Jazyková příručka

MUMPS

Pro školní účely sestavil
Jaroslav Kučera

1. Seznámení s MUMPSem

Cílem této kapitoly je zodpovědět několik základních otázek, které si položí většina lidí při prvním setkáni se slovem MUMPS. Tato kapitola není nutná pro zvládnutí MUMPSu. Pro každého budoucího uživatele je však vhodné znát odpovědi na různé otázky jako jsou:

- co je to MUMPS ?
- k čemu je použitelný a k čemu ne ?
- jak vypadá porovnání s jinymi databázovými systémy ?
- jaká je struktura souborů ?
- jak vypadá programování v MUMPSu ?
- dostupnost MUMPSu ?
- dostupnost školení a literatury ?

1.1. Co je to MUMPS ?

MUMPS je databázový systém se stromovou strukturou databází, která je vlastní ekonomickým, technickým, biologickým a dalším systémům. Navíc však nejde o klasickou stromovou strukturu, ale o zdokonalenou stromovou strukturu, která se velice blíží snové struktuře. Programování v MUMPSu je jednodušší než například v BASICU či v Dbase III+. Začátečník v programování může již po několika měsících jít daleko za hranice možností většiny známých programovacích jazyků, a to i dvou výše uvedených. MUMPS je programovací jazyk s nezvykle vysokou produktivitou programování. Produktivita je zhruba 3 - 5 krát vyšší v porovnání s jazyky, jako je například BASIC, COBOL, PASCAL, FORTRAN, Dbase a její klony, REDAP, FAND a další. Tato porovnání byla již několikrát popisována v různých časopisech a každý, kdo má zájem hledat dokonalé programovací prostředky, se o tom může přesvědčit.
MUMPS je standardizován v ANSI normě (první ANSI norma je z roku 1977 a nyní je platná norma z roku 1984). ANSI normu dodržuje každý implementátor jazyka MUMPS a lze zodpovědně říci, že MUMPS je implementován na všech ve světě běžně používaných operačních systémech počínaje systémem CP/M a konče systémy pro počítače jako je IBM 380. Implementátoři MUMPSu mohou systém dle určitých pravidel rozvíjet a implementovat nové prvky jazyka. Některá tato rozšíření se pak mohou stát součástí nové ANSI normy.
MUMPS je při používání standardizovaných prvků jazyka stoprocentně přenositelný mezi všemi počítači. Pro danou aplikaci je pak rozhodující pouze velikost vnějších rychle přístupných pamětí.

1.2. K čemu je použitelný a k čemu ne ?

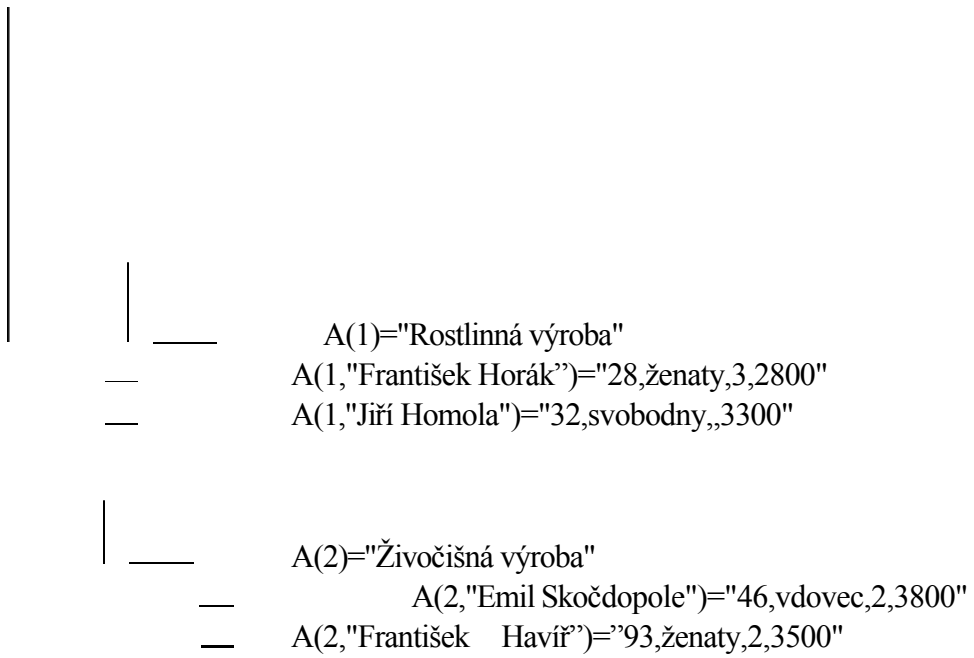
MUMPS je orientován na interaktivní zpracovávání velkého množství dat, a to jak v jednouživatelských systémech, tak i v rozsáhlých informačních sítích.MUMPS je dále s výhodou použitelný při tvorbě expertních systémů.
MUMPS je možno dále používat i pro dávkové zpracovávání rozsáhlých datových souborů vzniklých pod MUMPSem nebo pod jiným programovacím jazykem. Soubory vzniklé pod jiným jazykem je možno zpracovávat přímo nebo po převodu do struktury MUMPSu.
MUMPS, stejně jako každý jiný systém, má svoje omezení. Jde o nevhodnost jeho používání pro oblast v niž převládají složité vědeckotechnické výpočty, popřípadě řízení technologických procesů. Tento handicap se však u komplikačních verzí MUMPSu výrazně snižuje, a to hlavně pro oblast vědeckotechnických výpočtů.

1.3. Jak vypadá porovnání s jinými databázovými systémy ?

MUMPS, v porovnání s databázovými prostředky jako jsou Dbase a její klony, IDMS, REDAP, FAND a podobné, umožňuje vše co je potřebné ke tvorbě rozsáhlých informačních systémů, včetně "žurnálování". Žurnálování je autonomní systém ochrany před neblahými důsledky ztráty části informací v případě výpadku počítače a jeho obdoba je pouze u IDMS.
MUMPS používá velice úsporný instrukční kód s jednoduchými syntaktickými pravidly, důsledkem čeho je velmi krátký zápis i značně složité programové konstrukce.Porovnáním délky (v bytech) vychází opět 3 - 5 krát kratší program vytvořeny v MUMPSu oproti výše jmenovaným jazykům.
MUMPS umožňuje a většina jeho implementaci podporuje tvorbu strukturovaných programů. MUMPS existuje ve verzích interpretačních (většina starších verzí), ve verzích polokompilačních a kompilačních. Z tohoto plyne různá rychlost verzí od různých implementátorů na stejném počítači.
MUMPS je ve své podstatě systém virtualizace paměti,což znamená,že u počitače,které spadají do kategorie minipočítačů, středních a u velkých počítačů umí kontinuálně využívat více diskových médii najednou. Uživatel se vůbec nemusí starat o fyzické ukládáni dat na disk a MUMPS si sám ukládá data postupně na disk první, druhý a další, podle toho s kolika disky byl nainstalován. Je však pochopitelně možné, aby si uživatel sám řekl, na který disk chce která data ukládat. U počítačů kategorie mikro je možno mít databáze na vice diskových médiích, avšak uživatel si musí sám řídit přístup na jednotlivé disky. Po spuštění MUMPSu je možno disketu se systémem vyjmout a založit datovou či programovou disketu, a to i u interpretačních verzí.

1.4. Jaká je struktura souborů ?

MUMPS nepoužívá tradiční souborovou koncepci, ale databankovou koncepci. MUMPS na sebe přebírá veškerou starost s otevíráním a uzavíráním souborů. Uživatel je dokonce zbaven i práce s definováním souborů, takže pracuje vlastně na úrovni jednotlivých údajů či skupin údajů. Přistup k jednotlivým údajům a ke skupinám údajů je přímý. Struktura uložení dat je zdokonalená stromová struktura. Objasněni pojmu bude demonstrováno na příkladě. Máme zadána následující data:



```
|_____    A(5)="Mechanizace"
__        A(5,"Petr Kubeš")="32,ženatý,2,2900"
```

MUMPS používá přímý přístup ke skupinám údajů jako jsou údaje A (což je název celé databáze), A(číslo,"jméno"), tak i k jednotlivým údajům jako jsou A(1), A(2), A(5). Údaje, které jsou ve skupině, jsou dále jednotlivě přístupné po jejich jednoduché separaci. V příkladu je jako separátor použita čárka. Logická interpretace skupiny údajů A(1,"Jiří Homola") pak vypadá následovně: první údaj je věk, druhý je stav, třetí je počet dětí a čtvrtý údaj je plat. Třetí údaj není vyplněn, což znamená, že je prázdny, takže v databázi budou uloženy za sebou pouze dva separátory (2 čárky). Logickou interpretaci jednotlivých údajů si určuje tvůrce databáze při jejím návrhu. V datech nebyl definován údaj A(3) ani údaj A(4) a MUMPS pro ně na diskovém médiu nezabírá žádné místo. Stejně tak řetězce v jednotlivých údajích mohou mít rozdílnou délku, na disku jsou však uloženy dle aktuální délky. Bude-li v databázi kdykoli založen údaj A(4), pak se logicky umístí mezi údaje A(2) a A(5). Z uvedeného jasně vyplývá, že MUMPS dovede velmi šetrně využívat místo na diskových médiích, přestože používá přímý přístup k datům.
MUMPS pro přístup k datům používá indexy, které se nachází přímo v databázi. Příklad názorně ukazuje, že přístupové indexy mohou být tvořeny jak čísly, tak i textovými řetězci (ty musí být v uvozovkách). Přístupový index u údaje označeného A(1,"Jiří Homola") se skládá z čísla 1 a textového řetězce "Jiří Homola". Indexy však mohou být obsahem proměnné, či mohou být přes proměnnou nepřímo volány.
MUMPS má tzv. zdokonalenou stromovou strukturu ukládání dat, což znamená, že údaje jsou přístupny nejen na základě uvedení celého indexu, ale údaje je možno procházet i na úrovni právě zadané větve. Z dat v příkladu to znamená. Jsme-li například na prvku A(1,"František Horák"), pak můžeme na této úrovni přímo přistoupit k sousednímu prvku A(1, "Jiří Homola"). Řetězec "Jiří Homola" začíná písmenem "J", a proto tento celý prvek je za prvkem s indexem "František Horák", který začíná písmenem "F". Takto zdokonalená stromová struktura se začíná přibližovat ideálu databázových datových struktur, a to struktuře síťové.

1.5. Jak vypadá programování v MUMPSu

MUMPS používá velice úsporný instrukční kód s jednoduchými syntaktickými pravidly. Základních příkazů je kolem dvaceti, funkcí dvanáct a speciálních systémových proměnných je sedm (viz ANSI norma z roku 1984). Rozšiřující příkazy, funkce a speciální proměnné, dopracovává každý implementátor dle svých představ. Příkazy je možno používat zkrácené na první znak. Rozšiřující příkazy na dva znaky, přičemž první znak je písmeno "Z". Obdoba je u funkcí a systémových proměnných, přičemž se před nimi píše ještě znak dolar ($). Vykonání každého příkazu je možno podmínit (výjimku tvoří příkaz IF, ELSE a FOR). Příkaz READ pro vstup dat je možno omezit časově (toto platí i pro některé další příkazy).
MUMPS používá příkaz cyklu FOR, kde obsah řídící proměnné cyklu může byt číslo nebo i text. Je tedy možno napsat např.

```
FOR A=1:1:10
```

což znamená cyklus od 1 po 1 do 10, dále možno

```
FOR A=1:1:10,-5:-1.5:-20
```

což znamená od 1 po 1 do 10, dale od -5 po -1.5 do -20. čísla lze též zadávat přes proměnné matematickým vztahem. Následující příkaz umožní vytvořit cyklus přes text "Petr","Jan" a "Jiří".

```
FOR A="Petr", "Jan","Jiří".
```

MUMPS používá pro větvení programu příkazy GOTO, DO a funkce. Příkazy skoku GOTO a příkazy skoku do podprogramu DO lze ovlivnit podmínkami dokonce ve dvou úrovních. Takto napsaný příkaz

GOTO NAV

skočí bez jakékoli podmínky na návěští NAV.

GOTO:A="Petr" NAV

skočí na návěští NAV jen za podmínky, že obsah proměnně A je text "Petr".

GOTO:A="Petr" NAV1:B="ano",NAV2:C=25,NAV3:C=funkce(..).

Tento příkaz bude vykonán jen za podmínky,že A="Petr".
Pak bude program větven na NAV1 za podmínky,že B="ano", nebude-li toto provedeno,tak bude testována možnost odskoku na NAV2 za podmínky, že C=25, eventuálně nebude-li proveden odskok na NAV2,tak bude testována možnost odskoku na NAV3 za podmínky,že bude splněna naznačená funkce.
MUMPS umožňuje psaní velmi pěkně přehledných programů, ale také umožňuje psaní naprosto nečitelných a i autorem ne-doladitelnych programů. Není-li psaní programů podrobeno určitým konvencím, jako například programovat strukturovaně, pak není možno spolupracovat ve velkých programátorských kolektivech. V obráceném případě je možno provádět spolupráci ve velkém tvůrčím týmu.

2. POZOR NEPŘEHLÉDNĚTE TUTO ČÁST

2.1 Co je nutno vědět před započetím práce s jazykem MUMPS

Prvním předpokladem pro zdárnou práci s jazykem MUMPS je správná instalace interpreteru jazyka MUMPS na mikropočítači. Proto je nutno spinit několik základních podmínek, které jsou podrobně popsány v dalším referenčním manuálu.
V případě, že nemáte MUMPS nainstalován a nemáte možnost o toto požádat odborníka, lze si MUMPS nainstalovat alespoň provizorně. Zde je však nutné upozornit, že po takto provedené instalaci Vám nemusí pracovat některé jiné programy, které dříve pracovaly (v podstatě jde o to, že bude změněn soubor CONFIG.SYS).

