: A gentle introduction to the art of programming*.[1] Proč americký profesor informatiky pojmenoval svého robota Karel a ne Charles? Bylo to na počest českého spisovatele Karla Čapka, autora slavné divadelní hry RUR, ve které bylo slovo robot (jako novotvar vzniklý z českého slova robota, tj. těžká fyzická práce) poprvé použito.

Programovat robota Karla znamená ovládnout jeho mikrosvět. Karel se může pohybovat na jakési šachovnici, čtvercové hrací desce s čtvercovými poli. Deska, nazývaná také dvorek či město, má typický rozměr 10 x 10 polí, ale může být i větší (záleží na konkrétní implementaci). Na začátku je Karel „novorozeně“, které rozumí jen několika slovům. Umí udělat pouze čtyři základní pohyby:

- krok směrem, do kterého je otočen (příkaz KROK),
- otočení o 90° proti směru hod. ručiček (příkaz VLEVO-VBOK),
- položit značku na pole, na kterém stojí (příkaz POLOŽ),
- zvednout značku z pole, na kterém stojí (příkaz ZVEDNI).

Neumí tedy např. udělat krok vzad, úkrok stranou, otočit se o 90° ve směru hodinových ručiček, nebo otočit se o 180°. Toto všechno může

[1] Robot Karel: Jemný úvod do umění programovat.

udělat pomocí opakování a kombinování základních čtyř příkazů, nebo, což je pochopitelně lepší, ho můžeme naučit vykonávat nové příkazy.

Kromě základních pohybů má Karel také smysly. Jsou velmi jednoduché, takže pomocí nich je schopen určit pouze:

- zda před ním JE ZEĎ, nebo NENÍ ZEĎ; zeď může být před Karlem ze dvou důvodů,
  - buď stojí na kraji hrací desky (někdy se jí také říká dvorek, což lépe vysvětluje představu zdi okolo desky) a je natočen tak, že dalším krokem by se ocitl mimo desku,
  - nebo jsme zeď Karlovi umístili na hrací desku, a to na políčko, na které by Karel vstoupil dalším krokem,
- zda pod ním, tedy na políčku, na kterém právě stojí, JE ZNAČKA, nebo NENÍ ZNAČKA; není však schopen spočítat, kolik značek je zde umístěno,
- do třetice má Karel v sobě zabudován kompas, který je schopen rozlišit, do které ze čtyř základních světových stran je natočen (sever je nahoru, jih dolů, západ vlevo a východ vpravo); vyhodnotí tedy pravdivost výroků: JE SEVER, NENÍ SEVER, JE JIH, NENÍ JIH, JE ZÁPAD, NENÍ ZÁPAD, JE VÝCHOD, nebo NENÍ VÝCHOD,
- posledním smyslem je schopnost poznat, zda políčko, na kterém Karel stojí je jeho výchozí pole, jedno z polí na šachovnici (na dvorku, ve městě) můžeme označit jako domov; Karel tedy rozliší, zda pole, na kterém právě stojí, JE DOMOV, nebo NENÍ DOMOV.

Třetí oblastí, kterou Karel ovládá, jsou řídící struktury programů. Konkrétně podmíněný příkaz, čili větvení programu a několik různých příkazů cyklu. Tyto řídíci struktury si vysvětlíme na příkladech v dalších podkapitolách.

Příkazy Karlova jazyka jsme uváděli v češtině. Společným rysem všech dětských programovacích jazyků a jejich odlišností od běžných programovacích jazyků je lokalizace do mateřského jazyka dítěte (žáka základní školy), které chceme naučit algoritmickému myšlení a základům programování. Lokalizace usnadní žákům pochopení příslušného

programovacího jazyka, ale na druhou stranu vede k drobným odlišnostem v překladu mezi jednotlivými implementacemi.

1.2 KAREL dělá obraty a pochoduje

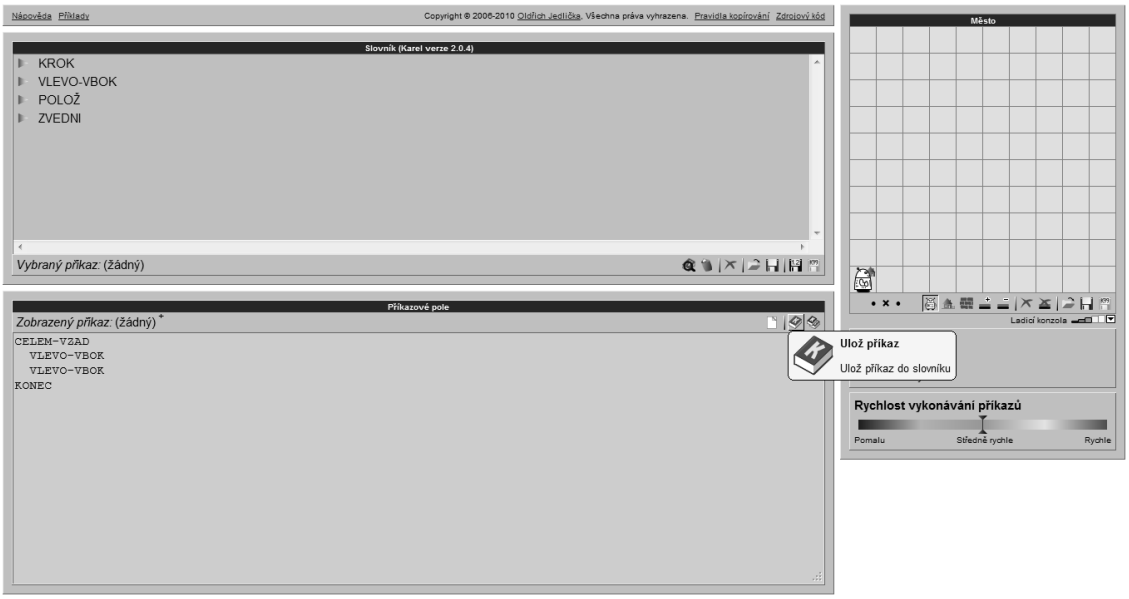
V této kapitole si ukážeme jak snadné je naučit Karla další prvky pořadové přípravy (vojenský dril není nijak intelektově náročný). Začněme nejprve tím, že naučíme Karla dva nové obraty, a to o 180°, tj. čelem vzad, a o 90° ve směru hodinových ručiček, tj. vpravo vbok. Vyzkoušíme si práci s Karlem a přidání těchto jednoduchých příkazů do jeho slovníku v interaktivním webovém prostředí bez nutnosti instalace.[2]

Program začíná vždy názvem nového příkazu a končí slovem KONEC. Mezi ně zapisujeme další příkazy v pořadí, v jakém se mají provádět.

