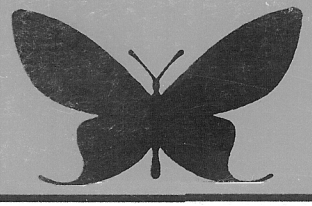
COMMPAS
vydavatelství, služby a distribuce
Dr. Ivan Pavlíček, Pavel Škvrně
p. o. box 60
140 00 Praha



COMMPAS
vydavatelství a služby
uživatelům mikropočítačů COMMODORE

ZÁKLADY PROGRAMOVÁNÍ

1. díl

základy programování v jazyku BASIC

COMMODORE 64

OBSAH:

I. ZÁKLADNÍ PRAVIDLA PROGRAMOVÁNÍ

Úvod

Tato kapitola hovoří o tom, jak programovací jazyk BASIC ukládá data a jak s nimi manipuluje. Předkládaná látka obsahuje:

1) Krátkou informaci o operačním systému, jeho částech, funkcích a znacích, používaných v Commodoru 64.
2) Vytváření konstant a proměnných, které typy proměnných existují, jak jsou konstanty a proměnné uložené v paměti.
3) Pravidla aritmetických výpočtů, testy vzájemných vztahů, manipulace se znakovými řetězci a logické operace, pravidla pro vytváření výrazů a způsoby konverze dat, které používá BASIC pro práci s daty smíšeného typu.

ZNAKY NA OBRAZOVCE

(základní množina)

Operační systém (OS)

Operační systém je uložen v paměti ROM (Read Only Memory, paměť, ze které je možno jen číst) a je kombinací tří základních programovacích modulů:

1) BASIC INTERPRETER
2) KERNAL
3) Screen Editor (obrazovkový editor)

1) BASIC Interpreter analyzuje syntaxi příkazů BASIC (správný zápis a použití) a provádí požadované výpočty a manipulaci s daty. Má slovník 65 klíčových slov (keywords); každé má speciální význam. Pro zápis těchto klíčových slov a pro zápis jmen proměnných se používá velkých a malých písmen a číslic 0 - 9.

Interpreter zná i některá diakritická znaménka a zvláštní znaky. Jejich seznam a význam je uveden v tabulce 1-1.

2) KERNAL je programový modul, který řídí většinu přerušení v systému (interrupt level processing). Podrobnosti jsou v kapitole 5. KERNAL také řídí vstup a výstup dat (input/output).
3) Screen Editor řídí výstup informací na obrazovku a edituje (umožňuje opravy) programy psané v BASICU. Screen Editor dále přijímá povely z klávesnice a rozhoduje, zda se mají bezprostředně provádět, či zda mají jít nejdříve do BASIC Interpreteru.

Tabulka 1-1 Seznam znaků v CBM BASICU

ZNAK	POJMENOVÁNÍ A VÝZNAM
Mezera	odděluje klíčová slova a jména proměnných
;	upravuje výstup v seznamu proměnných
=	znaménko přiřazení a testování vztahu
+	znaménko pro sčítání v aritmetických operacích a pro spojování znakových řetězců
-	odečítání, označení záporného čísla
*	(hvězdička) znak pro násobení
/	znak pro dělení
Šipka	(vzhůru) znak pro umocňování
()	v matematických výrazech a funkcích
%	deklaruje celočíselnou proměnnou
#	označuje logické číslo souboru
$	deklaruje znakovou proměnnou
,	upravuje výstup, oddělovač v příkazech
.	oddělovač desetin v desetinných číslech
"	uzavírají znakové konstanty
:	odděluje příkazy BASICU na jednom řádku
?	zkratka pro příkaz PRINT
<	pro testování
>	pro testování
PI	číselná konstanta 3.141592654

Operační systém umožňuje dva způsoby práce v BASICU:

1) Přímý mód (DIRECT MODE)
2) Programový mód (PROGRAM MODE)

1) V přímém módu se nepoužívá číslo řádku před příkazem. Příkazy se provádějí bezprostředně, okamžitě po stisknutí tlačítka **RETURN**.
2) Programový mód se užívá při práci s programy. V tomto módu musí mít všechny BASIC příkazy před sebou číslo řádku. V jednom řádku (za jedním číslem řádku) můžete mít několik příkazů, ale jejich počet je omezen tím, že na jednom logickém řádku obrazovky může být nejvýše 80 znaků. To znamená, že překročíme-li 80 znaků, musíme celý příkaz, který se do dané řádky nevešel, napsat do nové řádky.

Vždy než začneš psát nový program napiš **NEW** a stlač **RETURN**!

Commodore 64 má dvě úplné sady znaků, které můžeš používat z klávesnice nebo ve svém programu.

1. sada

- bez přeřazovačů jsou použitelná velká písmena a číslice **0-9**
- s přeřazovačem **SHIFT** lze psát grafické znaky znázorněné vpravo na přední straně zvoleného tlačítka
- při stlačené klávese **C=** se užívají grafické znaky vlevo na přední straně tlačítka

Stlačíme-li **SHIFT** pro tlačítko bez grafického symbolu, je užit symbol znázorněný na horní plošce tlačítka.

2. sada

- bez přeřazovačů se používají malá písmena a číslice 0 - 9
- při stlačené klávese **SHIFT** jsou to velká písmena
- při stlačené klávese **C=** jsou to opět grafické znaky z levé přední strany tlačítka

Pro přepnutí z jedné sady na druhou stiskni současně **SHIFT** a **C=**.

PROGRAMOVÁNÍ ČÍSEL A PROMĚNNÝCH

Celočíselné, reálné a řetězcové konstanty

Konstanty jsou hodnoty vkládané do příkazů v BASICU. BASIC tyto hodnoty používá při provádění příkazů. CBM (Commodore) BASIC umí rozeznat a zpracovat tři typy konstant:

1) Celá čísla (Integer Numbers)
2) Reálná čísla (Floating-Point Numbers)
3) Řetězce (Strings); alfanumerické proměnné, složené z písmen a číslic

Celá čísla

Pro C64 musí jejich hodnoty ležet mezi -32768 a +32767. Celočíselné konstanty nemají mezi číslicemi desetinnou čárku ani tečku. Je-li vynecháno znaménko plus (+), počítač předpokládá, že konstanta je kladné číslo. Nuly zapsané před konstantou jsou ignorovány a nesmějí se používat, protože zbytečně zabírají paměť a zpomalují program. Nezpůsobují však chyby. Celočíselné konstanty jsou v paměti uloženy jako dvojbytová binární čísla.

Několik příkladů celočíselných konstant:

-12 +44
8765 0
-32768 -876

POZNÁMKA: Nevkládej čárku ani mezeru pro oddělení tisíců mezi číslice! Počítač tě upozorní na chybu oznamem: ?SYNTAX ERROR.

Reálná čísla (čísla s pohyblivou řádovou čárkou)

Jsou to kladná nebo záporná čísla, která mohou obsahovat zlomkovou část. Pro oddělení této části se používá desetinná tečka. Opět použití čárky nebo mezery mezi číslicemi je chyba. Není-li před konstantou znaménko +, předpokládá počítač kladnou konstantu. Nuly před konstantou jsou ignorovány.

Reálné konstanty mohou být užity dvojím způsobem:

1) Jednoduché číslo (Simple Number) - např.: 5.1
2) Vědecká notace (Scientific Notation); semilogaritmický tvar - např.: 2.5E-6

Reálná čísla se zobrazují nejvýše devíti číslicemi, tj. jejich hodnota může být mezi -999999999 až +999999999. Vložíš-li do počítače více než 9 číslic, číslo bude zaokrouhleno podle desáté číslice.

V případě, že číslice na desátém místě je větší než 5, bude poslední (deváté) místo zaokrouhleno nahoru, jinak se zaokrouhluje dolů. To může být důležité při posuzování konečných výsledků. K uložení reálného čísla je potřeba 5 bytů, výpočet je prováděn s přesností na 10 desetinných míst. Výsledek pro tiskový výstup je ale zaokrouhlen na 9 míst.

Několik příkladů jednoduchých reálných čísel:

1.23 -.998877
+3.1459 .77777777
-333. .01

Čísla menší než .01 nebo větší než 999999999 se zobrazí ve vědecké notaci. Zápis se skládá ze tří částí:

1) Mantisa
2) Písmeno E
3) Exponent

Mantisa je běžné reálné číslo, např. 1.14; 215.3 atd. Měla by být dávána přednost mantisám v základním tvaru, tj. první nenulová číslice vlevo, ostatní vpravo od desetinné tečky, tedy 1.14; 2.153 atd.

Písmeno **E** informuje, že číslo je psáno s použitím mocnin deseti (v "exponenciálním tvaru").

Exponent, který je poslední částí zápisu, říká, jakou mocninou deseti se mantisa násobí.

Mantisa i exponent mohou být kladné i záporné. C64 může pracovat s exponenty od -39 do +38. Nejmenší zpracovatelné

kladné číslo je 2.93873588E-39, největší pak 1.70141183E+38. V případě, že výsledek výpočtu je menší než uvedená dolní mez, počítač napíše nulu, v případě, že výsledek přesáhne horní mez, počítač oznámí chybu zprávou ?OVERFLOW ERROR.

Několik příkladů reálných čísel ve vědecké notaci:

235.988-3 či lépe 2.35988E-3 (.235988)
2359E6 2.359E9 (2359000000)
-7.09E-12 (-.000000000000709)

Řetězce (String constants)

Jsou to skupiny číslic, písmen a znaků. Retězec může mít libovolnou délku až do maximální hodnoty, kterou je počet míst v 80ti místném řádku, po odečtení čísla řádku a případných dalších náležitostí příkazů.

Řetězec může obsahovat prázdná místa, písmena, číslice, diakritická znaménka, znaménka pro řízení barev či kurzoru, v libovolné kombinaci. Dokonce je možné i zařadit čárky mezi číslice. Jediný znak, který nesmí být použit, jsou uvozovky ("). To proto, že uvozovky se používají pro vyznačení začátku a konce řetězce.

Řetězec může mít také nulovou hodnotu - to znamená, že neobsahuje žádná data. Závěrečné uvozovky je možné vynechat, je-li string posledním na řádce nebo následuje-li dvojtečka (:).

Několik příkladů řetězců:

"" (nulový řetězec)
"HALO"
"Kčs 25,000.00"
"Počet zaměstnanců"

POZNÁMKA: Chceš-li zahrnout uvozovky (") do řetězce, použij CHRS(34).

Celočíselné, reálné a řetězcové proměnné

Proměnné jsou jména, která reprezentují hodnoty dat v příkazu v BASICU. Hodnota proměnné se přiřazuje buď rovnicí jako konstanta (např. X=5), nebo může být výsledkem výpočtu v programu. Proměnné, podobně jako konstanty, mohou být celá čísla, reálná čísla nebo řetězce. V případě, že použijete proměnnou dříve, než jí byla přiřazena určitá hodnota, interpreter jí automaticky přiřadí nulu v případě číselné proměnné nebo nulový řetězec v případě řetězcové proměnné.

Jména proměnných mohou mít libovolnou délku, ale pouze dva první znaky jsou respektovány v BASICU pro C64. To znamená, že žádné dvě použité proměnné nesmějí mít stejná dvě první místa.

Dále jména proměnných nesmějí být stejná jako klíčová slova v BASICU a tato klíčová slova v nich nesmějí být ani obsažena. Klíčová slova jsou všechny povely, příkazy, jména funkcí a jména logických operátorů. Použijete-li náhodou klíčové slovo ve své proměnné, ukáže se při vykonávání programu na obrazovce chybové sdělení ?SYNTAX ERROR.

Pro jména proměnných se používají písmena a číslice. Prvním znakem musí být písmeno. Jako poslední znak jména musí být v případě celočíselné proměnné použit znak %. Pro jméno řetězce musí být posledním znakem dolar. Není-li na konci proměnné ani % ani znak dolaru, počítač předpokládá, že se jedná o reálnou proměnnou.

Několik příkladů jmen proměnných, způsobu přiřazení a typu dat:

$="NADNORMATIVNI ZASOBY" - řetězcová proměnná
MTH$="NASE"+A$ - řetězcová proměnná
CNT%=CNT%+1 - celočíselná proměnná
FP=12.5 - reálná proměnná

Celočíselná, reálná a řetězcová pole

Pole (ARRAY) je tabulka nebo seznam dat, kterému je přiřazeno jméno proměnné. Pole je užitečný způsob, jak zkrátit zápis velkého počtu proměnných, které jsou k sobě v nějakém

vztahu. Vezmeme např. tabulku čísel o dvaceti řádcích, s dvanácti hodnotami v každém řádku (měsíční příjmy dvaceti zaměstnanců po dobu jednoho roku). Normálně bychom museli přiřadit každé hodnotě zvláštní jméno - tedy užít 240 jmen. Zavedeme-li jedno jméno pro pole, stačí pro identifikování hodnoty toto jméno a umístnění v poli - který řádek, který sloupec.

Jména polí mohou být opět celočíselného, reálného nebo řetězcového typu. Všechny prvky pole jsou stejného typu, jaký deklaruje jméno pole. Pole mohou být jednodimenzionální - jednorozměrná - např. seznam žáků ve třídě, dvourozměrná (tabulka platu uvedená výše), třírozměrná (elementy v Rubikově kostce) nebo vícerozměrná. Každý prvek pole je jednoznačně určen jménem pole a za ním následujícím indexem (indexy), či indexovou proměnnou zapsanou v závorce ( ).

Maximální rozměr pole může být teoreticky 255 (kvádr v 255 rozměrném prostoru) a maximální počet prvků v jednom směru by mohl být až 32767. Prakticky ale jsou rozměry polí omezeny kapacitou paměti a 80 znaky v logickém řádku programu. Má-li pole jen jeden rozměr a jeho index nikdy nepřesáhne 10, pak pole bude vytvořeno a zaplněno nulami (nulovými řetězci) při prvním vyvolání prvku. Jinak musí být použit příkaz DIM pro definování tvaru a velikosti pole. Požadovaná velikost paměti pro pole je určena následovně:

5 bytů pro jméno pole
+2 byty pro každý rozměr pole
+2 byty pro každý celočíselný prvek
nebo +5 bytů na každý prvek pro reálná čísla
nebo +3 byty na každý prvek pro řetězce

Indexy mohou být celá čísla, proměnné nebo aritmetické operace, které dávají celočíselné výsledky. Pro každý rozměr pole musí být samostatný index oddělený čárkou. Indexy mohou být čísla od nuly do maximálního počtu prvků v daném rozměru pole. Hodnoty, které jsou mimo tyto meze, způsobí chybové sdělení

?BAD SUBSCRIPT. Několik příkladů jmen polí, přiřazených hodnot a typu dat:

A$(0)="NADNORMATIVNI ZASOBY" - stringové pole
MTH$(K%)="NASE" - stringové pole
SUM(CNT%(1)=FP K% - reálné pole
A(5)=0 - číselné pole
C(2,7,9)=5 - číselné pole

VÝRAZY A OPERÁTORY

Výrazy se tvoří z konstant, proměnných, případně polí. Výrazem může být jednoduchá konstanta či proměnná nebo kombinace konstant a proměnných spojených aritmetickými, relačními či logickými operátory tak, aby dávaly určitou hodnotu. Funkce operátorů bude popsána později. Výrazy mohou být rozděleny do dvou tříd:

1) aritmetické
2) řetězcové

Výrazy se obvykle skládají ze dvou nebo více členů, nazývaných operandy. Každý operand je oddělen jednoduchým operátorem tak, abychom dostali požadovaný výsledek. Obvykle se to dělá přiřazením hodnoty výrazu pojmenované proměnné. Všechny dosud uvedené příklady konstant a proměnných byly současně příklady výrazů. Operátor je speciální symbol, který BASIC Interpreter vašeho C64 pochopí jako pokyn k provedení určité operace s proměnnými nebo daty. Jeden či několik operátorů v kombinaci s jednou či více proměnnými a konstantami tvoří výraz. BASIC C64 rozeznává aritmetické, relační a logické operátory.

Aritmetické výrazy

Aritmetické výrazy po vypočítání dávají celá nebo reálná čísla. Aritmetické operátory (+, -, *, /, mocnina) představují sčítání, odčítání, násobení, dělení, a mocnění.

Aritmetické operátory

Aritmetický operátor definuje aritmetickou operaci, která se provádí na dvou operandech na obou stranách operátoru. Aritmetické operace používají reálných čísel. Celá čísla jsou převedena na reálná dříve, než se začne aritmetická operace provádět a výsledek se opět převede na celé číslo v případě, že je přiřazen celočíselné proměnné.

Sčítání (+)

Plusové znaménko (+) určuje, že operand na pravé straně je přičten k operandu vlevo.

Příklady:

2+2
A+B+C
X%+1

Odčítání (-)

Mínusové znaménko určuje, že operand na pravé straně je odečten od operandu vlevo.

Příklady: 4-1
100-64
A-B
55-142

Znaménka (-) může být také použito k označení záporného čísla. Je to totéž jako odečíst dané číslo od nuly.

Příklady: -5
-9E4
-B
4-(-2)

Násobení (*)

Hvězdička (*) určuje, že operand nalevo je násoben operandem vpravo.

Příklady: 100 * 2
50 * 0
A * X1
R% * 14

Dělení (/)

Dělící znaménko (/) určuje, že operand nalevo je dělen operandem vpravo.

Příklady: 10/2
6400/4
A/B
4E2/XR

Umocňování (šipka nahoru)

Šipka nahoru určuje, že operand nalevo je umocněn operandem vpravo (exponentem). Exponent může být libovolné číslo, pokud výsledek operace je platným reálným číslem.

Příklady: 2 (šip.nah.) 2 = 2*2
3 (šip.nah.) 4 = 3*3*3*3
AB (šip.nah.) CD
3 (šip.nah.) -2 = 1/(3*3)

Relační operátory

Relační operátory jsou především používány k porovnání hodnot dvou operandů, ale také mohou dávat aritmetický výsledek. Relační operátory a logické operátory (AND, OR, NOT = a, nebo, ne) užity pro porovnání vytvářejí aritmetické zhodnocení pravdivosti výrazu (true/false = pravdivý/nepravdivý). Je-li výraz pravdivý, je výsledku přiřazena celočíselná hodnota -1, je-li nepravdivý, je to 0. Relační operátory jsou tyto:

< menší než
= rovný
> větší než
<= menší nebo rovný
>= větší nebo rovný
>< nerovný

Příklady:

1 = 5-4 - pravdivý výsledek (-1)
14 < 66 - pravdivý výsledek (-1)
15 = 15 - pravdivý výsledek (-1)
6 > 83 - nepravdivý výsledek (0)

Relační operátory mohou být použity také pro porovnání řetězců. Pro tyto účely mají písmena abecedy pořadí A B C D atd. Řetězce jsou porovnávány tak, že jsou hodnoceny relace mezi odpovídajícími znaky odleva doprava (podrobněji v odstavci o řetězcích).

Příklady:

"A" < "B" - pravdivý výsledek (-1)
"X" = "XY" - nepravdivý výsledek (0)
BB$ >< CC$ - pravdivý (-1)

Numerická data mohou být porovnávána zase jenom s numerickými daty. Stejně řetězce mohou být porovnávány jen s řetězci, jinak se objeví chybové hlášení ?TYPE MISMATCH. Numerické operandy se při porovnávání převádějí na reálná čísla, pak se porovnají a výsledek je buď výrok "true" (pravdivý) (-1) nebo "false" (nepravdivý) (0).

Výsledkem porovnání je tedy vždy celé číslo, buď -1, nebo 0, ať jsou vstupní operandy jakékoliv, třeba i řetězce. Toto číslo může být použito v dalších krocích k čemukoliv, kromě dělení, protože dělení nulou není v matematice definováno.

Logické operátory

Logické operátory (AND, OR, NOT) mohou být použity pro modifikaci významu relačních operátorů nebo k získání aritmetického výsledku. Výsledek logické operace může být jiný než -1 nebo 0. Ale každý nenulový výsledek je považován za pravdivý při testování podmínek pravda/nepravda (true/false).

Logické operátory (někdy nazývané Booleovské operátory) mohou být také užity k provádění logických operací s jednotlivými binárními čísly (bity) dvou operandů, kromě operace NOT, která se týká jenom operandů vpravo od operátoru. Operandy musí být v oboru celých čísel mezi -32768 a 32767 (reálná čísla jsou převedena na celá) a logické operace dávají celočíselný výsledek.

Logické operace jsou prováděny na vzájemně si odpovídajících bitech obou operandů. Výsledek logické operace AND je pro daný bit log. 1, jen když jsou oba příslušné bity operandu rovny 1. Bitový výsledek logické operace OR je 1, jestliže alespoň jeden z odpovídajících bitů je roven 1. Logické NOT vytváří opačnou hodnotu každého bitu. Máme-li provést NOT 1 je výsledek 0 a naopak výsledek NOT 0 je 1.

Exclusive OR (XOR) nemá samostatný logický operátor, ale je součástí příkazu WAIT. Exclusive OR znamená, že jsou-li dva bity rovny, výsledek je 0. V opačném případě je to 1.

Logické operace jsou definovány skupinou příkazů, které jsou uvedeny v Booleanské "tabulce pravdivosti"; viz Tab. 1 - 2:

Tab. 1 - 2 Booleanská tabulka pravdivosti

Výsledkem operace AND je 1, pouze když oba bity jsou 1:

1 AND 1 = 1
0 AND 1 = 0
1 AND 0 = 0
0 AND 0 = 0

Výsledkem operace OR je 1, když alespoň jeden bit je 1:

1 OR 1 = 1

0 OR 1 = 1
1 OR 1 = 1
0 OR 0 = 0

Logické NOT vytváří opačnou logickou hodnotu:

NOT 1 = 0
NOT 0 = 1

Exclusive OR (XOR) je součástí operace WAIT a platí:

1 XOR 1 = 0
0 XOR 1 = 1
1 XOR 0 = 1
0 XOR 0 = 0

Logické operátory AND a OR určují Booleovské operace, které se provádějí na operandy, stojící po obou stranách operátoru; operátor NOT působí pouze na operand vpravo. Logické operace jsou prováděny až po skončení všech aritmetických a relačních operací ve výrazu.

Příklady:

IF A=100 AND B=100 THEN 10
jsou-li oba členy = 100, potom je výsledek pravdivý; program pokračuje na řádce 10

A=96 AND 32: PRINT A
výsledek: A = 32

IF A=100 OR B=100 THEN 20
výsledek je pravdivý (program pokračuje na řádce 20), je-li A nebo B rovno 100

A=64 OR 32: PRINT A
výsledek: A = 96

IF NOT X < Y THEN 30
výsledek je pravdivý pro X >= Y

X=NOT96
výsledek je -97 (dvojkový doplněk)

Hierarchie operací

Při provádění výrazů existuje pevná hierarchie operací, tj. určité operace jsou vždy prováděny dříve než jiné.

BASIC Interpreter provádí nejdříve aritmetické operace, pak relační a nakonec logické. Rovněž mezi aritmetickými operacemi existují pevná pravidla o nadřazenosti, totéž platí pro skupinu logických operací. Pouze relační operace žádné nadřazenosti nemají a provádějí se odleva doprava. Rovněž sejdou-li se operace stejného řádu (např. pět součtů za sebou), provádějí se odleva.

Hierarchie aritmetických a logických operací v pořadí od nejvyšší po nejnižší je uvedena v tabulce 1-3:

Tabulka 1 - 3 Hierarchie operací při provádění výrazů

Aritmetické operace:	umocňování
	negace (-A)
	násobení, dělení
	sčítání, odčítání
Relační operace:	větší než, menší než, rovný,
	větší nebo rovný, menší nebo rovný
Logické operace:	logické NOT
	logické AND
	logické OR

Někdy je potřeba tuto hierarchii operací změnit, což je možné závorkami. Uvnitř závorky se vytvoří "podvýraz", ten bude proveden jako první, a teprve na výslednou hodnotu se provádějí operace umístěné vně závorky.

Při používání závorek musíme dát pozor na to, abychom měli stejný počet levých (otvíracích) a pravých (zavíracích) závorek, jinak nám BASIC zahlásí chybu ?SYNTAX ERROR.

Výrazy s členy v závorkách mohou být samy umístěny do závorek. Může být použito až hloubky deseti úrovní dvojic závorek. Při výpočtu se nejdříve provádějí operace v nejvnitřnější závorce.

Operace s řetězci

Řetězce se porovnávají použitím stejných relačních operátorů (= = =), jako při porovnávání čísel.

Při porovnávání řetězců vezme počítač nejlevější znak od obou řetězců a vyhodnotí jeho číselnou pozici v tabulce znaku. Je-li kódová pozice obou porovnávaných znaků stejná, jsou si rovny. Znak s nižším kódovým číslem je menší. Následuje porovnání následujících dvou znaků až do konce jednoho řetězce.

V případě, že obsah obou řetězců je stejný, je za menší považován řetězec s menší délkou. V úvahu jsou brány i mezery před a za řetězcem.

Stringové výrazy

Stejně jako s čísly je možné provádět operace s řetězcovými proměnnými. Jediný aritmetický operátor, který CBM BASIC rozeznává při práci s řetězci, je znaménko (+) použité pro sečítání řetězců. Při této operaci je string vpravo od znaménka + připojen na konec stringu vlevo, a tím vznikne nový spojený řetězec. Výsledek může být okamžitě vytisknut, použit pro porovnávání nebo přiřazen ke jménu proměnné.

Je-li řetězec porovnáván s číselnou konstantou (proměnnou) nebo je přiřazen číselné proměnné, objeví se chyba ?TYPE MISMATCH.

Příklady řetězcových výrazů a jejich sečítání:

```
10 A$="FILE": B$="NAME"
20 NAM$ = A$ + B$ (dá řetězec: FILENAME)
```

Poznámka: Za slovem NEW je mezera!

PROGRAMOVACÍ TECHNIKY

Převod (konverze) dat

Je-li potřeba, převede BASIC Interpreter celočíselná data na reálná čísla a obráceně. Při tom platí následující pravidla:

Všechny aritmetické i relační operace jsou prováděny s reálnými čísly. Celá čísla jsou převedena na reálná, s nimi jsou provedeny výpočty a výsledek je převeden zpět na celé číslo. Logické operace převádějí své operandy na celá čísla a dávají celočíselný výsledek.

Je-li proměnná jednoho typu přiřazena číselné proměnné jiného typu, bude číslo převedeno a zapsáno jako typ podle deklarace nové proměnné.

Při převodu reálného čísla na celočíselné je desetinná část zanedbána. Celé číslo je proto menší či nejvýše rovno původnímu reálnému číslu. Je-li výsledek mimo obor -32768 až +32767, ukáže se chyba ?ILLEGAL QUANTITY.

Použití příkazu INPUT (vstup)

Nyní, když již víte, co jsou proměnné, využijeme tuto informaci spolu s příkazem INPUT pro praktické programové aplikace. V našem prvním příkladu můžeme pokládat jméno proměnné za identické s paměťovým místem, kam počítač uloží odpověď - jméno uživatele vašeho programu. V programu, který bude od uživatele žádat zapsání svého jména, zvolíme pro proměnnou, resp. paměťové místo, kam se toto jméno uloží, označení N$ (N=name; $ označuje znakovou proměnnou). Před začátkem psaní programu napište příkaz NEW a stlačte tlačítko RETURN.

```
10 PRINT "VASE JMENO:":INPUT N$
20 PRINT "DOBRY DEN, PANE "N$
```

Na spuštění vašeho programu použijete příkaz RUN. Z předcházejících částí manuálu si jistě vzpomínáte na rozdělení proměnných na dva základní typy:

1) číselné, numerické proměnné
2) znakové řetězce (stringy)

Pro jména číselných proměnných volíme kombinace písmen a číslic; na začátku označení musí vždy stát písmeno a jen dva první znaky slouží pro identifikaci v BASICU. Bude-li proměnná celočíselná, dostane jméno navíc označení %. Jméno znakového řetězce musí být zakončeno znakem dolar.

Několik příkladů označení proměnných a užití příkazu INPUT:

```
10 PRINT"VLOZ CISLO":INPUT A
20 PRINT A
10 PRINT"NAPIS SLOVO":INPUT A$
20 PRINT A$
10 PRINT"NAPIS CISLO":INPUT A
20 PRINT A;"NASOBENO 5 ROVNA SE ";A*5
10 PRINT"NAPIS DVE CISLA":INPUTA:INPUTB
20 C=A*B
30 PRINTA;" NASOBENO ";B;" ROVNA SE ";C
```

Při spuštění programu si jistě všimnete, že počítač vytiskne na obrazovce vše, co je mezi uvozovkami, ale jinak s tím nic nedělá. V případě, že některá zkratka nebo jméno se používá častěji, je výhodnější přiřadit mu jméno řetězce, než ho vždy vypisovat do uvozovek. Např.:

```
10 A$="KRAL KAREL IV"
20 B$="Kcs"
30 C$="$"
```

Užití příkazu GET

Nejjednodušší způsob zadání dat do počítače je prostřednictvím příkazu INPUT. Máme-li složitější potřeby, jako

ochrana proti zapsání chyby, je příkaz GET pružnější. Tento oddíl ukazuje užití příkazu GET ve spojení s úpravou textu na obrazovce.

Commodore 64 má klávesnicovou vyrovnávací paměť (keyboard buffer) o obsahu 10 znaků.

Vykonává-li počítač nějakou operaci, nečte klávesnici a my přesto můžeme připravit až 10 znaků do tohoto buffru, které počítač zpracuje po dokončení začaté operace. Pro ilustraci zkusme tento program:

```
NEW
10 TI$="000000"
20 IF TI$ "000015" THEN 20
```

Nyní napiš RUN, stiskni RETURN a zatímco program běží, napiš na klávesnici HELLO.

Nic se nestane po 15 sekund od odstartování programu. Poté se zpráva HELLO objeví na obrazovce. Zkusíme-li zapsat více znaků než 10, budou nadbytečné znaky ztraceny. Protože příkaz GET pracuje, i když žádný znak není stisknut, je nutno příkaz GET dát do cyklu, v němž program zůstane, dokud není stisknuta nějaká klávesa nebo dokud neobdrží znak z programu.

Předchozí program vymažeme příkazem NEW. Doporučený způsob použití příkazu GET je následující:

```
10 GET A$: IF A$="" THEN 10
```

Mezi uvozovkami není žádná mezera. To znamená prázdný znak; program se vrací zpět na řádek 10, dokud někdo nestiskne klávesu na počítači. Když je klávesa stisknuta, program pokračuje na následující řádce. Připojme k programu tuto řádku:

```
100 PRINT A$;:GOTO 10
```

Odstartujte program příkazem RUN. Na obrazovce se neobjeví žádný kurzor, ale zobrazí se znaky, odpovídající stisknuté klávese. Tento dvouřádkový program lze použít spolu se systémovým programem na úpravu textu na obrazovce (screen editor) v mnoha aplikacích.

Je možno programově odpojit některé klávesy, jako např. CLR/HOME (ochrana před náhodným vymazáním obrazovky), nebo využít funkční klávesy pro definici celých slov nebo frází.

Následující program dává každému funkčnímu klíči speciální význam. Tento program má mnoho využití.

```
20 IF A$=CHR$(133) THEN POKE 53280,8: GOTO 10
30 IF A$=CHR$(134) THEN POKE 53281,4:GOTO 10
40 IF A$=CHR$(135) THEN A$="DEAR SIR:"+CHR$(13)
50 IF A$=CHR$(136) THEN A$="SINCERELY,"+CHR$(13)
```

Argument stringové funkce pochází z tabulky CHR$ kódů uvedené v dodatku C.

Funkční klíče jsou definovány tak, aby představovaly instrukci za slovem THEN. Změnou argumentů u CHR$ mohou tuto instrukci vykonávat různé klíče. Rovněž informace za THEN může být libovolně měněna.

JAK ZMENŠIT PROSTOR, KTERÝ PROGRAM ZAUJÍMÁ V PAMĚTI

Proces zkracování programu je nazýván "crunching". Crunching umožňuje vložení co největšího počtu instrukcí do programu. Pomáhá redukovat velikost programů, které by při dané velikosti nemohly být prováděny; provádíme-li např. inventuru s mnoha položkami, zabírají data mnoho prostoru a potřebujeme program co nejkratší.

Užitím zkratek klíčových slov

Seznam zkratek klíčových slov je uveden v dodatku A. Jsou užitečné, neboť jejich prostřednictvím můžeme dát více informací na řádek. Nejužívanější zkratka je otazník (?), který BASIC interpretuje jako příkaz PRINT. Listujeme-li takový program, Commodore automaticky vytiskne klíčová slova v plné délce. Jestliže programová řádka potom přesáhne délku logické řádky, tj. 80 znaků

na řádek, takovou řádku není možno opravovat přímo. Chceme-li ji změnit, musíme ji přepsat celou ve zkratkách nebo v nezkrácené podobě ji rozdělit na dva řádky o různých číslech řádky. Program může být uschován (SAVE) se zkratkami, ale nesmí být předtím použit příkaz LIST, který dosazuje nezkrácené tvary příkazů a může způsobit překročení povolené délky řádku.

Zkrácením čísel programových řádků

Většina programátorů začíná program řádkem 100 a pokračuje po 10 (tj. následují řádky 110, 120, ..). To umožňuje během vývoje programu vkládat další řádky (111, 112, ...). Program můžeme zkrátit, jestliže po jeho dokončení užijeme co nejnižší čísla řádek (tj. 1, 2, 3 ..), neboť delší číslo řádky zabere více místa v paměti než kratší při užití v příkazech GOTO, GOSUB ap. Například číslo 100 zabírá 3 byty v paměti, číslo 1 pouze 1 byte.

Vkládáním více instrukcí na řádek

Na jeden programový řádek je možno vložit více instrukcí, oddělených dvojtečkou (:). Jediným omezením je, že všechny instrukce musí být na téže logické řádce; ta včetně dvojteček nesmí být delší než 80 znaků. Příklad dvou programů před a po zkracování:

Před zkrácením:

```
10 PRINT"HELLO",
20 FOR T=1 TO 500:NEXT
30 PRINT "HELLO,AGAIN..."
40 GOTO 10
```

Po zkrácení:

```
10 PRINT"HELLO...";:FORT=1TO500:NEXT:NEXT:
20 PRINT"HELLO, AGAIN...":GOTO10
```

Odstraněním příkazů REM

Příkaz REM je užitečný, aby připomněl nám nebo dalším programátorům, co která část programu dělá. Avšak když je

program dokončen, tyto poznámky už budou pravděpodobně zbytečné, proto je můžeme vypustit. Plánujeme-li program ještě v budoucnu přepracovávat či doplňovat, je užitečné si uschovat též kopii s poznámkami.

Užitím proměnných

Jestliže nějaké slovo, číslo nebo věta je opakovaně užita v programu, je výhodné jim přiřadit jistou proměnnou o délce jeden nebo dva znaky. Číslům může být přiřazena numerická proměnná, slovům a větám stringová proměnná se znakem dolaru na konci.
Následuje příklad:

Před zkracováním:	Po zkracování:
10 POKE 54296,15	10 V=54296:F=54273
20 POKE 54276,13	20 POKEV,15:POKE54276,13
30 POKE 54273,10	30 POKEF,10:POKEF,40:POKEF,70
40 POKE 54273,40	40 POKEV,0
50 POKE 54273,70	
60 POKE 54296,0	

Užitím příkazu READ a DATA

Data můžeme vkládat do programu buď jednu položku po druhé nebo užitím příkazu DATA, pomocí něhož můžeme vytvořit dlouhý seznam vstupních hodnot.

Užitím polí a matic

Pole a matice usnadňují a zkracují manipulaci s velkým množstvím dat. Pole mohou být vícedimenzionální.

Vyloučením mezer

Jeden z nejsnadnějších způsobů, jak redukovat program, je vyloučení mezer. Mezery jsou vkládány obvykle do programu kvůli přehlednosti; ve skutečnosti jsou zbytečné a jejich vyloučením ušetříme prostor v paměti.

Užitím příkazu GOSUB

Užíváme-li jistou část programu vícekrát, je užitečné ji zapsat pouze jednou, na konec umístit příkaz RETURN a vyvolávat ji příkazem GOSUB .

Užitím TAB a SPC

Chceme-li vytisknout informaci do jisté pozice na obrazovce, můžeme užít několik znaků pro pohyb kurzoru nebo užít funkce SPC a TAB, které jsou ekonomičtější.

II. SLOVNÍK JAZYKA BASIC

ÚVOD

Tento článek vysvětluje klíčová slova CBM BASICU. Nejprve je dán jejich seznam, zkratky klíčových slov a vzhled jednotlivých znaků na obrazovce.

Dále je vysvětlena detailně syntaxe a účinek jednotlivých klíčových slov, jsou dány příklady jejich použití v programu.

Commodore 64 BASIC povoluje zkratky většiny klíčových slov. Zkratky jsou voleny tak, aby byly jednoznačné pro jednotlivá klíčová slova a jejich poslední písmeno tvoří SHIFT plus znak.

Zkratky, užité v programu, nešetří prostor v paměti, neboť všechna klíčová slova jsou INTERPRETREM redukována na jednoznakové "tokens". Listujeme-li program, obsahující zkratky, jsou klíčová slova vypsána celá. Užitím zkratek můžeme vložit na řádek více příkazů, i když při jejich nezkráceném zapsání by převýšily 80 znaků na logický řádek. Ovšem SCREEN EDITOR pracuje pouze na 80 znakové řádce. Užijeme-li zkratky, potom řádku, která by při plném vypsání přesáhla 80 znaků, nebudeme moci opravovat. Takovou řádku musíme buď celou přepsat včetně zkratek, nebo ji rozdělit na dvě řádky s různými čísly řádky ap.

Kompletní seznam klíčových slov, jejich zkratek a vzhledu na obrazovce je v tabulce 2-1 originálu. Následuje abecední popis všech příkazů, povelů a funkcí použitelných pro Commodore 64.

Tento článek rovněž vysvětluje vnitřní funkce BASICU, vestavěné do BASIC Language Interpreteru. Vnitřní funkce je možno užít v přímém módu i uvnitř programu, aniž bychom je dále definovali. To ovšem neplatí pro uživatelské funkce. Výsledek vnitřní funkce může být užit jako výstup nebo může být přiřazen proměnné jistého typu a užit dále v programu. Existují dva typy BASIC funkcí:

1) numerické funkce
2) stringové funkce

Argument vnitřní funkce je vždy uzavřen v závorkách, přímo za klíčovým slovem. Mezi klíčovým slovem a závorkou nesmí být žádná mezera!

3. typ funkce je určen typem výsledku. Funkce, jejichž výsledkem je string, mají poslední znak klíčového slova znak dolaru ($). V některých případech stringová funkce obsahuje jeden nebo více číselných argumentů.

Numerická funkce provádí, je-li třeba, konverzi mezi reálnou a celočíselnou hodnotou. V následujícím popisu je dán typ argumentu i typ výsledku užité funkce.

POPIS KLÍČOVÝCH SLOV JAZYKA BASIC

A B S

Typ: numerická funkce
Formát: ABS (výraz)
Účinek: Výsledkem je absolutní hodnota čísla. Absolutní hodnota záporného čísla je toto číslo násobeno -1.
Příklady užití funkce ABS:

```
10 X=ABS(Y)
10 PRINT ABS(X J)
10 IF X=ABS(X) THEN PRINT "POSITIVE"
```

AND

Typ: operátor
Formát: (výraz) AND (výraz)
Účinek: AND je užito v booleanských operacích k testování bitů. Dále se užívá ke zjišťování pravdivosti obou operandů.

V booleanské algebře výsledek operace AND je 1 tehdy a jen tehdy, jsou-li obě čísla, na která je použit operátor AND, rovna 1. Výsledek je nulový, je-li alespoň jedno z čísel rovno nule.
Příklady na 1-bitovou operaci AND:

```
  0       1       0       1
AND 0   AND 0   AND 1   AND 1
-------  -------  -------  -------
  0       0       0       1
```

Commodore 64 provádí operaci AND na celá čísla v intervalu (-32768,+32767). Užití čísel mimo tento interval má za následek chybové hlášení ?ILLEGAL QUANTITY. Celé číslo v binární formě představuje 16 bitů. Provedeme- li operaci AND na odpovídající bity dvou celých čísel, je výsledek opět 16 bitové celé číslo s hodnotou v intervalu (-32768,+32767).

Vyhodnocujeme-li pravdivost či nepravdivost čísla, počítač předpokládá pravdivost, pokud není hodnota čísla nulová. Při porovnávání je hodnota -1 přiřazena pravdivému výsledku, hodnota 0 nepravdivému. V binárním tvaru -1 znamená, že všechny bity jsou rovny 1, a 0, že všechny bity jsou nulové. Výsledek je pravdivý, jestliže každý bit výsledku je pravdivý.
Příklad užití AND při zjišťování pravdivosti (nepravdivosti):

```
50 IF X=7 and W=3 THEN GOTO 10
```

ASC

Typ: numerická funkce
Formát: ASC(string)

Účinek: Po použití funkce ASC dostaneme číslo od 0 do 255, které odpovídá Commodore ASCII hodnotě prvního písmene ve stringu. Tabulka Commodore ASCII hodnot je uvedena v dodatku C originálního manuálu.
Příklady na použití ASC funkce:

```
10 PRINT ASC("Z")
20 X=ASC("ZEBRA")
30 J=ASC(J$)
```

Je-li J$ v příkazu číslo 30 nulový string, potom funkce ASC nepracuje a dostaneme chybové hlášení ?ILLEGAL QUANTITY. Chceme-li přečíst nulový string, musíme užít příkazy GET, resp. GET#. Abychom se vyhnuli případnému chybovému hlášení při čtení stringů, přičteme CHR$(0) na konec stringu.

Příklad na ASC funkci v případě, kdy se chceme vyhnout chybovému hlášení:

```
30 J=ASC(J$+CHR$(0))
```

ATN

Typ: numerická funkce
Formát: ATN(číslo)
Účinek: Výsledek je arcustangens čísla vyjádřený v úhlové míře (radiánech). Pohybuje se v intervalu (- /2,+ /2). Tangens výsledku je dané číslo.
Příklady použití funkce ATN:

```
10 PRINT ATN(0)
20 X=ATN(J) 180/PI: REM Převádí na stupně
```

CHR$

Typ: stringová funkce
Formát: CHR$(číslo)
Účinek: Tato funkce převádí Commodore ASCII kód na jeho písmenový ekvivalent. Seznam písmen a jejich kódů je uveden v

dodatku C. Číslo musí mít hodnotu v intervalu (0,255), jinak dostaneme chybové hlášení ?ILLEGAL QUANTITY.

Příklady užití CHR$ funkce:

```
10 PRINT CHR$(65): REM 65=velké písmeno A
20 A$=CHR$(13): REM 13=klíč RETURN
50 A=ASC(A$): A$=CHR$(A): REM převod do ASCII a zpět
```

CLOSE

Typ: příkaz vstupu a výstupu (I/O statement)

Formát: CLOSE (číslo souboru)

Účinek: Příkaz uzavírá datový soubor nebo kanál k přístroji. Číslo souboru musí být stejné jako při otevírání souboru, resp. kanálů (viz příkaz OPEN a oddíl týkající se programování vstup/výstup).

Pracujeme-li se zařízeními, na které je možno ukládat informace, jako je magnetická páska nebo disky, operace CLOSE ukládá všechny nekompletní vyrovnávací paměti (buffry). Není-li to provedeno, je soubor nekompletní na pásce a nečitelný na disku. CLOSE operace není povinná pro ostatní zařízení, ale uvolňuje paměť pro další soubory. Detaily lze nalézt v manuálu k příslušnému zařízení.

Příklady na příkaz CLOSE:

```
10 CLOSE 1
20 CLOSE X
30 CLOSE 9 (1+J)
```

CLR

Typ: příkaz

Formát: CLR

Účinek: Tento příkaz uvolňuje paměť RAM, která byla použita a není již dále potřeba. Program v BASICU, který je v paměti, zůstává nezměněn, ale všechny proměnné, pole, adresy GOSUB, smyčky FOR...NEXT, uživatelské funkce a soubory jsou

vymazány z paměti a jejich místo je uvolněno pro nové proměnné, atd.

Soubory na disku nebo magnetické pásce nejsou správně uzavřeny (CLOSE) příkazem CLR. Informace o souborech je ztracená pro počítač, včetně nekompletních buffrů. Disk drive se bude stále domnívat, že soubor je otevřen (OPEN). Další informace viz příkaz CLOSE.

Příklady užití příkazu CLR:

```
10 X=25
20 CLR
30 PRINT X
R U N
0
READY
```

CMD

Typ: příkaz vstupu a výstupu (I/O statement)

Formát: CMD(číslo souboru)(,string)

Účinek: Příkaz přepíná výstup od televizní obrazovky na zařízení definované číslem souboru. Tím může být disk, magnetická páska, tiskárna nebo jiné zařízení vstupu a výstupu (např. modem). Číslo souboru musí být specifikováno předem příkazem OPEN. Řetězec, je-li dán, je připojen k souboru, což je užitečné pro označení výpisu ap. Pokud tento příkaz není zrušen, každý příkaz PRINT a povel LIST je proveden ve stejném formátu nikoli na televizní obrazovku, ale na zařízení definované číslem souboru.

Chceme-li výstup zpět na televizní obrazovku, musíme nejprve zrušit příkaz CMD povelem PRINT#, potom uzavřít dané zařízení příkazem CLOSE, které tím přestane očekávat data.

Každá systémová chyba (jako ?ILLEGAL QUANTITY) způsobí návrat výstupu na televizní obrazovku. Přístroj tím však není

odpojen, je nutno poslat i v případě chybového hlášení prázdný řádek. (Další detaily viz manuál pro disk nebo tiskárnu.)

Příklady příkazu CMD:

```
OPEN 4,4:CMD 4,"TITLE":LIST:REM výpis programu (list)
    na tiskárnu
PRINT#4:CLOSE 4: REM odpojení a uzavření tiskárny
10 OPEN 1,1,1,"TEST": REM otevření sekv. souboru
20 CMD 1: REM výstup na mag. pásku, ne na obrazovku
30 FOR L=1 TO 100
40 PRINT L: REM vlož číslo do buffru pro magn. pásku
50 NEXT
60 PRINT# 1:REM odpojení
70 CLOSE 1: REM vypíše buffer a správně uzavře
```

CONT

Typ: povel

Formát: CONT

Účinek: Tento povel znovu spustí výpočet programu, který byl zastaven příkazy STOP nebo END nebo stisknutím klíče RUN/STOP. Program pokračuje z místa, na kterém byl zastaven.

Byl-li program zastaven, může uživatel zjišťovat nebo změnit hodnoty jakékoli proměnné nebo si prohlédnout program. Při výpočtu programu po částech umisťujeme příkaz STOP do výhodné pozice pro zjištění hodnot proměnných a pro kontrolu chodu programu.

Provedeme-li jakoukoliv opravu v programu po jeho zastavení (třeba jen stiskneme klíč RETURN s kurzorem na nezměněné řádce) dostaneme po použití povelu CONT chybové hlášení CAN'T CONTINUE. Stejné hlášení dostaneme, chceme-li pokračovat v programu, který se zastavil kvůli libovolné chybě nebo způsobil-li uživatel chybu před napsáním povelu CONT.

Příklady povelu CONT:

```
10 PI=0:C=1
```



```
20 PI=PI+4/C-4/(C+2)
30 PRINT PI
40 C=C+4:GOTO 20
```

Tento program počítá hodnotu PI. Odstartuj program příkazem RUN a po chvíli stiskni klíč RUN/STOP. Na displeji se objeví:

```
BREAK IN 20 (nebo jiné číslo)
```

Napiš povel PRINT C ke zjištění, kolik toho počítač spočetl. Dalším užitím povelu CONT počítač pokračuje z místa, kde byl zastaven.

COS

Typ: funkce

Formát: COS(číslo)

Účinek: Funkce počítá kosínus daného čísla; číslo je úhel v radiánech.

Příklady funkce COS:

```
10 PRINT COS(0)
20 X=COS(Y PI/180): REM převod stupňů na radiány
```

DATA

Typ: příkaz

Formát: DATA (seznam konstant)

Účinek: Příkaz DATA ukládá informaci uvnitř programu. Program využívá této informace prostřednictvím příkazu READ, který naplňuje příslušné proměnné konstantami z příkazu DATA.

Příkaz DATA není výkonný příkaz v programu, musí být pouze přítomen. Obvykle bývá umístěn na konec programu. Všechny příkazy DATA v programu vytvářejí spojitý seznam. Data jsou čtena příkazem READ zleva doprava, od nejnižšího čísla řádku k nejvyššímu. Jestliže příkaz READ nalezne data, která neodpovídají

typem požadované proměnné (požaduje číslo, nalezne string), obdržíme chybové hlášení.

Data může tvořit libovolný znak. Některé však musí být uzavřeny v uvozovkách (" "). Jsou to především interpunkční znaménka čárka (,), dvojtečka (:), mezera, písmena zapsaná s použitím kliče SHIFT (shiftovaná písmena), grafické znaky, znaky pro řízení kurzoru.

Příklady příkazu DATA:

```
10 DATA 1,10,5,8
20 DATA JOHN,PAUL,GEORGE,RINGO
30 DATA "DEAR MARRY,HOW ARE YOU, LOVE, BILL"
40 DATA -1.7E-9,3.33
```

DEFFN

Typ: příkaz

Formát: DEF FN (jméno) (proměnná) = (výraz)

Účinek: Definuje uživatelskou funkci, která může později být užita v programu. Uživatelská funkce šetří místo v programech, kde by byl týž dlouhý vzorec užit vícekrát na různých místech. Vzorec je specifikován pouze jednou, dále je užit pouze jako jméno funkce.

Jméno funkce tvoří písmena FN následovaná jakoukoliv pojmenovanou proměnnou. Ta se může skládat z jednoho až dvou znaků, z nichž první musí být písmeno a druhý písmeno nebo číslice.

Příklady příkazu DEF FN:

```
10 DEF FN A(X)=X+7
20 DEF FN AA(X)=YZ
30 DEF FNA9(Q)=INT(RND(1) Q+1)
```

Funkce je vyvolána v programu užitím jména funkce s proměnnou v závorce. Toto jméno funkce je užito stejně jako jiná proměnná a její hodnota je spočtena automaticky.

Příklady užití funkce FN:

```
40 PRINT FN A(9)
```



```
50 R=FNAA(9)
60 G=F+FN A9(10)
```

V řádce 50 číslo 9 v závorce neovlivňuje hodnotu funkce, neboť v řádce 20, která definuje funkci AA, není užita proměnná v závorce. Výsledek je Y Z, bez ohledu na hodnotu X. V obou dalších příkladech hodnota v závorce ovlivňuje výsledek.

DIM

Typ: příkaz

Formát: DIM ((proměnná) (index),(proměnná)(index)...)

Účinek: Příkaz definuje pole nebo matici proměnných. To umožňuje užít pojmenovanou proměnnou s indexem. Index označuje, kolikátý prvek bude užit. Nejnižší prvek v poli má index 0, nejvyšší má index daný číslem v příkazu DIM. Jeho maximální hodnota je 32767.

Příkaz DIM je prováděn pouze jednou pro každé pole. Je-li řádka s příkazem DIM znovu prováděna, objeví se chyba REDIM'D ARRAY.

Příkaz DIM může mít až 255 indexů libovolné velikosti do 32767. Omezení ovšem tvoří množství paměti RAM vyhrazené pro proměnné. Pole mohou vytvářet běžné numerické proměnné, stringy nebo celá čísla. Stringové pole je označeno znakem $, celočíselné pole znakem % za jménem pole. Použijeme-li v programu prvek pole, jehož velikost nebyla definována příkazem DIM, předpokládá se automaticky po jeho prvním použití, že pole má 11 prvků.

Příklady na užití příkazů DIM:

```
10 DIM A (100)
20 DIM Z (5,7)
```

Výpočet paměti při užití příkazu DIM:

```
5  bytů pro jméno pole
2  byty pro každou dimenzi
2  byty / prvek celočíselné proměnné
```

5 bytů / prvek běžné numerické proměnné
3 byty / prvek stringové proměnné
1 byte pro každý znak v každém prvku stringu

END

Typ: -říkaz
Formát: END
Účinek: Příkaz ukončuje výpočet. Na obrazovce se po jeho provedení objeví zpráva READY a řízení počítače je předáno uživateli. Program může obsahovat větší množství příkazů END. Příkaz END není povinný, ale pouze doporučený. Příkaz END je podobný příkazu STOP. Rozdíl je pouze v tom, že po provedení příkazu STOP se na displeji objeví zpráva BREAK IN LINE XX, zatímco po provedení příkazu END zpráva READY. Po obou příkazech je povoleno pokračovat ve výpočtu užitím povelu CONT.
Příklady příkazu END:

```
10 PRINT "CHCES UKONCIT PROGRAM ?"
20 INPUT A$
30 IF A$="ANO" THEN END
40 REM ZBYTEK PROGRAMU...
999 END
```

EXP

Typ: numerická funkce
Formát: EXP(číslo)
Účinek: Tato matematická funkce počítá konstantu e (2.71828183) umocněnou na dané číslo. Hodnota větší než 88.0296919 způsobí chybové hlášení ?OVERFLOW.
Příklady funkce EXP:

```
10 Print EXP(1)
20 X=Y EXP(Z Q)
```


FN

Typ: numerická funkce
Formát: FN (jméno) (číslo)
Účinek: Tato funkce se vztahuje k dříve definované formuli DEF. V programu je její vyvolání nahrazeno číslem po předchozím vyčíslení jejího definičního vzorce určeného příkazem DEF FN.

Vyskytne-li se FN v programu dříve než její definice příkazem DEF FN, obdržíme chybové hlášení UNDEF'D FUNCTION.
Příklady uživatelem definované funkce FN:

```
PRINT FN A(Q)
1100 J=FN J(7)+FN J(9)
9900 IF FN B7(I+1)=6 THEN END
```

FOR ... TO ... (STEP...)

Typ: příkaz
Formát: FOR (proměnná)=(start) TO (limit) STEP (přírůstek)
Účinek: Toto je speciální BASIC příkaz, který umožňuje lehce užít proměnnou jako počítadlo. Je nutno specifikovat jisté parametry: reálnou pojmenovanou proměnnou, její počáteční a konečnou hodnotu a kolik máme přičíst v každém cyklu.

Příklad jednoduchého BASIC programu, v němž se hodnota proměnné L mění od 1 do 10, vytiskne se každé číslo a program se zakončí END:

```
100  L=1
110  PRINT L
120  L=L+1
130  IF L=10 THEN 110
140  END
```

Nyní stejná úloha s užitím příkazu FOR...:

```
100  FOR L=1 TO 10
110  PRINT L
```

```
120  NEXT L
130  END
```

Jak vidíte, program je kratší a snáže pochopitelný. Provádění příkazu FOR se skládá z několika operací. Počáteční hodnota "start" je umístěna do "proměnné" cyklu. V předchozím příkladu je do proměnné cyklu L umístěna hodnota 1.

Když se dostaneme k provádění příkazu NEXT, je hodnota "přírůstek" přičtena k "proměnné". Není-li STEP uveden, předpokládá se roven +1. Když se program prvně dostane na řádku 100, je hodnota 1 přičtena k proměnné cyklu L, takže nová hodnota L je 2.

Nyní je hodnota (proměnné) porovnána s hodnotou (limit). Jestliže hodnoty "limit" nebylo dosud dosaženo, program se vrací na řádek za příkazem FOR. V našem případě hodnota 2 proměnné cyklu L je menší než limit 10, proto se program vrací na řádek 110.

V případě, že hodnota "proměnné" je větší než "limit", smyčka je ukončena a program pokračuje na řádce následující příkaz NEXT.

V našem případě, když hodnota L je 11, což je vetší než limit 10, program pokračuje řádkou 130.

Poznámka: Smyčka se provádí vždy alespoň jednou.

Příklady FOR...TO...STEP... příkazu:

```
100  FOR L= 100 TO 0 STEP -1
100  FOR L=PI TO 6 STEP .01
100  FOR AA=3 TO 3
```

FRE

Typ: funkce

Formát: FRE(proměnná)

Účinek: Funkce nám umožňuje zjistit, jaké množství paměti RAM je k dispozici pro náš program a proměnné. Použije-li program více paměti, než je k dispozici, dostaneme chybové hlášení OUT OF MEMORY.

Číslo v závorce může nabývat libovolné hodnoty a není užito pro výpočet.

Poznámka: Je-li výsledek funkce FRE záporný, přičteme k němu 65536 a dostaneme počet volných bytů v paměti. Příklady funkce FRE:

```
PRINT FRE (0)
PRINT FRE (0) -(FRE(0)) 65536 (zobrazí okamžitou velikost volné paměti)
```

GET

Typ: příkaz

Formát: GET (seznam proměnných)

Účinek: Příkaz čte každý klíč (znak) stisknutý na klávesnici uživatelem. Znaky jsou ukládány do Commodore 64 klávesnicové vyrovnávací paměti (keyboard buffer). Zde může být uloženo až 10 znaků, stisknutí dalších znaků znamená ztrátu znaku. Přečtením znaků příkazem GET se vytvoří volný prostor pro další znak.

Je-li proměnná v příkazu GET numerická, potom při zaslání jiného typu proměnné obdržíme chybové hlášení ?SYNTAX ERROR. Abychom se tomu vyhnuli, je výhodné číst všechny znaky jako řetězce a převést je na čísla později.

Příkaz GET nám pomáhá vyhnout se některým omezením příkazu INPUT. Podrobnosti viz oddíl o využití příkazu GET v programovací technice.

Příklady příkazu GET:

```
10 GET A$:IF A$="" THEN 10:REM smyčka na řádku 10 do stisknutí klíče
20 GET A$,B$,C$,D$,E$: REM čte 5 klíčů
30 GET A,A$
```

GET#

Typ: příkaz vstupu/výstupu
Formát: GET# (číslo souboru),(seznam proměnných)
Účinek: Příkaz čte po jednom znaky ze specifikovaného zařízení. Pracuje stejně jako příkaz GET, pouze data jsou čtena z jiného vstupního zařízení než klávesnice. Jestliže žádný znak není obdržen, proměnná je položena rovno prázdnému stringu ("") nebo v případě numerické proměnné 0. Znaky, užívané k oddělování dat, jako čárka (,) nebo RETURN (jeho ASCII kód je 13), obdržíme stejné jako každý jiný znak.

Při práci s TV obrazovkou (zařízení #3) příkaz čte po jednom znaky z obrazovky. Každé užití příkazu GET# posune kurzor o jednu pozici vpravo.

Znak na konci logické řádky je změněn na CHR$(13) = RETURN.

Příklady na příkaz GET#:

```
1  GET# 1,A$
10 OPEN 1,3:GET#1,Z7$
20 GET#1,A,B,C$,D$
```

GOSUB

TYP: příkaz
Formát: GOSUB (číslo řádku)
Účinek: Příkaz je zvláštním tvarem příkazu GOTO s tím rozdílem, že si počítač pamatuje, odkud přišel. Po provedení příkazu RETURN (pozor, tento nemá nic společného s klávesou RETURN!) se program vrací na příkaz, následující za příkazem GOSUB.

GOSUB je zkratka pro GO TO a SUBROUTINE (skoč do podprogramu). Využíváme ho tam, kde se jistá část výpočtu v programu ma vícekrát opakovat, takže tyto řádky nemusíme stále na různých místech opisovat a šetříme prostor v paměti. Příkaz

GOSUB je podobný příkazu DEF FN. DEF FN šetří prostor využitím vzorce pro několikanásobný výpočet, GOSUB využitím několikařádkového podprogramu.

Při každém provedení příkazu GOSUB je číslo řádku a pozice programu na řádce uschována ve speciální oblasti nazývané "stack", která zabírá v paměti 256 bytů. To omezuje množství dat uložených ve stacku. Počet návratových adres podprogramu je tedy omezen. Je nutno dbát, aby každý podprogram byl zakončen příkazem RETURN, jinak je možno se dostat mimo paměť, i když je v paměti spousta volného místa.

GOTO

Typ: příkaz
Formát: GOTO (číslo řádky) nebo GO TO (číslo řádky)
Účinek: Příkaz umožňuje, aby program byl počítán v jiném sledu než v numerickém pořadí řádek. Slovo GOTO následované číslem řádky způsobí, že program pokračuje ve výpočtu na řádce dané "číslem řádky". GOTO bez čísla řádky je totež jako GOTO 0, tedy číslo řádky musí následovat.

Pomocí GOTO je možno vytvořit nekonečnou smyčku.
Nejjednodušší její příklad je:

```
10 GOTO 10
```

Tato smyčka může být přerušena pouze stiskem klíče RUN/STOP.

Příklady příkazu GOTO:

```
GOTO 100
10 GOTO 50
20 GOTO999
```

IF ... THEN ...

Typ: příkaz

Formát: IF (výraz) THEN (číslo řádky)
IF (výraz) GOTO (číslo řádky)
IF (výraz) THEN (příkaz)

Účinek: Tento příkaz dává BASICU jistou inteligenci; dovoluje vyhodnotit kladené podmínky a podle nich zajistit vhodný další běh programu.

Slovo IF je následováno výrazem, který může zahrnovat proměnné, stringy, čísla, porovnání, logické operátory. Slovo THEN je na stejné programové řádce a je následováno buď číslem řádky nebo jedním či více BASIC příkazy. Je-li výraz nepravdivý, pak vše za slovem THEN na téže programové řádce je ignorováno a program pokračuje ve výpočtu na další řádce. Je-li výraz pravdivý, program buď provede příkaz následující hned za slovem THEN, nebo se větví a pokračuje ve výpočtu na řádce dané "číslem řádky".

Příklad příkazu IF...GO....:

```
100  INPUT "TYPE A NUMBER";N
110  IF N = 0 GOTO 200
120  PRINT "DRUHA MOCNINA =";SQR(N)
130  GOTO 100
200  PRINT "CISLO MUSI BYT VETSI NEZ 0"
210  GO TO 100
```

Tento program napíše druhou odmocninu každého kladného čísla. Příkaz IF je zde užit ke zhodnocení čísla daného příkazem INPUT. Je-li N=0, program skočí na řádek 200 (pravdivý výsledek porovnání), jinak pokračuje na řádce 120. Zápis GOTO 200 ve skutečnosti znamená THEN GOTO 200.

Příklad IF...THEN... příkazu:

```
100  FOR L=1 TO 100
110  IF RND(1) .5 THEN X=X+1: GOTO130
120  Y=Y+1
130  NEXT L
140  PRINT "HEADS=" X
150  PRINT "TAILS=" Y
```


V řádce 110 je testováno náhodné číslo, zda je menší než .5. Když je výsledek pravdivý, pokračuje výpočet řadou příkazů za slovem THEN; X se zvýší o 1 a program skočí na řádek 130. Při nepravdivém výsledku program pokračuje řádkou 120.

INPUT

Typ: příkaz

Formát: INPUT "text"; (seznam proměnných)

Účinek: Slouží k předání vstupních informací do počítače. Když je prováděn, příkaz vytiskne na obrazovce otazník (?) a posune kurzor 1 mezeru vpravo od otazníku. Potom počítač čeká s blikajícím kurzorem na odpověď operátora a stisknutí klíče RETURN.

Slovo INPUT může být doprovázeno vysvětlujícím textem uzavřeným v uvozovkách. Ten je pak vytištěn na obrazovku před otazníkem.

Za textem následuje středník (;) a jména jedné nebo několika proměnných, oddělených čárkami. Do těchto proměnných počítač ukládá informace, zapsané operátorem na obrazovku. Proměnné mohou mít libovolné povolené jméno.

Příklady příkazu INPUT:

```
100  INPUT A
110  INPUT B,C,D
120  INPUT "NAPOVIDAJICI TEXT";E
```

Po spuštění programu příkazem RUN se objeví na obrazovce otazník, oznamující, že počítač očekává vstup pro řádek 100. Zapíšeme nějaké číslo do A pro další užití programu. Jestliže odeslané znaky nebyly číslo, dostaneme chybové hlášení ?REDO FROM START, což znamená, že počítač dostal string místo čísla. Zmáčkne-li operátor pouze klíč RETURN bez zapsání čehokoliv, zůstává hodnota proměnné nezměněna. Nyní se na obrazovce objeví otazník pro řádek 110.

Zapíšeme-li na obrazovku pouze jedno číslo a stiskneme RETURN, objeví se na obrazovce dva otazníky, oznamující, že Commodore 64 požaduje další vstup, na což počítači odpovíme příslušným vstupem. Nebo je možno hned zapsat příslušný počet hodnot vstupu, oddělených čárkou. Zapíšeme-li větší počet hodnot než je požadováno, dostaneme zprávu ?EXTRA IGNORED, znamenající, že hodnoty navíc nejsou vloženy do žádné proměnné.

Při provádění řádky 120 se na obrazovce zobrazí napovídající text s otazníkem.

Input příkaz nesmí být užit mimo program, neboť Commodore 64 potřebuje prostor pro buffer vstupních proměnných.

INPUT#

Typ: Příkaz vstup/výstup
Formát: INPUT# (číslo souboru), (seznam proměnných)
Účinek: Toto je nejrychlejší a nejsnadnější způsob, jak nahrát data uložená na disku nebo magnetické pásce do počítače.

Data jsou získávána ve tvaru celých proměnných až do délky 80 znaků na rozdíl od příkazu GET#, kdy jsou získávána po jednom znaku. Soubor musí být nejprve otevřen (příkaz OPEN) a potom příkazem INPUT# jsou naplněny jednotlivé proměnné. Příkaz INPUT# předpokládá, že proměnná je naplněna, když přečte kód RETURN (CHR$(13)), čárku (,), středník (;) nebo dvojtečku (:). Jsou-li tyto znaky požadovány pro vstup, musí být uzavřeny v uvozovkách (viz příkaz PRINT#). Přečte-li počítač stringovou proměnnou v místě, kde očekává numerickou, zobrazí chybové hlášení BAD DATA. Příkaz INPUT# může číst stringy dlouhé až 80 znaků, delší způsobí chybové hlášení STRING TOO LONG.

Je-li vstupní zařízení #3 (televizní obrazovka), příkaz čte celý logický řádek a posune kurzor na další řádek.

Příklady příkazu INPUT#:

```
10 INPUT#1,A
20 INPUT#2,A$,B$
```


INT

Typ: celočíselná funkce
Formát: INT (číslo, resp. číselný výraz)
Účinek: Navrací celočíselnou část výrazu. Je-li výraz kladný, je zlomková část odříznuta. Je-li výraz záporný se zlomkovou částí, je výsledek nejbližší nižší celé číslo.

Příklady funkce INT:

```
120  PRINT INT(99.4343),INT(-12.34)
                99          -13
```

LEFT$

Typ: stringová funkce
Formát: LEFT$ (string, celé číslo)
Účinek: Funkce vyjme ze stringu zleva počet znaků daných "celým číslem". To musí být v intervalu (0,255). Je-li "celé číslo" větší než počet znaků ve stringu, navrací funkce celý string. Je-li "celé číslo" rovno nule, obdržíme nulový string (string délky 0).

Příklady funkce LEFT$:

```
10 A$="COMMODORE COMPUTERS"
20 B$= LEFT$(A$,9):PRINT B$
RUN
COMMODORE
```

LEN

Typ: celočíselná funkce
Formát: LEN (string)
Účinek: Výsledkem je počet znaků ve stringu (délka stringu). Jsou počítány i netištěné znaky a mezery.

Příklad funkce LEN:

```
CC$="COMMODORE COMPUTER":PRINT LEN (CC$)
18
```

LET

Typ: příkaz
Formát: (LET)"proměnná"="výraz"
Účinek: Příkaz je užíván pro přiřazení hodnoty proměnné. Slovo LET není povinné, proto je zkušenější programátoři vynechávají, aby nezabíral prostor v paměti. Znaménko rovnosti (=) dostatečně naznačuje, že proměnné je přiřazena jistá hodnota nebo výraz.
Příklad příkazu LET:

```
10 LET D=12 (totéž jako D=12)
20 LET E$="ABC"
30 F$="WORDS"
40 SUM$=E$+F$ (SUM$ bude rovno ABCWORDS)
```

LIST

Typ: povel
Formát: LIST (první řádka) - (poslední řádka)
Účinek: Povel LIST umožňuje prohlédnutí řádek programu, který je uložen v paměti počítače. Tak je možno využít editor počítače a opravovat snadno řádky programu, který jsme právě Listovali.

Systémový povel LIST dovoluje vypsat celý program, který je právě v paměti, nebo jeho část na příslušné zařízení. Neuvedeme-li jiné zařízení, vypíše se LIST programu na TV obrazovku. Chceme-li použít pro LIST jiné zařízení (tiskárnu, disk ap.), musíme je nadefinovat příkazem CMD. Povel LIST se může vyskytnout i na libovolné řádce programu, ale BASIC vždy po jeho zpracování vrací systémové hlášení READY.

Listujeme-li program na obrazovce, odvíjí se jednotlivé řádky. Jejich pohyb můžeme zpomalit stisknutím klíče CTRL. LIST je přerušen po stisknutí klíče RUN/STOP.

Nejsou-li dána žádná čísla řádek, je Listován celý program. Zadáme-li jen první číslo řádky a pomlčku (-), je vylistovaná pouze

daná řádka a řádky s vyšším číslem. Zadáme-li pomlčku (-) a číslo řádky, zobrazí se řádky od začátku až do dané řádky. Zadáme-li obě čísla, zobrazí se řádky v daném intervalu včetně těchto řádek.
Příklady povelu LIST:

```
LIST (je Listován celý program v paměti)
LIST 500 (LIST řádky 500)
LIST 150- (LIST řádek od 150 do konce programu)
LIST -1000 (LIST od začátku do řádky 1000 včetně)
LIST 150-1000 (LIST od řádky 150 do 1000 včetně)
10 PRINT "THIS IS LINE 10"
20 LIST (LIST je užit v programu)
30 PRINT "TOTO JE RADKA 30"
```

LOAD

Typ: povel
Formát: LOAD "jméno souboru" ,(zařízení) ,(adresa)
Účinek: Příkaz načte programový soubor (program) z disku nebo magnetické pásky do paměti. Ten je pak možno přímo použít nebo jej změnit. Číslo zařízení není povinné, ale není-li uvedeno, počítač automaticky předpokládá, že je rovno 1, tj. kazetový magnetofon. Disk má číslo zařízení rovno 8. Load uzavírá všechny soubory a je-li užit v přímém módu, provádí rovněž CLR před načtením programu. Je-li LOAD užit uvnitř programu, je loadovaný program hned spuštěn. To znamená, že příkaz LOAD může být užit ke zřetězení několika programů dohromady. Během této operace se nemění hodnoty proměnných.

Použijeme-li "jméno souboru", je načten první soubor tohoto jména, který počítač najde. Hvězdička v uvozovkách ("*") jako "jméno souboru" způsobí, že je načten soubor, který je v adresáři disku (disc directory) uveden první. Nenalezne-li počítač "jméno souboru" nebo není-li daný soubor programovým souborem, dostaneme BASIC chybové hlášení ?FILE NOT FOUND.

Při zavádění programu z magnetické pásky může být "jméno souboru" vynecháno a je zaveden nejbližší program na pásce. Po stisknutí tlačítka PLAY má obrazovka barvu rámečku (border colour). Po nalezení programu získá obrazovka původní barvu (background colour) a zobrazí se zpráva FOUND. Stiskneme-li klávesu C=, CTRL nebo mezeru, soubor bude zaveden do paměti (naloadován), a to od adresy 2048, pokud nepoužijeme sekundární adresu 1 "adresa". Je-li sekundární "adresa" rovna 1, bude program zaveden do paměti od adresy, ze které byl uschován (SAVE).

Příklady užití LOAD:

```
LOAD (čte nejbližší program na pásce)
LOAD A$ (hledá program jména A$)
LOAD"*",8 (načte první program z disku)
LOAD"*",1,1 (vyhledá první program na kazetě a umístí ho
    do stejné části paměti, ze které byl nahrán)
LOAD "STAR TREK" (načte soubor z magnetické pásky)
PRESS PLAY ON TAPE
FOUND STAR TREK
LOADING
READY.
LOAD "FUN",8 (načte soubor z disku)
SEARCHING FOR FUN
LOADING
READY.
LOAD"GAME ONE",8,1 (soubor je načten z d. do stejné
    oblasti paměti, ze které byl uschován - SAVE)
SEARCHING FOR GAME ONE
LOADING
READY.
```

LOG

Typ: reálná funkce

Formát: LOG(číslo nebo číselný výraz)

Účinek: Počítá přirozený logaritmus (logaritmus o základě argumentu). Je-li hodnota argumentu nulová nebo záporná, dostaneme BASIC chybové hlášení ?ILLEGAL QUANTITY.

Příklady funkce LOG:

```
25 PRINT LOG (45/7)
1.86075234
10 NUM=LOG(ARG)/LOG(10) (počítá dekadický logaritmus
    argumentu)
```

MID$

Typ: stringová funkce

Formát: MID$(string,číselný výraz 1, (číselný výraz 2))

Účinek: Funkce vrací část řetězce vybraného z původního většího.

Začátek výběru je určen hodnotou číselného výrazu 1 a délka vybraného řetězce číselným výrazem 2. Oba výrazy mohou mít hodnoty od 0 do 255.

Jestli je hodnota 1. výrazu větší než délka původního řetězce nebo hodnota 2. výrazu je nula, potom dá funkce nulový řetězec.

Jestli je 2. výraz vynechán, počítač určí konec nového řetězce shodně s koncem původního řetězce. V případě, že základní řetězec má od určené počáteční hodnoty pro vybírání do svého konce méně znaků než udává 2. výraz, přebere se celý zbytek základního řetězce. Příklady funkce MID$:

```
10 A$="GOOD"
20 B$="MORNING EVENING AFTERNOON"
30 PRINT A$ + MID$(B$,8,8) výsledek: GOOD EVENING
```

NEW

Typ: funkce

Formát: NEW

Účinek: Slouží k vymazání programu, který je právě v paměti, a k vynulování obsahu všech proměnných. Povel NEW v přímém módu by měl být užit vždy, než začneme zapisovat nový program. New může být užit též v programu; pak ale vymaže vše, co bylo dosud v paměti počítače. To může způsobit potíže při odladování programu.

Pozor: Nenapíšeme-li NEW před zapsáním nového programu, můžeme dostat směs obou programů! Příklady povelu NEW:

```
NEW (vymaže program a všechny proměnné)
10 NEW (provede operaci NEW a STOP - zastaví program)
```

NEXT

Typ: příkaz

Formát: NEXT (proměnná cyklu),(proměnná cyklu)...

Účinek: Příkaz NEXT je užíván k ukončení smyčky FOR...NEXT. Fyzicky NEXT nemusí být posledním příkazem smyčky, ale je to poslední prováděný příkaz smyčky. Proměnná cyklu je název indexu v příkazu FOR. Jediný příkaz NEXT může ukončit několik cyklů, následuje-li za ním výčet proměnných cyklů (oddělených čárkami), které mají být uzavřeny. Proměnné cyklů musí zachovávat jisté pořadí. Nejvnitřnější cyklus musí být uzavřen nejdříve, čili jeho proměnná bude hned za slovem NEXT, zatímco proměnná cyklů nejvíce vně bude poslední. Jinými slovy: cykly se nesmějí křížit. Počet cyklů v sobě může mít až 9 úrovní. Je-li proměnná cyklů u NEXT vynechána, je zvýšená v příkazu FOR ta proměnná, která odpovídá pravidlům o seskupování cyklů (cykly se nesmějí křížit).

Po provedení příkazu NEXT je hodnota proměnné cyklů zvýšena o 1 nebo o hodnotu STEP. Potom je provedeno porovnání proměnné cyklů s konečnou hodnotou cyklů. Je-li proměnná cyklů vyšší, je cyklus ukončen.

Příklady příkazu NEXT:

```
10 FOR J=1 TO 5: FOR K=10 TO 20:FOR N=5 TO-5 STEP -1
```



```
20 NEXT N,K,J (ukončení seskupení cyklů)
10 FOR L= 1 TO 100
20 FOR M=1 TO 10
30 NEXT M
400 NEXT L (smyčky se nekříží!!)
10 FOR A=1 TO 10
20 FOR B=1 TO 20
30 NEXT
40 NEXT (použito bez jmen proměnných cyklů)
```

NOT

Typ: logický operátor

Formát: NOT "výraz"

Účinek: Logický operátor NOT provádí doplněk (NOT1=0,NOT0=1) každého bitu jako samostatného operandu; výsledek operace je celočíselný dvojkový doplněk (twos-complement; vysvětlení viz Booleovská algebra a poznámka na konci). NOT říká "jestliže není...". Pracujeme-li s čísly v pohyblivé desetinné tečce, je operand převeden na celé číslo a desetinná část je ztracena. Operátor NOT může být užit rovněž při porovnávání k negaci hodnoty (pravdivé či nepravdivé), čímž získáme opačný význam hodnoty. V následujícím příkladě "twos-complement" čísla "AA" je číslo "BB"; jestliže "BB" není rovno "CC", potom výraz je pravdivý.

Příklady operátoru NOT:

```
10 IF NOT AA=BB AND NOT(BB=CC) THEN...
NN%=NOT96: PRINT NN%
```

Poznámka: K nalezení hodnoty NOT užij vzorce X=(-(X+1)). (Twos complement celého čísla je jeho bitový doplněk plus jedna.)

ON

Typ: příkaz

Formát: ON "proměnná" GOTO/GOSUB "číslo řádky",(číslo řádky)...

Účinek: Provedení příkazu má stejný účinek jako příkaz GOTO. Příkaz slouží podobně jako klíč. Podle hodnoty "proměnné" příkaz skočí na číslo řádky v seznamu, jehož pořadí je dané "proměnnou". Např. je-li "proměnná"=3, příkaz skočí na číslo řádky, které je v seznamu na třetím místě.

"Proměnná" může nabývat hodnot od nuly až do počtu řádek programu. Je-li záporná, dostaneme chybové hlášení ?ILLEGAL QUANTITY. Je-li nulová nebo větší než počet řádek programu, je příkaz ignorován, program pokračuje ve výpočtu na další řádce.

Příkaz ON je jistou variantou příkazu IF..THEN... Místo abychom užili celé řady příkazů IF, užijeme jediný příkaz ON.

V následujícím příkazu jeden příkaz ON nahrazuje čtyři IF...

Příklady příkazu ON:

```
ON-(A-7)-2 (A+3)-3 (A*3)-4 (A/7)GOTO 400,900,1000,100
ON X GOTO 100,130,180,220
ON X+3 GOSUB 9000,20,9000
100  ON NUM GOTO 150,300,320,390
500  ON SUM/2+1 GOSUB 50,80,20
```

OPEN

Typ: příkaz vstup/výstup

Formát: OPEN "číslo souboru", (zařízení), (adresa), (jméno souboru), (typ), (mód)

Účinek: Příkaz otevírá kanál pro vstupní a výstupní periférie. Ne vždy jsou třeba všechny části příkazu OPEN. Některé požadují pouze dva kódy:

1) logické číslo souboru
2) číslo zařízení

"Číslo souboru" je logické číslo souboru, které se vztahuje k příkazům OPEN, CLOSE, CMD, GET#, INPUT#, PRINT#. Je spojeno se jménem souboru a s typem užitého zařízení. Logické číslo souboru může být libovolné číslo z intervalu (1,255).

Poznámka: Často číslo souboru větší než 128 bývá užito k jiným účelům, proto je lépe užívat čísla do 127.

Každé periférní zařízení (tiskárna, disk drive, kazet. mg.) má své vlastní číslo, na které reaguje. Číslo "zařízení" příkaz OPEN užívá ke specifikaci zařízení, na kterém bude nebo je datový soubor. Periférie jako kazeťák, tiskárna, disk drive mají přiřazeno rovněž několik sekundárních adres, což jsou kódy, které říkají každému zařízení, jaká operace bude vykonána. Logické číslo zařízení je užito s každým příkazem GET#, INPUT#, PRINT#. Je-li logické číslo zařízení vynecháno, počítač automaticky předpokládá, že informace bude zaslána nebo získávána z Datassette (kazetový magnetofon), jehož číslo zařízení je 1. Jméno souboru může být rovněž vynecháno, ale později pak nelze soubor žádným jménem vyvolat. Ukládáme-li soubory na magnetickou pásku, počítač předpokládá, že sekundární adresa je 0, neuvedeme-li jinak (operace READ).

Sekundární adresa 1 otevírá soubory na kazetě pro zápis, hodnota 2 způsobí tisk značky konec pásky (end of tape) při uzavření souboru. Tato značka znemožňuje čtení dat za koncem dat, což by způsobilo chybu ?DEVICE NOT PRESENT. Pro diskové soubory sekundární adresy 2 až 14 jsou určeny pro datové soubory, ostatní čísla mají speciální význam v DOS (diskový operační systém). Při užití disk drive sekundární adresa musí být užita (viz diskový manuál).

Jméno souboru je string o 1 až 16 znacích a pro kazetové soubory a soubory na tiskárně není povinný. Vynecháme-li "typ souboru", předpokládá se, že se jedná o programový soubor (pokud není uveden "mód"). Sekvenční soubory jsou otevřeny pro čtecí "mód" = R (reading), pokud nespecifikujeme zápisový "mód" = W (writing). "Typ" může být užit pro otevření existujícího relativního souboru, "typ" = REL. Relativní a sekvenční soubory mohou existovat pouze na disku.

Snažíme-li se číst ze souboru, který nebyl otevřen, hlásí počítač chybu ?FILE NOT OPEN. Snažíme-li se přečíst neexistující

soubor, hlásí ?FILE NOT FOUND. Je-li soubor otevřen pro zápis na disku a soubor tohoto jména již existuje, hlásí počítač upozornění FILE EXIST. Podobná kontrola pro pásku neexistuje, je nutno být pozorný při zápisu, abychom si nepřepsali soubory již uložené. Otevíráme-li znovu již otevřený soubor, dostaneme chybové hlášení FILE OPEN. (Detaily viz manuál pro tiskárnu.)

Příklady příkazu OPEN:

10 OPEN 2,8,4 "DISK-OUTPUT,SEQ,W" (otevírá sekvenční soubor na disku)

10 OPEN 1,1,2,"TAPE-WRITE" (zápis konec souboru (End-of-file) po CLOSE)

10 OPEN 50,0 (vstup z klávesnice)

10 OPEN 12,3 (výstup na obrazovku)

10 OPEN 130,4 (výstup na tiskárnu)

10 OPEN 1,1,0,"NAME" (čti z kazety)

10 OPEN 1,1,1,"NAME" (zápis na kazetu)

10 OPEN 1,2,0,CHR$(10) (otevři kanál pro RS-232)

10 OPEN 1,4,0,"STRING" (zašli na tiskárnu shiftovanou grafiku (upper-case graphic))

10 OPEN 1,4,7,"STRING" zašli na tiskárnu shift. i neshiftované znaky (upper/lower case))

10 OPEN 1,5,7,"STRING" (zašli na tiskárnu s číslem zař. 5 upper/lower case)

10 OPEN 1,8,15, "POVEL" (pošli povel na disk)

O R

Typ: logický operátor

Formát: "operand" OR "operand"

Účinek: Stejně jako relační operátory mohou být užity k získání rozhodnutí týkajících se chodu programu, logické operátory mohou spojovat dvě nebo více relací a určovat jejich pravdivost (nesprávnost), což může být užito v dalším výpočtu.

Logické OR dává hodnotu 1 (bitově), jestliže jeden nebo oba operandy jsou 1. Celočíselný výsledek tedy závisí na hodnotě obou operandů. Operátor OR může být též použit k porovnání výrazů. Je-li alespoň jeden z výrazů pravdivý, je výsledek pravdivý (hodnota -1). V následujícím příkladě: jestliže AA je rovno BB nebo jestliže XX je rovno 20, je výraz pravdivý.

Logické operátory pracují tak, že číslo převedou na 16 bitový twos-complement se znaménkem -, v intervalu (- 32768,32767). Je-li operand mimo tento interval, dostaneme chybové hlášení. Každý bit výsledku je určen odpovídajícími bity obou operandů.

Příklad operátoru OR:

```
100  IF(AA=BB) OR (XX=20) THEN...
230  KK%=64 OR 32: PRINT KK% (64 bitově je 1000000,
     32 je 100000)
1000000
OR 100000
--------------
96 (výsledek) 1100000, tj. 96
```

P E E K

Typ: celočíselná funkce

Formát: PEEK (číslo nebo číselný výraz)

Účinek: Výsledek je celé číslo v intervalu (0,255), přečtené z paměťového místa zadaného "číslem nebo číselným výrazem", jehož hodnota musí být z intervalu (0,65535). Při nesplnění těchto podmínek obdržíme chybové hlášení ?ILLEGAL QUANTITY.

Příklady funkce PEEK:

10 PRINT PEEK(53280) AND 15 (výsledkem je hodnota barvy rámečku)

5 A%=PEEK(45)+PEEK(46) 256 (navrací adresu tabulky basicových proměnných)

POKE

Typ: příkaz
Formát: POKE"umístění","hodnota"
Účinek: Příkaz slouží k zapsání 1 bytu (8 bitů) do daného místa v paměti nebo do vstupního/výstupního registru. Umístění může být též aritmetický výraz s hodnotou v intervalu (0,65535). "Hodnota" je číslo nebo výraz s hodnotou v intervalu (0,255). Je-li některá z hodnot mimo tyto intervaly, dostaneme chybu ?ILLEGAL QUANTITY.

Příkazy PEEK a POKE (vestavěné funkce umožňující nahlížet do paměti) jsou užitečné pro ukládání dat, řízení grafického displeje, pro tvorbu zvukových efektů, k zavádění podprogramů v Assembleru, předávání argumentů do a z assemblerovských podprogramů. Rovněž parametry operačního systému mohou být zkoumány užitím PEEK příkazu, mohou být měněny nebo s nimi může být manipulováno užítím POKE příkazu.

V dodatku G je kompletní mapa paměti a užitečných paměťových míst.

Příklady příkazu POKE:

```
POKE 1024,1 (vlož "A" do pozice 1 na obrazovce)
POKE 2040,PTR (vlož hodnotu PTR do sprite #0 data pointeru)
10 POKE RED,32
20 POKE 36879,8
2050  POKE A,B
```

POS

Typ: celočíselná funkce
Formát: POS (prázdný argument)

Účinek: Navrací polohu kurzoru na obrazovce (hodnota od 0 do 79 na 80 znakové logické řádce obrazovky). Protože řádek Commodore 64 je 40 znakový, čísla od 40 do 79 znamenají druhý řádek. Prázdný argument je ignorován.

Příklad funkce POS:

```
1000  IF POS(0) 38 THEN PRINT CHR$(13)
```

PRINT

Typ: příkaz
Formát: PRINT ((proměnná)) ((,/;)(proměnná))...
Účinek: Příkaz PRINT je určen k vypsání dat na obrazovku. Užitím příkazu CMD můžeme výstup převést na jiné zařízení v systému. (Proměnné) ve výstupním seznamu mohou být výrazy všech typů. Neobsahuje-li výstupní seznam žádnou proměnnou, je vytištěn prázdný řádek. Poloha vytištěné proměnné je určena oddělovacími znaménky mezi proměnnými ve výstupním seznamu.

Oddělovací znaménka mohou být mezera, čárka a středník. 80ti znaková logická řádka obrazovky je rozdělena do 8 zón po 10 listech. Ve výstupním seznamu čárka způsobí, že další hodnota bude vytisknuta na začátek další zóny. Středník způsobí tisk další hodnoty hned za předcházející. Existují dvě výjimky z tohoto zákona:

1) Za numerickými položkami následuje mezera.
2) Před kladným číslem je zapsána mezera.

Nepřítomnost mezer a jiných interpunkcí mezi stringovými konstantami má stejný efekt jako středník. Mezera mezi numerickými položkami nebo mezi stringem a numerickou položkou způsobí zastavení výstupu bez vytištění další položky.

Čárka nebo středník za poslední položkou seznamu způsobí tisk v dalším příkazu PRINT na stejný řádek, přičemž mezery jsou provedeny podle předchozích pravidel. Není-li na konci seznamu

žádné znaménko interpunkce, následuje návrat vozíku a posun o řádek a další příkaz PRINT je vytištěn na začátek nové řádky. Máme-li výstup na obrazovku a tištěná data jsou delší než 40 sloupců, pokračuje tisk na další řádce. V BASICU není příkaz s větším množství variací. Existuje mnoho symbolů, funkcí, parametrů spojených s tímto příkazem, které mohou být považovány za jazyk uvnitř BASICU; jazyk speciálně navržený pro zápis na obrazovku.
Příklady příkazu PRINT:

1) 5X = 5
   10 PRINT -5*X,X-5,X+5,X(šipka nahoru)
   -25 0 10 3125
2) 5 X=9
   10 PRINT X;"SQUARED IS";X*X;"AND";
   20 PRINT X "CUBED IS" X(šipka nahoru)3
   9 SQUARED IS 81 AND 9 CUBED IS 729
3) 90 AA$ = "ALPHA": BB$ = "BAKER": CC$ =
   "CHARLIE": DD$ = "DOG": EE$ = "ECHO"
   100 PRINT AABB;CC$ DD$,EE$
   ALPHABAKERCHARLIEDOG ECHO

Uvozovkový mód (QUOTE MODE)

Když napíšeme uvozovky (SHIFT a 2), řízení kurzorem se přeruší a na obrazovce se objeví reverzní znaky podle toho, který klíč byl stisknut. To umožňuje naprogramovat si tyto znaky, které po vytištění PRINT vykonají svou funkci. Pouze klíč INST/DEL není ovlivněn uvozovkovým módem.

1. Pohyb kurzoru

Můžeme "naprogramovat tyto znaky řízení kurzorem":
Skutečné zobrazení klíče:

CLR/HOME (reverzní S)
SHIFT CLR/HOME (reverzní srdce)
CRSR UP (reverzní Q)
SHIFT CRSR DOWN (reverzní kolečko)

CRSR RIGHT (reverzní J)
SHIFT CRSR LEFT (reverzní I)

Chceme-li napsat diagonálně na obrazovce HELLO, píšeme: PRINT "CLR/HOME H CRSR E CRSR L CRSR L CRSR O", což se objeví na obrazovce jako

PRINT " S H Q E Q L Q L Q O"

2. Reverzní znaky

Stiskneme-li současně klíč CTRL a 9 v uvozovkovém módu, objeví se uvnitř uvozovek R. To způsobí, že všechny znaky budou tištěny v reverze video, tj. jako negativní obraz. Chceme-li ukončit reverzní tisk, stiskneme CTRL a 0, což se vypíše jako (reverzní vodorovná čárka) nebo vytiskneme PRINT CHR$(13), tj. (RETURN klíč). Stejný výsledek má příkaz PRINT bez středníku nebo čárky na konci.

3. Řízení barev

Stisknutí klíče CTRL nebo C= a kteréhokoliv z 8 klíčů označujících barvu způsobí po uvozovce tisk speciálních reverzních znaků. Při vytisknutí tohoto znaku PRINT se změní barva obrazovky.

KLÍČ	BARVA
CTRL 1	černá
CTRL 2	bílá
CTRL 3	červená
CTRL 4	modrá
CTRL 5	purpurová
CTRL 6	zelená
CTRL 7	modrá
CTRL 8	žlutá
C= 1	oranžová
C= 2	hnědá
C= 3	světle červená
C= 4	šedivá 1
C= 5	šedivá 2

C= 6 světle zelená
C= 7 světle modrá
C= 8 šedivá 3

Chceme-li napsat HELLO červeně a THERE bíle, píšeme:

PRINT " CTRL 3 HELLO CTRL 2 THERE"

4. Insert mód (vkládání)

Prostor vytvořený stisknutím klíče INST/DEL má některé z charakteristik uvozovkového módu. Znaky řízení kurzorem a znaky pro barvy jsou zobrazeny jako reverzní znaky. Jediný rozdíl je, že INST a DEL jsou nyní zobrazeny jako inverzní znaky. (V uvozovkovém módu zůstává jejich funkce zachována.) INST vkládá mezery normálně.

Z tohoto důvodu je možno vytvořit PRINT příkaz, obsahující znaky DEL (nemožné v závorkovém módu).

10 PRINT"HELLO"INST/DEL SHIFT INST/DEL SHIFT INST/DEL INST/DEL INST/DEL P"

Spustíme-li tento řádek příkazem RUN, objeví se na displeji HELP, neboť dvě poslední písmena byla vymazána a na jejich místo bylo vloženo P.

Varování: DEL pracuje i když listujeme program, oprava takových řádek je obtížná.

Insert mód je ukončen stisknutím klíče RETURN nebo SHIFT RETURN nebo když byl zaplněn prostor vytvořený klíčem INST.

5. Další zvláštní znaky

Existují jisté znaky se zvláštní funkcí, které nemohou být zapsány přímo z klávesnice pro příkaz PRINT. Abychom je mohli umístit do uvozovek, musíme pro ně vynechat místo, stisknout RETURN nebo SHIFT RETURN a vrátit se na příkaz zpět pomocí kurzoru. Nyní stiskneme CTRL RVS/ON (způsobí tisk reverzních znaků) a píšeme:

FUNKCE	PÍŠEME
SHIFT RETURN	SHIFT M
aktivuj velká písmena	N
aktivuj malá písmena	SHIFT N
vypni klíč na změnu písmen	H

PRINT#

Typ: příkaz vstup/výstup

Formát: PRINT# "soubor-číslo"((proměnná)) ((,/;)(proměnná))....

Účinek: Příkaz PRINT# je určen pro zápis dat do logického souboru, který musí mít stejné číslo, jako bylo použito v příkazu OPEN. Výraz pro "proměnnou" může být libovolného typu. Oddělovací znaménka mezi jednotlivými částmi jsou stejná, jako povoluje příkaz PRINT a mají stejné použití. Účinek rozdělovacích znamének se však ve dvou významných případech liší.

Při použití PRINT# pro soubory na pásce má čárka místo vytváření tiskových zón stejný účinek jako středník. Proto nezáleží, zda se pro rozdělení použije mezera, čárka, středník nebo žádný rozdělovací znak. Jednotlivé datové položky jsou zapisovány jako souvislý proud znaků. Numerické položky jsou ukončeny mezerou a mezera se vkládá i před kladná čísla.

Jestliže není seznam výstupních položek ukončen žádným oddělovacím znaménkem, data jsou ukončena znakem CR (návrat) a LF (nový řádek). V případě, že výstup je ukončen čárkou nebo středníkem, jsou znaky CR a LF potlačeny. Bez ohledu na oddělovače nový příkaz PRINT# začne výstup na nejbližší použitelné pozici. Znak LF bude použit v příkazu INPUT# pro ukončení vstupu. Znak LF může být potlačen nebo nahrazen, jak bude uvedeno v příkladech.

Nejjednodušší způsob pro zapsání více proměnných do souboru na pásce nebo disku je použití řetězcové proměnné obsahující CHR$(13), což je znak LF, a vložit tuto proměnnou mezi ostatní proměnné při zápisu do souboru.

Příklady použití příkazu PRINT#:

```
10 OPEN 1,1,1,"PASKOVY SOUBOR"
20 R$ = CHR$(13) (změnou CHR$(13) na CHR$(44)
   můžeme vložit do výstupu čárku "," a pro CHR$(59)
   (středník ";")
30 PRINT# 1,1;R$;2;R$;3;R$;4
40 PRINT# 1,6
50 PRINT# 1,7
10 CO$=CHR$(44): CR$=CHR$(13)
20 PRINT#1 ,"AAA"CO$"BBB", AAA,BBB CCCDDDEEE
   "CCC" ; "DDD" ; "EEE" CR$ (návrat LF) "FFF"CR$; FFF
   (LF)
30 INPUT#1,A$,BCDE$,F$
 5 CR$=CHR$(13)
10 PRINT#2,"AAA";CR$;"BBB" (10 mezer) AAA
20 PRINT#2,"CCC"; BBB
30 INPUT#2,A$,B$,PRAZDNY$,C$
```

READ

Typ: příkaz

Formát: READ "proměnná"(,(proměnná))...

Účinek: Příkaz slouží k naplnění proměnných přečtením konstant z příkazu DATA. Typ dat a proměnných musí souhlasit, jinak dostaneme chybové hlášení ?SYNTAX ERROR. Prvky v seznamu DATA musí být odděleny čárkami.

Jedním příkazem READ můžeme přečíst jeden či více příkazů DATA a naopak. Data jsou proměnným v příkazu READ přiřazována v pořadí, jak byla zapsána. Chceme-li přečíst více dat, než je uvedeno v příkazech DATA, dostaneme chybové hlášení ?OUT OF DATA. Je-li počet proměnných menší než počet prvků příkazu DATA, bude další příkaz READ pokračovat v nedokončeném seznamu DATA. Viz rovněž příkaz RESTORE.

Poznámka: ?SYNTAX ERROR se objeví u čísla řádky příslušející příkazu READ, nikoli DATA.

Příklady příkazu READ:

```
110  READ A,B,C$
120  DATA 1,2,HELLO
100  FOR X=1 TO 10: READ A(X): NEXT
200  DATA 3.08,5.19,3.12,3.98,4.24
210  DATA 5.08,5.55,4.44,3.16,3.37
```

položky v řádce 1 jsou naplněny konstantami z řádky 5

```
1  READ CITY$,STATES$,ZIP
5  DATA DENVER,COLORADO,80211
```

REM

Typ: příkaz

Formát: REM ((poznámka))

Účinek: Slouží k lepšímu pochopení programu, který je listován. Připomíná, co jsme chtěli naprogramovat v jednotlivých sekcích programu (například k čemu slouží jistá proměnná atd). REM může tvořit libovolný text, slova nebo znaky včetně dvojtečky nebo BASIC klíčová slova.

BASIC ignoruje celý řádek za příkazem REM, ale je-li program listován, vše za příkazem REM se vypíše. Na REM může být rovněž proveden skok příkazem GOTO nebo GOSUB, výpočet pak pokračuje příkazem za GOTO nebo GOSUB.

Příklady příkazu REM:

```
10 REM CALCULATE AVERAGE VELOCITY
20 FOR X=1 TO 20: REM SMYCKA PRO 20 HODNOT
30 SUM=SUM+VEL(X):NEXT
40 AVG=SUM/20
```

RESTORE

Typ: příkaz

Formát: RESTORE

Účinek: BASIC ma vnitřní ukazovátko, které říká, která položka z příkazu DATA bude přečtena následujícím READ. Toto ukazovátko je vynulováno po užití příkazu RESTORE v programu a následující příkaz READ bude číst ta DATA, která jsou uvedena první (jsou čtena znovu). Příkaz může být umístěn kdekoliv v programu.

Příklady příkazu RESTORE:

```
  100  FOR X=1 TO 10: READ A(X):NEXT
  200  RESTORE
  300  FOR Y=1 TO 10: READ B(Y):NEXT
 4000  DATA 3.08,5.19,3.12,3.98,4.24
 4100  DATA 5.08,5.65,4.00,3.16,3.37
```

(spojuje dvě pole se stejnými daty)

```
10 DATA 1,2,3,4
20 DATA 5,6,7,8
30 FOR L=1 TO 8
40 READ A:PRINT A
50 NEXT
60 RESTORE
70 FOR L=1 TO 8
80 READ A:PRINT A
90 NEXT
```

RETURN

Typ: příkaz

Formát: RETURN

Účinek: RETURN označuje výstup z podprogramu volaného příkazem GOSUB. RETURN odstartuje zbytek programu na další výkonné řádce za GOSUB. Při seskupení podprogramů (nesting) musí každý mít alespoň jeden příkaz RETURN. Podprogram může obsahovat RETURN vícekrát, ale první provedený způsobí výstup z podprogramu.

Příklady příkazu RETURN:

```
  10 PRINT "TOTO JE PROGRAM"
  20 GOSUB 1000
  30 PRINT "PROGRAM POKRACUJE"
  40 GOSUB 1000
  50 PRINT "DALSI CAST PROGRAMU"
  60 END
1000 PRINT "TOTO JE PODPROGRAM VYVOLANY
     GOSUB": RETURN
```

RIGHT$

Typ: stringová funkce

Formát: RIGHT$ ((string),(číslo nebo číselný výraz))

Účinek: Výsledkem je podstring vyjmutý zprava z argumentu funkce RIGHT$. Délka podstringu je dána (číslem nebo čís. výrazem) a může být libovolné číslo od 0 do 255. Je-li nula, dostaneme nulový string (""). Je-li argument (string) kratší než délka podstringu, dostaneme celý string.

Příklady funkce RIGHT$:

```
10 MSG$="COMMODORE COMPUTERS"
20 PRINT RIGHT$(MSG$,9)
RUN
COMPUTERS (výsledek)
```

RND

Typ: reálna funkce

Formát: RND((číslo nebo čís.výraz))

Účinek: RND vytváří náhodná čísla od 0.0 do 1.0. Počítač generuje náhodná čísla provedením výpočtu na jistou startovací hodnotu v matematickém žargonu zvanou "seed", která je určena při zapnutí počítače. Argument RND je prázdný až na znaménko. Je-li argument kladný, dostaneme stejnou pseudonáhodnou

řadu při stejné hodnotě seed. (Pro různé seed dostaneme různé posloupnosti náhodných čísel, každou posloupnost lze zopakovat při startu od stejné hodnoty seed.) To je důležité při odlaďování programu.

Je-li argument nula, potom počítač generuje čísla přímo od hardwareových hodin počítače ("jiffy clock"). Záporný argument způsobí, že při každém vyvolání funkce RND je nové seed.

Příklady funkce RND:

```
220 PRINT INT(RND(0) 50) (náhodná čísla od 0 do 50)
100 X=INT(RND(1) 6)+INT(RND(1) 6)+2 (simuluje 2 náh. čísla)
100 X=INT(RND(1) 1000)+1 (náhodná čísla od 1 do 1000)
100 X=INT(RND(1) 150)+100 (náhodná čísla od 100 do 249)
100 X=RND(1) (U-L)+ (náhodná čísla v intervalu (U,L))
```

RUN

Typ: povel

Formát: RUN ((číslo řádky))

Účinek: Systémový příkaz RUN odstartuje program, který je právě v paměti. Způsobí, že nejprve je provedena operace CLR. Tomu se můžeme vyhnout tak, že k opětnému odstartování programu užijeme příkaz CONT nebo GOTO místo RUN. Je-li dáno (číslo řádky), program začne výpočet od této řádky. Jinak RUN odstartuje výpočet od první řádky programu. RUN může být užit též uvnitř programu. Jestliže (číslo řádky) neexistuje, dostaneme chybové hlášení UNDEF'D STATEMENT.

Odstartovaný program se zastaví při nalezení příkazu STOP a END, po provedení poslední programové řádky a při výskytu BASIC chyby při výpočtu.

Příklady povelu RUN:

```
RUN (start od první řádky programu)
RUN 500 (start od řádky 500)
```



```
RUN X (start od řádky X; neexistuje-li, dostaneme
    UNDEF'D STATEMENT ERROR)
```

SAVE

Typ: povel

Formát: SAVE"jméno souboru", (číslo zařízení), (adresa))

Účinek: Slouží k uložení programu, který je v paměti, na disk nebo magnetickou pásku. Program, který je ukládán, je ovlivňován pouze příkazem SAVE. Po skončení operace SAVE program zůstává v paměti počítače, pokud nepoužijeme jiný povel ke změně. SAVE vytváří soubory typu "prg" (program). Je-li "číslo zařízení" vynecháno, C64 automaticky předpokládá, že program bude uložen na kazetu, mající číslo zařízení 1. Je-li číslo zařízení 8, bude program uložen na disk. SAVE může být rovněž užit uvnitř programu a výpočet bude pokračovat na dalším řádku po jeho provedení.

Programy na pásce jsou automaticky uloženy dvakrát, takže C64 může kontrolovat správnost zavádění při užití LOAD. Při ukládání programu na kazetu není (jméno souboru) ani sekundární adresa povinná. Ovšem uvedeme-li v příkazu SAVE jméno programu v uvozovkách nebo stringovou proměnnou (---$), může C64 snáze nalézt program při jeho pozdějším vyvolání. Sekundární adresa 1 říká KERNALU, že později bude program zaveden od adresy, kde je nyní v paměti, nikoli od 2048. Sekundární adresa 2 způsobí zápis značky end-of-tape (konec pásky) za programem. Sekundární adresa 3 kombinuje obě tyto funkce.

Ukládáme-li program na disk, je (jméno souboru) povinné.

Příklady povelu SAVE:

```
SAVE (zápis na kazetu bez jména)
SAVE"ALPHA",1 (zápis na kazetu pod jménem "ALPHA")
SAVE "ALPHA",1,2 (zápis "ALPHA" se značkou
    end-of-tape)
```

SAVE "FUN.DISK",8 (zápis na disk, tj. 8)
SAVE A$ (zápis na pásku pod jménem A$)
10 SAVE "HI" (zapíše program a pokračuje na další řádce)
SAVE "ME",1,3 (zabezpečí při LOAD zpětné uložení do původního prostoru v paměti; dá značku EOT na konec souboru)

SGN

Typ: celočíselná funkce
Formát: SGN ((číslo, num. výraz))
Účinek: SGN dává celočíselnou hodnotu podle znaménka argumentu. Je-li kladné, výsledek je 1, při nulovém argumentu je 0, při záporném argumentu je výsledek -1.
Příklady funkce SGN:

```
90 ON SGN(DV)+2 GOTO 100,200,300 (skoč na řádek 100,
   je-li DV 0, na 200 při DV=0, na 300 pro DV 0)
```

SIN

Typ: reálná funkce
Formát: SIN ((číslo nebo čís. výraz))
Účinek: Výsledek je sínus argumentu v radiánech. Hodnota COS(x) je rovna SIN(x+3.14159265/2)
Příklad funkce SIN:

```
235  AA=SIN(1.5): PRINT AA
.997494987
```

SPC

Typ: Stringová funkce
Formát: SPC((číslo nebo čís. výraz))
Účinek: SPC funkce je užívána k formátování dat v logickém výstupním souboru i na obrazovce. Vytiskne se počet mezer (SPaCes) daných argumentem, počínaje první volnou pozicí. Pro

soubory na kazetě a na obrazovce je hodnota argumentu od 0 do 255, pro disk do 254. U souboru na tiskárně se automaticky provede návrat vozíku a posun o řádek, je-li posledním tištěným znakem mezera. Na nový řádek se žádná mezera nezapíše.
Příklady funkce SPC:

```
10 PRINT "RIGHT ";"HERE ";
20 PRINT SPC(5)"OVER"SPC(14)"THERE"
RUN
RIGHT HERE      OVER          THERE
```

SQR

Typ: reálná funkce
Formát: SQR((číslo nebo čís. výraz))
Účinek: Výsledkem je druhá odmocnina argumentu. Je-li hodnota argumentu záporná, dostaneme chybové hlášení ?ILLEGAL QUANTITY.
Příklad funkce SQR:

```
FOR J=2 TO 5: PRINT J*5, SQR(J*5): NEXT
```

Výsledek:

```
10 3.16227766
15 3.87298335
20 4.47213595
25 5
```

STATUS

Typ: celočíselná funkce
Formát: STATUS
Účinek: Vrací kompletní STATUS pro poslední operaci vstup/výstup na otevřeném souboru. STATUS kterékoli periférie může být přečten. STATUS (krátce ST) je systémem definované jméno proměnné, do které KERNAL vkládá STATUS vstupní/výstupní operace.

Tabulka hodnot *STATUS* pro pásku, tiskárnu, disk a RS-232:

Bit číslo	Číselná hodnota	Casette Read	Serial Bus R/W	Tape Verify + Load
0	1		time out write	
1	2		time out read	
2	4	krátký blok		krátký blok
3	8	dlouhý blok		dlouhý blok
4	16	chyba při čtení		různé
5	32	chybná kontrolní suma		chybná kontrolní suma
6	64	EOF; konec souboru	EOI	
7	128	EOT; konec pásky	zařízení není připojeno	EOT

Příklady funkce STATUS:

```
10 OPEN 1,4:OPEN 2,8,4," MASTER SOUBOR,SEQ,W"
20 GOSUB 100:REM ZKONTROLUJ STATUS
30 INPUT#2,A$,B,C
40 IF STATUS AND 64 THEN80: REM POKYN PRO
   END-OF-FILE
50 GOSUB 100: REM ZKONTROLUJ STATUS
60 PRINT#1,A$,B;C
70 GO TO 20
80 CLOSE 1: CLOSE2
```



```
90 GOSUB 100:END
100 IF ST=0 THEN 9000: REM POKYN PRO CHYBU
    VSTUP/VYSTUP
```

STEP

Typ: příkaz

Formát: (STEP výraz)

Účinek: Nepovinné slovo STEP následuje konečnou hodnotu v příkazu FOR. Definuje přírůstek pro proměnnou cyklu. Jakákoli hodnota ve STEP je přípustná. Hodnota 0 však vytvoří nekonečnou smyčku. Je-li klíčové slovo STEP vynecháno, je přírustek +1. Proměnná cyklu je změněna o hodnotu STEP při dosažení příkazu NEXT. Je provedeno testování proměnné cyklu a konečné hodnoty. (Další informace viz příkaz FOR...)

Poznámka: Hodnota STEP nesmí být měněna uvnitř smyčky.

Příklady příkazu STEP:

```
25 FOR XX=2 TO 20 STEP2 (opakuje smyčku 10krát)
35 FOR ZZ=0TO -20 STEP -2 (opakuje smyčku 11krát)
```

STOP

Typ: příkaz

Formát: STOP

Účinek: Příkaz slouží k zastavení běžícího programu a k návratu do přímého módu. Stejný efekt má stisknutí klíče RUN/STOP. Na obrazovce se objeví zpráva ?BREAK IN LINE nnnn, za ní READY. nnnn je číslo řádky, na které byl program zastaven. Všechny otevřené soubory zůstávají otevřené a hodnoty proměnných zachovány po provedení STOP. Program může být znovu odstartován užitím CONT nebo GOTO.

Příklady příkazu STOP:

```
10 INPUT#1,AA,BB,CC
20 IF AA=BB AND BB=CC THEN STOP
```

```
30 STOP
(Je-li proměnná AA rovna -1 a BB rovno CC, potom:) BREAK
    IN LINE 20
BREAK IN LINE 30 (pro všechna ostatní data)
```

STR$

Typ: stringová funkce
Formát: STR$((číslo nebo čís. výraz))
Účinek: Výsledek je stringová reprezentace čísla v argumentu. Převod je proveden pro každou hodnotu argumentu, za každým číslem následuje mezera a je-li kladné, též ho mezera předchází.
Příklady funkce STR$:

```
100  FLT=1.5E4: ALPHA$=STR$(FLT)
110  PRINT FLT,ALPHA$
15000 15000
```

SYS

Typ: příkaz
Formát: SYS (adresa v paměti)
Účinek: Toto je nejobvyklejší způsob, jak spojovat programy v BASICU a ve strojovém kódu. Program ve strojovém kódu začíná na adrese udané příkazem SYS. Systémový příkaz SYS může být užit buď v přímém módu nebo v programovém k předání řízení mikroprocesoru existujícímu strojovému programu v paměti. "Adresa v paměti" je číslo nebo číselný výraz udávající umístění kdekoli v paměti ROM nebo RAM.

Užijeme-li příkaz SYS, musí oddíl strojového programu být zakončen instrukcí RTS (ReTurn from Subroutine - návrat z podprogramu), takže po skončení provádění strojového programu budou prováděny další BASIC příkazy za povelem SYS.
Příklady příkazu SYS:

```
SYS 64738 (skok do systému - Cold Start v ROM)
```



```
10 POKE 4400,96: SYS 4400 (jde do strojového programu
    na adrese 4400 a hned se vrací)
```

TAB

Typ: stringová funkce
Formát:TAB((číslo nebo čís. výraz))
Účinek: Funkce TAB přemístí kurzor doprava o počet míst daných argumentem, počínaje od nejlevější pozice na dané řádce. Hodnota argumentu musí být v řadě od 0 do 255. Funkce TAB může být užita pouze s příkazem PRINT, její spojení s příkazem PRINT# nemá smysl.
Příklady funkce TAB:

```
100  PRINT "NAME" TAB(25)" MNOZSTVI";PRINT
110  INPUT#1,NAME$,AMT$
120  PRINT NAM$,TAB(25)AMT$
NAME MNOZSTVI
G.T. JONES 25
```

TAN

Typ: reálná funkce
Formát: TAN ((číslo nebo čís. výraz))
Účinek: Počítá tangens argumentu v radiánech. Při přetečení funkce TAN dostaneme chybové hlášení ?DIVISION BY ZERO.
Příklad funkce TAN:

```
10 XX=.785398163: YY=TAN(XX): PRINT YY
1
```

TIME

Typ: numerická funkce
Formát: TI
Účinek: Tato funkce čte čas (interval Timer). Tento typ hodin se nazývá "jiffy clock". Hodnota "jiffy clock" je položena rovno nule

(inicializovaná) při zapnutí systému. Čas se mění po 1/60 sekundy. Hodiny jsou vypnuty během vstupních a výstupních operací s páskou.

Příklad funkce TI:

```
10 PRINT TI/60"SEKUNDY OD ZAPNUTI PRISTROJE"
```

TIME$

Typ: stringová funkce

Formát: TI$

Účinek: Pokud je počítač zapnut, pracuje funkce TI$ stejně jako normální hodiny. Jsou přečteny "jiffy clock" a jejich hodnota převedena do stringové funkce TI$. Nebo mohou být nastaveny jako normální hodinky na přesný čas a hodnota TI je k tomuto času přičítána. Tento čas ovšem není přesný po vstupní/výstupní operaci s páskou.

Příklad funkce TI$:

```
1 TI$="000000"; FOR J=1 TO 1000: NEXT: PRINT TI$
000011
```

USR

Typ: reálná funkce

Formát: USR((číslo nebo čís. výraz))

Účinek: Funkce USR (User SubRoutine) provede skok do uživatelského podprogramu ve strojovém kódu, jehož startovací adresa je uložena na adrese 785-786. Tato adresa musí být naplněna užitím příkazu POKE před vyvoláním funkce USR. Pokud se tak nestane, dostaneme chybové hlášení ?ILLEGAL QUANTITY.

Hodnota argumentu je uložena v akumulátoru na adrese 97, pro přístup z assembleru, a výsledkem funkce USR je hodnota ....which ends up there... při návratu z podprogramu do BASICU.

Příklady funkce USR:


```
10 B=T SIN(Y)
20 C=USR(B/2)
30 D=USR(B/3)
```

VAL

Typ: numerická funkce

Formát: VAL((string))

Účinek: Vrací číselnou hodnotu (VALue) dat ve stringovém argumentu. Není-li první neprázdný znak ve stringu znaménko + nebo - nebo číslice, je hodnota funkce VAL nulová. Stringová konverze je ukončena na konci stringu nebo při nalezení nečíslicového znaku (s výjimkou desetinné tečky a znaku exponentu e).

Příklady funkce VAL:

```
10 INPUT #1, NAM$,ZIP$
20 IF VAL(ZIP$) 19400 or VAL(ZIP$)>96699
THEN PRINT NAM$ TAB(25) "GREATER PHILADELPHIA"
```

VERIFY

Typ: povel

Formát: VERIFY ("(jméno souboru)") (,(zařízení))

Účinek: VERIFY se užívá v přímém nebo programovém módu k porovnání programu v paměti s programem na kazetě nebo na disku. Obvykle se používá hned po SAVE k ujištění, že program byl správně nahrán na kazetu nebo disk.

Vynecháme-li (zařízení), předpokládá se porovnání s Datassette (kazet. mg.) s číslem 1. Je-li vynecháno "jméno souboru" při práci s páskou, bude porovnán nejbližší nalezený soubor na kazetě. Pro soubory na disku je "jméno souboru" povinné. Při nalezení rozdílu v zápisu proti programu v paměti se objeví na displeji ?VERIFY ERROR.

Jméno programu může být zadáno v uvozovkách ("") nebo jako stringová proměnná. VERIFY lze též užít k nastavení pásky za poslední program (porovnáme poslední zapsaný program s programem v paměti), takže nový program může být zapsán bez nebezpečí náhodného přepsání jiného programu.

Příklady povelu VERIFY:

```
VERIFY (zkontroluje první program na pásce)
PRESS PLAY ON TAPE
OK
SEARCHING
FOUND (jméno souboru) VERIFYING
9000 SAVE "ME",8
9010 VERIFY "ME",8 (zkontroluje program na disku)
```

WAIT

Typ: příkaz

Formát: WAIT (umístění),(mask-1)(,(mask-2))

Účinek: Příkaz WAIT způsobí přerušení programu, dokud daná adresa v paměti není naplněna jistou hodnotou. Jinými slovy, program je zastaven, dokud se nevyskytne jistá vnější událost. To se může stát naplněním status bitu vstupních/výstupních registrů. Data užitá s WAIT mohou být numerické výrazy, ale jejich hodnota je převedena na celé číslo.

Většina programátorů tento příkaz nikdy nepoužije. Slouží k zastavení programu, dokud bity jistého místa v paměti nejsou určitým způsobem změněny. Toho se užívá pro jisté vstupní/výstupní operace a téměř nikde jinde.

Příkaz WAIT vezme hodnotu v (umístění) a provede logické AND s hodnotou v (mask-1). Je-li v příkazu (mask-2), je na výsledek předchozí operace (AND) provedena operace XOR (exclusive OR) s hodnotou (mask-2). Jinými slovy, mask-1 odfiltruje všechny bity, které nechceme testovat (tj. všechny bity, které mají nulovou hodnotu v mask-1). (Mask-2) otočí (flips) všechny bity, takže

můžeme provádět testování na platnost i neplatnost podmínky. Každý bit testovaný na 0 by měl mít 1 v odpovídající pozici (mask-2). Liší-li se odpovídající bity (mask-1) a (mask-2), je bitový výsledek operace XOR jedna, shodují-li se bity, je výsledek operace XOR nulový. Příkazem WAIT můžeme vytvořit nekonečnou pauzu, kterou lze přerušit pouze RUN/STOP a RESTORE klíčem. Stiskni klíč RUN/STOP a RESTORE. První příklad dole čeká na stisknutí klíče na páskové jednotce, aby program mohl pokračovat. Druhý příklad čeká na kolizi spritu s pozadím obrazovky.

Příklady příkazu WAIT:

```
WAIT 1,32,32
WAIT 53273,6,6
WAIT 36868,144,16 (144 a 16 jsou masky. 144=10010000
    binárně, 16=10000 binárně. Příkaz WAIT zastaví
    program až do nastavení bitu o hodnotě 128 (7. bit)
    nebo do vymazání bitu o hodnotě 16 (4. bit).)
```

KLÁVESNICE COMMODORE 64

Operační system má 10-znakovou klávesnicovou vyrovnávací paměť (keyboard buffer), přebírající stisky klávesnice, dokud nemohou být využity. Buffer si pamatuje stisky kliků v pořadí, jak přišly, a stejně je využívá. (Nejprve zpracuje první klíč, pak druhý...)

Jinými slovy, keyboard buffer umožňuje "psát dopředu". (Využito v příkazech INPUT a GET.) Problém vznikne při náhodném stisku klávesnice, z bufferu je pak dosazen nesprávný znak.

Obyčejně nesprávné stisky klávesnice nejsou problémem, lze je snadno opravit klíči CRSR a DEL s následným přepsáním. Po stisknutí klíče RETURN není žádná korekční akce možná, neboť znak již byl zaslán do bufferu. Tomu se vyhneme smyčkou, která přečte celý buffer:

```
10 GET JUNK$: IF JUNK$ "" THEN 10 : REM Vyprázdní
    keyboard buffer
```

Kromě příkazu INPUT a GET může k přečtení bufferu být užito příkazu PEEK (adresa 197 obsahuje celočíselnou hodnotu právě stisknutého klíče). Nebyl-li stisknut žádný klíč, vrací PEEK hodnotu 64. V dodatku C jsou uvedeny číselné hodnoty znaků na klávesnici a jejich znakové ekvivalenty (CHR$). Následuje příklad smyčky, která skončí, teprve až je klíč stisknut a jeho celočíselná hodnota převedena na znakovou hodnotu.

```
10 AA= PEEK(197): IF AA=64 THEN 10
20 BB$=CHR$(AA)
```

Keyboard se chová jako řada klíčů organizovaných do matice o 8 řádcích a 8 sloupcích. Tato matice je prohlížena KERNALEM využívaje CIA#1 I/O chip (MOS 6526 Complex Interface Adapter).

CIA registr #0 (adresa 56320 $DC00) prohlíží sloupce matice, registr #1 (adresa 56321 ,$DC01) řádky matice. Bity 0-7 adresy 56320 odpovídají sloupcům 0-7. Bity 0-7 adresy 56321 odpovídají řádkům. Přečtením sloupcových a potom řádkových hodnot KERNAL dekóduje hodnotu CHR$(N) stisknutého klíče.

Osm řádků krát osm sloupců představuje 64 možných hodnot. Po stisku klíčů RVS, CTRL nebo C= nebo stiskneme-li SHIFT a některý další klíč, generujeme další hodnoty. KERNAL dekóduje tyto klíče odděleně a pamatuje si stisk těchto řídících klíčů. Výsledek je na adrese 197.

Znaky mohou být čteny přímo z vyrovnávací paměti klávesnice na adrese 631-640 užitím příkazu POKE. Počet těchto znaků je na adrese 198. Toho může být využito k automatickému provádění série povelů v přímém módu vytištěním těchto příkazů na obrazovku, vložením návratu vozíku do bufferu a vložením počtu znaků. Následující program vytiskne programy a odstartuje výpočet.

```
10 PRINT CHR$(147)"PRINT#1:CLOSE1:GOTO 50"
20 POKE 631,19:POKE #",13:POKE633,13:POKE198,3
30 OPEN1,4:CMD1:LIST
40 END
50 REM PROGRAM ODSTARTUJE ZDE
```


SCREEN EDITOR

Screen editor umožňuje pohodlnou opravu textu programu. listujeme-li program na obrazovce, můžeme užít klíče kurzoru a další speciální klíče k pohybu kurzoru po obrazovce, abychom provedli příslušné opravy. Po provedení příslušné opravy na řádce stisknutím klíče RETURN na kterékoli pozici na řádce SCREEN EDITOR přečte celý 80ti znakový logický řádek. Text je potom zaslán do Interpreteru a zařazen do programu. Opravená řádka nahradí starou verzi této řádky v paměti. Další kopie kterékoli řádky může být vytvořena změnou čísla řádky a stiskem RETURN.

Při užití zkratek klíčových slov, kdy počet znaků je větší než 80, budou znaky přes 80 ztraceny, neboť SREEN EDITOR čte pouze dvě fyzické řádky obrazovky. To je rovněž důvod, proč INPUT čte pouze 80 znaků. Z těchto důvodů je doporučeno pracovat s pouze 80 znaky na řádce.

Za určitých podmínek se znaky pro řízení kurzoru chovají odlišně (viz pohyb kurzoru, uvozovkový mód a insert mód u příkazu PRINT).