2.2, Prozatímní instalace interpreteru jazyka MUMPS

Nejprve je potřeba vytvořit kopii distribuční diskety na winchester disk (HD). Toto za Vás udělá program INSTAL, který je na distribuční disketě. V tomto bodě jsou na Vás kladeny následující požadavky:
1 - zapnout počítač
2 - zajistit aby na HD bylo volné místo alespoň 350 kB
3 - distribuční disketu zasunout do vrchní mechaniky a poté zatlačit na páčku - uzavřít mechaniku diskety dle nákresu
4 - napsat na klávesnici následující dva znaky a: (malé či velké písmeno A s dvojtečkou), poté stisknout klávesu označenou nápisem RETURN nebo ENTER
5 - nyní napište na klávesnici slovo INSTAL a poté stiskněte klávesu <ET>
6 - tímto okamžikem je zahájena instalace a její ukončení se ohlásí na obrazovce naprosto stejným způsobem jako po zapnutí počítače
7 - po ukončení instalace si dobře uschovejte Vaši distribuční disketu.
Tímto, jste ukončili instalaci MUMPSu. MUMPS je možno spustit tak, že na klávesnici napíšete slovo MUMPS a stlačíte klávesu <ET>. Totéž můžete provést i nyní,čímž poprvé vstoupíte do systému MUMPS. Zde je opět na místě upozornit, že takto nainstalovaný MUMPS může sloužit pouze pro výuku, nikoli pro seriózní práci. Pro vážnou práci je MUMPS potřeba nainstalovat dle popisu v druhé části manuálu.
Po spuštění se Vám MUMPS ohlásí vypsáním úvodního textu a svoji připravenost k práci oznámí znaménkem > umístěném na novém řádku v první pozici. Od této chvíle jste řízeni přímo MUMPSem a není možno používat příkazy operačního systému MS-DOS.

3. Práce v přímém příkazovém režimu a základní příkaz MUMPSu

MUMPS, který vlastníte je interpreter, což Vám umožňuje používat jej jako kalkulačku, případně můžete zadávat přímo programové příkazy a tyto jsou ihned vykonány. Tato vlastnost předurčuje interpreter MUMPSu k výukovým účelům. Proto prvotní seznamování s MUMPSem budeme provádět v přímém režimu. Programovací režim bude vysvětlován od kapitoly 5.4.

3.1. Velmi důležité upozornění

MUMPS Vás bude ze začátku trápit tím, že je pro něj v některých případech nesmírně důležité dodržování počtu mezer. Proto si dobře zapamatujte:
- příkaz může začínat na libovolném místě v řádku, ale parametry příkazu musí být od příkazu odděleny právě jednou mezerou; viz příklad:
W "ano",!,"ne"<ET>

W je příkaz pro psaní na obrazovku, za nim je přesně jedna mezera, pak následují tři parametry pro tisk.
- parametry příkazu jsou od sebe oddělovány čárkou a za posledním parametrem následuje bezprostředně stlačení klávesy <ET>, tedy žádná další mezera; viz příklad:
W "ano",!,"ne"<ET>
tři parametry jsou od sebe odděleny dvěmi čárkami a toto vše je bezprostředně následováno stlačením klávesy <ET>.
W "ano, to je ono"<ET>
zde je pouze jeden parametr, protože cokoli je započato uvozovkami a uvozovkami ukončeno, je bráno jako jeden řetězec a čárka zde není jako oddělovač parametrů, ale jako normální čárka v textu.
- použijete-li na řádku další příkazy, pak jsou od parametrů předchozích příkazů odděleny jedinou mezerou.
W "ano" W ! W "ne"<ET>
příkaz W (WRITE) je zde použit třikrát za sebou (což je zdlouhavé oproti původnímu zápisu), nicméně je zřetelné, že za parametry příkazu musí vždy následovat jediná mezera.
- některé příkazy lze používat i bez parametrů, což má speciální význam. V takovémto případě je příkaz následován stlačením klávesy <ET> bez jakékoli mezery, což je shodné jako ukončení běžného příkazu, kdy je klávesa <ET> stlačena bezprostředně za parametry
W<ET>
použijete-li příkaz bez parametrů, pak za ním stlačíte bezprostředně klávesu <ET>
- použijete-li v jednom řádku za příkazem bez parametrů další příkaz, pak tento další příkaz oddělíte od příkazu bez parametrů dvěmi mezerami
W W "ano"<ET>
první příkaz W je bez parametrů a od dalšího příkazu W je oddělen dvěmi mezerami.

3.2. Používání matematických výrazů a příkaz WRITE

Zadáte-li například příkaz WRITE (což lze zkráceně napsat jen jako písmeno velké W nebo malé w), následovaný jednou mezerou a poté například 2+3 (ukončeno stiskem klávesy <ET>)

WRITE 2+3<ET>

Nebo

W 2+3<ET>

pak se na obrazovce objeví v následujícím řádku výsledek 5. Takto lze za příkaz WRITE psát cokoli. Např. texty v uvozovkách (W "pracujete s MUMPSem"), matematické výpočty (viz příklad W 2+3), výpis záznamu z databáze, nebo výpis časti textového řetězce (W $E("jazyk MUMPS",7,11)) což způsobí výpis slova MUMPS. Řadu dalších možností poznáte v průběhu výuky.
První důležitá poznámka se vztahuje k používání matematických výrazů v MUMPSu. Použijete-li jakýkoli přikaz (tedy i příkaz W) následovaný matematickým výpočtem, pak musíte vědět, že není používáno matematických priorit, ale výraz je vyhodnocován zleva doprava. Toto však lze změnit použitím závorek. Vyzkoušejte si následující příklady a zjistíte, že nejde o žádný vážný nedostatek MUMPSu.
W 2+3*5<ET> což dá výsledek 25 a nikoli jak jsme zvyklí 17 W 2+(3*5)<ET>
dá výsledek 17

Jistě zajímavé zjištění pro Vás bude následující výraz.
W "10 koní běžících"*"2 sloni ležící"+3<ET> kdy výsledek tohoto výrazu bude 23
avšak výraz
W "Jan Novák 4."+"jeho 1 dobrý přítel"*5<ET> dá výsledek 0.
Vysvětlení je velmi jednoduché a smysl i význam tohoto tvoří jednu ze základních filosofických myšlenek MUMPSu. Jde o to, že MUMPS přijímá vše jako řetězce, to jest jako posloupnost znaků. Což znamená, že "10

koní" je řetězec stejně jako libovolné číslo (+976.34 nebo -0.25 nebo .345 nebo "23.5" nebo "-1").
U všech řetězců vč. numerických) závisí jejich matematická hodnota na prvním znaku v řetězci. Pak tedy následují
cí řetězce "10 koní" "15dívek " 3.45" " -.25" "list3"
.mají následující matematickou hodnotu:

"10 koní" má hodnotu 10 protože první dva znaky mají numerický význam
"15dívek" má hodnotu 15 -""- -""- -""
" 3.45" má hodnotu 0 protože první znak je mezera, což je nenumericky znak !
" -.25" má hodnotu 0 -""- -""- -""
"list3" má hodnotu 0 protože první znak je "1", tedy nenumerický znak

Matematická hodnota všech čísel je jednoznačná, to znamená:

+976.34 má hodnotu 976.34
-0.25 má hodnotu -0.25
.345 má hodnotu .345
"23.5" má hodnotu 23.5 první znak je numericky
"-1" má hodnotu -1 první znak "-" je numerický

3.2. Používání matematických výrazů a příkaz WRITE

Každý řetězec ve kterém se objeví alespoň jeden nenumericky znak (což může byt i znaménko + či - uvedené za číslem je potřeba uvádět v uvozovkách. Např. "5peněz" "ABCD" " 25" "25.5-5" "123+456"
Jedno ze základních využití matematické interpretace libovolného řetězce bylo naznačeno v uvedeném příkladě. Tedy začíná-li řetězec numerickou hodnotou, pak jej lze použít také k matematickým operacím. Hlavní však je to, že i čislo je vlastně bráno jako řetězec a v celém MUMPSu není potřeba deklarovat číselné, logické, binární, řetězcové a jiné hodnoty proměnným, což velice zjednodušuje používání MUMPSu. Toto mohou zhodnotit programátoři z FORTRANU, PASCALU a dalších jazyků. Jakákoli námitka, že MUMPS vlastně neumí šetřit pamětovy prostor není na místě, což vyplývá z úvodních kapitol této příručky.

Další příklady matematickych výpočtů:
W 10/2*3<ET> = 15
W 10+"5koní"<ET> = 15
W 1+2*3/3*5<ET> = 15

3.3. Práce v přímém příkazovém režimu při používání libovolných příkazů MUMPSu

Názvy příkazů v MUMPSu lze psát ve třech základních modifikacích:
- celé slovo
- první znak
- první dva znaky u příkazů začínajících na Z

viz příkaz WRITE, popisovaný v kapitole 3.1. To znamená, že lze psát WRITE i W. Navíc je možno psát příkazy i malými písmeny, takže WRITE, write, W i w jsou čtyři naprosto shodné příkazy. Z uvedené možnosti zkracování názvů všech příkazů vyplývá, že MUMPS může mít jen 26 příkazů. V podstatě jich má jen 25, přičemž písmeno H je použito ve dvou příkazech (viz kapitoly 5.8 příkaz HANG a 5.9 příkaz HALT), a písmeno Z, které má výsadní postaveni, v šesti příkazech. Příkazy začínající písmenem Z patři do množiny příkazů, které jsou používány v převážné míře pro ladění a editování programů a v programech samotných se většinou

nepoužívají. Tyto příkazy lze zkracovat na první dva znaky, tedy vždy na písmeno Z a ještě jeden znak. Mezinárodní doporučení pro MUMPS jsou taková, aby programátoři nepoužívali příkazy začínající na Z ve svých programech, protože pak nejsou tyto programy přenositelné mezi různými počítači. Doporučeni je velmi rozumné a mělo by platit i pro Vás.

3.3.1 Příkaz WRITE

Příkaz WRITE (W) neboli napsání čehokoli na obrazovku bylo předvedeno v kapitole 3.2. Příkaz WRITE lze ale použit i pro výpis textu na tiskárnu a pro další účely. S tím vším se postupně seznámíte v druhé polovině této příručky, zatím budete postupně pronikat do základních významů jednotlivých příkazů.
Příkaz W má ještě tzv. formátovací znaky. Pomoci těchto znaků lze upravovat tisk na obrazovku, nebo tiskárnu. Jde o následující znaky:
výmaz obrazovky (nová str.na tiskárně) ! tisk na nový řádek
?X posun na pozici sloupce X v řádku
Například napište tento příkaz,

W #,!,!,!,?25,"dobry den"<ET>

a MUMPS odpoví výmazem obrazovky, odřádkováním o tři řádky od vrchu obrazovky (tj. dostane se na čtvrtý řádek) a počínaje 25. sloupcem vypíše text "dobrý den". Cely příkaz lze ještě zjednodušit. Přestože #, ! a ?25 jsou parametry příkazu W, není nutno mezi těmito parametry (platí pouze pro formátovací znaky) dělat čárku. Pak je možno (a bývá to dobrým zvykem) psát příkazy takto:

W #!!!?25,"dobrý den"<ET>

Druhý základní význam příkazu WRITE je WRITE bez parametrů. Tento příkaz Vám vypíše obsah všech proměnných v operační paměti. Vyzkoušejte si následující příklad:

S A=10,B="libovolný text"<ET> W<ET>

Příkazem S jste si do operační paměti uložili dvě proměnné a to proměnnou A a proměnnou B. Příkazem W jste si vypsali obsah těchto dvou proměnných na obrazovku. Stejného účinku jako u příkazu W dosáhnete příkazem VIEW 0 (V 0), viz kapitola 3.3.4.

3.3.2. Příkaz SET a proměnné MUMPSu

rozpoznatelnost proměnných a řetězců v MUMPSu
Příkaz SET (S) se používá pro uložení hodnot do proměnných. Proměnná je pamětová buňka označená jménem. Jméno může být tvořeno až z osmi znaků, přičemž první znak musí byt písmeno nebo procento (A.), další (tj. 2. až 8.) mohou byt libovolné alfanumerické znaky.Obsah proměnné se v průběhu programu může měnit. Proměné v MUMPSu dělíme do dvou skupin. První skupina jsou tzv. lokální proměnné a druhá tzv. globální proměnné. Hlavním a skoro jediným rozdílem je to, že lokální proměnné jsou uloženy přímo v operační paměti uživatele, kdežto globální proměnné jsou uloženy na disku. To, zda jde o lokální či globální proměnnou,je odlišeno pouze jediným znakem a tím je znak stříška (^) před prvním znakem názvu proměnné. Lokální proměnné neobsahuji tento znak, před globální proměnné je ho nutno psát (prostřednictvím znaku ^ říkáme MUMPSu "jdi se podívat na disk").

A B c %mci ACCOUNTR jsou možné lokální proměnné
^A ^B ^C ^%mci ^ACCOUNTR jsou možné globální proměnné

přičemž mohou existovat vedle sebe lokální proměnné a A
a globální proměnné ^a ^A.
Příklad naplnění proměnných. tyto proměnné zadejte do Vašeho počítače, budete je za chvíli potřebovat). První příkaz SET uvádíme vypsán celý, další již zkráceně jako S.

SET A=10<ET> uloží řetězec "10" do lokální proměnné A

S BAA="Adam a Eva"<ET> uloží řetězec "Adam a Eva" do lokální proměnné BAA

S ^HLP="napoveda"<ET> uloží řetězec "napoveda" do globální proměnné ^HLP
S ^A=999.999<ET> uloží řetězec "999.999" do globální proměnné ^A

Napíšete-li příkaz W<ET> nebo V 0<ET> , tak se na Vaši obrazovce objeví ve dvou řádcích pod sebou vypsány:

A "10"
BAA "Adam a Eva"

Globální proměnné ^HLP a ^A se nevypíší, protože jsou globálními proměnnými a jsou tudíž na disku a nikoli v operační paměti.