ČELEM-VZAD
    VLEVO-VBOK
    VLEVO-VBOK
KONEC
	VPRAVO-VBOK
	    ČELEM-VZAD
	    VLEVO-VBOK
	KONEC

Karla můžeme naučit, aby se choval rozumně, např. aby nenarážel do zdi, ale ve chvíli, kdy ji před sebou vidí, raději zatočil. K tomu nám slouží podmíněný příkaz KDYŽ *podmínka* ‖ *příkaz(y) 1* ‖ KONEC, JINAK ‖ *příkaz(y) 2* ‖ KONEC[3]. Podobně můžeme zařídit, aby se Karel nesnažil zvednout značku, když pod ním žádná značka není. V tomto případě použijeme jednodušší verzi podmíněného příkazu KDYŽ *podmínka* ‖ *příkaz(y)* ‖ KONEC.

[2] Vývojové prostředí robota Karla naprogramované Oldřichem Jedličkou jako skript na straně klienta v jazyce JavaScript najdete na webu http://karel.oldium.net/.
[3] Symbol dvojité svislé čáry ‖ používáme pro oddělení částí příkazu, které musí začínat na novém řádku.



Obr. 1.1 - Webové prostředí robota Karla

CHYTRÝ-KROK KDYŽ NENÍ ZEĎ KROK KONEC, JINAK VLEVO-VBOK KONEC KONEC	CHYTŘE-ZVEDNI KDYŽ JE ZNAČKA ZVEDNI KONEC KONEC

Ještě rozumnější chování by bylo, kdybychom Karlovi řekli: „Jdi ke zdi!“ a on by šel tak dlouho, dokud by ke zdi nedošel. Jindy bychom chtěli, aby se Karel, když už má v sobě zabudovaný kompas, natočil na určitou světovou stranu, třeba na sever.

JDI-KE-ZDI DOKUD NENÍ ZEĎ KROK KONEC KONEC	NA-SEVER DOKUD NENÍ SEVER VLEVO-VBOK KONEC KONEC

K takovým úkolům nám poslouží příkaz cyklu s podmínkou. Provádí se tak dlouho, dokud je splněna podmínka. Pokud není podmínka splněna hned na začátku, příkazy uvnitř cyklu se neprovedou vůbec. Podmínka může být uvedena také na konci cyklu. Představme si, že chceme, aby Karel, který stojí na poli označeném jako „domov“ v levém dolním rohu dvorku, obešel pomocí svého „chytrého kroku“, který jsme ho před chvílí naučili, celý dvorek kolem dokola a po návratu domů se opět zastavil. Kdybychom použili podmínku na začátku cyklu, Karel by se vůbec nehnul. Hned na začátku by zjistil, že je doma a už by neudělal ani jeden krok. Proto použijeme příkaz cyklu s podmínkou na konci, který má strukturu OPAKUJ ‖ *příkaz(y)* ‖ AŽ *podmínka*. Cyklus končí, až když je podmínka splněna, ale vždy se provede aspoň jednou.

```
OBCHŮZKA
    OPAKUJ
        CHYTRÝ-KROK
    AŽ JE DOMOV
KONEC
```

Posledním typem cyklu je cyklus s předem daným počtem opakování. Příkaz má strukturu OPAKUJ *n*-KRÁT ‖ *příkaz(y)* ‖ KONEC. Ukažme si jeho použití, jestliže chceme, aby Karel udělal trojkrok, nebo aby obešel čtvercovou trasu. Během provádění těchto dvou příkazů nevyhodnocuje Karel situaci ve svém okolí. Když nechceme, aby narazil do zdi, musíme mu příkaz dát jen tam, kde vidíme dost místa pro jeho vykonání.

```
TROJKROK
    OPAKUJ 3-KRÁT
        KROK
    KONEC
KONEC
```

```
ČTVEREC
    OPAKUJ 4-KRÁT
        TROJKROK
        VLEVO-VBOK
    KONEC
KONEC
```

1.3 Grafické znázornění algoritmů, vývojové diagramy a kopenogramy

Programy představují vlastně slovní zápis algoritmů v určitém formálním jazyce. Některým žákům bude bližší grafické znázornění struktury algoritmu. K tomuto účelu běžně užíváme jednak vývojové diagramy, jednak kopenogramy. Právě kopenogramy jsou spojeny s implementací jazyka Karel v českém prostředí[4].

Vývojové diagramy[5] rozlišují jednotlivé typy řídících struktur různými tvary obrazců. Ovál znázorňuje začátek a konec algoritmu, obdélník jednotlivé výkonné příkazy, nebo jejich bloky, kosočtverec větvení algoritmu pomocí podmíněného příkazu, nepravidelný šestiúhelník řídící příkaz cyklu s předem daným počtem opakování. Chceme-li zdůraznit, že některý výkonný příkaz bude popsán samostatným algoritmem, zdvojíme čáry na levém a pravém okraji obdélníku. Pro znázornění příkazů vstupu a výstupu za běhu programu se používá rovnoběžník.

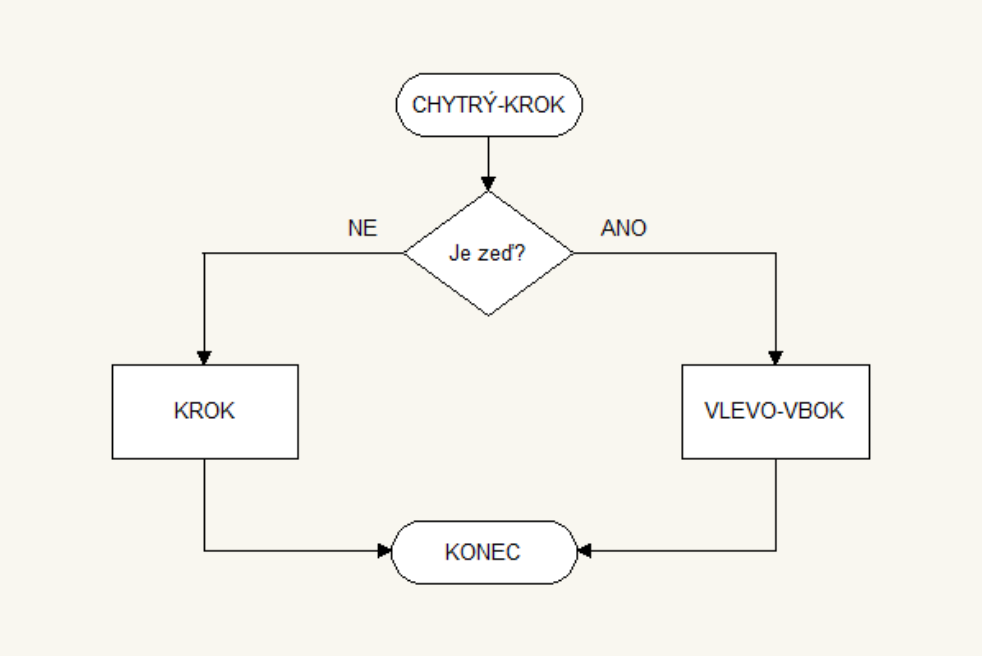
Oproti tomu kopenogramy rozlišují jednotlivé typy řídících struktur barevně. Záhlaví příkazu vybarvíme žlutě, výkonné příkazy růžově, podmíněný příkaz světle modře[6] a začátek a konec cyklu světle zeleně. K těmto černobílým skriptům je třeba si pořídit pastelky (nebo zvýrazňovače) a kopenogramy vybarvit. Kopenogramy jsou český vynález, i když částešně inspirovaný Nassi-Schneidermanovými diagramy, jejich název je odvozen z prvních dvou písmen příjmení jejich autorů (Jiří Kofránek, Rudolf Pecinovský a Petr Novák).

---

[4] S kopenogramy pracuje důsledně např. metodický materiál *Programovací jazyk KAREL* (Vejvoda – Rytíř, 2001), bibliografické údaje této příručky jsou uvedeny v seznamu literatury doporučené k samostatnému studiu.

[5] Vývojové diagramy v tomto skriptu byly vytvořeny pomocí programu DiagramDesigner.

[6] V kopenogramu příkazu CHYTRÝ-KROK tedy vybarvíme pole CHYTRÝ-KROK žlutě, pole NENÍ ZEĎ a pole se šipkou světle modře a pole s výkonnými příkazy KROK a VLEVO-VBOK růžově.

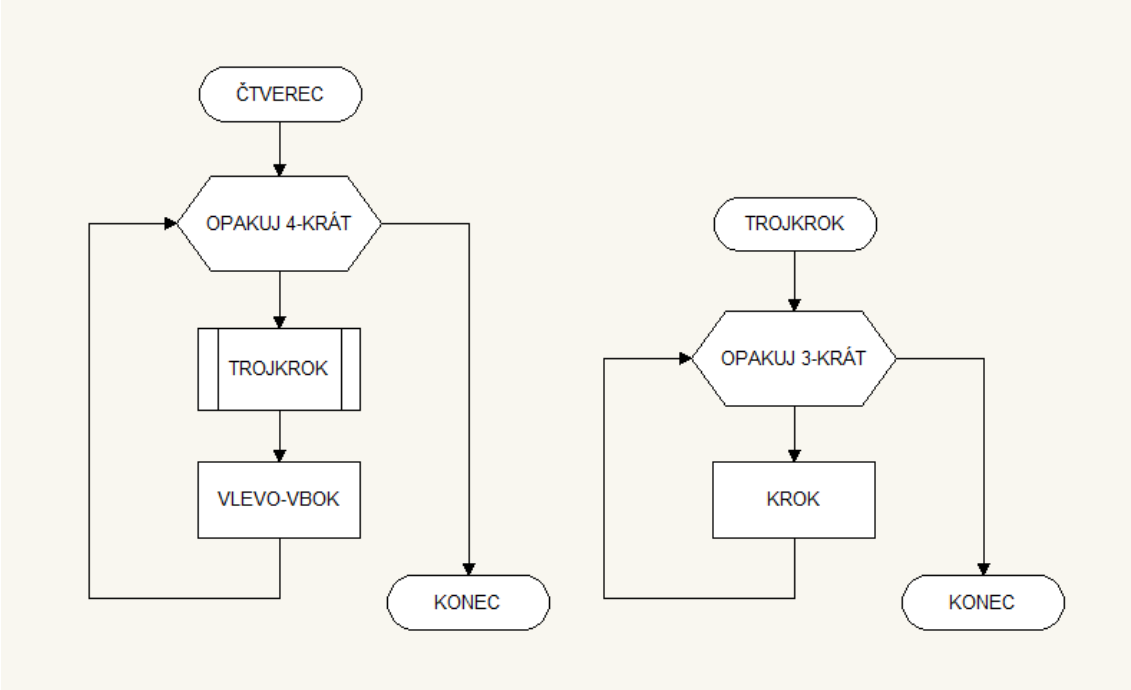


Obr.1.2 - Vývojový diagram příkazu CHYTRÝ-KROK

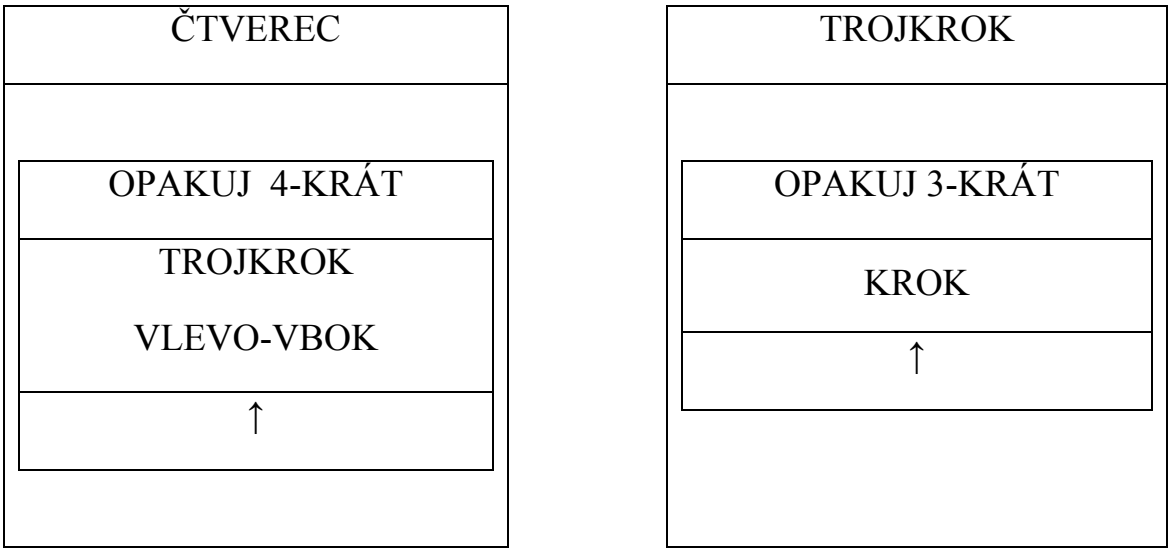
CHYTRÝ-KROK	
NENÍ ZEĎ	↓
KROK	VLEVO-VBOK

Obr.1.3 - Kopenogram příkazu CHYTRÝ-KROK

Kopenogramy jsou jedním z typů stukturogramů. Používáním sturkturogramů ve výuce vytváří učitel u svých žáků správné představy a správné návyky, pokud jde o strukturované programování.



Obr.1.4 - Vývojový diagram příkazů ČTVEREC a TROJKROK



Obr.1.5 - Kopenogramy příkazů ČTVEREC a TROJKROK

1.4 KAREL staví ze značek schody a značí si jimi cestu

V této kapitole si pohrajeme s Karlovou schopností pokládat a zvedat značky. Začněme něčím jednoduchým, nechme Karla postavit schody. Protože Karlovy příkazy nemají žádné příkazy, musíme pevně zvolit výšku schodu. Zvolíme výšku 2 značek (použijeme klasické značky, nikoliv kuličky). Nejprve navrhneme strukturu celé stavby a teprve pak dořešíme dílčí příkazy. Zadávat jednotlivé algoritmy musíme ovšem v opačném pořadí. Příkaz pro stavbu schodů ukazuje, že je možné v jazyce Karel psát i delší programy.

```
SCHODY
        NA-VÝCHOD
        KROK; POLOŽ2
        KROK; POLOŽ4
        KROK; POLOŽ6
        KROK; ZAPLŇ
        KROK; ZAPLŇ
        ÚKROK-VLEVO; POLOŽ2
        ÚKROK-VPRAVO
        KROK; ZAPLŇ
        ÚKROK-VLEVO; POLOŽ4
        ÚKROK-VPRAVO
        KROK; ZAPLŇ
        ÚKROK-VLEVO; POLOŽ6
        ÚKROK-VPRAVO
        KROK; ZAPLŇ
        ÚKROK-VLEVO; ZAPLŇ
        ÚKROK-VPRAVO
        KROK
KONEC
```

```
NA-VÝCHOD
        DOKUD NENÍ VÝCHOD
                VLEVO-VBOK
        KONEC
KONEC
```

```
POLOŽ2
        POLOŽ; POLOŽ
KONEC
```

```
POLOŽ4
        POLOŽ2; POLOŽ2
KONEC
```

```
POLOŽ6
        POLOŽ4; POLOŽ2
KONEC
```

```
ZAPLŇ
        POLOŽ4; POLOŽ4
KONEC
```

```
ÚKROK-VLEVO
        VLEVO-VBOK
        KROK
        VPRAVO-VBOK
KONEC
```

Značky může do města pokládat nejen Karel, ale můžeme mu je do jeho města předem umístit a tím mu například vyznačit cestu, po které se má pohybovat. Při pohybu po značkách se snaží Karel jít vždy nejprve rovně, když to nejde, zkusí zahnout doleva a když ani to nejde, pak zahnout doprava. Jako úplně poslední možnost si ponechává obrat o 180° a cestu zpátky po cestě, kterou předtím přišel. Spouštěcím příkazem programu je příkaz PO-ZNAČKÁCH.

```
NA-ZNAČKU
    KDYŽ NENÍ ZEĎ
        KROK
    KONEC, JINAK
        VLEVO-VBOK
        NA-ZNAČKU2
    KONEC
    KDYŽ NENÍ ZNAČKA
        ZPĚT
        VLEVO-VBOK
        NA-ZNAČKU2
    KONEC
KONEC
```

```
NA-ZNAČKU2
    KDYŽ NENÍ ZEĎ
        KROK
    KONEC, JINAK
        ČELEM-VZAD
        NA-ZNAČKU3
    KONEC
    KDYŽ NENÍ ZNAČKA
        ZPĚT
        ČELEM-VZAD
        NA-ZNAČKU3
    KONEC
KONEC
```

```
NA-ZNAČKU3
    KDYŽ NENÍ ZEĎ
        KROK
    KONEC, JINAK
        VPRAVO-VBOK
        KROK
    KONEC
    KDYŽ NENÍ ZNAČKA
        ZPĚT
        VPRAVO-VBOK
        KROK
    KONEC
KONEC
```

```
ČELEM-VZAD
    VLEVO-VBOK
    VLEVO-VBOK
KONEC
```

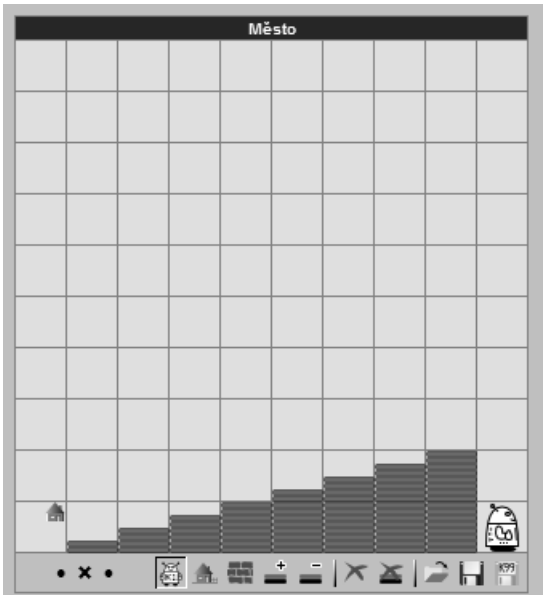
```
VPRAVO-VBOK
    ČELEM-VZAD
    VLEVO-VBOK
KONEC
```

```
PO-ZNAČKÁCH
  OPAKUJ 100-KRÁT
    NA-ZNAČKU
  KONEC
KONEC
```

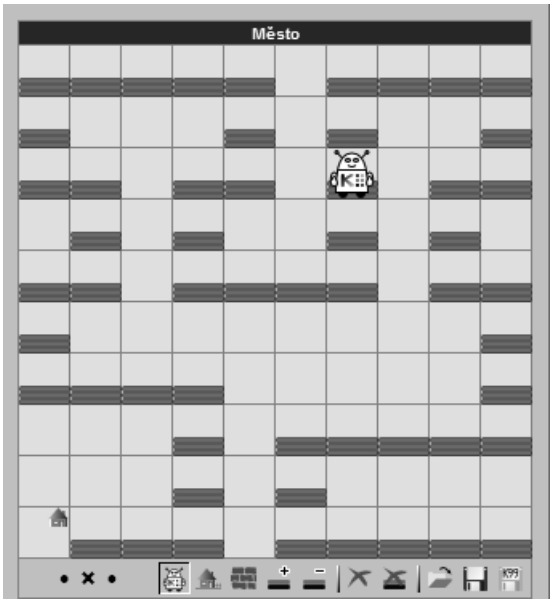
1.5 Procvičení algoritmizace a programování v prostředí robota KARLA

Abyste si upevnili získané vědomosti, vypracujte následující cvičení:

a) Vybarvěte podle konvence popsané v kapitole 1.3 kopenogramy příkazů CHYTRÝ-KROK (obr.1.3), ČTVEREC a TROJKROK (obr.1.5).
b) Znázorněte pomocí vývojového diagramu algoritmus „chytrého úkroku“, při kterém se Karel nejprve otočí vlevo, podívá se, zda před ním není zeď a pokud ne, tak udělá krok a otočí se vpravo. Kdyby ovšem před sebou viděl zeď, otočí se o 180° a podívá se, zda je zeď směrem vpravo od jeho původního směru. Pokud ne, udělá krok a otočí se vlevo, pokud ano, otočí se na místě vlevo a zůstane stát.
c) Znázorněte pomocí kopenogramu algoritmus popsaný ve cvičení b).
d) Naprogramujte a odlaďte příkaz CHYTRÝ-ÚKROK, který realizuje postup popsaný ve cvičení b).
e) Navrhněte algoritmus, jehož realizací Karel systematicky projde všechna pole svého dvorku a sesbírá všechny značky na něm rozmístěné. Postup můžete popsat slovně, nebo znázornit buď vývojovými diagramy, nebo kopenogramy. Vyberte si jednu z možností.
f) Algoritmus navržený ve cvičení e) realizujte jako příkaz UKLIĎ-DVOREK, podpříkazy potřebné k realizaci celého postupu volte podle potřeby a dle svého návrhu ve cvičení e).
g) Naprogramujte a odlaďte příkaz PYRAMIDA, pomocí kterého Karel postaví stupňovitou pyramidu. Můžete si ji představit také jako osově souměrné schody, které nejprve stoupají a pak zase klesají.
h) Navrhněte algoritmus, ve kterém bude Karel chodit po značkách, ale na rozdíl od postupu popsaného v kapitole 1.4 bude značky zvedat a značení tak během své cesty rušit. Ve chvíli, kdy dojde na konec trasy a nebude mít kolem sebe již žádné pole se značkou, se zastaví. Postup můžete popsat slovně, nebo znázornit buď vývojovými diagramy, nebo kopenogramy. Vyberte si jednu z možností.



Obr.1.6 - KAREL staví schody



Obr.1.7 - KAREL chodí po značkách

i) Algoritmus navržený ve cvičení h) realizujte jako příkaz PO-STOPĚ, podpříkazy volte podle potřeby a dle svého návrhu ve cvičení e).

1.6 Seznamujeme KARLA s rekurzí

Karla jsme naučili spoustu zajímavých příkazů, ale dosud jsme mu neprozradili jedno velké překvapení. Spočívá v možnosti použít uvnitř definice příkazu, kterou teprve vytváříme (ale ještě jsme ji neukončili), tento nově vytvářený příkaz, jako kdyby jej už Karel znal. Takové volání sama sebe nazýváme rekurze a je to velmi silný nástroj v řadě programovacích jazyků.

Skutečnost, že jazyk robota Karla umožňuje rekurzi, nám nejen pomůže zapsat jednodušším způsobem některé dříve definované příkazy, ale také zapsat zcela nové příkazy, které by bez rekurze vůbec nebylo možné realizovat. Například při stavbě schodů už nemusíme předem

určovat počet položených značek, stačí dát Karlovi chytře najevo, aby na každé další pole dal vždy o jednu značku víc.[7]

```
SCHODY-N
    KROK
    SCHODY-R
    ZPĚT
KONEC
```

```
ČELEM-VZAD
    VLEVO-VBOK
    VLEVO-VBOK
KONEC
```

```
ZPĚT
    ČELEM-VZAD
    KROK
    ČELEM-VZAD
KONEC
```

```
SCHODY-R
    KDYŽ NENÍ ZEĎ
        KDYŽ JE ZNAČKA
            ZVEDNI
            KROK
            POLOŽ
            ZPĚT
            SCHODY-R
            POLOŽ
        KONEC, JINAK
            KROK
            POLOŽ
            SCHODY-R
            ZPĚT
        KONEC
    KONEC
KONEC
```

Příkladem příkazu, který by bez rekurze vůbec nešel naprogramovat, je příkaz JDI-DO-PŮLKY, který zajistí, že Karel dojde do poloviny vzdálenosti mezi svoji okamžitou pozicí a zdí.

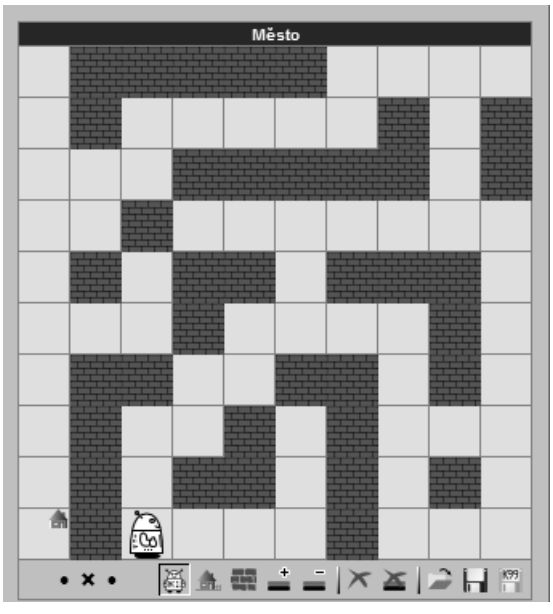
[7] Karel neumí rozpoznat kolik značek má pod sebou, přitom ale pokus, položit více než osm značek na jedno pole, skončí chybným hlášením. Proto je tento program potřeba spouštět příkazem SCHODY-N, který vložením nadbytečného kroku zaručí nepřekročení maximálního počtu položených značek.

```
JDI-DO-PŮLKY
        KDYŽ JE ZEĎ
                ČELEM-VZAD
        KONEC, JINAK
                KROK
                KDYŽ NENÍ ZEĎ
                        KROK
                KONEC
                JDI-DO-PŮLKY
                KROK
        KONEC
KONEC
```

```
ČELEM-VZAD
        VLEVO-VBOK
        VLEVO-VBOK
KONEC
```

Pomocí rekurze můžeme robota Karla naučit hledat cestu z labyrintu. Postavíme labyrint ze zdí a umístíme Karla daleko od políčka označeného DOMOV a dáme mu pokyn, aby se vydal na cestu. Před každým krokem Karel položí značku a tím si vyznačí, na kterém políčku už byl. Používání značek zaručí, že Karel nakonec najde cestu domů.

```
LABYRINT
   KDYŽ NENÍ DOMOV
        KDYŽ NENÍ ZEĎ
            POLOŽ;  KROK
            KDYŽ                NENÍ
ZNAČKA
                LABYRINT
            KONEC, JINAK
                ZVEDNI;  ZPĚT
                ZVEDNI
                VLEVO-VBOK
                LABYRINT
            KONEC
        KONEC, JINAK
            VLEVO-VBOK
            LABYRINT
```



Obr.1.8. - KAREL bloudí labyrintem

```
        KONEC
    KONEC
KONEC
```

Poslední ukázkou rekurze, kdy jedna z nově definovaných procedur volá sama sebe, bude soubor příkazů pro vytvoření Pascalovo trojúhelníka ze značek. Jsme omezeni maximálním počtem značek na jednom poli a také velikostí Karlova dvorku. Přesto je příkaz zajímavý tím, že Karla musíme naučit sčítat počty značek v určených dvou polích a výsledek zobrazit ve třetím poli.

```
PASCAL
        NA-SEVER; KE-ZDI
        VPRAVO-VBOK
        OPAKUJ 4-KRÁT
                KROK
        KONEC
        POLOŽ; KE-ZDI; CR-LF
        OPAKUJ 4-KRÁT
                PASCAL-RADEK
                CR-LF
        KONEC
        VLEVO-VBOK
        KROK; POLOŽ
        ČELEM-VZAD; KE-ZDI
KONEC
```

```
PRESUN
        L
        KDYŽ JE ZNAČKA
                ZVEDNI
                VLEVO-VBOK; L
                POLOŽ
                VLEVO-VBOK
                PRESUN
                VLEVO-VBOK; L
                POLOŽ
                VLEVO-VBOK; L
        KONEC, JINAK
                VLEVO-VBOK; L
        KONEC
KONEC
```

```
PASCAL-RADEK
        KROK
        OPAKUJ 8-KRÁT
                PRESUN
                ČELEM-VZAD
                PRESUN
                KROK
        KONEC
KONEC
```

```
L
        KROK; VLEVO-VBOK
        KROK
KONEC
```

```
CR-LF
        VPRAVO-VBOK; KROK
        VPRAVO-VBOK; KE-ZDI
        ČELEM-VZAD
KONEC
```

V tabulce příkazů již znovu neopakujeme dříve odladěné příkazy ČELEM-VZAD, VPRAVO-VBOK a KE-ZDI.

1.7 Instalujeme KARLA s rozšířenou nabídkou příkazů

Práce s Karlem ve webovém rozhraní plně vyhovuje pro úvod do algoritmizace a programování. Jeho výhodou je jednoduchost a dostupnost. Co se žáci naučí ve škole, mohou si hned doma zopakovat bez nutnosti cokoliv instalovat do svého domácího počítače. Prostředí i obrázek robota jsou navíc velmi zdařilé z hlediska grafického.

Pokud se však rozhodneme pro instalaci klasického spustitelného programu, můžeme získat některá zajímavá rozšíření[8]. Jednotlivé implementace prostředí a jazyka robota Karla se také mohou navzájem lišit v některých detailech. My si rozdíly ukážeme na příkladu porovnání webové verze (autor Oldřich Jedlička) a spustitelné verze KarelWin (autor Petr Laštovička). Prvním rozdílem je definice nového příkazu. Zatímco ve webové verzi jsme uváděli pouze název příkazu, implementace KarelWin vyžaduje jedno ze tří klíčových slov:

- PŘÍKAZ … spustitelná procedura, na vyžádání se provede,
- FUNKCE … samostatně nelze spustit, ale lze ji volat z procedury, vrací číselnou hodnotu,
- PODMÍNKA … podobně jako FUNKCE, ale místo čísla vrací logickou hodnotu (pravda, nebo nepravda).

Za klíčovým slovem teprve následuje název příkazu, funkce, nebo podmínky, za kterým navíc mohou (ale nemusí) bezprostředně navazovat kulaté závorky, v nichž je uveden parametr (nebo několik parametrů) oddělený čárkami.

Další odlišnost je v základních příkazech:

[8] Tak tomu je např. v implementaci KarelWin autora Petra Laštovičky, která je dostupná na webové stránce http://petr.lastovicka.sweb.cz/ostatni.html#karel. V tomto případě se dokonce nejedná o klasickou instalaci, ale pouze o prosté překopírování souborů, z nichž nejdůležitější je spustitelný soubor (aplikace) KarelWin.exe.

- Web Karel … KROK, VLEVO-VBOK, POLOŽ, ZVEDNI,
- KarelWin … KROK, VLEVO, VPRAVO, POLOŽ, ZVEDNI, MALUJ *'znak'*, DOMŮ.

Pokud bychom chtěli používat příkazy dříve odladěné ve webovém prostředí, není v tom velký problém, musíme však na začátek každé definice přidat slovo PŘÍKAZ, dodefinovat příkaz VLEVO_VBOK jako alias základního příkazu VLEVO a ve všech příkazech s pomlčkou použít místo pomlčky (-) podtržítko (_). Nový příkaz MALUJ '*znak*' napíše na pole pod Karlem znak uvedený mezi apostrofy. Je také možné psát MALUJ *ascii*, kde *ascii* je číslený kód znaku podle ASCII[9]. Příkaz DOMŮ umístí Karla do levého dolního rohu města a otočí jej směrem na východ.

Kromě písmen a podtržítka můžeme v názvech příkazů používat znaky ? ! : ` @ # a číslice. Název však nesmí začínat číslicí! Příklad definic příkazů:

```
PŘÍKAZ KROKY(n)
    OPAKUJ n
        KDYŽ NENÍ ZEĎ
            KROK
        KONEC
    KONEC
KONEC
```

```
PŘÍKAZ VLEVO_VBOK
    VLEVO
KONEC
```

```
PŘÍKAZ KE_ZDI
    DOKUD NENÍ ZEĎ
        KROK
    KONEC
KONEC
```

Podmínky v obou implementacích musí předcházet buď klíčové slovo JE, nebo klíčové slovo NENÍ. Nabídka aplikace pro OS Windows je opět o něco širší:

- Web Karel … SEVER, JIH, VÝCHOD, ZÁPAD, ZEĎ, ZNAČKA,
- KarelWin … SEVER, JIH, VÝCHOD, ZÁPAD, ZEĎ,

[9] ASCII … American Standard Code for Information Interchange

ZNAČKA, MÍSTO, ZNAK, ZNAK=‘*znak*‘

Podmínka MÍSTO rozhodne, zda je na daném poli ještě místo k položení značky, podmínka ZNAK, zda je na poli napsán nějaký znak, a podmínka ZNAK=‘*znak*‘, zda je zde zapsán konkrétní znak, uvedený mezi apostrofy. Další podmínky je možné definovat pomocí programů označených klíčovým slovem PODMÍNKA.

Zajímavým rozšířením jsou příkazy pro řízení běhu programu RYCHLE, POMALU a ČEKEJ *ms* (kde *ms* je počet milisekund, po který má Karel čekat), resp. ČEKEJ -1 (čekání na stisk klávesy). Pravděpodobně ještě zajímavější je možnost pracovat s proměnnými (i když pouze s celočíselnými) a využívat vestavěné a uživatelem definované funkce. Novou proměnnou deklarujeme příkazem ČÍSLO *p*, kde *p* je proměnná, ale ještě vhodnější je přiřadit proměnné zároveň nějakou hodnotu, příkaz pak má tvar ČÍSLO *p* = *číselný výraz*. S čísly pak můžeme provádět běžné početní operace +, –, *, / a % (% znamená zbytek po celočíselném dělení).

```
PŘÍKAZ ABECEDA
    DOMŮ
    ČÍSLO I=0
    OPAKUJ 26
        I = I+1
        KROK
        MALUJ 64 + I
    KONEC
    KROK
KONEC
```

```
PŘÍKAZ NAPIŠ
    DOMŮ
    ČÍSLO p = 32
    DOKUD NENÍ p = 13
        MALUJ p
        KROK
        ČEKEJ -1
        p = KLÁVESA
    KONEC
KONEC
```

Všimněte si, že abeceda s 26 znaky se nám do Karlova města pohodlně vešla. Rozměry plochy, po níž se pohybuje KarelWin, jsou totiž implicitně 40 x 20 polí. Šikovným využitím proměnných a funkcí dokážeme zadávat Karlovi stiskem kláves znaky, které má vypsat. Funkce KLÁVESA totiž vrací ASCII kód znaku. Dalšími novými funkcemi, kromě funkce KLÁVESA, jsou funkce NÁHODA(*n*), která vrací náhodné

přirozené číslo z množiny {0, 1, 2, …, n-2, n-1}, a funkce ZNAK, která vrací hodnotu znaku, který se právě nachází pod Karlem.

```
PŘÍKAZ CHAOS
    OPAKUJ 300
        OPAKUJ NÁHODA(3)
            VLEVO
        KONEC
        OPAKUJ NÁHODA(20)
            KDYŽ NENÍ ZEĎ
                KROK
            JINAK
                ČELEM_VZAD
            KONEC
        KONEC
        OPAKUJ NÁHODA(10)
            KDYŽ JE MÍSTO
                POLOŽ
            KONEC
        KONEC
    KONEC
KONEC
```

```
PŘÍKAZ ZNAČKY_NA_ČÍSLICE
    DOMŮ
        OPAKUJ 20
        DOKUD NENÍ ZEĎ
            PŘEVOD
            KROK
        KONEC
        KDYŽ JE ZEĎ
            PŘEVOD
            VLEVO
            KDYŽ NENÍ ZEĎ
                KROK
            KONEC
            VLEVO
            KE_ZDI
            ČELEM_VZAD
        KONEC
    KONEC
KONEC
```

```
PŘÍKAZ PŘEVOD
    ČÍSLO k=0
    DOKUD JE ZNAČKA
        ZVEDNI
        k=k+1
    KONEC
    KDYŽ k>0
        MALUJ 48+k
    KONEC
KONEC
```

Příkazy cyklů se v implementaci KarelWin téměř neliší, pouze u cyklů s daným počtem opakování se uvede pouze číslo *k* (nikoliv *k*–KRÁT)

a místo čísla lze použít také proměnnou. V podmíněných příkazech s dvěma bloky příkazů neukončujeme první blok slovy … KONEC, JINAK …, ale pouze slovem … JINAK … Zcela novým nástrojem pro větvení programu je příkaz ZVOLIT. Má následující strukturu:

ZVOLIT *výraz* PŘÍPAD *výraz, n. seznam výrazů* *blok příkazů* PŘÍPAD *výraz, n. seznam výrazů* *blok příkazů* JINAK *blok příkazů* KONEC	*Priorita operátorů:* *, /, % +, - =, <>, >, <, >=, <= JE, NENÍ A NEBO

Nové jsou také možnosti opustit okamžitě prováděnou proceduru a vrátit se na místo, odkud byla volána, příkazem NÁVRAT, nebo možnost okamžitě ukončit provádění všech procedur příkazem STOP. Vedle struktury příkazu zvolit jsme uvedli prioritu operátorů používaných při práci s proměnnými a s podmínkami. Tím jsou odlišnosti a rozšíření implementace KarelWin proti základní verzi programovacího jazyka robota Karla v podstatě vyčerpány. Zbývá už jen říci, že programy můžeme komentovat, a to buď po dvou limítkách //*komentář*, nebo mezi dvojznaky /* *komentář* */. Ukažme si nakonec, jak je lze využít k zobrazování víceciferných čísel na příkladu příkazu SEČTI(x,y), který zobrazí dvě zadaná čísla, sečte je a pak zobrazí jejich součet. Neuvádíme znovu výpisy příkazů ZPĚT a KE_ZDI.

```
PŘÍKAZ SEČTI(x,y)
    //nastavení pozice
    DOMŮ
    VLEVO
    KE_ZDI
    ZPĚT
    VPRAVO
    KE_ZDI
    //první sčítanec
    PIŠ(x)
    VPRAVO
    KROK
    VLEVO
    //druhý sčítanec
    PIŠ(y)
    VPRAVO
    KROK
    VLEVO
    //podržení
    OPAKUJ 10
        ZPĚT
        MALUJ '-'
    KONEC
    KE_ZDI
    VPRAVO
    KROK
    VLEVO
    //součet
    PIŠ(x+y)
KONEC
```

```
PŘÍKAZ PIŠ(n)
    KDYŽ n > 9
        VYPIŠ(n)
    JINAK
        ZPĚT
        MALUJ n+48
        KROK
    KONEC
KONEC
```

```
PŘÍKAZ VYPIŠ(c)
    KDYŽ c > 9
        ČÍSLO z = c % 10
        ČÍSLO p = (c-z)/10
        ZPĚT
        KDYŽ p>9
            VYPIŠ(p)
        JINAK
            ZPĚT
            VYPIŠ(p)
            KROK
        KONEC
        VYPIŠ(z)
        KROK
    JINAK
        MALUJ c+48
    KONEC
KONEC
```

1.8 Procvičení programování v implementaci KarelWin

Abyste si upevnili získané vědomosti, vypracujte následující cvičení:

a) Naprogramujte příkaz ČÍSLICE_NA_ZNAČKY, po jehož zadání Karel projde systematicky všechna pole a nalezené číslice zamění za příslušný počet značek.
b) Naprogramujte příkaz ATBAŠ, po jehož zadání Karel projde systematicky všechna pole a nalezená písmena velké či malé abecedy zamění podle převodové tabulky substituční šifry atbaš.
c) Naprogramujte příkaz ALBAM, po jehož zadání Karel projde systematicky všechna pole a nalezená písmena velké či malé abecedy zamění podle převodové tabulky substituční šifry albam.
d) Naprogramujte příkaz PODLE_PLOTU, který vhodným způsobem realizuje transpoziční šifru nazývanou „podle plotu“.
e) Naprogramujte příkaz ODEČTI(x,y), po jehož spuštění a následném vložení dvou přirozených čísel Karel zobrazí pod sebe čísla x a $-y$ a pak vypočte a zobrazí jejich rozdíl, včetně případného znaménka mínus.
f) Naprogramujte příkaz ROZKLAD(n), po jehož spuštění Karel načte přirozené číslo a zobrazí jeho rozklad na prvočinitele.
g) Naprogramujte příkaz NSD(x,y), po jehož spuštění a následném vložení dvou přirozených čísel Karel zobrazí pod sebe tato čísla a potom vypočte a zobrazí jejich největší společný dělitel.
h) Naprogramujte příkaz PRVOČÍSLA, po jehož zadání Karel pod sebe vypíše posloupnost prvních dvaceti prvočísel.
i) Naprogramujte příkaz FIBONACCI, po jehož zadání Karel pod sebe vypíše prvních dvacet členů Fibonacciovy posloupnosti.
j) Naprogramujte příkaz PASCAL_TROJ, po jehož zadání Karel zobrazí prvních devět řádků Pascalova trojúhelníka. Poslední zobrazený řádek tedy bude: 1 8 28 56 70 56 28 8 1 .
k) Naprogramujte příkaz MAGIC, po jehož zadání Karel zobrazí magický čtverec o velikosti 4 x 4. Vyplní jej tedy čísly 1 až 16.

l) Pro desky zaplněné jedno a dvojcifernými čísly – viz cvičení j) a k) – určete příkaz NAJDI(n), který systematicky prochází všechna pole a zastaví se při nalezení zadaného čísla *n*.
m) Naprogramujte příkaz PATNACTKA, který vhodným způsobem realizuje známý hlavolam Samuela Loyda s čísly 1 až 15 a jednou volnou pozicí uvnitř čtverce o velikosti 4 x 4. Volnou pozici Karel posune vždy podle právě stiknuté kurzorové klávesy.
n) Naprogramujte příkaz MINCE, který vhodným způsobem realizuje známý hlavolam Martina Gardnera s mincemi. Možné tahy nechť jsou vyznačeny písmeny a program reaguje na jejich stisk.
o) Naprogramujte příkaz LABYRINT, který způsobí, že Karel najde cestu ven z bludiště, přičemž si vhodným způsobem značí cestu pokládáním a zvedáním značek.
p) Naprogramujte příkaz JEZDEC, při jehož provádění se Karel pohybuje jako šachový jezdec, cílová políčka označuje (např. písmeny abecedy) a na dříve označená pole již nevstupuje, ale přeskakuje je.