2007

KAREL (verze 4.1)

NÁPOVĚDA K PROGRAMU

PETR LAŠTOVIČKA

Edit by Dvestezar www.dvestezar.cz 2015 v 2015.09.15

Obsah

Úvod

Program Karel je určen na výuku programování. Je vhodný hlavně k procvičení používání procedur a rekurzivních algoritmů. Karel je jméno robota, který umí vykonávat příkazy podle programů, které mu napíšete. V levé části je obrazovky okno vyhrazené pro psaní a úpravy vašich programů. V pravé části se nachází plocha, ve které je v levém dolním rohu umístěn robot Karel.

Při psaní lze používat obvyklé klávesové zkratky. Blok textu se označuje buď myší, nebo držením `SHIFT` spolu s klávesami pro přemístění kurzoru. Slovo se dá označit dvojím kliknutím myší.

Při otevírání nebo ukládání souboru není nutné psát ke jménu souboru příponu. Zdrojové texty programů mají příponu KRL. Pokud existuje na disku soubor KAREL.KRL, tak bude otevřen automaticky při spuštění. Ve Windows je dobré si vytvořit zástupce a umístit ho do nabídky Start. V zástupci správně nastavte pracovní adresář.

Edit by Dvestezar :

Tento návod jsem poupravil pro mého syna pro lepší čitelnost a orientaci a přidal sekci „První kroky“. Stejně jako mě kdysi, ho začalo bavit „nečemu“ přikazovat a radovat se z toho, že „to funguje“. Kdoví jaké budou jeho další kroky v životě, protože taťkům se popravdě to programování synů v období teen-rozkvětu moc nedaří ☺.

Základní příkazy a podmínky

- Příkazy jsou : `KROK, VLEVO, VPRAVO, DOMŮ, POLOŽ, ZVEDNI`
- Podmínky jsou : `ZEĎ, SEVER, JIH, VÝCHOD, ZÁPAD, ZNAČKA, MÍSTO, ZNAK`

Pohyb

Robot je zobrazen jako trojúhelníček, aby bylo poznat, do jakého směru je právě natočen. Na začátku je natočen směrem vpravo (na východ). Když dostane příkaz `KROK`, popojede o jedno políčko vpřed. Příkazem `VLEVO` se otočí o 90 stupňů doleva, příkazem `VPRAVO` se otočí o 90 stupňů doprava. Celá plocha je ohraničena zdí, do které robot nesmí narazit. Pokud se tak stane, pak robot přeruší vykonávání příkazů a na spodním řádku obrazovky se zobrazí chybové hlášení. V programu je možné podmínkou `ZEĎ` zjisti, jestli robot stojí před zdí. Robot také dovede zjistit, na kterou stranu je natočen. Směr natočení lze testovat podmínkami `SEVER, JIH, VÝCHOD, ZÁPAD`. Příkazem `DOMŮ` se Karel přesune do levého dolního rohu a natočí se směrem na východ.

Značky

Kromě změny polohy umí robot ještě pokládat značky (příkazem `POLOŽ`) a sbírat značky (příkazem `ZVEDNI`). Značky jsou na ploše zobrazeny číslem určujícím, kolik značek se na políčku nachází. Počet značek pod robotem se zobrazuje v levém dolním rohu plochy. Podmínkou `MÍSTO` se zjistí, jestli je ještě možné položit další značku. Naopak podmínkou `ZNAČKA` zjistíte, jestli pod robotem leží alespoň jedna značka. Pokud na políčku zrovna žádná není, nesmí robot dostat příkaz `ZVEDNI`, jinak přeruší provádění programu a zobrazí chybovou zprávu.

Používání znaků (písmenek) v kódu

Příkazem `MALUJ` lze na políčko napsat nějaký znak. V programu musejí být znaky napsány v apostrofech. Například `MALUJ '*'` nakreslí hvězdičku na pozici Karla. To samé udělá `MALUJ 42`, kde číslo udává kód znaku v ASCII[1] tabulce. Podmínkou `ZNAK='X'` lze zjistit, jestli je na políčku pod robotem znak **X**. Podmínkou `JE ZNAK` se testuje, jestli je na políčku nějaký (libovolný) znak.

Na jednom políčku nemohou být zároveň znak a značky. Proto se při položení značky smaže znak a naopak namalováním znaku se odstraní všechny značky.

Vysvětlení směrů

- SEVER = nahoru
- JIH = dolů
- VÝCHOD = vpravo
- ZÁPAD = vlevo

[1] Pro vice info o ASCII doporučuji v Googlu zadat : wiki ascii

Struktura programů

Každý program se skládá z procedur. Prvním řádkem procedury je slovo `PŘÍKAZ`, za kterým následuje název procedury. Název může obsahovat velká a malá písmena (včetně písmen s háčky nebo s čárkami), číslice a také znaky ? ! _ : ` @ # První znak názvu nesmí být číslice. Procedura je ukončena příkazem `KONEC`.

```
PŘÍKAZ název
   Příkazy
KONEC
```

Uvnitř procedury je posloupnost příkazů. To mohou být buď názvy jiných procedur, nebo to jsou základní povely pro ovládání robota, anebo to jsou speciální příkazy pro řízení běhu programu (příkazy cyklů nebo podmínek).

Bývá zvykem psát každý příkaz na nový řádek, protože je to přehlednější. Je však povoleno napsat na jeden řádek i více příkazů. V tom případě musejí být odděleny středníkem.

Do programu také můžete psát komentáře. Ty jsou dvojího druhu. Když napíšete //, tak všechno až do konce řádku se považuje za komentář. Druhý typ komentáře je text, který začíná /* a končí */. Takový komentář může být dlouhý i několik řádků.

```
// toto je jednořádkový komentář


/* toto je dvou
řádkový
nebo víceřádkový
komentář */
```

V programu nezáleží na pořadí jednotlivých procedur. Je proto možné napsat volání procedury v místě, které se nachází ještě před její definicí. Tzn. můžeme volat proceduru MOJE_PROCEDURA v proceduře ještě před samotným definováním procedury MOJE_PROCEDURA

Příklad

```
procedura první_procedura
   krok
   vlevo
   moje_procedura //toto je volání procedury ještě před její definicí
konec

/* teď teprve následuje definice moje_procedura
Nic ale samozřejmě nebrání v tom aby byla moje_procedura
definována před procedurou první_procedura*/
procedura moje_procedura
   krok
   krok
   vpravo
konec
```

Žádná procedura není hlavní. To znamená, že spustit můžete kteroukoli proceduru. Spouštění lze provádět dvěma způsoby.

1. Když zmáčknete klávesu F9, pak se spustí procedura, která je právě vidět na obrazovce a ve které bliká kurzor.
2. Druhý způsob je stisknout F12, čímž se zobrazí seznam všech procedur. Pak stačí zadat několik počátečních písmen nebo příkaz vyhledat použitím kurzorových kláves a potvrdit stiskem klávesy ENTER nebo

poklepáním myší a příkaz spustí. Spuštění můžete provádět vícekrát za sebou. Stiskem ESC se vrátíte zpět na zobrazení textu programu. Podobným způsobem lze také rychle najít proceduru v programu - klávesou F11.

Podmíněné příkazy

```
KDYŽ podmínka
  příkazy
JINAK
  příkazy
KONEC
```

Jestliže je podmínka splněna, pak se provedou příkazy následující za příkazem `KDYŽ`. Pokud podmínka není splněna, pak se vykonají příkazy, které následují po příkazu `JINAK`. Pokud mezi `JINAK` a KONEC nejsou žádné příkazy, tak lze příkaz `JINAK` vynechat.

```
ZVOLIT výraz
  PŘÍPAD výrazy
    příkazy
  PŘÍPAD výrazy
    příkazy
  JINAK
    příkazy
KONEC
```

Nejdříve se vyhodnotí číselný výraz u příkazu `ZVOLIT`. Ten se pak postupně porovnává s výrazy za příkazy `PŘÍPAD`. Když se nalezne shoda, pak se vykonají příkazy v části za odpovídajícím příkazem `PŘÍPAD`. Jestliže výraz za příkazem `ZVOLIT` se nerovná žádnému z výrazů za příkazy `PŘÍPAD`, pak se vykonají příkazy za slovem `JINAK`. Pokud mezi `JINAK` a `KONEC` nejsou žádné příkazy, tak lze příkaz `JINAK` vynechat. Za slovem `PŘÍPAD` může následovat více výrazů, které jsou odděleny čárkami. Čárka má na tomto místě podobný význam jako spojka NEBO.

Příklad

```
ZVOLIT a
  PŘÍPAD 5  // tzn. když a=5
    KROK
  PŘÍPAD 3  // tzn. když a=3
    VLEVO
  JINAK
    VPRAVO
KONEC

// nebo když není potřeba použít JINAK
ZVOLIT a
  PŘÍPAD 5  // tzn. když a=5
    KROK
  PŘÍPAD 3  // tzn. když a=3
    VLEVO
KONEC
```

Příkazy cyklů

Celkem existuje 6 druhů cyklů: s podmínkou na začátku, s podmínkou na konci, s podmínkou na začátku i na konci, nekonečný cyklus, cyklus s počtem opakování, cyklus s počtem opakování a ještě podmínkou na konci.

Na začátku cyklu je jeden ze tří příkazů:

1. `DOKUD` podmínka - Cyklus se vykonává tak dlouho, dokud je podmínka splněna. Když podmínka přestane být splněna, pak cyklus skončí a program pokračuje prováděním příkazů za koncem cyklu.
2. `OPAKUJ` výraz - číselný výraz udává, kolikrát se má cyklus provádět.
3. `OPAKUJ`

Na konci cyklu je jeden ze dvou příkazů:

1. `AŽ` podmínka - Když je podmínka splněna, pak cyklus skončí.
2. `KONEC`

Příklady

```
//Karel udělá 5 kroků
a=0
dokud a=5
   krok
   a=a+1
konec

//Karel udělá 10 kroků
opakuj 10
   krok
konec

//Karel se bude otáčet dokud nebude otočen na sever
opakuj
   vlevo
až sever
```

Ostatní příkazy

- `NÁVRAT` - ukončí prováděnou proceduru a pokračuje v proceduře, která ji zavolala
- `STOP` - zastaví program.
- `ČEKEJ` číslo - čeká daný počet milisekund.

Funkce

- `ZNAK` - vrací znak, který se nachází pod Karlem. Když tam není žádný znak nebo jsou tam značky, pak vrátí nulu
- `NÁHODA`(n) - vrací náhodné číslo z intervalu 0 až n-1
- `KLÁVESA` - přečte znak z klávesnice. Pokud žádná klávesa nebyla stisknuta, pak vrátí nulu. Pokud potřebujete čekat, až bude stisknuta klávesa, použijte příkaz `ČEKEJ -1`. Funkční klávesy mají kódy záporné. Například kurzor vlevo má kód -75, kurzor vpravo má kód -77, klávesa ENTER má kód 13
- **Je možné si naprogramovat další funkce**. Funkce mají úplně stejnou stavbu jako procedury, pouze místo klíčového slova `PŘÍKAZ` je slovo `FUNKCE`. Navíc musí každá funkce vracet nějakou hodnotu. To uděláte tak, že napíšete název funkce, rovnítko a pak číselný výraz, který má funkce vrátit.

```
FUNKCE muj_nazev(a)
  KDYŽ a=5
    muj_nazev=10
  JINAK
    muj_nazev=100
  KONEC
KONEC
```

Podmínky

- `SEVER`, `JIH`, `VÝCHOD`, `ZÁPAD` - tyto podmínky zjišťují, do jakého směru je právě robot Karel natočen.
- `ZEĎ` - tato podmínka je splněna, když je před Karlem zeď.
- `ZNAČKA` - tato podmínka je splněna, když se pod Karlem nachází alespoň jedna značka
- `MÍSTO` - tato podmínka je splněna, když lze položit značku
- `PRAVDA` - tato podmínka je vždy splněna
- `NEPRAVDA` - tato podmínka není nikdy splněna

Podmínky lze spojovat logickými spojkami `A`, `NEBO`. Zápornou podmínku vytvoříte slovem `NENÍ`.

Lze také používat aritmetické operátory =, <>, >, <, >=, <=.

Můžete si naprogramovat svoje podmínky. Podmínky se od funkcí odlišují pouze tím, že místo slova `FUNKCE` mají slovo `PODMÍNKA`. Kromě toho nevracejí číselný výraz, ale logický výraz.

Priority operátorů

```
*, /, %
+, -
=, <>, >, <, >=, <=
JE, NENÍ
A
NEBO
```

Pořadí provádění operací lze změnit pomocí závorek. Kromě operátorů, čísel a proměnných se mohou uvnitř výrazů vyskytovat také funkce a podmínky.

Když za slovem `JE` následuje podmínka, pak se slovo `JE` ignoruje. Když za slovem `JE` napíšete číselný výraz, pak vznikne podmínka, která je splněna právě tehdy, když je hodnota výrazu různá od nuly.

Operace % počítá zbytek po dělení.

Proměnné

Při provádění programu je často potřeba zapamatovat si nějaké číslo a použít ho později. K tomu slouží proměnné. Proměnné se vytvářejí příkazem `ČÍSLO`, za kterým následují názvy proměnných oddělené čárkami. Když chcete uložit číslo do proměnné, pak napište název proměnné, rovnítko a číselný výraz. Když chcete přečíst hodnotu proměnné, pak stačí napsat název proměnné kdekoli v číselném výrazu.

Příkaz `ČÍSLO` lze použít kdekoli uvnitř procedury. Navíc můžete hned v příkaze `ČÍSLO` do proměnné přiřadit hodnotu. Stačí za proměnnou napsat rovnítko a číselný výraz. Když to neuděláte, pak bude při vytvoření proměnná obsahovat nějaké nesmysly. Při skončení procedury zanikají všechny její proměnné. Není možné vytvářet globální proměnné.

Procedury s parametry

Procedury, funkce nebo podmínky mohou mít parametry. Když voláte takovou proceduru, pak musíte zadat hodnoty, které se předají proceduře. V definici procedury jsou názvy parametrů uvedeny v závorkách za názvem procedury. Dál už se uvnitř procedury pracuje s parametry stejně jako s proměnnými.

Když před názvem parametru je znak &, pak se parametr nepředává hodnotou, ale odkazem. To znamená, že při volání procedury se místo číselného výrazu zadává proměnná. Když uvnitř procedury změníte hodnotu parametru, pak se změní hodnota proměnné, která byla předána při volání procedury.

Příklad procedury s parametrem

```
PROCEDURA muj_nazev_procedury(a)
  příkaz
  KDYŽ a=10
    příkaz
  KONEC
KONEC
```

Pole

Když si potřebujete uložit do paměti větší množství čísel, pak použijete pole. Pole se vytváří podobně jako obyčejná proměnná, navíc se za název pole do hranatých závorek napíše počet prvků v poli. Tento počet nemusí být konstanta. Například je povoleno, aby velikost pole závisela na parametru procedury. K jednotlivým prvkům pole se přistupuje tak, že napíšete název pole a za ním v hranatých závorkách pořadí prvku v poli. První prvek pole je na nultém místě. Počet prvků v poli se dá zjistit funkcí `DÉLKA`.

Je možné vytvářet vícerozměrná pole. Například pole X[4][3] bude obsahovat 12 prvků, které budou uspořádány do 4 řádků a 3 sloupců. Funkce délka(X) vrátí hodnotu 4 a funkce délka(X[0]) vrátí hodnotu 3. Příkazem X[0][2]=5 přiřadíte hodnotu 5 do prvku v prvním řádku a ve třetím sloupci v poli X.

Plocha

Úpravy plochy

Na začátku jsou zdi postaveny pouze po obvodu. Je však možné vybudovat další zdi kdekoli uvnitř plochy a tím robotovi značně ztížit situaci. Někdy také chceme předem připravit značky, které by měl za úkol robot posbírat nebo podle nich měnil směr pohybu. K tomu slouží v menu *Plocha* volba *Upravit*. Na levé straně obrazovky se zobrazí všechny činnosti, pomocí nichž můžete plochu upravovat. Nejrychlejší způsob je používat myš. Pokud pracujete s klávesnicí, pak stiskem Ctrl+šipky přesunete kurzor na nejbližší zeď v daném směru. Do rohu se nejrychleji přemístíte pomocí Ctrl+HOME, Ctrl+END, Ctrl+PGUP, Ctrl+PGDW. Kromě značek a zdí lze na políčko vložit nějaký znak. Stačí stisknout odpovídající klávesu.

Když úpravu plochy ukončíte stiskem `ENTER`, pak se právě vytvořená plocha uchová do paměti. Kdykoli pak dáte *Obnovit*, překopíruje se z paměti zpět na plochu. Toho se dá využít při ladění vašich programů. Když spustíte nějakou proceduru, která přemístí robota jinam a ještě k tomu vysbírá nějaké značky, pak jednoduše stiskem F5 můžete vše vrátit do původního stavu. Když máte otevřeno více ploch, tak si svůj stav pamatuje každá z nich. Možnost obnovení je také dobrá pro případ, že v menu omylem zvolíte povel *Vymazat* nebo *Generovat*.

Práce s více plochami

Je možné mít najednou otevřeno více ploch. Každá má přitom svého robota a pamatuje si jeho polohu. Příkazy provádí vždy ten robot, který se nachází v právě aktivní ploše. Aktivní plocha je ta, která je vidět na obrazovce. Také všechny povely z menu *Plocha* (kromě *Nová* a *Otevřít*) se vztahují na právě aktivní plochu. V menu *Plocha* dole pod čarou najdete seznam všech otevřených ploch a zvolením některé z nich se daná plocha stane aktivní. Další plochu přidáte volbou *Nová*. Tím se vytvoří prázdná plocha. Když pak stisknete F5, překopíruje se do ní předchozí aktivní plocha. Naopak volbou *Zavřít* se plocha odstraní ze seznamu a také se smaže v paměti. Volbou *Vymazat* se vymažou všechny zdi a značky, ale plocha nadále zůstane v paměti.

Plochy můžete ukládat do souborů na disk. U jména souboru nezadávejte příponu, použije se přípona KPL. Při vytvoření nové plochy se jí přidělí název končící číslem, které se pro každou novou plochu zvětšuje o jedničku. Jestliže chcete jméno změnit, zvolte *Přejmenovat*. Tím se změní jméno v paměti, aniž by se něco měnilo na disku. Volbou *Uložit jako* se změní jméno a soubor se zároveň uloží na disk.

Ladění programů

Když program spustíte pomalu (klávesou F6), pak můžete měnit rychlost klávesami + a - nebo klávesami 0 až 9 na numerické klávesnici. Rychlost se zobrazuje na spodním řádku obrazovky a udává, kolik milisekund trvá vykonání jednoho kroku. Pokud nastavíte zpoždění mezi příkazy delší než 300ms, tak se zároveň ukazuje část programu, ve které právě běží program. Program lze kdykoli přerušit stiskem `ESC` nebo `BACKSPACE`.

Program je možné krokovat po jednotlivých řádcích. Je možné vstupovat do procedur (F7) nebo procedury provádět jako naráz jeden krok (F8). Dále přibyla možnost krokování po jednotlivých instrukcích (SHIFT+F7). To použijete v případech, když máte na jednom řádku více příkazů. Poslední možnost (SHIFT+F8) je nejrychlejší způsob krokování - program se zastavuje jen na příkazech `KROK`, `VLEVO` nebo `VPRAVO`. Během krokování je umožněno prohlížet zdrojový text. Jestliže v něm ale provedete jakoukoli změnu, bude program přerušen. Při prohlížení textu je barevně odlišen řádek, na kterém se zastavil program. Pokud se vám tento řádek ztratí z obrazovky, lze se na něj přemístit povelem Právě prováděný příkaz (v menu *Hledat*). Povelem *Ukončit běh* (v menu *Spustit*) se úplně přeruší provádění programu a další stisk F9 spustí proceduru zase od začátku. Pokud program skončí chybovým hlášením, lze chybu ignorovat a normálně pokračovat v provádění programu.

Dalším prostředkem pro ladění jsou *breakpointy*. Ty se vkládají nebo mažou kombinací CTRL+F8. Jestliže se při provádění programů narazí na breakpoint, pak se program zastaví. Pak můžete program krokovat nebo pokračovat v běhu. Také můžete přidat další breakpointy nebo původní breakpoint odstranit. Pokud vložíte breakpoint na řádek, který neobsahuje žádnou instrukci (např. na `KONEC` ukončující příkaz `KDYŽ`), pak se breakpoint umístí až na následující řádek. Volba *Jít ke kurzoru* funguje tak, že se na řádek s kurzorem umístí dočasný breakpoint a program se spustí jako při stisku F9. Pak by měl skončit na řádku s kurzorem. Může se ale zastavit ještě dříve na nějakém jiném breakpointu nebo se může stát, že se na řádek z kurzorem vůbec nedostane a skončí normálně po vykonání všech příkazů. Další podobnou volbou je povel *Dokončit proceduru*. To je dobré použít tehdy, když krokujete program a chcete, aby se rychle vykonaly všechny příkazy až do konce procedury včetně příkazu `KONEC`.

Seznam všech klíčových slov

Lze používat česká i anglická klíčová slova. Je povoleno střídat češtinu a angličtinu dokonce i uvnitř jednoho programu.

```
příkaz, funkce, podmínka,
když, dokud, opakuj, jinak, až, konec,
návrat, číslo, zvolit, případ,
není, je, a, nebo,
krok, vlevo, vpravo, zvedni, polož, maluj,
čekej, stop, domů, rychle, pomalu,
zeď, značka, místo, sever, západ, jih, východ,
znak, klávesa, náhoda, délka,
pravda, nepravda
```

```
procedure, function, condition,
if, while, repeat, else, until, end,
return, integer, switch, case,
not, is, and, or,
step, left, right, pick, put, paint,
delay, stop, home, fast, slow,
wall, mark, space, north, west, south, east,
char, key, random, length,
true, false
```

Možnosti nastavení editoru

V menu *Úpravy* zvolením *Možnosti* zobrazíte dialogové okno, kde lze měnit vlastnosti editoru. Pokud chcete změny uložit do souboru KAREL.INI, aby se projevily také při příštích spuštěních programu, zvolte v menu *Úpravy* položku *Uložit nastavení*.

- Trvalé bloky - blok zůstává označen tak dlouho, dokud nestisknete Ctrl+K+H, nebo neoznačíte jiný blok, nebo blok nesmažete
- Přepisovat bloky - blok se smaže, když začnete něco psát, nebo stisknete DELETE. Tato volba se ignoruje v případě, že jsou zapnuté trvalé bloky.
- Odsazování po ENTER - při ukončení řádku stiskem ENTER se kurzor posune pod první znak předchozího řádku
- Odsazování při Backspace - na začátku řádku nebo na prázdném řádku se smaže tolik mezer, aby se odsazení zmenšilo o jedna
- Hledat slovo pod kurzorem - při hledání se do kolonky Najít napíše slovo, kde je právě kurzor
- Vložení řádku posune kurzor - když stisknete ENTER na začátku řádku, pak se umístí kurzor o řádek výš do nově vytvořeného řádku
- Zpět je po skupinách - zpět se provádí po celých slovech místo po jednotlivých písmenech.
- Po uložení lze dát zpět - když tuto volbu vypnete, pak lze provádět zpět jen do toho místa, kdy byl soubor naposledy uložen. Tím získáte více volné paměti.
- Klíčová slova velkými písmeny - když tuto volbu zapnete, pak se při příštím překladu změní v celém souboru klíčová slova z malých písmen na velká

První kroky

Úvod prvních kroků

Možná se někdy zapomenu, ale pokud se někde zmíním o proceduře, tak tím myslím příkaz.

Při psaní je vhodné si vytvořit návyky, které nám do budoucna hodně pomohou, jako například

- komentovat v úvodu příkazu nebo funkce, co dělají a co popř.vracejí za výsledek.
- komentovat si jednotlivé bloky programu, stejně jako příkazy či funkce na začátku
- názvy příkazů a funkcí vytvářet bez diakritiky a využívat podtržítek jako mezer, pro zpřehlednění názvu, také lze používat kombinaci malých a velkých písmen, např. `moje_prvni_procedura`, nebo `MojePrvniProcedura`, je rozhodně čitelnější než `mojeprvniprocedura` ;) a proč podržítko? Protože to zatím nevíš, ale ostatní znaky se používají v programování jako hodně důležité funkce, např :
  - znak tečky (stejný znak jako na konci řádku) se používá jako oddělovač objektů – pokročilé programování v pokročilých programovacích jazycích
  - Znak složené závorky {} – se také hojně používají při objektovém programování
  - Zavináč @ - zapeklitý symbol, který nebudu vysvětlovat, ale hraje velmi důležitou roli
  - Znak | - v některých jazycích je to výraz pro NEBO
  - $ - vůbec nepoužívat, je to hojně používaný symbol, pro označování proměnných, někde všech, někde jen řetězcových (text) a někde je to vyjádření určitého čísla, nejedná se o dolar jako měnu ☺ ale pouze o symbol.
- Takže ještě jednou názvy příkazů a funkcí, : nejvhodnější je používat písmenka A až Z, stejně tak i malá písmenka a podtržítko, jako zástupný znak za mezeru

První procedura

I když je dnes ve zvyku všude používat jako příklad Hello World (možná jste se již setkali s tímto novodobým trendem) tak my tímto směrem nepůjdeme.

Jako první proceduru by jsme mohli nadefinovat jednoduchý pohyb Karla a to čelem vzad. Na tomto příkladu si ukážeme jaké jsou různé možnosti programování a jak lze někdy stejného výsledku dosáhnout různými cestami.

Takže první způsob :

```
příkaz celem_vzad
  vlevo
  vlevo
konec
```

Druhým způsobem je obrácený směr ale stejný výsledek :

```
příkaz celem_vzad2
  vpravo
  vpravo
konec
```

Tímto je ukázáno, že v některých případech lze dosáhnout naprosto stejných výsledků. Jasný výběr z těchto dvou procedur by nastal, kdyby se Karel neuměl otáčet doleva, nebo to měl prostě zakázáno[0]. V následující kapitole využijeme stejný pohyb ale s využitím cyklů.

Procedura a cykly

Stejně jako v předchozím příkladu budeme chtít po Karlovi aby se otočil směrem vzad, ale v tomto případě využijeme cyklů jako např. :

Samozřejmě že ohledně optimalizace (viz dále) jsou v případě čelem vzad cykly méně výhodné než napsat 2x příkaz vpravo – zabírají víc místa a Karel musí poslouchat/číst (viz.dále) více příkazů. Cykly se používají

v případech, kdy máme spoustu příkazů a chceme je několikrát opakovat, této spoustě příkazů říkáme **blok programu,** stejně tak je blok programu příkaz, funkce nebo cyklus.

```
příkaz cyklus1_celem_vzad
  opakuj 2
    vpravo // nebo vlevo, dle chuti
  konec
konec

příkaz cyklus2_celem_vzad
číslo ax=0
  dokud ax<2
    vpravo // nebo vlevo, dle chuti
    ax=ax+1
  konec
konec

příkaz cyklus3 celem vzad
číslo ax=0
  opakuj
    vpravo // nebo vlevo, dle chuti
    ax=ax+1
  az ax=2
konec
```

Optimalizace

Proč používat cykly a ne posloupnost příkazů jako na začátku, když v druhém příkladu byly nevýhodné? Protože ve složitějších operacích, kde se opakuje více příkazů (které vytvářejí blok programu), je kvůli úspoře místa a rychlejšímu čtení vhodné použít cyklu., příkazu nebo funkce Např.

```
příkaz start
  //nastavíme výchozí bod
  domů
  vlevo

  // a píšeme 2x tah koně což jsou 2 bloky programu
   // 1.blok
   krok
   krok
   vpravo
   krok

   // 2.blok
   krok
   krok
   vpravo
   krok
konec
```

Pro ušetření místa a počtu příkazů[1] - „Optimalizaci“ - (protože myslíme dopředu a protože se příkaz může hodit ještě vícekrát …) vytvoříme proceduru tah_kone, nakonec, máme tu dva naprosto shodné bloky, tak proč je opisovat.

```
příkaz start
  //nastavíme výchozí bod
  domů
  vlevo

   // a máme tu možnost využít cyklus což je blok programu
   opakuj 2
     tah_kone
   konec
   /*tady se ušetřilo 5 řádků i když následná definice to dohoní, ale pokud
by jsme opakovali již 3x nebo vícekrát, tak by se jednalo o značné ušetření
místa a v případě cache i času, tomuto se říká optimalizace*/
konec

příkaz tah_kone
   krok
   krok
   vpravo
   krok
konec
```

Nastanou ale případy kdy je cyklus nevhodný. Předchozí příklad donutí Karla chodit ve tvaru čtverce, ale co když chceme, aby pohybem šachového koně šel neustále nahoru. V tomto případě cyklus postrádá smysl a musíme před druhým krokem upravit jeho směr.

```
příkaz start
  //nastavíme výchozí bod
  domů
  vlevo

  // a píšeme
  tah_kone
  vlevo   /* nasměrujeme tak aby šel nahoru,
    co se stane při vynechání tohoto příkazu ? */
  tah_kone
konec

příkaz tah_kone
   krok
   krok
   vpravo
   krok
konec
```

První oprávněné využití cyklu

Procedura a cyklus jdi dokud není zeď …

```
příkaz jdi_dokud_neni_zed
  dokud není zeď
    krok
  konec
konec
```

Jak by vypadala procedura bez cyklu? Ani by vytvořit nešla ….Karel by udělal jen jeden krok.

```
příkaz jdi_dokud_zed
  když není zeď
     krok
  konec
konec
```

A když Karel udělá jen jeden krok, kolikrát by jsme museli opsat blok

```
  když není zeď
     krok
  konec
```

aby došel Karel až ke zdi? 20x ? Co když Karel nebude stát ve výchozím bodě (domečku), ale uprostřed mapy? Polovina vypsaných bloků by se stala přebytečnou, to by bylo plýtváním jak místa, tak zbytečného čtení podmínek, které Karel nakonec nesplní.

První funkce

Nastanou případy kdy je potřeba vytvořit funkci. Funkce je něco jako příkaz, ale chceme po tomto příkazu, aby nám vrátil nějakou hodnotu, třeba číselnou. Takže takovou první funkci s využitím dosavadních znalostí bude : spočítej značky na místě kde Karel stojí. Jak může Karel spočítat značky? Jednoduše, prostě je jednu po druhé zvedne, dokud tam žádná nezůstane ;) .

```
funkce kolik_znacek
číslo cn=0 /* nadefinujeme si proměnnou
            ve které si Karel bude pamatovat počet značek */
   dokud znacka // dokud leží na Kalově pozici značka tak proveď
     zvedni // zvedni značku
     cn=cn+1 // a připočti jenda (zvedni prst na ruce ;) )
   konec
   kolik_znacek=cn /* přiřadíme zjištěný počet názvu funkci, která v této
                   chvíli přebírá funkci proměnné */
konec
```

Problém nastane tehdy, když chceme aby Karel jen spočítal značky, to znamená aby tam i zůstaly, protože momentálně Karel tímto způsobem značky posbírá a nic na jeho pozici nezůstane. A proto myslíme dopředu a už i optimalizujeme a řekneme si „Karel je musí zase položit“. A protože bude Karel nejspíš pokládat značky často, nebo spíš to bude asi oblíbená činnost kterou mu budeme často přikazovat, tak vytvoříme proceduru polož_značky. No a protože to nebude pokaždé stejný počet, tak si vytvoříme příkaz s proměnnou.

Příkaz s proměnnou

Jak jsem napsal, tak uděláme

```
příkaz poloz_znacky(kolik)
   dokud kolik>0
     polož
     kolik=kolik-1
   konec
konec
```

Tento příkaz položí na pozici Karla tolik značek, kolik mu zadáme jako číslo místo proměnné kolik, samozřejmě je Karel limitovaný omezeným počtem značek na pozici.

No a teď náš příkaz skloubíme s předchozí funkcí, tzn. ve funkci přidáme jeden řádek s námi vytvořeným příkazem.

```
funkce kolik_znacek
číslo cn=0
  dokud značka
    zvedni
    cn=cn+1
  konec
  kolik_znacek=cn

  poloz_znacky(cn) /* přidáme příkaz pro položení tolika značek
                  kolik jich našel ;) */
konec

příkaz poloz_znacky(kolik)
  dokud kolik>0
    polož
    kolik=kolik-1
  konec
konec
```

Po spuštění funkce posbírá a spočítá značky, potom číslo „kolik napočítal“ předá příkazu poloz_znacky a tím vlastně dojde k tomu, že se na jeho místě nezmění počet značek po vykonání funkce a ještě nám řekne kolik jich tam bylo.

[0] V prvních verzích Karla se Karel uměl otáčet jen jedním směrem a pro opačný směr se musela vytvořit procedura, tak naschvál : Pokud by Karel znal jen příkaz VPRAVO, jak by asi vypadala definice příkazu VLEVO ;)

[1] Ve skutečnosti každé volání procedury a rozhodnutí podmínky je také příkaz, což lze zjistit, pokud si dáme při ladění krokovat program příkaz po příkazu s vnořením do příkazů. Takže ve skutečnosti jde hlavně o úsporu místa. Pokud se ale jedná o často využívanou proceduru – blok programu, může jí procesor (mozek Karla) tzv. nakešovat (CPU cache – viz wiki) a provádění programu se o to urychlí (někdy rapidně), a to je v našem případě to „rychlejší čtení“.