Lokální i globální proměnné mohou byt indexované, viz následující příklad přiřazení hodnot:

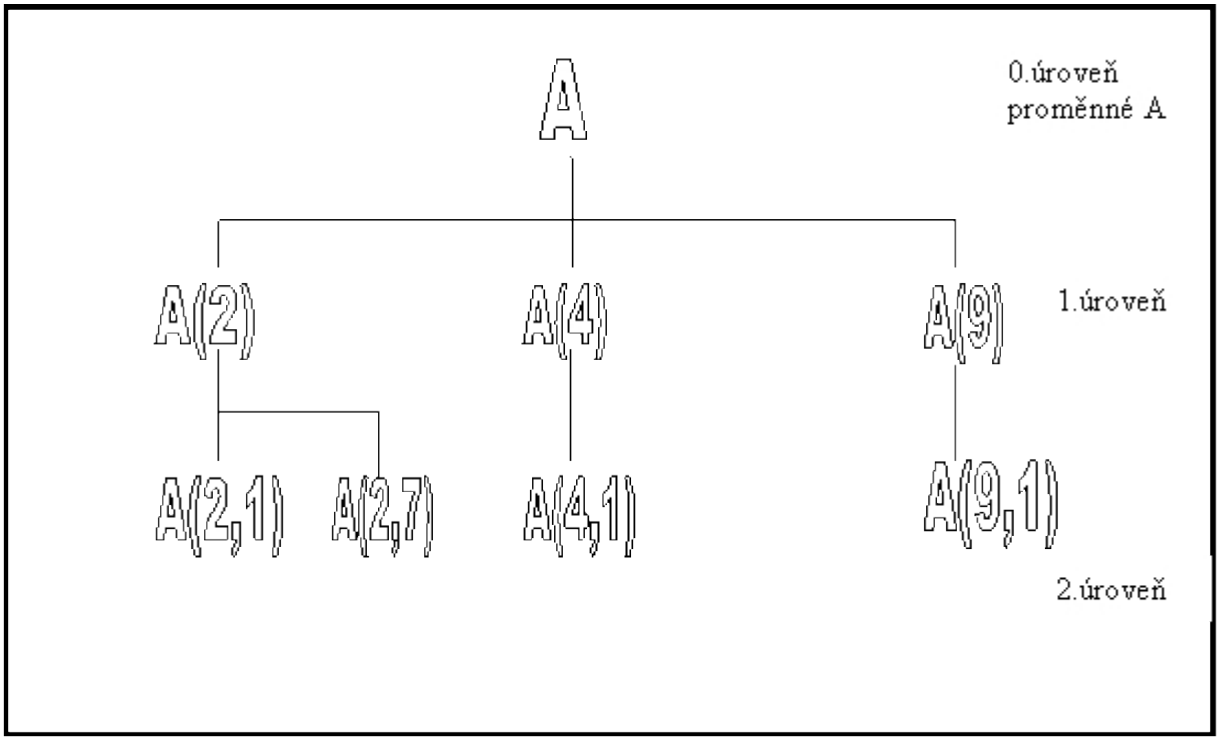
S A(2,1)="Josef Novák#10.11.52#ženaty"<ET>

Tímto způsobem zadáte indexovanou lokální proměnnou A(2,1) a při použití příkazu V 0 se nám tento text neobjeví, pouze před proměnnou A si všimněte hvězdičky. Tato hvězdička vám říká, že lokální proměnná A je indexovaná a vidět je pouze prvek uložený pod proměnnou A bez indexu. Použijete-li příkaz V 1, pak se na obrazovce vypíše vše co patři pod indexované lokální proměnné.

Pak tedy uvidíte:
*A "10"
2,1 "Josef Novák#10.11.52#ženatY"
BAA "Adam a Eva"

Upozornění:
Proměnné, které zadáte například jako A21 nebo ^Al,jsou neindexované proměnné.Index musí byt vždy v závorce.
Interpretace indexovanych proměnných je velice zajímavá a jde o druhou zde předloženou základní myšlenku MUMPSu. Pro vysvětlení bude nejlepší obrázek. Při jeho prohlížení vycházejte z již zadaných proměnných A a A(2,1), k nimž si přidáme další indexované proměnné A. Viz následující příkazy:

S A(2)=1,A(4)=5,A(9)=17<ET>
S A(4,1)="Petr Kubeš#25.06.56#ženatý"<ET>
S A(9,1)="František Havíř#4.04.55#ženaty"<ET> S A(2,7)="Kurzy šití#Kurz počítačů"<ET>



Z obrázku tedy vyplyvá, že indexované proměnné tvoří vlastně strom, a to strom s tolika úrovněmi, kolik indexů je v závorce. To znamená, že A(1,1,1000,22) je vlastně čtvrtou úrovní indexované proměnné A, tedy vlastně čtvrtou úrovní A databázového stromu. Jestliže před všechny proměnné A z předešlého příkladu napíšete znak stříšky viz takto uvedený příklad:

S ^A=10,^A(2)=1,^A(4)=5,^A(9)=17
S ^A(4,1)="Petr Kubeš#25.06.56#ženatý"
S ^A(9,1)="František Havíř#14.04.55#ženaty"
S ^A(2,7) ="Kurz šití#Kurz počítačů",^A(2, 1) ="Josef Novák#10.11.52#ženatý"
pak utvoříte svoji první databázi na disku. Interpretace této databáze může být následující. První indexy mohou být např. čísla pracovníků (teď pozor jde o čísla 2, 4 a 9) a hodnoty uložené na první úrovni pod těmito indexy jsou vlastním obsahem databázových proměnných pod A(2) je 1, pod A(4) je 5 a pod A(9) je 17 . Tyto hodnoty mohou být například číslo oddělení podniku, v němž pracovníci pracují. Druhá úroveň bude interpretována jako jednotlivé záznamy vztahující se k pracovníkům číslo 2, 4 a 9. Přitom pod indexem A(X,1) bude vždy řetězec s údaji:

1. jméno a příjmení
2.datum narození
3. stav

a tyto údaje jsou od sebe vždy odděleny znakem #. Další řetězce jsme zadali až pod index A(X,7),kde je uvedena

organizovanost pracovníka (konkrétně v tomto případě u pracovníka s osobním číslem 2, který se jmenuje Josef Novák jsou uvedeny školení. Prvky jsou opět odděleny znakem #.
Tato vámi vytvořená databáze je vlastně dle tradičního pojetí v programování souborem. Chcete-li se přesvědčit, že jste opravdu vytvořili databázi, pak stačí provést příkaz V 2, který Vám vypíše jména všech databází na disku. Bude tam i Vaše první databáze označená jako A (bez stříšky).

Práce se soubory jako založení souboru, vyhledání věty souboru, uzavření souboru a další práce se soubory bývají většinou jedny z nejtvrdších oříšků při výuce programovacích jazyků. Nyní jste bez jakýchkoli problémů založili soubor a tento můžete kdykoli smazat buď celý, nebo po částech, viz příkaz KILL v kapitole 3.3.5.
V této chvíli by bylo vhodné říci něco ohledně uložení databáze na disku, protože indexy nebyly uváděny po pořádku a řetězce u jednotlivých indexovaných globálních proměnných mají různou délku. Ukládání se děje dle určitých optimalizačních postupů při využívání tzv. řídkých polí. Jde o třetí zde popisovanou základní myšlenku MUMPSu a tou je optimální úspora paměti. MUMPS má ke všem datům přímý přistup, takže po zadání indexu vždy přímo najde místo, kde je uložen požadovaný prvek, nebo místo,kam má daný prvek uložit. Pro představu o uloženi na disku je možno toto zjednodušeně napsat následujícím způsobem.
A=10,A(2)=1,A(2,1)=Josef Nová#10.11.52#ženatý,A(2,7)=ČSOP#R OH,A(4)=5,A(4,1)=Petr Kubeš#25.06.56#ženatý,A(9)=17,A(9,1)=F rantišek Havíř#14.04.55#ženatý
Z uvedeného vyplývá, že neexistující prvky (A(1),A(3),A(5) až A(8), A(1,1 až A(1,7), A(2,2) až A(2,6),A(3,1) až A(3,7) a dále A(4,2) až A(4,7), A(5,1) ... až A(8,7) a ani A(9,2) až A(9,7)) nezabírají na disku žádné místo. Vložíte-li do databáze například prvek:

S ^A(4,3)="3500#21.5#200"

pak se tento vsune mezi prvek A(4,1) a A(9). Navíc se prvky řadí těsně za sebou, takže mezi prvky není zbytečný volný prostor. Jediné omezeni je pro délku řetězce, kdy žádné proměnné nesmí být přiřazen řetězec delší než 255 znaků. Pro jednoznačnost vysvětlení uvedeme příklad. A=123456789 a A(1,3,7)="ano" jsou proměnné, jimž jsou přiřazeny řetězce znaků. V prvém případě je to devět znaků "123456789", v druhém případě jsou to tři znaky "ano". Z výše uvedeného tvrzeni jednoznačně vyplývá, že MUMPS nepotřebuje třídění, protože všechny prvky jsou ihned po založeni tam, kam patří.

Obdobným způsobem,jakým jste založili databázi A,můžete zakládat další databáze. Pro zjednodušení dalšího výkladu si založte druhou databázi, neboli druhý GLOBÁL,jak se říká mezi uživateli MUMPSu. Nazvěte jej globál B, viz příklad:

S ^B("ředitel")=2,^B("ekonomicky nám.")=4<ET>
S ^B("vedouci prov.17")=9<ET>

V tomto globálu použijete jako indexy nikoli čísla, ale řetězce. MUMPS totiž vše interpretuje jako řetězce, takže je mu naprosto jedno, jestli je index zadán jako číslo (viz ^A(2,1 , nebo jako řetězec viz ^B(“ředitel"). Toto platí jak pro globály (globální proměnné), tak pro lokální proměnné. Pro upřesnění je však nutné podotknout, že zápisy proměnných

^B("ředitel") a ^B(ředitel)

nejsou shodné. V prvním případě je v závorce uveden jako index řetězec ředitel, ve druhém případě je v závorce uvedena jako index proměnná s názvem ředitel !! MUMPS jednoznačně chápe vše co je uvedeno v uvozovkách (číslo může být uvedeno bez uvozovek, protože proměnná nemůže začínat číslicí) jako řetězec. Cokoli je uvedeno bez uvozovek MUMPS chápe jako proměnnou (výjimkou je opět číslo, protože proměnná nemůže začínat číslicí).' Jde o jedno ze základních pravidel
MUMPSu.

Nyní zadejte následující příkaz:

W ^B("ředitel")," ",^A(2,1)<ET>

a na obrazovce se vypíše:

2 Josef Novák#10.11.52#ženatý

Toto je názorná ukázka, jak lze k datům v databázích přistupovat. Přístup do databáze jste si nemuseli nijak připravit (to znamená v tradičních programovacích jazycích otevřít soubor a případně jej popsat). Odpadá tedy problém zda můžete pracovat pouze s jedinou databází či třeba se sto databázemi současně (databáze = globál = soubor).
Příkaz S umí zadávat řetězce úspornou formou. Chcete-li uložit do několika proměnných najednou stejný řetězec, pak můžete použít následující formu příkazu:

S (A,B,C,D)=100<ET>

Takto uložíte jediným příkazem do proměnných A, B, C a D řetězec 100.

MUMPS UMOŽŇUJE V KTEROUKOLIV CHVÍLI PŘISTUP KE VŠEM DATŮM
MUMPS JE SYSTEM VIRTUALIZACE PAMETI

Tím, že MUMPS přistupuje ke všem datům a indexům jako k řetězcům, že je ke všem datům prakticky okamžitý přístup a dále tim,že minimalizuje nároky na použití vnějších pamětí, je MUMPS předurčen pro hromadné interaktivní systémy zpracování dat. MUMPS má však spoustu dalších velice důležitých vlastností, které vyplynou z dalšího výkladu.

3.3.3. Příkaz READ

Předchozí kapitola končila tím, že MUMPS je výhodně použitelný pro interaktivní systémy zpracování dat. Dost možná Vás zarazí, že má pouze jediný příkaz pro vstup dat z obrazovky. To znamená, že nemá nějaký generátor * vstupních formulářů jak bývá zvykem. Na tomto místě můžeme pouze konstatovat (v dalším dílu vysvětlit) jak je tato zdánlivá nevýhoda relativní. Je totiž možno používat nejrůznější důmyslné formuláře, tyto je však potřeba naprogramovat jako stavebnicové moduly a pak máte generátor vstupních formulářů dle vlastní představy. Takovýchto generátorů pak můžete požívat několik, vždy podle konkrétních požadavků a nejste omezováni jedním (posléze zjistíte, že špatně použitelným) standardním generátorem.
Příkaz READ (R) je běžně používán následujícím způsobem, viz příklad:
R A<ET>
Po takovémto zadání se blikající čára (dále budeme používat počítačově spisovného kurzor) přesune na začátek dalšího řádku a provokativně bliká, čeká na vstup řetězce z klávesnice. Napište jakýkoli řetězec (počítač se neuráží) a po jeho dopsání stiskněte klávesu <ET>. Poté se na dalším řádku objeví opět znak > a za ním blikající kurzor. Tímto postupem jste zadali řetězec do lokální proměnné A. Obsah proměnné si můžete vypsat známým příkazem

W ^A<ET>,

kdy se Vám na obrazovce znovu objeví text zadany na vyšším řádku.
Použijete-li příkaz V 1<ET>, vypíše se obsah všech lokálních proměnných, které jste dosud zadali. Důležitou informací je, že příkazem R nelze zadávat řetězce do globálů.
Nelze tedy napsat:

R ^A<ET>

Tento pokus bude následován chybovou zprávou a neprovede se. Do globálu lze zadávat hodnotu pouze přes příkaz S, takže příkaz R je omezen pouze na práci s lokálními proměnnými.
Příkaz R používá, stejně jako příkaz W,formátovací znaky pro práci s obrazovkou. Jsou jimi stejné znaky jako u příkazu W. Pro zopakováni:

- # výmaz obrazovky (nová str.na tiskárně)
- ! tisk na nový řádek
- ?X posun na pozici sloupce X v řádku

Navíc je možno vypsat před načítáním proměnné libovolnou zprávu (již lze kombinovat s formátovacími znaky).
Takže například příkaz:
R #!!!?25,"vstup do A : ",A
smaže obrazovku počítače, odřádkuje o tři řádky a odskočí o 25 znaků od levého okraje. Přesně od této pozice vypíše zprávu a za ní bude čekat na zadáni proměnné A.
Upozornění ! Máte-li například na obrazovce popsán řádek a Vy na tomto řádku použijete formátovací znak ?25 (a to jak v příkazu R,tak v příkazu W), pak je 24 znaků smazáno a od 25. znaku je v příkazu R proveden výpis zprávy. MUMPS pak čeká na vstup proměnné A. V příkazu W je od 25. znaku vypisován řetězec.

Další modifikací příkazu R je použití speciálních znaků hvězdička a mřížka (* #). Znak # dovoluje vložit do proměnné pouze tolik znaků, kolik označuje číslo uvedené bezprostředně za ním, viz příklad:
R A#10<ET>
Tento příkaz umožni zadat do proměnné A maximálně 10 znaků. Zadáte-li méně znaků, tak jejich zadání ukončíte normálně stlačením klávesy <ET>. Zadáte-li deset znaků, tak se ukonči vstup řetězce do proměnné automaticky, tedy jako kdybyste stlačili <ET>.

Znak * umožňuje vstup pouze jednoznakového řetězce do proměnné, avšak tento znak zanechá v proměnné pouze svoji ASCII hodnotu. Viz příklad:
R *A<ET>
Stlačíte-li například písmeno velké C, tak po jeho stlačeni se automaticky ukonči vstup a v proměnné A najdete dekadickou hodnotu znaku C dle ASCII tabulky, což je 67. Zkonfrontujte si tento fakt s ASCII tabulkou, která je jako příloha č. 2 této příručky.

3.3.4. Příkaz VIEW

K příkazu VIEW (V) jen několik málo slov. Jde o příkaz, který Vám umožní vypsat obsah lokálních proměnných, jména jednotlivých globálů na disku, adresář programu napsaných pod MUMPSem a adresář disku. Příkaz V se píše pouze jako V mezera a číslo. číslem může být 0, 1, 2, 3 a 4.

V 0<ET>

zobrazí všechny neindexované lokální proměnné a u indexovaných napíše jen znak hvězdičku *, jak bylo ukázáno v kapitole 3.3.2 a vypíše eventuální obsah proměnné na nulté úrovni. Všechny proměnné jsou řazeny dle pozice v ASCII tabulce a dle indexu. To znamená, že proměnná %ZZZ je před proměnnou AAA, protože znak % má nižší hodnotu dle ASCII tabulky, než písmeno A. Proměnná A("Jiří") bude před proměnnou A("Petr") a proměnná A("Petr",1) bude následovat za A("Petr").

V 1<ET>

pracuje obdobně jako V 0, avšak vypíše i obsah všech indexovaných proměnnych.
V následujícím textu k příkazu V se objeví několikrát pojem disk. Vždy jde o vybraný disk, tedy ten disk s nímž pracujete. Máte-li však MUMPS nainstalován na dva různé disky (viz instalace v referenčním manuálu), pak se jedná v případě výpisu názvů globálů o disk na němž pracujete s globály. V případě výpisu jmen programů v MUMPSu či názvů všech souborů na disku, tak jde o disk na němž jste nastaveni pro práci s programy. Nevíte-li, co jsou to soubory na disku, co je vybraný disk, nebo cokoli uvedené v tomto odstavci, a ve vysvětlování příkazu V 2 až V 4 , tak se tím nenechte vyvést z rovnováhy a klidně pokračujte dále ve studiu MUMPSu.

V 2<ET>

zobrazí jména všech globálů na disku. Zobrazeny jsou jen názvy, takže nejsou vidět žádné indexované globální proměnné.

V 3<ET>

zobrazí názvy všech programů napsaných pod MUMPSem a uložených na disku. Jde o soubory s příponou .MMP.
V 4<ET>
zobrazí cely adresář disku, to znamená všechny soubory. Obdoba příkazu DIR v operačním systému.

3.3.5. Příkaz KILL

Příkaz KILL (K) je používán k rušení proměnných a to jak lokálních, tak globálních. Pro jednoduchost vysvětlování si zadejte následující proměnné pomocí příkazu S. A, A(1), A(1,5), A(1,9), A(3) , A(7), s libovolným

obsahem. Po zadání lokálních proměnných (lze říci lokální báze A) si vyzkoušíte dle následujících instrukcí činnost příkazu KILL. Napíšete následující příkaz:

K A(3)<ET>

Příkaz zruší pouze lokální proměnnou A(3). Pravdivost tohoto tvrzeni lze otestovat Vám již známým příkazem W. Zadáte-li W A(3)<ET>, tak MUMPS oznámí, že daná proměnná neexistuje, což znamená, že je opravdu zrušená. Ostatní proměnné si nechejte vypsat například příkazem V 1. Dále zadejte příkaz:

K A(1)<ET>

a MUMPS zruší proměnné A(1) a A(1,5). Zrušeni proměnné A(1) není třeba komentovat. Zrušení proměnné následující za uzlem A(1) lze vysvětlit názorně tak, že jsme uřízli celou větev A(1), včetně všech jejich větviček. V našem případě byl následovník jenom jeden a to A(1,5). Všichni následovníci proměnné A(1) musí mít na místě prvního indexu znak 1 (např. A(1,100), A(1,"Jiří",29), A(1,55,"00123",12,77) atd. To znamená, že všichni tito následovnici by byli smazáni použitím příkazu K A(1). Názorné zobrazení uřezání větve Vám vysvětli obrázek v textu na následující straně.

Použijete-li na původně uvedená data A, A(1), A(1,5), A(1,9), A(3),- A(7) příkaz:

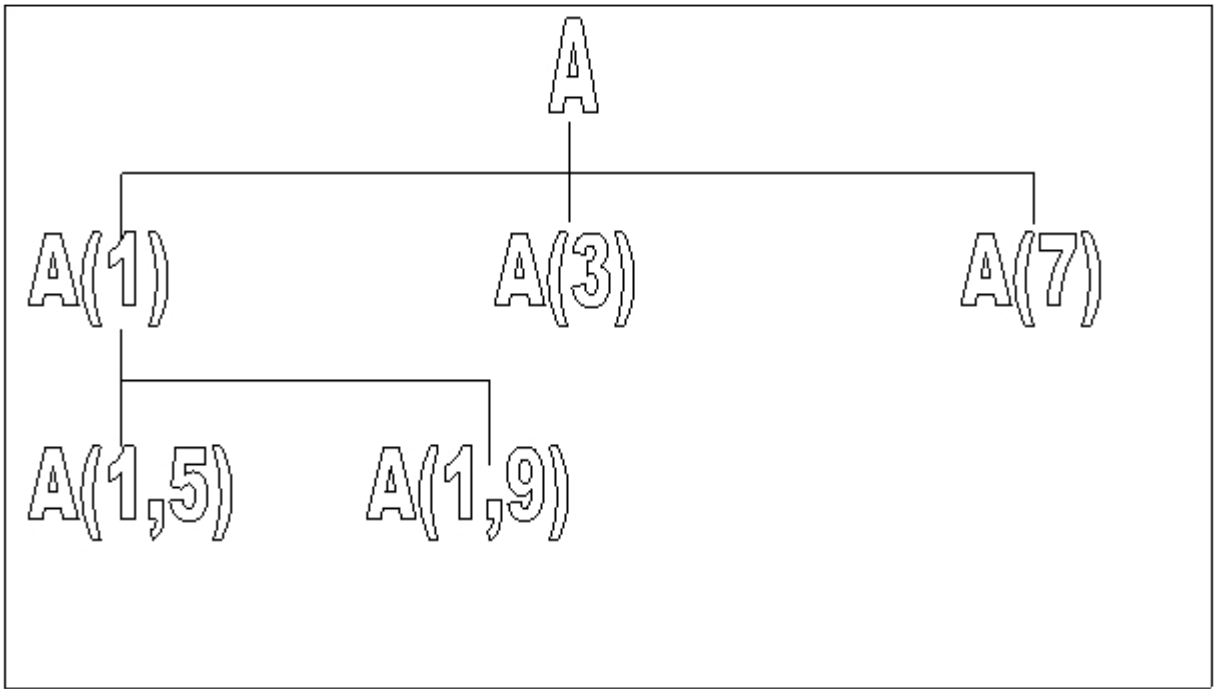
K A(1,9)<ET>

pak zrušíte pouze proměnnou A(1,9) a všechny ostatní proměnné zůstanou zachovány. Použitím příkazu:

K A<ET>

zrušíte celou lokální bázi A. Tato pravidla platí i pro rušení globálních proměnných a jejich částí. Příkaz K může mít i více parametrů než jeden. Například lze rušit jen vybrané části báze A viz příklad K A(1,9),A(3)<ET>, případně i více různých proměnných
jako např.

K A,B,C<ET>.



Dále uvedená struktura příkazu KILL je platná jen pro lokální proměnné. Jde o dvě modifikace příkazu, a to příkaz K bez parametrů a tzv. exkluzívní KILL.
K bez parametrů zruší všechny lokální proměnné, to znamená, že jakmile použijete příkaz K, tak v operační paměti nezůstane žádná dříve definovaná lokální proměnná.
Exkluzívní K ukážeme na následujícím příkladu. Zadejte si několik proměnných (např. A, A(1,5), B, C, ABB).
Nyní zadejte příkaz:
K (A,C)<ET>
a při výpisu například příkazem V 1 zjistíte, že v paměti zůstaly zachovány pouze proměnné A, A(1,5) a C. Tento příkaz nelze použít na globály z pochopitelných důvodů, protože při náhodném použití příkazu jako K (^A,^B), by se Vám zničily všechny databáze, kromě databáze ^A a ^B.

4. Podmiňování příkazů v MUMPSu

Podařilo-li se Vám zdárně projít kapitolou 2 a 3, tedy prostudováním a hlavně odzkoušením příkazů S, R, W, K a V, tak můžete bez problémů pokračovat touto kapitolou.
Dosud jsme Vám předložili dva základní způsoby používání příkazů. Jde o příkaz s parametry, například S A=10 nebo W "ano" a o příkazy bez parametrů. Důležité je dodržování mezer za příkazy, tedy jedna mezera mezi příkazem a parametry, další mezera v tom případě, když za parametry začíná nový příkaz. Následuje-li za příkazem bez parametrů další příkaz, pak jsou tyto dva příkazy odděleny dvěmi mezerami.
Další ze základních způsobů používání příkazů jsou příkazy s podmínkou (velice důležitá varianta používání, která nemá obdobu v běžných programovacích jazycích). Skoro všechny příkazy v MUMPSu umí tento způsob používat. Všechny příkazy, které jste se až dosud učili, mohou byt používány s podmínkou.
Pro ukázku podmíněně vykonaného příkazu si nejprve vyplňte lokální proměnnou A, S A=10<ET> a poté můžete vyzkoušet Váš první podmíněný příkaz (například W).

W:A=10 "ano, A je 10"<ET>

Po vykonáni tohoto příkazu se na novém řádku obrazovky objeví text "ano, A je 10". Zadáte-li například:

W:A=3 "ano, A je 3"<ET>

tak se Vám nevypíše na obrazovce žádný text. Vypsání či nevypsání textu je závislé na podmínce, která následuje příkaz hned za dvojtečkou. Příkaz tedy lze čist takto: napiš na obrazovku řetězec "ano, A je 3", ale pouze je-li lokální proměnná A rovna třem. Protože lokální proměnná A je rovna deseti, tak tisk je proveden pouze v prvním případě. Další příklad uvedeme na příkazu R.

R:A=10 B<ET>

Je-li proměnná A=10, pak je provedeno čteni do proměnné B. Je-li v A jiný řetězec než "10", pak příkaz není vykonán. To, že příkaz není vykonán znamená,že je ignorován a provádí se další následující příkaz. Vyzkoušejte následující příkaz.
S A=20 R:A=20 B#1 W:B="A" "ANO" W:B="N" "NE"<ET>
Po stlačeni <ET> je očekáván vstup jednoho znaku z klávesnice. Stlačíte-li písmeno velké A, tak se na obrazovce vytiskne nápis ANO. Stlačíte-li velké N, tak se vytiskne nápis "NE".
Nyní si celý řádek rozebereme. Příkazem S A=20 jste napinili lokální proměnnou A řetězcem "20". Příkaz R je vykonán jen v tom případě, je-li A=20. Tato podmínka je splněna, takžeMUMPS čeká na zadání znaku. Zadat můžete libovolný jeden znak, avšak tisk bude proveden pouze v případě, že proměnná B je naplněná bud velkým písmenem A, nebo velkým N.
Stejně jako u příkazů R a W je podmínku možno použit i u příkazů K, S a V. Jistě Vás samotné napadne spousta příkladů.
Pro inspiraci uvádíme některé další:

S A=10,B=20,C=30 K:C=30 A,B<ET> S A=10 S:A=10 B=20<ET>
S A=10 V:A=10 1<ET>

U podmíněně vykonávaných příkazů je příkazem nejen název (např. W), ale též dvojtečka a celá podmínka za dvojtečkou. Hlavní věc na kterou nesmíte zapomenout je to, že podmíněný příkaz je od svých parametrů oddělen vždy jednou mezerou a mezi parametry a dalším příkazem je opět jedna mezera. Podmíněný příkaz bez parametrů je od dalších příkazů oddělen dvěmi mezerami. Končí-li příkazový řádek podmíněným příkazem bez parametrů, pak za ním nesmí být žádná mezera. Stejně tak, je-li poslední příkaz bez parametrů, pak za ním nesmí byt žádná mezera!

5. Práce v přímém příkazovém režimu a řídicí příkazy MUMPSu

Tato kapitola se zabývá přikazy pro řízeni cyklů, rozhodování, skoky a volání programů. Další příkaz, kterým se tato kapitola zabývá, je příkaz vykonání obsahu proměnné jako programového řádku. V kapitole je již běžně používáno všech tří základních modifikací příkazů, viz kapitola 4.

5.1. Příkaz FOR pro řízení cyklu

Cyklus FOR (F) v MUMPSu má odlišný charakter od běžných zvyklostí z jiných programovacích jazyků. Vykonávání cyklu se vztahuje pouze k řádku na němž je cyklus vykonáván. Zkušenější možná namítnou, že jeden řádek v programu nic neznamená a tudíž cyklus v jednom řádku je nesmysl. Jestliže si ale uvědomíte, že:

- délka příkazového řádku v MUMPSu může být až 252 znaků
- příkazy lze psát jedním písmenem
- z cyklu lze skákat na podprogramy

tak zjistíte, že do jednoho řádku lze vměstnat celkem solidní program. Pro první odzkoušení zadejte následující příkazový řádek.

```
F I=2:1:10 W I," ",I*I,!<ET>
```

Nyní si rozebereme činnost tohoto příkazového řádku. Příkaz cyklu F má řídící proměnnou I. Řídící proměnná může byt pouze lokální, tedy nikdy globální. 2:1:10 znamená, že na počátku bude do proměnné I vložena dvojka a proběhne první cyklus, po provedení všech příkazů následujících na řádku bude řídicí proměnná cyklu I zvětšena o jednu,a tak to půjde stále dokola až do hodnoty deset. Jakmile bude hodnota řídící proměnné cyklu deset, tak se naposledy vykonají všechny příkazy na řádku a cyklus skončí. Po ukončení cyklu zůstane v řídící proměnné hodnota posledního kroku cyklu, to je deset. To, že příkaz bude tisknout čísla od dvojky do desítky pod sebe a ve druhém sloupečku druhou mocninu daného čísla by Vám již mělo být jasné. Pokud ne, tak nahlédněte do kapitoly 3.2 a 3.3.1. Po ukončení cyklu bude MUMPS čekat na další příkaz, eventuálně v programovém režimu přejde na vykonávání nového řádku. Následující příklad Vám ukáže jak jinak lze zadávat hodnoty řídící proměnné cyklu.

```
S ^A(2,1)=3,^B=2,C(3)=10<ET>
F A=-2:1.2:2,8:1:11,"ano","ne",^A(2,1):^B:C(3) W A,I<ET>
```

Tento příklad je už na první pohled podezřelý. Nebudeme Vás napínat jako ve špatné detektivce. Nejde o vloženy cyklus, ale o jeden cyklus s pěti parametry. Řídící proměnná cyklu postupně nabývá hodnot prvního, druhého atd. až posledního parametru (v tomto případě pátého). První parametr je -2:1.2:2, což znamená, že řídící proměnná bude postupně nabývat hodnot -2, -0.8, 0.4 a 1.6 . Poté se uplatní druhý parametr a řídící proměnná bude nabývat hodnot 8, 9, 10 a 11. Pak řídící proměnná nabude postupně třetí parametr, tj. řetězec "ano" a dále čtvrtý parametr, tedy řetězec "ne". Pátý parametr vloží do řídicí proměnné A obsah globální proměnné ^A(2,1). Cyklus bude dále prováděn v kroku, jehož hodnota je
v globální proměnné ^B,a to tak dlouho, dokud řídící proměnná A nenabude hodnotu rovnající se, nebo nejbližší nižší než je v lokální proměnné C(3). Páty parametr bude tedy nabyvat hodnot 3, 5, 7 a nakonec 9. Po ukončeni cyklu bude v řídící proměnné A číslo 9 (lépe by bylo říci jednoznakový řetězec "9"). Na tomto místě jste se mohli opět přesvědčit o tom, že MUMPS chápe vše jako řetězce, takže řídící proměnná cyklu nabývá hodnot nikoli však s numerickou interpretací, ale s řetězcovou interpretací.
Následující poznámka k příkazu F Vám představí nekonečny cyklus, který je nutno zastavovat podmínkou uloženou na řádku za příkazem cyklu.

```
F I=5:1 Q:I=10 W I,I<ET>
```

Po spuštění tohoto příkazového řádku se budou na obrazovce postupně ve sloupci pod sebou zobrazovat čísla 5 až

9. Příkaz F pracuje opět tak, že nejprve naplní řídící proměnnou cyklu první hodnotou za rovnítkem, tedy číslem 5 a pak přičítá k řídící proměnné stále číslo jedna. Přítrž tomuto nekonečně se opakujícímu cyklu je schopen vykonat pouze nějaký odskok, nebo příkaz QUIT. Tento podmíněný příkaz QUIT ukončí cyklus ve chvíli, kdy zjistí, že podmínka je splněna,
tedy že řídící proměnná cyklu nabyla hodnotu deset. Poté se již nebude vykonávat zbytek příkazů na řádku. Příkaz QUIT bude blíže vysvětlen v kapitole 5.6.
Nejspíš Vás již napadla otázka, jak utvořit vložený cyklus. Toto je předvedeno v následujícím příkaze:

F I=1:1:5 F J=1:1:3 W I,J,!<ET>

Příkaz zobrazí na obrazovce sloupec čísel 11, 12, 13, 21 ... až 51, 52, 53 a poté skončí. Drobný program na ukázku tří vložených cyklů je uveden v příloze čislo 3. Jeho studium si však radši ponechte na konec výuky.
Poslední informací k používání příkazu F je to, že jeho vykonání nelze podmínit. Nelze tedy napsat příkaz:

F:A=1 I=1:1:10<ET>

Takovýto příkaz je nesprávný a odezvou je chybová zpráva. Příkaz F nelze použit bez parametrů, to je opět chyba.

5.2. Příkaz IF pro podmíněné vykonání části programu, speciální systémová proměnná $TEST a časovaný READ,zkracování názvu systémových proměnných a tři logické operátory &, ! a '

Rozhodovací příkazy IF a ELSE jsou používány opět nezvyklým způsobem oproti běžným programovacím jazykům. Jejich provádění není společně svázáno jako běžné IF s THEN a ELSE, ale pracují zcela autonomně a většinou je ani nelze společně používat.

5.2.1. Speciální systémová proměnná $TEST a Časovaný READ

Základní princip práce příkazů IF a ELSE je spojen se speciální systémovou proměnnou nazvanou $TEST ($T . Tato systémová proměnná je nastavovaná časovaným příkazem READ na hodnotu 0 nebo 1 a dále též příkaz IF může nastavit hodnotu proměnné $T na 0 či 1. Hodnotu proměné $T však nelze nastavit příkazem S ani R. Zkratka $T je zkrácený název speciální systémové proměnné $TEST, protože zkracovat lze nejenom příkazy, ale také speciální systémové proměnné. Princip zkracování je naprosto shodný se zkracováním příkazů. První znak všech speciálních systémových proměnných je $ a za nim následuje první písmeno z názvu proměnné. Nyní si vyzkoušejte následující příkaz:

W $T<ET>

5.2.1. Proměnná $TEST a časovaný READ

Po odeslání tohoto příkazu Vám MUMPS slušně vysvětlí (pro-střednictvím chybové zprávy), že takovouto proměnnou nezná. A má naprostou pravdu, protože doposud nebyla proměnná $T žádným způsobem nastavena. Nyní zadejte příkaz:

R A:5<ET>

a pak nic nezadávejte. Kurzor bude chvíli čekat na vstup řetězce do proměnné A a po určité chvíli mu dojde trpělivost a ozve se MUMPS (prostřednictvím znaku >), že je přichystán přijímat další příkazy. Vysvětlení parametru A:5 je následovné. MUMPS čeká zhruba 5 vteřin (čas je závislý od rychlosti, kterou pracuje

mikroprocesor) na zadáni řetězce do proměnné A. Po uplynuti čekací doby (nebyla-li stlačena klávesa <ET>) se ukončí čekání na vstup řetězce do proměnné A a MUMPS čeká na další příkazy. Nyní zadejte:

R A,I,B:5,1,C<ET>

a můžete zadávat. Vstup do proměnné A bude trpělivě čekat dokud nestisknete <ET>, nebo dokud nezadáte řetězec a <ET>. Nyní je proveden přeskok na nový řádek a očekávání vstupu do proměnné B. Vstup <ET>, nebo řetězec a <ET>, ukončí toto očekávání. Pokud se však vstup neuskuteční, tak je automaticky vstup. do proměnné B po cca pěti vteřinách ukončen a opět je trpělivě očekáván vstup do proměnné C.
Další příklad Vám ukáže vztah časovaného R k proměnné $T. Zadejte časovaný vstup do proměnné A, avšak vyčkejte až se ozve MUMPS, tedy nezadávejte řetězec do A, a neukončujte vstup klávesou <ET>.

R A: 5<ET>

pak zadejte příkaz:
W $T," text",A<ET>
a MUMPS Vám odpoví vytištěním následujícího textu: 0 text
Z tohoto plyne, není-li v průběhu časového období časovaného příkazu R stlačeno <ET> , tak proměnná $T nabude hodnoty 0. Bude-li zadán řetězec do proměnné A, avšak v časovém období nebude stlačeno <ET>, pak v proměnné A sice řetězec bude,ale v $T bude opět 0. Takže pozor, vztah k proměnné $T je jen na základě stlačeni <ET> a nemá žáhou souvislost s řetězcem v proměnné A. Na druhé straně, jestliže v časovém limitu stlačíte <ET> nebo zadáte řetězec a stlačíte <ET>, tak v proměnné $T bude čisto 1.

5.2.2, Příkaz IF

Konečně lze přistoupit k příkazu IF I). Provedte následující příkazovy řádek. Avšak pozor Příkaz I je zde bez parametrů, takže musí byt následován dvěmi mezerami a pak teprve příkazem W.

R A:100 I W "A obsahuje ",A," a bylo stlačeno <ET>"<ET>

Po jeho spuštěni zadejte nějaký kratší řetězec a stlačte <ET>. Příkaz W za příkazem I se provede, protože časovany příkaz R nastaví proměnnou $T na hodnotu 1. Provedte tento příkazový řádek znovu (nyní však čas zkraťte na 5 tj.
R A:5) a po spuštění příkazu nestlačujte <ET>. Přitom je možno nějaký řetězec napsat nebo nenapsat. Po uplynutí časového období se příkaz ukončí a nebude provedena část řádku za příkazem I, protože proměnná $T je nyní nastavena na hodnotu 0.

Příkaz I dovede také nastavovat hodnotu proměnné $T. Ukázka je v následujícím příkazu I s parametry. Zadejte:

S A=10 I A=10 W "ano"<ET>

Dobře si všimněte, že v příkladě je příkaz I s parametry, takže je zde jedna mezera mezi I a parametry, a dále mezi parametry a příkazem W. Příkaz I se zde provede, protože logická podmínka je splněna a právě tato logická podmínka způsobí to, zda obsah proměnné $T bude 0 či 1 (zde je 1), což lze zjistit příkazem W $T. Vypíše se hodnota 1.
Velmi důležitou poznámkou je to, že podmínky příkazu I lze řetězit, což znamená, že je možno použít i několik parametrů. Pro ukázku si napište příkaz se třemi parametry:

S A=10,B=20,^C=30<ET>
I A=10,B=20,^C=30 W "ano"<ET>

Příkaz I funguje tak, že je vyhodnocována jedna podmínka za druhou. Ve chvíli, kdy některá z řady podmínek není vyhodnocena jako pravdivá, je do proměnné $T vložena hodnota 0 a příkaz je ukončen, zbytek řádku se neprovede. Jsou-li však splněny všechny podmínky, pak v proměnné $T je hodnota 1 a zbytek řádku se provede. Uvedený příkaz lze nahradit naprosto shodnými příkazy. Viz náledující dva příkazy:

I A=10 I B=20 I ^C=30 W "ano"<ET>
I (A=10)&(B=20)&(^C=30) W "ano"<ET>

Příkazy jsou však delší a navíc jejich vykonávání zabere MUMPSu více času ! Znak & viz kapitola 5.2.3.
Vykonání příkazu I, stejně tak jako příkazu F, nelze podmínit pomocí dvojtečky. Nelze tedy napsat příkaz:

I:A=10 (B=20),(^C=30) W "ano"<ET>

Takovýto příkaz je nesprávný a způsobí chybu. Chyba je signalizována chybovou zprávou.
Poslední informace k příkazu I. Není-li nastavena hodnota systémové proměnné $T, tak se příkaz I bez parametrů vyhodnotí vždy tak, jako by byla v proměnné $T nastavena 1, přičemž ani po provedení příkazu I nebude proměnná $T nastavena ! Tato poznámka se vztahuje pouze k příkazu I bez parametrů (parametry příkazu I totiž vždy nastaví obsah systémové proměnné $T).

5.2.3. Tři logické operátory &, ! a '

Znak & v posledním příkazovém řádku znamená AND,to jest "a zároveň". Interpretace poslední modifikace příkazu I pak zni: proved zbytek řádku za podmínky, že A=10 a zároveň
B=20 a zároveň ^C=30.

MUMPS umí používat tři logické operátory. Jsou jimi operátor & (AND), ! (OR) a operátor ' (NOT).
Operátor & byl již vysvětlen na příkladu. Pro úplnost lze dodat, že AND znamená splnění podmínky před nim a za nim. Například (A=10) je podmínka před AND a (B=20) je podmínka za AND. Protože W jsou splněny, znamená to pro další logické AND, že podmínka před ním je splněna a podmínka za ním, což je v našem případě (^C=30) je splněna také.
Dalším operátorem je ORLze jej překládat jako slovo "nebo". Následující příklad je s příkazem I.

I (A="JAN")!(B=20)!(^C=32.5) W "ano"<ET>

MUMPS pak interpretuje příkaz takto: je-li A="JAN", nebo B=20, nebo ^C=32.5 tak provede zbytek řádku.
Znamená to tedy, že zbytek řádku bude proveden jestliže alespoň jedna podmínka bude splněna.
Posledním logickym operátorem je ' (NOT). Tento operátor vysvětlíme na následujících příkazech:

I A=20 W "A je 20"<ET>
I A'=20 W "A není 20"<ET>
I A>20 W "A je větší než 20"<ET>
I A'>20 W "A je rovno, nebo menší než 20"<ET>
I A<20 W "A je menši než 20"<ET>
I A'<20 W "A je rovno, nebo větší než 20"<ET>

Pro lepší vysvětlení byla použita relační znaménka =, < a >. MUMPS používá logické NOT k negacím relačních operátorů, protože neumí používat spojení jako <= (menší nebo rovno), <> (různé od ~, >= (větší nebo rovno).
Takže pak relace A'=20 znamená A je různé od 20 atd. viz texty v příkazovych řádcích uvedených příkladů.
Operátor NOT lze využít ve spoustě dalších případů, což bude ukázáno v průběhu další výuky.

5.3. Příkaz ELSE pro podmíněné vykonání části programu

Příkaz ELSE (E) je, co se vyhodnocování týká obráceným pólem příkazu IF. Zbytek řádku za příkazem E se provede jen tehdy, když systémová proměnná $T bude mít hodnotu 0 (nebo když nebude proměnná $T vůbec nastavena, což je shodné s příkazem I). Příkaz E, na rozdíl od příkazu I, se používá pouze ve variantě bez parametrů, viz příklad:
S A=10 I A=15 W "A=15"<ET> E W "A není rovno 15"<ET>

Po provedení obou řádků se na obrazovce objeví pouze nápis z druhého příkazu, tj. A není rovno 15. Příkaz za I na prvním řádku se nevykonal, protože proměnná $T byla vyhodnocením podmínky A=15 naplněna číslem 0.Příkaz za E se vykonal, protože proměnná $T je nastavena na 0. Z uvedeného jasně vyplývá, že příkaz E je bytostně spjat se systémovou proměnnou $T. Použití příkazu E bez nastavené $T je nekorektní činnost, která však nevyvolá chybovou zprávu a provede další příkazy za příkazem E.
Paradoxním případem použiti příkazů I a E je to, když není nastavena proměnná $T a příkaz I, je použit bez parametrů (E pochopitelně též bez parametrů). V takovémto případě budou vykonány činnosti za příkazy I a E 1 v případě, že jsou oba příkazy na společném řádku.
Vykonání příkazu E nelze podmínit dvojtečkou s podmínkou. To znamená, že nelze napsat příkaz:

E:A=10 W "ano"<ET>

Takovýto příkaz je nesprávný a odezvou je chybová zpráva.

5.4. Příkaz GO pro provádění skoků bez návratu, zápis krátkých zkušebních programů,informace o návěštích a poznámkách v MUMPSu a znaky ^S,^C

MUMPS umožňuje prováděni skoků v programech, a to prostřednictvím příkazu GO. Na tomto místě by tedy bylo vhodné povědět si něco málo o programech a o jejich tvorbě.

5.4.1. Zápis programu do paměti počítače, používání návěští a poznámek v programu

Program v MUMPSu lze do počítače zadávat několika způsoby. Nyní Vám ukážeme ten nejjednodušší, ale zároveň nej-pracnější způsob. Zadávejte následujíc programové řádky s tím, že tam kde uvidíte napsáno <TAB> stlačíte klávesu označenou jako TAB

ZAC<TAB>;pokus1<ET>
<TAB>S A=0<ET>
Nl<TAB>F I=1:1:10 W !,I<ET>
<TAB>S A=A+1 W "konec ",A G Nl<ET>

Nyní zadejte příkaz ZP<ET> a na vlastní oči se přesvědčte co jste provedli. To,co jste napsali pomocí klávesy <TAB>,je váš první program. Není veliký, ale pro účely vysvětleni příkazu G naprosto postačující. Nejprve tedy jak je to s tím zadáváním programu do paměti. První řádek začíná návěštím ZAC a za ním jste stlačili klávesu <TAB>. Dále jste napsali středník a text pokus1. Toto vše jste uložili jako programový řádek stlačením klávesy <ET>. Tim, že jste napsali řetězec a za ním stlačili klávesu <TAB>,jste.MUMPSu oznámili, že zadáváte programový řádek s návěštím.
Návěští je řetězec o maximální délce 8 znaků, přičemž první znak může být libovolný alfabetický znak nebo znak %. Další znaky mohou byt libovolné alfanumerické znaky, případně může byt jako návěští použito číslo bez desetinné tečky o délce maximálně 8 znaků. Návěští musí začínat na první pozici v řádku, to znamená,že mu nesmí

předcházet žádná mezera, ani znak <TAB>. Návěští je ukončeno stlačením klávesy <TAB>. Po stlačení klávesy <TAB> pokračují buď jednotlivé příkazy, nebo může být použita poznámka.
Před poznámkou se vždy píše středník. Vše co je za středníkem se neprovádí, jde pouze o komentář. Poznámku lze psát i za příkazy. V případě psaní poznámky za příkaz bez parametrů, je potřeba mezi středníkem a příkazem vynechat 2 mezery.
Druhý řádek Vašeho programu nezačíná návěštím, takže nejprve stlačíte klávesu <TAB>,kterou MUMPSu oznámite, že následující text je program a nemá byt bezprostředně vykonán. Dále napíšete příkaz (v tomto případě S A=0) a poté stlačíte klávesu <ET>.
Z uvedeného vyplývá, že programové řádky se do paměti zadávají pomocí klávesy <TAB>, před kterou může anebo nemusí být návěští. Lze tedy použít pouze dvě varianty zadávání programových řádků. Napsat, a to hned od kraje řádku bu
ď

1. návěští, <TAB> (nikoli mezeru) a pak zbytek řádku, nebo
2. <TAB> (nikoli mezeru) a pak zbytek řádku. Řádek je vždy nutno ukončit stlačením <ET>.

Výmaz programu z paměti se provádí napsáním editačního příkazu ZREMOVE. Jde o příkaz začínajíce na písmeno Z, takže jeho zkráceny zápis jsou dvě písmena (viz kapitola 3.3). V dalším textu budeme běžně používat zkráceně ZR. Program si zatím nemažte, protože nejprve se naučíte nejjednodušší způsob opravy řádků.
Pomocí příkazu ZR totiž můžete vymazat i jednotlivé řádky. Napíšete-li například příkaz:
ZR N1<ET>, připadně ZR ZAC+2<ET>,
tak smažete řádek programu v paměti, konkrétně řádek:

```
N1      F I=1:1:10 W !,I
```

Přičemž ZAC+2 znamenă druhy řádek za řádkem s návěštím ZAC+0 smazáni řádku se můžete přesvědčit po napsáni příkazu:

```
ZP<ET>
```

čímž se Vám vypíše program uloženy v paměti na obrazovku. Na místo vymazaného řádku teď vložte novy programový řádek. Napíšete-li nyní například příkazový řádek:

```
N1<TAB>F I=1:1:10 W !,I*I<ET>
```

tak bude uložen opět na místě řádku smazaného. Další vkládaný řádek by se pak dostal až za tento řádek. Příkazy ZR a ZP v budoucnu asi nebudete využívat a programy budete psát některým editorem MUMPSu, nebo úplně jinými textovými editory.
Z tohoto důvodu není potřebná perfektní znalost několika dalších příkazů jako je ZINSERT, ZMOVE a ZDELETE. Zkušební programy budou vždy jen krátké, takže není potřeba podrobné znalosti těchto příkazů. Vhodné bude zapamatovat si příkaz ZP, který Vám vypíše program uložený v paměti.

5.4.2. Příkaz GO pro provádění skoků bez návratu a znaky ^S, ^C

Příkaz skoku GO (G), stejně tak jako příkaz DO je snad příkaz s největší škálou modifikací. Základní význam příkazu je skok na návěstí uvedené jako parametr. Tento skok může byt směrován jak na návěští v daném programu,tak na návěští jiného programu,viz kapitola 5.4.3. V uvedeném programu je prováděn neustále skok na návěští N1 a program takto běhá v nekonečném cyklu.0 nápisu konec je vždy možno přečíst číslo, po kolikáté se program opakuje. Program spusťte příkazem:

D ZAC<ET>

a program se rozběhne v již popsaném nekonečném cyklu. Chcete-li zastavit toto řádění, pak jen pouze stlačením a podržením klávesy označené <CTRL> (nebo <STAG> dle typu klávesnice) a stlačením znaku <S>, nebo <C>. Stlačení kláves <CTRL> a <S> je běžně označováno jako ^S, stlačeni kláves <CTRL> a <C> je běžně označováno jako ^C. Stlačením ^S se pouze přeruší vykonávání programu, stlačením libovolné další klávesy se program opět rozběhne. Stlačením ^C se provádění programu ukončí a MUMPS se ozve znakem připraven, tj. >.Znak ^ zde nemá vztah ke globálům.
Nyní proveďte další příkazy:

ZR N1+1<ET>
<TAB>R "konec A/N ",ANO I (ANO="N")!(ANO="N") G N1<ET> ZP<ET>

Tímto je program změněn a vypsán na obrazovku. Vypadá takto:

```
ZAC   ;pokus1
      S A=0
N1    F I=1:1:10 W I*I
      R "konec A/N",ANO I (ANO="N")!(ANO="n") G N1
```

Máte-li na obrazovce cokoli jiného, tak napište příkaz ZR<ET> a vložte program znovu výše uvedeným postupem, avšak již s novým posledním řádkem. Program spusťte příkazem:

D ZAC<ET>
nebo příkazem:

G ZAC<ET>

Výsledek bude naprosto shodný, protože program se rozběhne v obou případech od začátku. Od této chvíle se bude program chovat zdvořileji,protože Vám po každém cyklu umožní rozhodnout se zda jej ukončíte, nebo zda může pokračovat. Jak je z posledního řádku patrno, jestliže je splněna podmínka za příkazem 1, program skočí (dík příkazu G Nl) na návěští N1. Takto pracuje příkaz G v té nejjednodušší variantě.
Skok může byt směrován též na řádek, který nemá návěští. I takto lze spouštět program, nebo skákat v programu. Příkaz pro spuštění Vašeho programu od druhého řádku pak vypadá následovná:

G ZAC+1<ET>
V uvedeném programu je možno změnit poslední řádek například tak, že příkaz G N1 přepíšeme na G ZAC+2. Toto provedete pomocí následujících příkazů:

ZR N1+1<ET>

<TAB>R "konec A/N",ANO I (ANO="N")!(ANO="n") G ZAC+2<ET>
takže výsledkem bude následující program:

```
ZAC   ;pokus1
      S A=0
N1    F I=1:1:10 W I*I
      R "konec A/N",ANO I (ANO="N")!(ANO="n") G ZAC+2
```

Program bude skákat na druhý řádek za řádkem s navěštím ZAC, tedy na řádek s návěštím Nl. Program můžete

upravit tak, že návěští N1 (které nyní nepotřebujete) vynecháte. Viz následující příkazy:

```
ZR ZAC+2<ET>
<TAB>R I=1:1:10 W !,I*I<ET>
```

Opravený program musí vypadat takto:

```
ZAC   ;pokus1
      S A=0
      F I=1:1:10 W !,I*I
      R "konec A/N",ANO I (ANO="N")!(ANO="n") G ZAC+2
```

Program bude tedy skákat na třetí řádek, to je na druhý řádek za řádkem s návěstím ZAC.
Vykonání příkazu G lze podmínit. Znamená to, že lze použít následující zápis příkazu:

```
      G:A<10 ZAC+2
```

Nyní si upravte program do následující podoby:

```
ZAC   ;pokusí
      S A=0
      S A=A+1
      W !,A G:A<1O ZAC+2
```

Tento program vypíše pouze sloupec čísel od 1 do 10 a sám ukonči svoji činnost, protole za čtvrtým řádkem nejsou již další příkazy.
Příkaz G lze použit i na vícenásobné větveni, viz příklad:

```
G N1:A=10, N2:A=20, N3: B=" ano", KONEC:^C (1, 5) ="ne"
```

Jednotlivé parametry příkazu G jsou odděleny čárkou a parametr se skládá ze dvou částí. První částí je návěští, na které bude proveden skok v případě, že podmínka za dvojtečkou bude splněna. Druhá část je podmínka za dvojtečkou. Rozepsání podmínek pak vypadá takto:

```
N1:A=10
N2: A=20
N3:B="ano"
KONEC:^C(1,5)="ne"
```

Tento příkaz je interpretován následovně. Jdi na návěští N1, jestliže lokální proměnnA A je rovna 10 (obsahuje řetězec "10"), nevykonáš-li předcházející skok na návěští N1, tak testuj zda A=20. Nevykonáš-li ani tento skok na návěstí N2, tak pokračuj stejným způsobem dále. Není-li splněna žádná z uvedených podmínek, tak program pokračuje dalším příkazem uvedeným za tímto příkazem A. Je-li splněna kterákoli z uve- dených podmínek (jedna i vísce), tak je řízení přeneseno na návěští s první splněnou podmínkou. Bude-li například A=20 a B="ano", tak bude skok proveden na návěští N2.
Největší lahůdkou používání příkazu G je podmíněný příkaz G s vícenásobným větvením. Pro pochopení tohoto příkazu zadejte do počítače následující program (před jeho psaním smažte předcházející program příkazem ZR).

```
ZAC ;POKUS
      R "konec prace? A/N ",KONEC#l Q:(KONEC="W")!(KONEC="a")
      R !,"vstup 1,2,3 ",KAM#l
      W !,"Zadáno cislo "
      G:(KONEC="N")!(KONEC="n") Nl:KAM=1,N2:KAM=2,N3:KAM=3
      W "není v rozsahu 1 - 3, nebo KONEC není N",! G ZAC
```

```
N1      W 1,1 G ZAC
N2      W 2,! G ZAC
N3      W 1,1 G ZAC
```

Program se nejdříve zeptá zda chcete ukončit práci, nebo zda budete pokračovat. Vaši odpověd si uloží do proměnné KONEC (vstup je umožněn jen jednomu znaku). Je-li v proměnné KONEC jiný znak než velké či malé A, tak program pokračuje. Třetí řádek umožní vstup jednomu znaku do proměnné KAM. čtvrtý řádek programu vždy vytiskne text "Zadano cislo". Patý řádek je nejzajímavější. O tom, zda příkaz G bude vůbec vykonán rozhodne podmika G:(KONEC="N")!(KONEC="n"). V případě jejího splnění jsou testovány postupně podmínky jednotlivých parametrů. Odskok je pak vykonán na první návěští, jehož podmínka je spiněna. Bude-li například v proměnné KAM číslo 2, pak parametr N1:KAM=l neni spiněn a je testována podmínka v druhém parametru, to je KAM=2. Tato podmínka je splněna a program skočí na návěští N2. Po vytištění čísla 2 odřádkuje a vrátí se na návěští ZAC.
Nesplnění podmínky za příkazem G na pátém řádku, předá řízení na šestý řádek a po vytisknutí textu v příkazu W "neni v rozsahu 1 - 3, nebo KONEC není N" je provedeno odřádkování, a pak se provede skok na návěští ZAC.
Nesplněním žádné z podmínek parametrů pokračuje program na šestém řádku, opět vytiskne text a vrátí se na návěští ZAC.

5.4.3. Uložení programu na disk a zavedení programu z disku do paměti, skok GO do programu na disku

MUMPS umožňuje mít v paměti počítače vždy jen jeden program. Chcete-li skočit na návěští jiného programu, pak tento jiný program je vždy uložen na disku. Pro vyzkoušení skoku na návěští jiného programu je potřeba uložit program na disk. Uloženi provedete pomocí příkazu ZSAVE (ZS) s uvedením jména programu. Jméno programu je tvořeno maximálně z 8-mi znaků, přičemž první znak musí být písmeno nebo procento (%), další znaky mohou být libovolné alfanumerické znaky.
Pravidla pro tvorbu názvů programů jsou shodná s pravidly pro tvorbu názvů proměnných i programových návěští (u návěští může být použito navíc celociferné číslo jako jméno návěští).
Příklad uložení programu s názvem POKUS: ZS POKUS<ET>
Nyní můžete vymazat program z paměti počítače příkazem ZR. Obrácený příkaz k příkazu ZSAVE, je příkaz ZLOAD (ZL) pro nahrátí programu z disku do paměti počítače. Program si můžete kdykoliv nahrát z disku do paměti příkazem:

ZL POKUS<ET>

Takto zavedený program si můžete prohlédnout použitím příkazu ZP.

5.4.3.1 Ukládání a zavádění programu

Nyní Vám ukážeme, jak je možno skočit do programu, který je uložen na disku. Protože budete skákat do programu jenž jste si uložili na disk, tak si pro věrohodnost dalšího příkladu vymažte paměť pomoci příkazů:

ZR <ET>

Pozor, za příkazem ZR jsou dvě mezery. Nyní proveďte příkaz:

G ^POKUS<ET>

a spustí se program POKUS. Za příkazem G je mezera a znak ^ , pak teprve následuje jména programu. Znak ^ znamená, že jde o program na disku a nikoli o návěští programu v paměti. Znak ^ tu má stejnou funkci jako u rozlišení Lokální a globální proměnné. Je-li program na disku, tak ho spustíme za pomoci znaku ^ napsaného před jménem programu, toto neplatí u pří kazů ZS a ZL). Je-li program v paměti nahraný příkazem ZL nebo právě napsaný), tak jej spustíte skokem na návěští, nikoli skokem na název programu ! Například ukončíte-li činnost uvedeného programu zadáním znaku A, tak jej znovu spustíte příkazem:

G ZAC<ET>

Program uložený na disku však lze spustit i od libovolného návěští, a to přímo bez zavádění příkazem ZL. Pro věrohodnost pokusu si opět zrušte program v paměti příkazem ZR, a zadejte následující příkaz:

G ZAC^POKUS<ET>

kterým se spustí program POKUS. Skok je směrován na návěští programu, za návěštím je znak ^ a pak teprve následuje jméno programu. Pro jednoduchost zapamatováni si představte,jak by vypadalo vypsání parametru obráceně (G ^POKUSZAC). Tento příkaz říká spusť program POKUSZAC, protože znak ^ znamená, že jde o program na disku. Vy však nemáte na disku program POKUSZAC, takže půjde o chybné volání neexistujícího programu. Z těchto pohledů působí znak ^ jako informace o tom, že je volán program z disku a zároveň jako oddělovač názvu programu a návěští.

5.5. Příkaz DO a podprogram

Příkaz DO (D) je obdobný jako příkaz G, a to jak syntaxí, tak využitím. Největší a nejdůležitější rozdíl je to, že po provedeni volaného programu nebo podprogramu se řízení vrací na příkaz bezprostředně za příkazem skoku D. Tímto jsme chtěli velmi zjednodušeně přiblížit příkaz D, což může byt považováno za programátorské rouhání, avšak k takovému
zjednodušeni dojde po čase každý.
Základní použití příkazu D bylo již použito v kapitole 5.4.2. Jde o sputtěni programu, který je nahrán v paměti počítače. Příkaz vypadá následovně:

D ZAC<ET>

Takovýto příkaz spustí program v paměti od návěští ZAC. Když budete chtít spustit program který není v paměti počítače, pak uvedete název spouštěného programu (před názvem znak ^ , protože jde o program na disku).
Příkaz bude vypadat takto:

D ^POKUS<ET>

Příkazem spustíte program POKUS (program bude nejprve načten z disku), a to od prvního řádku programu. Chcete-li program spustit od jiného než prvního řádku, pak musíte v příkazu uvést i jméno návěští, od něhož se má program spustit. Například:

D ZAC^POKUS<ET>

nebo:

D ZAC+1^POKUS<ET>

V prvním případě bude program spuštěn od prvního řádku,
v druhém případě od druhého řádku (do paměti však bude zaveden vždy celý program).
Hlavni těžiště využití příkazu D je však v programování. Pro další seznámení s příkazem D si opět zadejte zkušební program a nazvěte jej například POKUSD. Použijete-li pro tento program opět jméno POKUS, tak se přepíše Váš původní program s názvem POKUS na disku novým programem POKUS. MUMPS neumožní uložit dva programy stejného jména (toto de facto neumožní MS-DOS).

Program vznikl modifikací programu z kapitoly 5.4.2.

```
ZAC ;POKUSD
        R "konec prace? A/N ",KONEC#1 Q:(KONEC="A")!(KONEC="a")
        R !,"vstup 1,2,3",KAM#1
        W !,"Zadané číslo
        D:(KONEC="N")!(KONEC="n") Nl:KAM=1,N2:KAM=2,N3:KAM=3
        I (KAM<1)!(KAK>3) W "není v rozmezí 1- 3",! G ZAC      W:(KONEC'="N")!(KONEC'="n") " neplatí,
KONEC není N",!
        G ZAC
        ;podprogramy
N1      W 1,1 Q
N2      W 2,! Q
N3      W 3,! Q
```

Na desátém až dvanáctém řádku jsou tři podprogramy. Podprogramy jsou jednoduše řečeno samostatné úseky programu, zapsané v daném programu, nebo uložené na disku jako samostatné programy s vlastním názvem a eventuelně i s vlastními návěštími.Podprogramy jsou volány v průběhu práce řídícího programu.V programu POKUSD jsou vlastně tři podprogramy. Podprogramy začínají návěštím a někde v těle podprogramu, případně na jeho konci je příkaz návratu, což je v MUMPSu příkaz QUIT. Ukázkou podprogramu z programu POKUSD je například:

```
N1      W 1,! Q
```

Podprogramem je možno nazvat i samostatný velký program, který je volán jiným programem. Takovýto podprogram je volán svým vlastním jménem, případně i návěštím.
Následující řádky popisují činnost zkušebního programu POKUSD. Program se nejdříve zeptá zda chcete ukončit práci, nebo zda budete pokračovat. Vaši odpověď si uloží do proměnné KONEC (vstup je umožněn jen jednomu znaku). Je-li v proměnné KONEC jiný znak než velké či malé A, tak program pokračuje. V tomto místě zadejte velké či malé N,pro názornost příkladu. Třetí řádek umožni vstup jednomu znaku do proměnné KAM. čtvrtý řádek programu vždy vytiskne text "Zadáno číslo". Pátý řádek je nejzajímavější. Avšak pozor, zde jsou již změny proti vyhodnocováni příkazu G. Zda příkaz D bude vůbec vykonán rozhodne podmínka D:(KONEC="N")!(KONEC="n").V případě jejího splnění jsou testovány postupně podmínky jednotlivých parametrů. Odskok je pak vykonán na první návěští, jehož podmínka je splněna. Bude-li napfiklad v proměnné KAM čislo 2, pak parametr N1:KAM=1 není splněn a je testována podmínka v druhém parametru, to je KAM=2. Tato podmínka je splněna a program skočím na návěští N2. MUMPS si však zapamatuje místo z kterého odskočila po vytištění čísla 2 a odřádkování vykoná příkaz Q (QUIT . Příkaz Q způsobí návrat na další parametr příkazu D, tedy na parametr N3:KAM=3. Podmínka tohoto parametru (KAM=3) není splněna, takže nebude vykonán odskok na návěští N3. Šestý řádek nebude opět vykonán, protože proměnná KAM je v intervalu 1-3. Sedmý řádek rovněž nebude v tomto případě vykonán, protože proměnná KONEC obsahuje znak N. Na osmém řádku bude vykonán příkaz G ZAC.
Zadáte-li v otázce "konec práce? A/N " jiná písmena než požadovaná (A, a, N, n), nebo u vstupu čisla jiné než požadované, pak bude vytištěna buď zpráva v šestém řádku a dále bude proveden skok na návěští ZAC, nebo bude vytištěna zpráva ze sedmého řádku a proveden odskok na ZAC až z osmého řádku.
Na začátku této kapitoly (5.5) jsou napsány příkaz jako D ZAC, D POKUS, D ZAC^POKUS a D ZAC+1^POKUS. Tyto příkazy je rovněž možno použít přímo v programu. Ještě jednu modifikaci příkazu D jsme nepřipomněli, je to příkaz D s jednou podmínkou. Příkaz vypadá například takto:

```
D:A=10 POKUS<ET>
```

Budete-li mít v proměnné A řetězec "10", pak spustite program POKUS z disku.

5.6. Příkaz ukončení QUIT a zřetězování proměnných

Příkaz QUIT (Q) byl již vlastně představen ve všech nejdůležitějších případech svého využití. Jde o ukončení viz kapitola 5.5 a dále ve funkci ukončení cyklu, viz kapitola 5.1.
Příkaz Q lze použít bez podmínky i s podmínkou. Při používání příkazu Q je vždy potřeba pamatovat na to, že jde o příkaz bez parametrů (vynechávat dvě mezery před následujícím příkazem nebo před středníkem s poznámku).
Použiti příkazu Q ve vložených příkazech F si zaslouží samostatný příklad. Napište si následující příkazový řádek:

```
F I=1:1:5 F J=1:1 Q:J=S W I,J,!<ET>¨
```

Příkaz Q ve vnořeném cyklu ukončuje vždy jen ten cyklus, za nimž se nachází a předává řízení předchozímu cyklu. V tomto příkazovém řádku ukončuje cyklus řízený proměnnou J a předá
vá řízení cyklu s řídící proměnnou I. Znamená to, že neukončí práci celého příkazového řádku, ale jen jeho vnořené

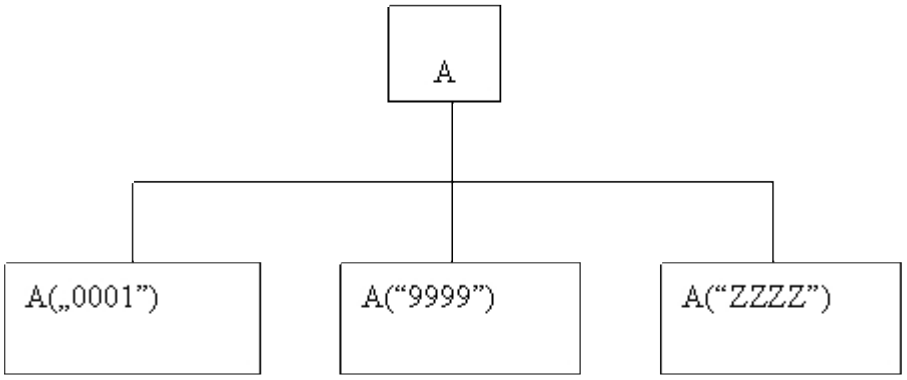
části.
Prostudováním předcházejících kapitol jste se dostali k prvnímu příkladu, který je i na distribuční disketě. Tento program se jmenuje PRIKLAD1.MMP. Program je zpracován pomocí čtyř příkazů (R Q S G) a logicky operátor OR (!) v příkazu konce. Příklad obsahuje znak ! ve dvou variantách. První varianta je použita v příkazu R jako formátovací znak pro odřádkování, druhá varianta je použita v příkazu Q jako logický operator OR. Další znak, který je použit v MUMPSu s více významy je znak #. Jeden význam je opět v příkazu R jako formátovací znak pro smazání obrazovky, druhy význam znaku # je v témže příkazu pro povoleni maximální délky řetězce. Tyto dva znaky a jejich významy si dobře zapamatujte, protože se v MUMPSu běžně používají. V dalším výkladu se dozvíte, že dva významy mají i znaky otazník (? a hvězdička (*).
V programu je dále použit znak podtržení Jde o tzv. zřetězovací znak. Jeho význam spočítá v tomhle spojuje řetězce (zřetězuje). Jako příklad lze uvést následující příkaz:

```
S A-"123"_"456",B=A_" 789"<ET>
```

Obsah proměnné A je řetězec "123456" a obsah proměnné B "123456 789".
program PRIKLAD1.MMP

```
ZAC    ;ZALOZENI DATABAZE,MP,
       ;založeni pracovníků do We A
       DALSI          R #!!!!,"ZALOZIT DALSIHO PRACOVNIKA A/N ",DALI
Q:(DALSI="N")!(DALSI="n")
OSCIS          R 111,"zadejte osobni cislo pracovnika ",OSCIS#4 R 1111,"jmeno : ",JMENO#20
       R 1,"prijmeni : ",PRIJMENI#20
       R l,"narozen : ",DATNAR#8,1,"stav : ",STAV#10
ULOZ S A(OSCIS)=JMENO_" "_PRIJMENI_"#"_DATNAR_"#"_STAV
       G DALSI
```

Program nakládá údaje JMENO, PRIJMENI, DATNAR a STAV ve zřetězené formě do databáze ^A. údaje jsou v databazi uloleny pod čísly pracovníků, takže vytvářenou databazi lze zobrazit takto:



Program pracuje v cyklu od návěští s názvem DALSI až po poslední řádek. O ukončeni cyklu je rozhodováno na řádku s návěštím DALSI a na řádku následujícím. Na řádku s návěštím OSCIS se načítá osobní číslo pracovníka o maximálně čtyřech znacích do proměnné OSCIS. Na dalších třech řádcích je proveden vstup údajů do lokálních proměnných JMENO, PRIJMENI, DATNAR a STAV. Tyto proměnné zřetězuje a ukládá do indexovaných globálních proměnných ^A s indexy OSCIS. Z uvedeného příkladu je též patrno, že návěští a proměnné mohou mít stejné názvy (DALSI, OSCIS). Nevýhodou tohoto programu je skutečnost, že jednou založené číslo pracovníka můžete přepsat novými údaji, aniž byste toto zjistili. Tato a další úpravy jsou provedeny v následujícím textu,

jakmile se seznámíte s potřebnými funkcemi.

5.7. Příkaz XECUTE pro vykonání příkazu z proměnné

Příkaz XECUTE (X) je velice zajimavý a výkonný příkaz. Pro první seznámení s ním si napište následující příkazy:

```
S A="F I=1:1:10 W I,!"<ET>
X A<ET>
```

Po zadáni těchto dvou příkazů se spustí cyklus tisku hodnoty řídící proměnné cyklu I, a to od čísla 1 do 10 po jedné. Vytiskněte si proměnnou A. Najdete v ní řetězec, který je vlastně příkazem F I=1:1:10 W I,!. Příkaz X tedy vykonal příkaz, který byl uložen v proměnné A. V proměnné může byt uložen řetězec o celkové délce až 255 znaků, programový řádek však může mít jen 252 znaků. Do libovolné proměnné (lokální i globální) je tedy možno uložit řetězec o délce až 252 znaků a ten provést jako program. Takto lze též zpracovat celé velké programy a uložit je v databázi. Jako malou ukázku si napište následující příklad:

```
S A="F I=1:1:10 X B",B="F J=1:1:2 W !,I,J"<ET>
X A<ET>
```

Takto vlastně spustíte programový řádek, který přepsán vypadá následovně:

```
F I=1:1:10 F J=1:1:2 W !,I,J
```

Vykonání příkazu X lze podmínit, viz příklad:

```
S C=10 X:C=10 A<ET>
```

V příkazu X lze použít i několik argumentů za sebou, to znamená , že lze spouštět za sebou různé příkazy. Jako příklad uvádíme následující příkazy:

```
S A="S X=10",B="W X"<ET>
X A, B<ET>
```

Teď si můžete vyzkoušet zajímavou činnost příkazu X. V přímém příkazovém režimu zadejte následující příkaz:

```
F I=1:1 R !,"zadejte příkaz MUMPSu",!,RAD,! X RAD<ET>
```

Nyní můžete zadávat libovolné příkazy MUMPSu a tyto budou vždy po stlačení klávesy <ET> vykonány. V případě chybného zadáni se vrátíte do přímého příkazového režimu. Pro ověření zadejte například následující řádky:

```
S A=10 W ABET>
F J=1:1:10 W J,I<ET> D ^POKUSD<ET>
```

5.8. Příkaz HANG pro pozastavení běhu programu

Příkaz HANG (H) pozastavuje vykonáváni činnosti MUMPSu po stanovenou dobu. Příkaz:

H 5<ET>

Pozastaví vykonávání jakékoli činnosti MUMPSu na 5 vteřin. pozastaví je stejně jako u časovaného příkazu R relativní jednotkou a závisí na rychlosti s níž pracuje mikroprocesor. Využití tohoto příkazu je například pro vysvícení nápovědy na obrazovce po určitý časový úsek. Příkaz H je možno podmínit. Lze tedy napsat:

H:A=10 3,5<ET>

Příkaz je splněn, když lokální proměnná A obsahuje řetězec "10". Splnění příkazu je pozastaveni běhu programu na cca 8 vteřin. Osm vteřin vznikne jako součet obou parametrů, tedy 3 + 5.' Příkaz HANG začíná na stejné písmeno jako následující příkaz HALT. Oba dva příkazy se zkracuji na písmeno H. Jak tedy poznat kdy jde o příkaz HANG a kdy o HALT ? Není to tak těžké, jak se Vám nyní možná zdá. Příkaz HANG má smysl jen s parametry (to jest s určením časového úseku pro pozastavení činnosti MUMPSu). Příkaz HANG tedy nelze použít bez parametrů. Na rozdíl od tohoto je příkaz HALT možno použít pouze bez parametrů, protože příkaz HALT nemám parametry. Pro zdůraznění: podmínka za dvojtečkou není parametr. Parametr je vždy od příkazu (to jest včetně příkazu s podmínkou za dvojtečkou) oddělen jednou mezerou a dale čárkami od ostatních parametrů vztahujících se ke stejnému příkazu. Takže u příkazu H:A=10 3,5 je H:A=10 příkaz a 3,5 jsou 2 parametry příkazu H:A=10. Viz kapitola 4 a kapitola 3.1.

5.9. Příkaz HALT pro ukončení programu a práce MUMPSu

Příkaz HALT (H) je příkaz bez parametrů. Lze jím zastavit vykonávání programu a vrátit se do přímého příkazového režimu, nebo jim lze ukončit práci pod MUMPSem. To, zda příkaz ukonči program nebo práci pod MUMPSem, záleží na režimu z kterého je volán. Zadáte-li příkaz H v přímém příkazovémvém režimu, tak ukončí práci pod MUMPSem a vrátí se do operačního systému MSDOS. Zadáte-li tento příkaz v programu, pak je prováděný program zastaven a ohlásí se MUMPS znakem > což znamená, že se nacházíte v přímém příkazovém režimu. Příkaz H přitom zruší všechny návratové adresy volání programů a podprogramů, takže volá-li první program pomocí příkazu D další program a v tomto volaném programu se provede příkaz H, tak se ukončí činnost programu a MUMPS je v přímém příkazového režimu. Program již nelze spustit od místa, kde byl přerušen (na rozdíl od přerušení způsobeného ladícím příkazem BREAK, který bude vysvětlen v další kapitole).Příkaz H v přímém režimu použijete například takto:

S A=10 W " konec programu " H:A=10<ET>

Po naplnění proměnné A řetězcem "10" a po vytištění textu "konec programu " se ukončí práce pod MUMPSem a vrátíte se do operačního systému. Použijete-li tento stejný příkazový řádek v programu, tak ukončíte činnost programu a vratíte se do přímého příkazového režimu. Jak je z příkladu patrno, lze vykonání příkazu H podmínit (H:A=10). Opět zdůrazňujeme, že jde o podmíněné vykonání příkazu H. Podmínka A=10 je tedy součástí příkazu a nikoli parametrem. Parametry by musely být odděleny od samotného příkazu mezerou.

6. Příkaz BREAK a ZGO ladění programů příkazy BREAK a ZGO,zamezení zastavení programu uživatelem

Program v MUMPSu lze vcelku pohodlně ladit používáním ladících příkazů. První ladící příkaz je BREAK (B). Tento příkaz se používá pro zastavení práce programu. Druhy příkaz je ZGO (ZG) , který se používá k pokračování v programu zastaveném příkazem B.

Nejprve si napište příklad, na němž budeme demonstrovat používání příkazů B a ZG. Demonstrační příklad je

sestaven pouze z příkazů S a ladících příkazů B. Příkazy S jsou záměrně psány s jedním parametrem.

```
ZAC S A=10 B  S B=20 S C=30 S D=40
S E=50 S F=60
S G=70 S H=80
S I=90  B:I=90 "ano, I=90" S J=100
```

Sami jste již určitě poznali, že příkaz B je v prvním řádku programu použit bez parametru a ve čtvrtém řádku s parametrem. Příkaz bez parametru pouze zastaví prováděni programu a na obrazovce zobrazí řádek a místo, kde se program zastavil. Spustíte-li demonstrační příklad, pak se na obrazovce vypíše následující zpráva:

```
BREAK ---
ZAC S A=10 B   S B=20 S C=30 S D=40
ZAC^               *
```

Text na prvním řádku Vám oznamuje, že v programu byl proveden příkaz B. Druhý řádek je opis celého příkazového řádku, ve kterém byl proveden příkaz B. Hvězdičkou je označeno místo, kde bude program pokračovat po zadáni příkazu ZG. Třetí řádek oznamuje název návěští a eventuelně za znakem ^ název programu,v němž byl příkaz B vykonán. V této chvíli si může te vypsat proměnné, které program naplnil. Příkaz W odhalí, že je zatím naplněna pouze jediná proměnná, a to A. Program nyní spustíte dále například příkazem:

```
ZG 1<ET>
```

Příkaz ZG 1 umožní vykonání pouze jediného příkazu v řádku, to jest příkaz S B=20. Na obrazovce se znovu objeví:

```
BREAK ---

ZAC S A=10 B   S B=20 S C=30 S D=40
ZAC^                       *
```

takže podle polohy hvězdičky je zřejmé, že byl vykonăn jen jeden příkazu. Toto si opět ověřte příkazem W. Dále byste mohli obdobným způsobem postupovat příkazu po příkazu. Nyní vyzkoušejte následující modifikaci příkazu ZG.

```
ZG 2<ET>
```

Příkaz ZG 2 provede zbytek řádku a první příkaz dalšího řádku. Poté se opět zastaví a podá zprávu. Postup by bylo opět možno opakovat. Nyní zadejte poslední variantu příkaz ZG, a to variantu bez parametrů.

```
ZG<ET>
```

Nyní se program rozběhne a zastaví se až na příkaze B (v našem případě navíc zkontroluje podmínku zda I=90; v případě nesplnění podmínky by se pochopitelně nevykonal). Výpis na prvním řádku bude vypadat takto:

```
B R E A K --- ano, I=90
```

Další řádky budou vypadat stejně jako u všech předchozích zpráv. Znovu napište příkazu ZG a program se ukončí. Na konci výkladu k ladění programů pomocí příkazů B a ZG, bychom chtěli podtrhnout následující. Pokud se testovaný program nachází v přerušení příkazem B, pak není možno vykonavat činnosti jako opravy v programu

atd. Účinek příkazu B lze eliminovat buďto stlačením ^C (<CTRL> a <C>), nebo normálním ukončením programu. ^C stlačíte v přímém příkazovém režimu, čímž ukončíte působení příkazu B.
Příkaz B může zajistit, aby uživatel nemohl zastavit běh programu jinak než korektním ukončením. K tomu louží speciální parametr příkazu B. Zadáte-li příkaz

B 0<ET>

tak nelze stlačením ^C vyvolat přerušení programu.Příkaz B 0 se používá jako první příkaz programu. Tím se zabezpečí, aby uživatel nemohl náhodně či záměrně ukončit činnost programu. Odblokování tohoto příkazu je možné příkazem:
B 1<ET>
Poté je již možno stlačením ^C přerušit program.

7. Práce s obrazovkou a tiskárnou

MUMPS nemá žádné speciální příkazy pro práci s obrazovkou a tiskárnou. Výpis na tiskárnu je umožněn po provedení příkazů OPEN a USE. Tyto příkazy se v MUMPSu používají na tzv. otevřeni vstupní a vystupní cesty (OPEN),a dále k nastaveni právě používané vstupní či výstupní cesty (USE). Příkazy O a U mají široké uplatněni pro přenosy dat mezi počítači, případně umožňuji přístup k sekvenčnim souborům formátu SDF (tj. skupin ASCII znaků oddělené čárkami a konec řádku označen CR a LF). Tyto možnosti jsou však vázány na další znalosti a podrobně jsou rozebrány v druhé příručce.

7.1. Práce s tiskárnou

Zůstanete-li u obrazovky a tiskárny, tak Vám zatím stačí vědět, jak vypnout vypis na obrazovku a jak zapojit výstup na tiskárnu. Výstup na tiskárnu se zapojí po vykonání příkazů:

O 1 U 1 W zbytek příkazového řádku <ET>
a zároveh se tímto vypne vystup na obrazovku. Příkaz říká, aby bylo otevřeno výstupní zařízení číslo 1 (0 1), což je tiskárna a dále, aby toto zařízeni bylo použito pro všechny příkazy W (toto říká příkaz U 1), a to až do doby než bude proveden příkaz C 1 (což platí jen pro programový režim), u přímého příkazového režimu do ukončení příkazového řádku. Všechny příkazy W použité v programovém režimu pak vysílají řádky k tisku přímo na tiskárnu a konec této činnosti, včetně přepojení výstupu na obrazovku je vykonán po příkazu C 1. Bude-li v programu například takováto sekvence příkazů:

```
TISK   O 1 U 1
W "tisk na tiskárnu",!!!
W ?30,"další řádek tisku"
C 1
```

pak bude na tiskárně proveden tisk dvou řádků. Formátovací příkaz přitom mají stejnou funkci jako při práci s obrazovkou, viz kapitola 3.3.1. To znamená, že po tisku textu "tisk na tiskárnu" se provede odřádkování o tři řádky a text "další řádek tisku" bude vypsán až od třicáté pozice v řádku. Vyzkoušíte-li zadat tyto příkazy v přímém režimu, tak pro tiskárnu bude použit pouze první řádek a odřádkování, druhý příkaz W bude vykonán již jen na obrazovce. Při výpisech na tiskárnu se nelze vracet v řádku zpět,takže příkazový řádek:

O 1 U 1 W ?30,"abcd",?20,"111"<ET>

vypíše na jeden řádek text "abcd", a protože nová pozice tabulátoru je nižší než předcházející, tak ji ignoruje a text "111" vypíše přímo za předešlým "abcd".

Většina tiskáren umí používat různé okrasné znaky, zvýrazněná písma a eventuálně grafiku. Toto vše je závislé na znalosti řídících znaků pro danou tiskárnu. Znáte-li tyto znaky, pak lze použít speciální varianty příkazu W, ktera Vám umožní vlastnosti tiskárny využít. Jde o příkaz W s hvězdičkou. Chcete-li například na tiskárně vypsat text, který bude mít dvojnásobnou šířku písmen, tak na velkém množství tiskáren uspějete s pomocí tohoto příkazu:

O 1 U 1 W *27,*33,*40,"dvojnásobná šířka tisku !”<ET>

Příkaz W vykoná vyslání řídících znaků 27, 33 a 40, a vyslání vlastního textu. Tiskárna použije řídící znaky k nastavení požadovaného tisknu a pak teprve tiskne text obsaženy v příkazu W. Řítící znaky musí být napsány v dekadickém tvaru, vždy po jednomu, oddělené čárkou a před každým řídicím znakem musí být hvězdička. Hvězdička před číslem (čísla od 0 do 255) říká, že bude vyslán jeden osmibitový znak.

7.2. Práce s obrazovkou a nepřímé volání přes znak @

Obrazovka není tzv. sekvenční zařízení jako je tiskárna, takže na ni lze vytvářet pohyby všemi směry. Pohyby jsou zprostředkovány prostřednictvím řídících znaků operačního systému MSDOS. Tyto jsou vysílány na obazovku naprosto stejným způsobem jako u tiskárny, tedy s použitím hvězdičky. Daleko jednodušší a rychlejší je využívání předdefinovaných řídících sekvencí. Takovéto předdefinované sekvence jsou v programu CURSOR. Jejich využiti je velice jednoduché, takže další výklad k pohybu po obrazovce věnujeme jim. