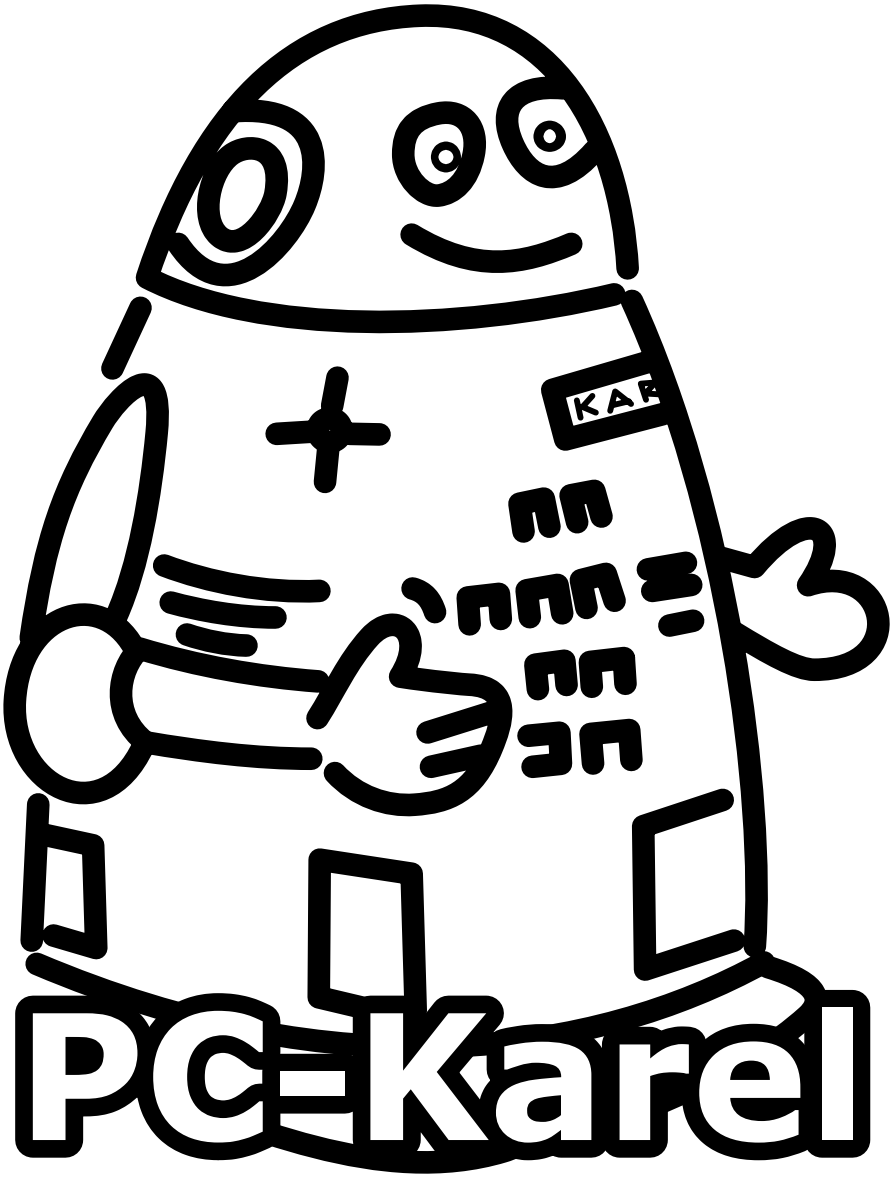

KAR
PC=Karel

PC-Karel (2001/07/12). Internet: www.pckarel.zde.cz (pckarel@zde.cz)

Za pomoc, nápady a myšlenky děkuji zvláště Davidu Breberovi a Petru Pelantovi.
Sazba systémem TEX.

PC–Karel

Vážený čtenáři,

dostává se Vám do tuky text, který si klade dva cíle: seznámit Vás se základy jazyka Karel a popsat jeho implementaci pro osobní počítače IBM-PC s názvem `PC-Karel`.

První část je obecná a měla by, s různými malými úpravami, platit na všech implementacích. Konkrétními vlastnostmi programu `PC-Karel` se zabývá druhá část.

Při studiu doporučuji vše rovnou zkoušet u klávesnice počítače – je to nejlepší metoda. Přináší ale jednu nevýhodu: Při postupném studiu první části bude třeba občas nahlížet do části druhé. To proto, že Karel má 2 části: svoji „duši“ – samotný jazyk se svými myšlenkami a konstrukcemi, a svoje „tělo“ – vlastní implementaci pro počítač. Protože jsem se v textu snažil oddělit „duši“ od „těla“, ale protože jedno bez druhého nemůže existovat, je třeba onoho listování. Ti, kteří již jazyk znají, mohou prostudovat pouze druhou část týkající se programu `PC-Karel`.

Tento program je ABSOLUTNĚ BEZ ZÁRUKY; jde o volné programové vybavení jehož šíření je za jistých podmínek vítáno. Podrobnosti najdete v části Licenční ujednání. Budu Vám vděčný za jakékoliv připomínky ať už k programu, nebo k textu samotnému.

Jiří Osoba.

1. KAREL – popis jazyka

1.1. Trocha historie nikoho nezabije

Jazyk Karel vytvořil profesor Richard E. Pettis ze Stanfordské univerzity v USA. Popsal jej ve své knize „KAREL – The Robot“ a nazval jej skutečně „KAREL“ – na počest českého spisovatele Karla Čapka – autora divadelní hry „R.U.R.“, ve které Čapek jako první člověk na světě použil slova *robot* (které ale vymyslel jeho bratr Josef).

V bývalém Československu se Karel poprvé objevil zásluhou doc. Hvoreckého v Bratislavě. V druhé polovině 80. let se Karel rozšířil na všechny tehdy používané mikropočítače (`IQ151`, `PMD85`, `Sinclair`, ...) a konaly se soutěže dětí a mládeže v jeho programování. Karel byl původně navržen jako „předvoj“ jazyka Pascal – má stejné programátorské konstrukce a stejné principy – stejně, jako Pascal podporuje strukturované programování a platí zde zásada rozkladu problému na jednodušší části. Není proto pro děti i dospělé problémem přejít od Karla např. k Logu, Pascalu, C a jiným programovacím jazykům.

Karel se obvykle vyučuje u dětí 10–13 let starých – základy jsou i pro mladší, ale nejkrásnější část – *rekurze* – je náročná i pro některé dospělé.

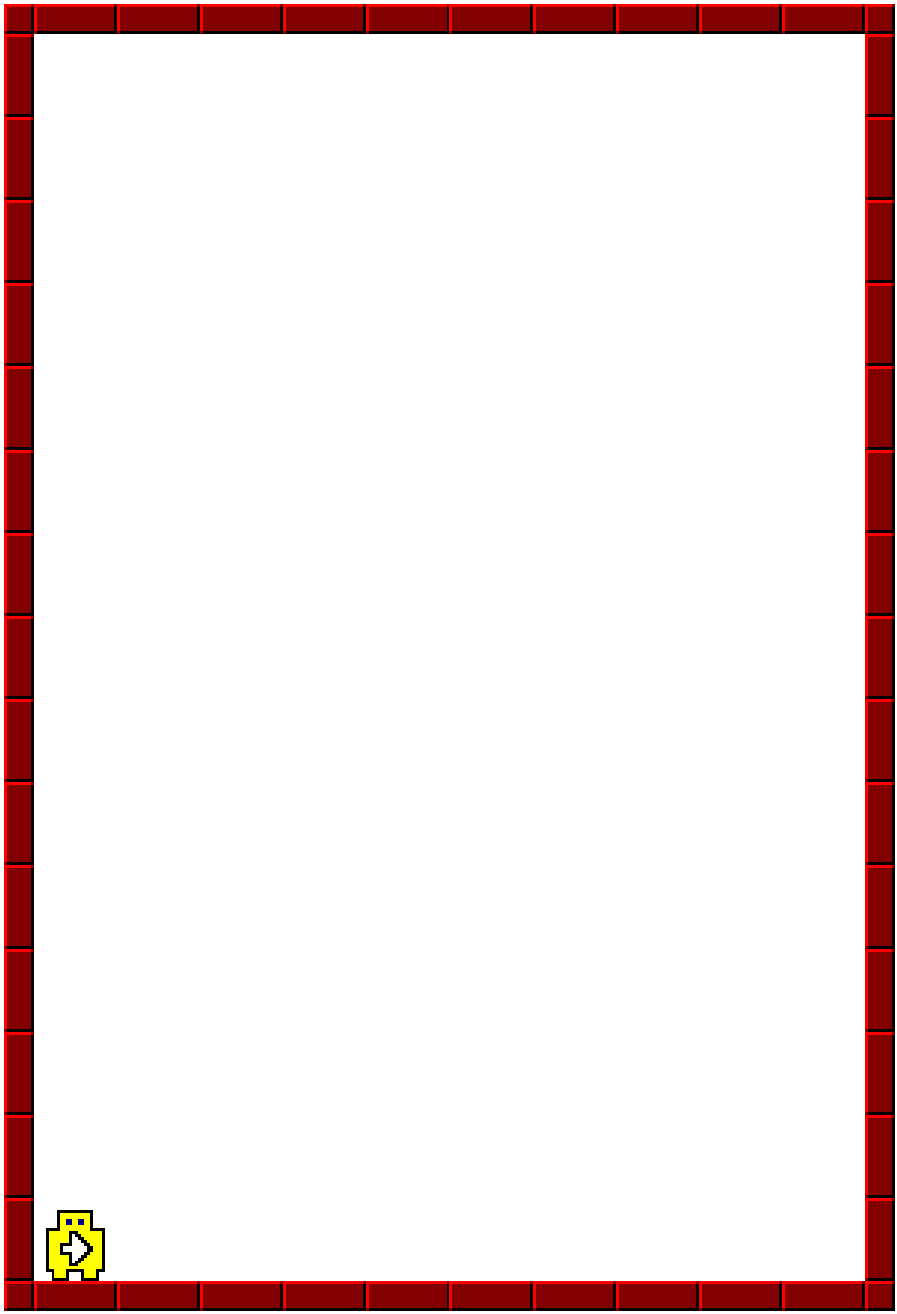
Ještě malou poznámku k používané CESTINE: Ta vznikla v době, kdy počítače ještě neuměly ČESKY a od té doby se v Karlovi zachovává.

1.2. Základní kameny Karla

Karel je malý, ale šikovný, robotek, který žije ve svém *městě* – čtvercovém nebo obdélníkovém *dvorečku*, po kterém se může pohybovat. Může být natočen do jedné ze 4 světových stran (*SEVER, JIH, VYCHOD, ZAPAD*) a na zádech nosí kouzelný baťůžek, ze kterého může vytahovat *značky* a klást je na místo, na kterém právě stojí, a opět je sbírat. Baťůžek se podobá bezedné slánce – je v něm dostatečná zásoba značek pro celé město.

Základní a nejdůležitější Karlovou vlastností je schopnost učit se nové příkazy. Tyto *složené příkazy* se Karel učí jako sled příkazů již známých. A aby bylo pro začátek z čeho sestavovat, zná Karel několik základních příkazů – *primitiv*:

- `SLOVNIK` – Karel vypíše všechny známé příkazy – primitiva i složené příkazy.
- `MESTO` – spouští jakýsi editor města, kterým můžeme do města postavit zdi, položit značky apod.
- `ROZKLAD` – Karel nám ukáže, z čeho a jak je složen příkaz.
- `CHYBA` – Karel vymaže ze svého slovníku poslední příkaz. Pokud Karla zrovna učíme příkazu novému, pak vymaže posledně zadanou sekvenci.
- `KDYZ`, `DOKUD`, `OPAKUJ`, `KONEC` – strukturované příkazy. Jejich funkci si vysvětlíme později.
- `KROK` – první ze 4 tzv. *výkonných primitiv* – Karel udělá jeden krok ve směru, do kterého je natočen. Pokud je před ním zeď, pak Karel do této zdi narazí, ohlásí to a ukončí vykonávání všech příkazů.
- `VLEVO-VBOK` – Karel se otočí o 90° proti směru hodinových ručiček.
- `POLOZ` – Karel vyndá ze svého kouzelného baťůžku značku a položí ji na políčko, na kterém stojí, ovšem pouze tehdy, pokud tam je ještě místo. Na každé políčko se vejde jenom několik značek (5, 9, ... – podle implementace). Víc jich tam dát nejde – komín značek by spadl – Karel to ohlásí a skončí.



Obr. 1: Karel ve svém městě

- `ZVEDNI` – Karel zvedne značku z políčka, na kterém stojí, a uloží ji do svého baťůžku. Pokud tam žádná značka není, Karel se trochu nazlobí, protože přece „Kde nic není, ani smrt nebere“ a skončí (jako když narazí do zdi . . .).

To jsou tedy základní příkazy, které Karel zná od začátku. Některé z nich si můžete rovnou vyzkoušet (`SLOVNIK`, `MESTO`, `KROK`, `POLOZ`, `ZVEDNI`) i s tím, jak se bude Karel chovat, pokud ho budete nutit dělat věci, které nemůže (`KROK` do zdi apod.).

Příkazy `SLOVNIK`, `MESTO`, `ROZKLAD` a `CHYBA` nejdou použít při sestavování nových příkazů.

1.3. Složené příkazy

Jistě sami uznáte, že by to nebylo nic moc, kdybychom mohli s Karlem jen ručně chodit po dvorečku. To je pravda. Ale jak jsem se již zmínil dříve, je Karel ochoten se učit novým příkazům. A to není málo.

Jak na to? Jednoduše.

Dejme tomu, že chceme Karla naučit novému obratu, `CELEM-VZAD`. Co vlastně musí Karel udělat? Po příkazu `CELEM-VZAD` musí zůstat otočen na druhou stranu. Což znamená, že musí udělat 2× `VLEVO-VBOK`.

Výborně, problém s `CELEM-VZAD` máme teoreticky vyřešen. Nyní se pokusíme tomu naučit Karla.

Pokud zadáme Karlovi příkaz, který ještě nezná, protože ho nemá ve svém slovníku, pak si myslí, že ho chceme něčemu novému naučit. Proto, pokud zadáme

```
CELEM-VZAD↵,
```

se zaraduje a odpoví nám:

```
NOVY PRIKAZ CELEM-VZAD ZNAMENA
```

a čeká, že mu budeme zadávat již ZNÁMÉ příkazy. Proto mu budeme odpovídat následovně:

```
VLEVO-VBOK↵
VLEVO-VBOK↵
KONEC↵
```

Nyní se můžeme přesvědčit, že to opravdu zabralo. Po příkazu `SLOVNIK` vidíme, že Karel zná i náš nový příkaz `CELEM-VZAD`. Po příkazu `ROZKLAD` a `CELEM-VZAD` vidíme, že:

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

Náš nový příkaz můžeme (a měli bychom) hned vyzkoušet. Zadáme

```
CELEM-VZAD↵
```

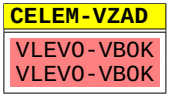
a – *Ó, přeslavný to den* – Karel provede dva `VLEVO-VBOK`y a zůstane stát čelem na druhou stranu.

Na tomto místě je myslím vhodné upozornit na jednu Karlovu vlastnost: obvykle není možné opravovat příkazy. Proto doporučuji nejprve pečlivě pracovat na papíře a poté teprve přepisovat do počítače.

Další záludností je to, že je možno (pomocí příkazu `CHYBA`) ze slovníku mazat příkazy od konce – proto pečlivě každý nový příkaz vyzkoušejte, než začnete definovat další. Vyhnete se tím mazání a opětnému definování více příkazů v okamžiku, kdy zjistíte, že třetí příkaz od konce slovníku je chybný.

1.4. Kopenogramy

Pro zápis složených příkazů můžeme používat přehlednější, grafickou formu, tzv. KOPENOGRAM. Název kopenogramu je odvozen od jmen jeho autorů: Kofránek–Pecinovský–Novák. Jak takový kopenogram vypadá? Ukažme si to na příkladu `CELEM-VZAD`:

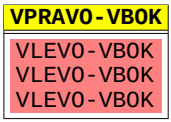


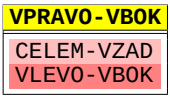
Každý příkaz se nějak jmenuje – `CELEM-VZAD` – název je oddělen od těla příkazu dvojitou čárou. A protože je název důležitý, je vybarven *žlutě*.

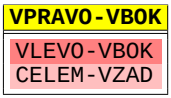
Následuje tělo příkazu – *blok* složený z dalších příkazů. Jednotlivé příkazy mají *červenou* (jedná-li se o primitiva) nebo *růžovou* (jedná-li se o složené příkazy) barvu. Blok končí vodorovnou čárou – ve slovním zápise to znamená `KONEC`.

Uvidíte, že kopenogram je mnohem přehlednější způsob zápisu především složitějších příkazů.

Nyní naučíme Karla dalšímu obratu na místě – `VPRAVO-VBOK`. Jak už jistě sami tušíte, bude se skládat ze 3 příkazů `VLEVO-VBOK`. Možná řešení jsou ale tři:





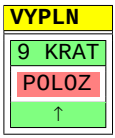


protože Karel už zná příkaz `CELEM-VZAD`. Zápis pro Karla jistě zvládnete sami. Po provedení příkazu `VPRAVO-VBOK` by se Karel měl otočit o 90° po směru hodinových ručiček.

1.5. Cyklus `OPAKUJ`

Představte si, že je třeba nějakou část programu zopakovat. Například v příkaze `VYPLN`, kterým Karel vyplní políčko, na kterém stojí, značkami, t.j. položí tam 9 (nebo 5 aj. – podle implementace) značek.

Mohli bychom to vyřešit podobným způsobem, jako při definici příkazu `VPRAVO-VBOK` ze tří příkazů `VLEVO-VBOK`, t.j. mohli bychom napsat 9× za sebou příkaz `POLOZ`. Ale jistě sami cítíte, že by to bylo příliš dlouhé a nepřehledné. Proto Karel umí zopakovat blok příkazů (nebo i jeden samotný) pomocí příkazu `OPAKUJ`:




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

A jak Karla naučit cyklu `OPAKUJ`? Snadno. Zadáme

`OPAKUJ`↵

a na otázku `KOLIKRAT?` odpovíme

`9`↵.

Všimněte si, že příkaz `OPAKUJ` nám přidá další příkaz `KONEC`. To proto, že je třeba nějak označit blok příkazů, které se mají opakovat – jsou ohraničeny příkazy `OPAKUJ` — `KONEC`.

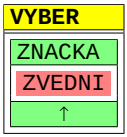
Dále vidíme, že tělo cyklu (a vlastně i tělo celého příkazu) je posunuto trochu doprava – to proto, aby bylo snáze viditelné, co je uvnitř čeho a k čemu patří který KONEC. Toto posouvání si Karel vytváří sám.

Cykly je samozřejmě možné vkládat do sebe – cyklus může obsahovat libovolnou sekvenci příkazů, tedy i další cyklus, podmínky atd.

Horní i dolní část kopenogramu pro opakování má *zelenou* barvu.

1.6. Cyklus DOKUD JE

Dejme tomu, že chceme Karla naučit příkaz VYBER, kterým vybere všechny značky, které jsou na políčku, na kterém stojí. Slovně bychom to vyjádřili asi tak, že bude provádět příkaz ZVEDNI dokud pod ním bude nějaká značka. Pro tyto případy je Karel vybaven příkazem DOKUD a celý příkaz VYBER bude vypadat takto:




```
VYBER ZNAMENA
  DOKUD JE ZNACKA
    ZVEDNI
  KONEC
KONEC
```

A jak naučit Karla cyklu DOKUD? Velmi podobně, jako cyklu OPAKUJ. Zadáme:

DOKUD ↵

a Karel se zeptá: DOKUD JE/NENI – to proto, že příkazy uvnitř DOKUD může vykonávat buď dokud podmínka JE splněna, nebo dokud splněna NENI. (Využití konstrukce DOKUD NENI si ukážeme za pár řádků.

Po volbě JE/NENI jsme dotázáni na podmínku. Zadáme

ZNACKA ↵.

A jaké podmínky umí Karel testovat?

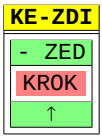
- ZNACKA – test, je-li na políčku, na kterém stojí, nějaká značka (jedna nebo více) – může-li tedy bez obav provést příkaz ZVEDNI.
- ZED – test, je-li na příštím políčku před Karlem, t.j. ve směru, ve kterém je natočen, zeď. Jinými slovy – jestli Karel příštím krokem narazí do zdi.
- SEVER, JIH, VYCHOD, ZAPAD – test, je-li Karel natočen do daného směru.

Pro cyklus DOKUD platí to samé, co pro cyklus OPAKUJ: kopenogram je zelený, cykly se mohou vnořovat atd. To proto, že oba příkazy jsou stejného druhu – opakují nějaký blok příkazů. Rozdíl je pouze v tom, že v OPAKUJ známe počet opakování, kdežto u DOKUD je opakování cyklu závislé na (ne)splnění vnější podmínky.

Je tu ale ještě jeden malý rozdíl: cyklus OPAKUJ proběhne právě tolikrát, kolikrát má napsáno v záhlaví (pokud nedojde k chybě), zatímco cyklus DOKUD nemusí proběhnout ani jednou – pokud není podmínka splněna hned na začátku, pak se blok příkazů DOKUD — KONEC přeskočí. Zadáte-li takto naučený příkaz VYBER na místě, na kterém není žádná značka, tak Karel nic neudělá. To je v pořádku, protože jsme říkali: dokud je nějaká značka ke zvednutí, tak ji zvedni. Nu a protože tam žádná značka není, tak Karel nemá co na práci.

1.7. Cyklus DOKUD NENI

Pokusme se nyní naučit Karla příkaz KE-ZDI: Karel dojde až ke zdi, která leží někde před ním, aniž by do ní narazil. To znamená: dělej KROK dokud není zeď. A v Karlově zápisu:




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

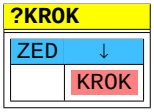
Chování cyklu DOKUD NENI je stejné – vykonává se, dokud je splněna podmínka, že NENI ZED. Znak „-“ v kopenogramu značí negaci podmínky.

O cyklu DOKUD NENI platí vše, jako o DOKUD JE.

1.8. Příkaz KDYZ

Existují případy, kdy je nutno vykonávání programu rozdělit na dvě části podle nějaké podmínky: je-li splněna, pak vykonej toto a pokud není, vykonej něco jiného. Karel je pro takové případy vybaven příkazem KDYZ.

Dejme tomu, že chceme Karla naučit příkazu ?KROK (opatrný krok) – Karel udělá krok, pokud před ním není zeď:




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

I v tomto případě je zápis kopenogramem přehlednější. Vidíme v něm 2 paralelní větve programu: ta levá, v našem případě prázdná, se vykoná, pokud je podmínka splněna; pokud není splněna, vykoná se pravá větev. Tomu odpovídá konstrukce KDYZ — KONEC JINAK — KONEC – vidíte, že nám přibyl další KONEC. To vyplývá i z kopenogramu – vodorovná čára, ukončující blok, znamená KONEC.

A ještě k barvičkám v kopenogramu: záhlaví (obě části) příkazu KDYZ mají *modrou* barvu.

Nu a to je ze základních kamenů Karla vše. Zdá se Vám to málo? A přece to stačí. Pravda, jednu, tu nejdůležitější, věc jsem zapomněl zdůraznit. Bez ní by Karel byl skoro k ničemu. Je to *rekurze*.

1.9. Rekurze

Co to je *rekurze*? Stručně řečeno: je to volání sebe sama. To znamená, že při psaní nového příkazu uvedeme někde v těle název příkazu samotného. Zdá se Vám to divoké? Nu, ano, přiznávám, ale až poznáte, jaký je ta rekurze mocný a kouzelný nástroj, nebude se Vám bez ní chtít programovat.

Začneme příkazem PIRUETA – Karel se roztočí jako krasobruslař a nikdy sám od sebe neskončí, pokud ho k tomu nedonutíme vnějším zásahem.


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

Ptáte se, jak je vůbec možné, že příkaz může volat sám sebe? Snadno. Pokud totiž učíte Karla, pak mu na začátku každého nového příkazu řeknete, jak se ten příkaz bude jmenovat (PIRUETA).

Karel si jej hned zařadí do svého slovníku (pokud se dá v době definování nového příkazu vypsat slovník, tak můžete vidět, že právě rozpracovaný příkaz už v něm je) a od té chvíle už je pro Karla známý v tom smyslu, že je jej možno použít v definici složených příkazů (tedy i sebe samotného).

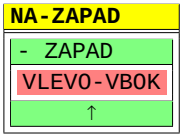
Připadá Vám příkaz `PIRUETA` divný? Možná, ale funkce je docela jednoduchá. Když zadáte Karlovi příkaz `PIRUETA` (předtím ho ale musíte `PIRUETU` naučit), podívá se, co ten příkaz znamená. A vidí, že znamená `VLEVO-VBOK`. To je primitivum, které umí okamžitě vykonat a proto se pootočí. Pak následuje příkaz `PIRUETA`. To není primitivum a proto se a podívá, co že znamená `PIRUETA`. A vidí, že znamená `VLEVO-VBOK` (primitivum, které umí rovnou vykonat → pootočí se) a `PIRUETA`. Opět se podívá, co znamená `PIRUETA`. A vidí, že ...

Jistě tušíte, že bychom to mohli rozepisovat donekonečna. Ano, správně. Karel se roztočí jako čamrda a nebude jevit vůli se zastavit. A přece – Karel si pokaždé, když narazí na složený příkaz (a ne na primitivum) poznamená, že odbíhá dělat něco jiného a že se musí potom někam vrátit, aby dokončil to, co rozdělal. Tento *zásobník* není bezedný, jako jeho baťůžek na značky. A proto se po nějaké době stane, že si už nemá kam zapsat další poznámky a zanaříká: `UZ ME TO NEBAVI` – nebo něco na ten způsob – a skončí. Není to ale regulérní konec příkazu – je to jako kdyby např. narazil do zdi.

Zde je jedna malá změna v barvách kopenogramu: řádek, kde se volá rekurze, se vybarvuje *žlutě* – stejně, jako hlavička programu – aby bylo hned na první pohled jasné, že jde o rekurzi.

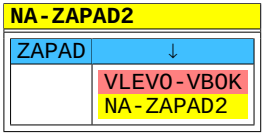
Určitě si teď říkáte – no nic moc, ta rekurze. To byl ale jen ten nejjednodušší případ, který byl vybrán pro začátek. Rekurzi je samozřejmě možno použít i efektivněji, než jen k nekonečnému opakování.

Dejme tomu, že bychom chtěli Karla naučit příkaz `NA-ZAPAD`, kterým se otočí tak, aby stál čelem na západ. Jistě Vás napadlo následující řešení s použitím cyklu `DOKUD`:




```
NA-ZAPAD ZNAMENA
  DOKUD NENI ZAPAD
    VLEVO-VBOK
  KONEC
KONEC
```

Nyní to ale zkusíme jinak, pomocí `KDYZ` a rekurze:




```
NA-ZAPAD2 ZNAMENA
  KDYZ JE ZAPAD
  KONEC JINAK
    VLEVO-VBOK
    NA-ZAPAD2
  KONEC
KONEC
```

Jak to funguje? Karel se nejprve podívá, zda je otočen na západ. Pokud ano, pak nic neudělá. To je v pořádku. Pokud není na západ, pak udělá primitivum `VLEVO-VBOK` a zavolá opět `NA-ZAPAD2`. Podívá se, je-li na západ. Pokud není, udělá primitivum `VLEVO-VBOK` a zavolá `NA-ZAPAD2` ...

Toto je příklad postupného zjednodušování úlohy. Dejme tomu, že máme vyřešit nějaký problém (otočení na západ), který se skládá z neznámého počtu zjednodušujících kroků (`VLEVO-VBOK`). A tak můžeme postupovat tak, že:

1. Otestujeme, zda-li už je problém vyřešen
2. Pokud není, uděláme dílčí zjednodušení (`VLEVO-VBOK`) a vrátíme se k bodu 1.
3. Pokud je problém vyřešen, máme vyhráno.

Všechno může být samozřejmě daleko složitější, ale pokud najdeme v problému tuto závislost, máme vyhráno.

Až dosud vlastně nezáleželo na tom, kam se Karel v rekurzi vrací – protože pak následuje stejně jen KONEC. Tomuto užití rekurze se říká *nepravá rekurze*. Jistě už tušíte, že když existuje rekurze nepravá, že bude existovat i rekurze nelevá. Ano, ale říká se jí *pravá*. A u pravé rekurze velmi záleží, jestli se vrátíme a kam.

Že Vám není jasné to vracení se zpátky? Dobrá, vraťme se teď my k příkazu VPRAVO-VBOK (tomu prostřednímu). Jak jej Karel vlastně vykonává? Podívá se, jak je definován. A vidí, že je tam příkaz CELEM-VZAD. Ten příkaz není primitivum, takže bude muset přerušit vykonáváni příkazu VPRAVO-VBOK a jít se věnovat příkazu CELEM-VZAD. Nejprve si ale do zásobníku poznamená, kam se pak má vrátit – na druhý příkaz ve VPRAVO-VBOK. Pak se vrhne na příkaz CELEM-VZAD. Jakpak je definován? Je tam VLEVO-VBOK – to je primitivum, které umí vykonat a tak jej hned vykoná, dále další VLEVO-VBOK – opět jej ihned vykoná a následuje KONEC. Karel by rád skončil, ale je poslušný (a dobře naprogramovaný) a tak se podívá do svého zápisníku/zásobníku, jestli se náhodou nemá k něčemu vracet. A skutečně – má tam napsáno, že se má vrátit na druhý příkaz příkazu VPRAVO-VBOK. Dobrá, to je VLEVO-VBOK – primitivum, umíme, okamžitě vykonáme – a pak je tam KONEC. Karel se opět podívá do zápisníku/zásobníku, ale tam už nic není, protože si předchozí poznámku škrtnul. Může tedy s klidným srdcem, které je určitě z elektronek, ukončit všechno vykonávání.

Graficky by se to dalo ztvárnit asi takto:

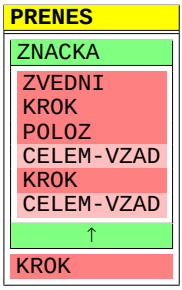
```
VPRAVO-VBOK ZNAMENA
  CELEM-VZAD          → CELEM-VZAD ZNAMENA
                           VLEVO-VBOK
                           VLEVO-VBOK
                        ← KONEC
  VLEVO-VBOK
KONEC
```

Že se Vám to zdá jednoduché? ALE ONO TO JE JEDNODUCHÉ. A proč se tomu tak obšírně věnuji? Protože u rekurze je ten rozdíl (on to vlastně ani rozdíl není), že se nevolá jiný příkaz, ale že se příkaz volá sám. To ale Karel netuší – pro něj jsou všechny složené příkazy rovnocenné – musí si poznamenat do zásobníku, kam se má vrátit.

1.10. Přenášení značek

Mějme tento úkol: Karel stojí na několika značkách a nemá před sebou zeď. Jeho úkolem je vybrat značky a přenést je o jedno políčko dál a tam je zase položit – tolik, kolik jich předtím nasbíral.

Tento problém je možno řešit i bez použití rekurze, pomocí DOKUD, a to takto:

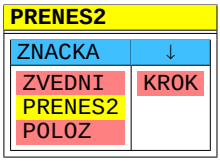



```
PRENES ZNAMENA
  DOKUD JE ZNACKA
    ZVEDNI
    KROK
    POLOZ
    CELEM-VZAD
    KROK
    CELEM-VZAD
  KONEC
  KROK
KONEC
```

Jak to funguje asi uhádnete sami. Dokud je na políčku, kde Karel stojí, značka, tak ji zvedne, udělá krok, tam ji položí a zase se vrátí. Nakonec, když tam už žádné značky nejsou, udělá Karel krok na cílové políčko.

Jistě, cíl to splnilo, ale co se Karel naběhá – s každou značkou zvlášť (a to má na zádech na značky batůžek), pořád se točí ...

Nešlo by to řešit jednodušeji, elegantněji? Šlo, ale jen s rekurzí:




```
PRENES2 ZNAMENA
  KDYZ JE ZNACKA
    ZVEDNI
    PRENES2
    POLOZ
  KONEC JINAK
    KROK
  KONEC
KONEC
```

Možná na to teď nevěřícně hledíte a říkáte si: to je nějaké podezřelé. Ale není. Je to naprosto regulérní příkaz. Pokusím se Vám popsat vykonávání tohoto příkazu pro případ, že pod Karlem jsou 2 značky. Pracuje to samozřejmě naprosto stejně pro libovolný počet značek, jen by se to sem větší nevešlo.

```
PRENES2 ZNAMENA
  KDYZ JE ZNACKA
    ZVEDNI
    PRENES2       → PRENES2 ZNAMENA
                      KDYZ JE ZNACKA
                        ZVEDNI
                        PRENES2       → PRENES2 ZNAMENA
                                          KDYZ JE ZNACKA
                                          KONEC JINAK
                                            KROK
                                          KONEC
                                      ← KONEC
                        POLOZ
                      KONEC JINAK
                      KONEC
                    ← KONEC
    POLOZ
  KONEC JINAK
  KONEC
KONEC
```

Je třeba poznamenat, že v podmínce `KDYZ` je rozepisována jen ta větev, která se vykonává vzhledem ke splněnosti podmínky.

Není to řešení nádherné? Karel si vlastně spočítal značky tím způsobem, že si postupně zapisoval do zásobníku návraty na příkaz `POLOZ`. I po stránce časové dělá Karel minimum příkazů – a vykonávání trvá minimum času.

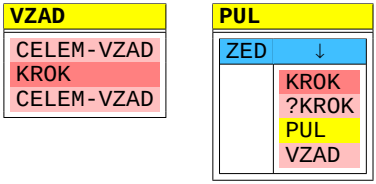
O tomto příkazu (a o všech s touto strukturou) je možné říci následující: část za rekurzí (`POLOZ`) se vykoná tolikrát, jako část před rekurzí (`ZVEDNI`). Druhá větev (`KROK`) se provede jednou, a to mezi nimi.

S touto znalostí se můžeme pustit do dalšího klasického příkladu – PUL – Karel dojde do poloviny vzdálenosti mezi ním a zdí.

Že nevíte, jak na to? Trochu Vám napovím. Karel si cestou ke zdi spočítá dvojkroky, které musel udělat a zpět udělá právě tolik kroků.

Ještě nic? Přemýšlejte, jenom číst – to není ta správná cesta programování.

Ale abyste se netrápili, tady je řešení:



1.11. Hanojské věže

Znáte problém Hanojských věží? Ne? Tak poslouchejte.

Kdesi ve starém Tibetu je klášter, kde mají zajímavou věc: zlatou, stříbrnou a platinovou tyč, na kterých je navlečeno 64 kroužků posázených diamanty. Kroužky jsou odstupňovány podle velikosti – dole je největší, nahoře je nejmenší. Na počátku byly všechny kroužky navlečeny na zlaté tyči. Úkolem tamních mnichů je přemístit celou pyramidu kroužků na tyč stříbrnou. Musí ale dodržovat následující pravidla:

- Vždy je možno přenášet pouze jeden kroužek z jedné tyče na druhou
- Není možno umístit větší kroužek na menší.

Ptáte se, co má společného Tibetský klášter s Karlem? Inu to, že tento problém je možno poměrně efektivně algoritmizovat a naprogramovat, a to i v Karlovi. Že nevěříte? Přesvědčte se na dalších řádcích.

Nejprve se pokusme najít postup, jakým je možno problém řešit. Asi již tušíte, že na to půjdeme přes rekurzi. Správně, bude to nejefektivnější.

Postup přenášení věže lze krásně popsat následujícím rekurzivním algoritmem:

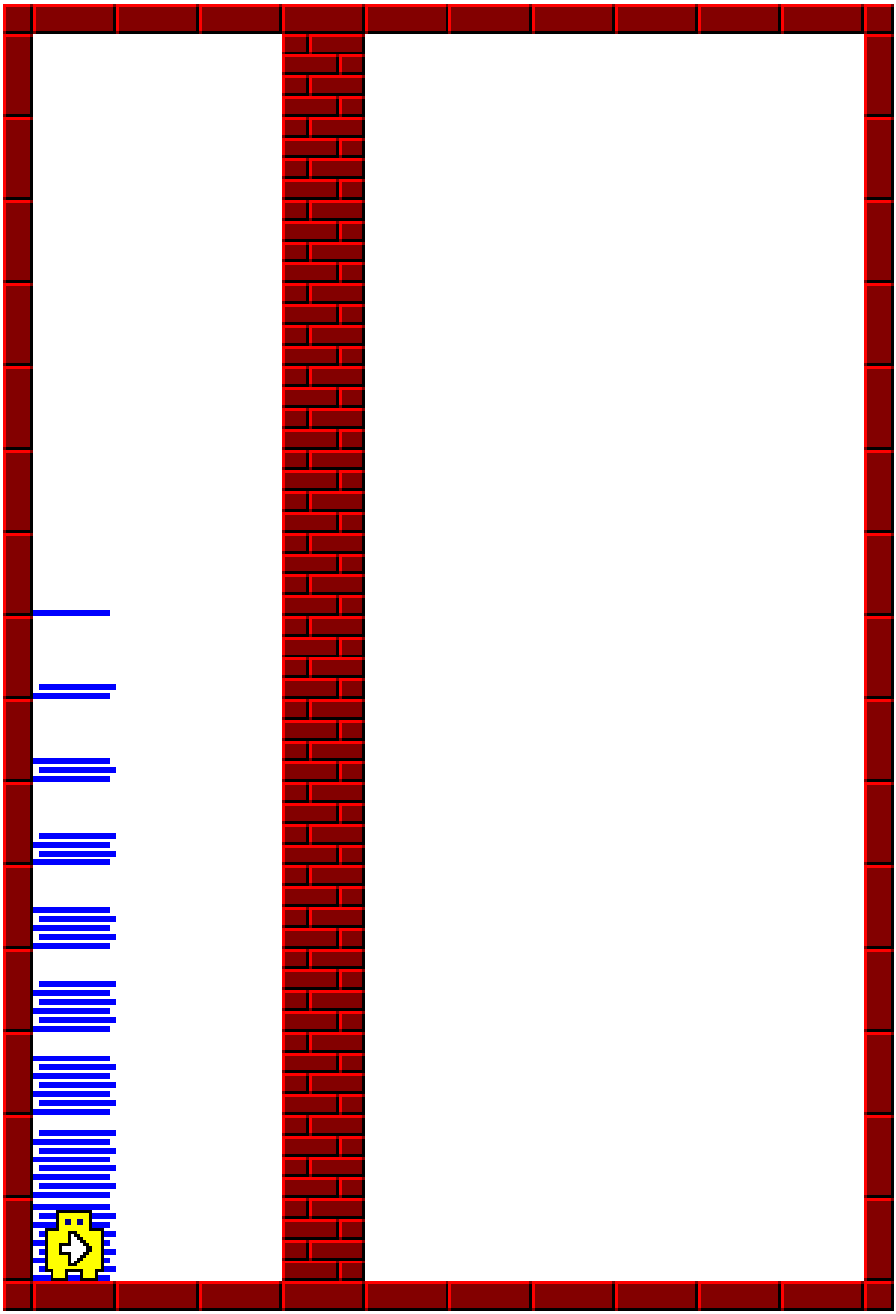
- Pokud není žádný kroužek k přenášení, pak není co dělat *(okrajová podmínka)*
- Přesuň věž od druhého kolečka odspodu na pomocnou tyč (vlastně rekurze)
- Přenes nejspodnější kroužek na cílovou tyč
- Přenes věž z pomocné tyče na cílovou (opět rekurze)

To je celé. Že to je jednoduché? Ano, to je tou rekurzí. Nu a jak budeme tyče reprezentovat v Karlovi? Tak třeba následovně:

- Jednotlivé kroužky budou znázorněny značkami nad sebou
- Jednotlivé tyče budou sloupce ve městě
- Ty tři tyče je třeba postavit „do kruhu“ – aby od každé byla jedna vpravo a jedna vlevo.
- Kroužků nebude 64, to bychom se konečného řešení nedočkali a ani nemáme tak velké město. Bude jich, dejme tomu, 9. POZOR – nad posledním je třeba ještě jedna volná řada ve městě!

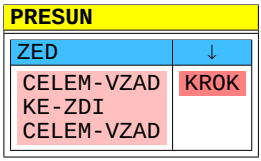
Obrázek 2 ukazuje možné základní postavení. Je vytvořena věž z 9-ti kroužků, Karel ji bude přenášet doprava.

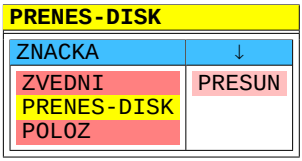
Velikost kroužku je reprezentována počtem značek – nemá to vliv na přenášení.



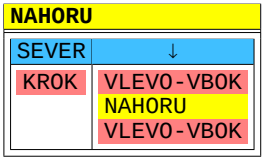
Obr. 2: Základní postavení na „Hanojské věže“

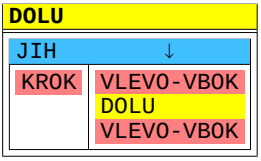
Říkali jsme, že Karel musí mít dojem, že ty tři tyče jsou do kruhu, t.j. že z každé se dá jít vlevo i vpravo. Aby toto bylo splněno, máme příkaz `PRESUN`. Pro přenesení libovolného počtu značek ve směru, kterým je Karel natočen, napišme příkaz `PRENES-DISK` – bude modifikací příkazu pro přenášení značek o jeden krok:



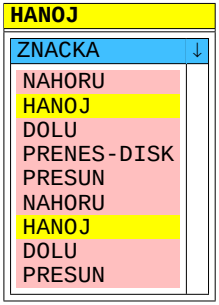


Nyní si musíme ujasnit, jak budeme přenášet subvěže: budeme je přenášet na druhou stranu, než potřebujeme přenést celou věž. Potřebujeme tedy udělat krok na sever a natočit se do opačného směru, než původně. Podobně potřebujeme poté slézat dolů:





Dejme tomu, že příkaz pro přenesení Hanojské věže se bude jmenovat `HANOJ`. Na něj Karel přenese věž od úrovně, na které stojí, ve směru, do kterého je natočen. Co bude obsahovat? Jednak onu okrajovou podmínku – pokud není značka, tak se nemusí nic přenášet. Pak bude obsahovat pokyn k přenesení subvěže na druhou stranu (`NAHORU` + `HANOJ` – rekurze), po přenesení subvěže skončí Karel tam, kde začal, takže musí zase slézt dolů (`DOLU`), přenést jeden disk (`PRENES-DISK`), přesunout se na přenesenou subvež (`PRESUN` + `NAHORU`), přenesení subvěže na už přenesený základní disk (`HANOJ` – další rekurze), navrácení se na dolní úroveň (`DOLU`) a přesunout se na výchozí místo, odkud jsme celou věž přenesli (`PRESUN`).



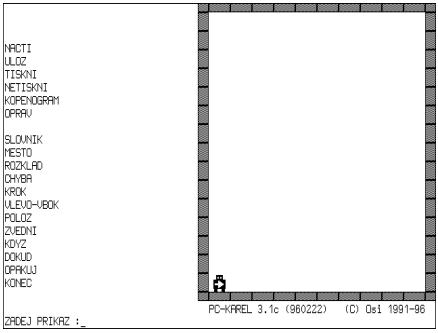
Samozřejmě byste si to měli zkusit na vlastní oči na počítači, jinak neuvěříte.

1.12. Závěr

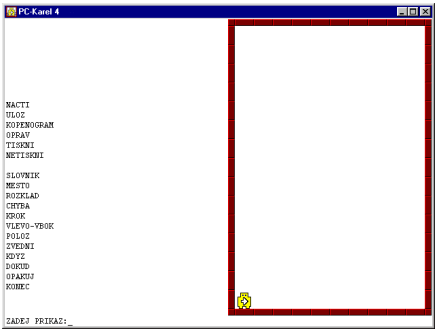
To je vše o základech jazyka Karel. Já se znám s Karlem od roku 1985, kdy mi bylo 13 let, a s odstupem času se k němu rád vracím. Považuji ho za geniální nápad – co věcí se dá dělat s osmi jednoduchými příkazy a malinkatým robotkem. Ne nadarmo se říká: V jednoduchosti je genialita. Přeji Vám, abyste si Karla zamilovali tak, jako já.

2. PC–Karel – Karel pro IBM-PC

Program `PC-Karel` je interpretrem jazyka Karel pro počítače IBM–PC. Existuje ve dvou verzích: `PC-Karel 3·1` je v provedení pro MS–DOS, `PC-Karel 4` je pro MS–Windows. Obě implementace jsou velmi podobné, případné odlišnosti jsou označeny **D** (platí pouze pro DOS verzi) nebo **W** (platí pouze pro Windows verzi).



PC–Karel 3·1



PC–Karel 4

D Interpret jazyka Karel `PC-Karel` se skládá pouze z programu `KAREL.EXE`. Práce je možná ve dvou režimech: grafickém (na kartách EGA/VGA a Hercules) a textovém. Implicitně se nastavuje grafický režim. Pokud vyžadujete textový, který je výrazně rychlejší, je třeba při spouštění zadat parametr `/t` – Text nebo `/tm` – Text Mono. Identifikace Vaší grafické karty by měla proběhnout automaticky. Pokud by ale nastaly problémy, pokuste se programu vnutit pomocí parametrů `/e` – EGA/VGA, `/m` – EGA/VGA Mono nebo `/h` – Hercules grafický adaptér, který je ve Vašem počítači. Program je dodáván jako samorozbalovací archiv EXE spolu s licenčním ujednáním (soubor `licence.kam` v kódování Kamenických, `licence.852` v kódování CP852).

W Interpret jazyka Karel `PC-Karel` se skládá pouze z programu `KAREL16.EXE` (verze Windows 3.1/3.11), případně `KAREL32.EXE` (verze Windows 95/98/ME/NT/2000). Verze pro Windows 3.1/3.11 je dodávána jako samorozbalovací archiv EXE spolu s licenčním ujednáním (soubor `licence.txt`). Verze pro Windows 95/98/ME/NT/2000 je dodávána jako instalační EXE a instaluje se standardním způsobem pro tyto operační systémy. Program může kromě barevného režimu pracovat i v černobílém – volba se provádí, stejně jako ovládání dalších Windowsích vlastností, přes systémové menu (ikona v levém horním rohu okna).

- Program `PC-Karel` vychází z programů `KAREL92` – `KAREL94` a je s nimi kompatibilní.
- Všechny příkazy se zadávají jako jednotlivá slova. O případné další (podmínky) si `PC-Karel` řekne sám.
- Příkazy je možno zkracovat – napsat jen několik počátečních písmen s tečkou na konci.
- V definici nového příkazu má „`Enter` (↵)“ význam příkazu `KONEC`.
- Je možno listovat v 9-ti posledně zadaných příkazech pomocí kláves „↑“ a „↓“.
- Maximální počet značek na jednom políčku je 9.

D V příkaze `MESTO` je zavedeno nastavení pauzy – doba v [ms], kterou se čeká vždy při vykonání výkonného primitiva (`KROK`, `VLEVO-VBOK`, `POLOZ`, `ZVEDNI`).

- Rychlost vykonávání (velikost pauzy) je možno při vykonávání plynule měnit pomocí kláves „+“ a „-“.

- Stiskem klávesy „Shift“ popř. zamknutím pomocí „CapsLock“ je příkaz prováděn maximální rychlostí (odpovídá nastavení =0 ms).
- Provádění příkazu se ukončuje stiskem klávesy „Esc“.
- Je možno ukládat a načítat obsah města: v režimu MESTO pomocí kláves „F2“ a „F3“. Město, které se zadá i s příponou „.MES“ při spouštění programu, se ihned načte.
- Je zaveden cyklus s podmínkou na konci OPAKUJ — POKUD. Zadává se jako OPAKUJ, na dotaz na počet opakování se odpoví pouze „Entrem (↵)“. Slovo POKUD se zadává jako KONEC, následuje otázka na podmínku.
- Je možno používat konstrukce KDYZ NENI — KONEC JINAK — KONEC.
- Načítání a ukládání slovníku je možné pomocí klíčových slov ULOZ a NACTI. Slovník, který se má načíst při startu programu, je možno zadat jako parametr příkazové řádky (nemusí obsahovat příponu „.KAR“).

W Slovník je možno uložit i načíst v textové podobě – tato forma není načtitelná do DOSové verze. Slovník lze vymazat pomocí klíčového slova SMAZ.

- Jsou zavedeny nové podmínky: STISKNUTO – test stisku „Ctrl“, 1–9 (počet značek), PLNO (plné políčko značek – totožné s 9), X1–X10 a Y1–Y15 pro pozici na dvorku počítanou od levého dolního rohu, NAHODA pro asi 50% splnění.
- Je možno generovat kopenogramy pomocí klíčového slova KOPENOGRAM.

D V kopenogramech je možno nahradit dvojitou čáru za jednoduchou pomocí parametru /kj – Kopenogram Jednoduchý, tisknout kopenogram ze znaků „=-+|“ pomocí parametru /kt – Kopenogram Textový. Pro zobrazování v CP852 je možné odpojit dvojitou čáru od jednoduché pomocí parametru /kl – Kopenogram Latin2.

- ROZKLAD, SLOVNIK a KOPENOGRAM je možno posílat do souboru. Jméno tohoto souboru se zadá po příkazu TISKNI. Nové věci se připojují na konec tohoto souboru. Tisk se končí příkazem NETISKNI.

W V příkazu TISKNI je možno zvolit výstupní formát: HTML nebo textový s volbou rámečků v KOPENOGRAMech – textové (složené ze znaků „=-+|“), jednoduché rámečky, dvojité rámečky nebo rámečky pro CP852 (odpojení dvojité vodorovné čáry od svislé).

D V režimu MESTO je možno vytisknout město – tiskne se buď textově do souboru příkazu TISKNI – pomocí klávesy „F5“, nebo graficky buď na prn (pro Epson, Star ...) pomocí klávesy „F8“ nebo do souboru .PCX klávesou „F9“.

W Město je možno kopírovat do schránky (clipboardu) přes nabídku v systémovém menu. Kopíruje se černobíle nebo barevně – podle aktuálního zobrazení.

D V režimu psaní příkazu je možno ovládat město: pomocí „L-Shift + Kurzory“ – pohyb Karla po dvorečku, „R-Shift + Kurzory“ – natáčení Karla, „Alt + N, P, S, Z, 0..9“ – Nové, Polož, Smaž, Zeď, počet značek – odpovídá režimu MESTO.

W V režimu psaní příkazu je možno ovládat město: pomocí „Ctrl + Kurzory“ – pohyb Karla po dvorečku, „Ctrl + Shift + Kurzory“ – natáčení Karla, „Ctrl + N, P, S, Z“ – Nové, Polož, Smaž, Zeď – odpovídá režimu MESTO.

D Klíčovým slovem OPRAV se spouští jednoduchý editor pro opravu příkazu (zadá se jeho jméno) nebo psaní příkazů nových (zadá se prázdné jméno). Editování se končí klávesami „Ctrl+E“ – akceptování napsaného, nebo „Esc“ – zrušení napsaného. Pokud se po zadání „Ctrl+E“ neobnoví Karlova obrazovka, nýbrž se zůstane v editoru, pak se Karlovi něco ve Vašem zápise nelíbilo – kurzor je nastaven na kritické místo a v poslední řádce svítí krátká nápověda.

W Klíčovým slovem `OPRAV` se spouští editor pro opravu příkazu (zadá se jeho jméno) nebo psaní příkazů nových (zadá se prázdné jméno). Po ukončení editoru je nutno stisknout některou klávesu, aby došlo k načtení nově nadefinovaných příkazů

- Nakonec nejdůležitější: `PC-Karel` se končí příkazem `KONEC`.

Tento program byl oceněn v soutěži Ministerstva školství, mládeže a tělovýchovy České republiky „Duhová disketa 93“.

Na závěr bych chtěl poděkovat všem, kteří mi říkali a psali své připomínky a náměty a tím pomohli k rozvoji programu.

Tento program je ABSOLUTNĚ BEZ ZÁRUKY; jde o volné programové vybavení jehož šíření je za jistých podmínek vítáno. Podrobnosti najdete v části Licenční ujednání. Budu Vám vděčný za jakékoliv připomínky ať už k programu, nebo k textu samotnému.

3. Licenční ujednání

1. Smíte používat tento program bez jakéhokoliv omezení. Toto používání je bezúplatné.
2. Smíte rozmnožovat a šířit kompletní kopii tohoto programu tak, jak jste ji obdrželi. Toto šíření je bezúplatné. Za fyzický akt přenesení kopie programu můžete žádat poplatek.
3. Žádné modifikace tohoto programu nejsou povoleny.
4. Vzhledem k bezplatnému poskytnutí licence k tomuto programu se na něj NEVZTAHUJE ŽÁDNÁ ZÁRUKA, a to v míře povolené zákonem. Tento program je poskytován „tak, jak je“, bez záruky jakéhokoliv druhu, ať výslovné nebo vyplývající, včetně, ale nikoli jen, záruk prodejnosti a vhodnosti pro určitý účel. Pokud jde o kvalitu a výkonnost programu, leží veškeré riziko na vás. V žádném případě, s výjimkou toho, když to vyžaduje platný zákon, vám nebude žádný z držitelů autorských práv odpovědný za škody, včetně všech obecných, speciálních, nahodilých nebo následných škod vyplývajících z užívání anebo neschopnosti užívat programu (včetně ale nikoli jen, ztráty nebo zkreslení dat, nebo trvalých škod způsobených vám nebo třetím stranám, nebo selhání funkce programu v součinnosti s jinými programy), a to i v případě, že takový držitel autorských práv nebo jiná strana byli upozorněni na možnost takových škod.
5. Všechna práva, která nejsou výslovně udělena, jsou vyhrazena nositeli autorských práv.
6. Platnost této licence není omezena územím České republiky.
7. Instalací a použitím tohoto programu souhlasíte s těmito licenčními podmínkami. Pokud s nimi nesouhlasíte, nepokračujte v instalaci nebo tento program odinstalujte.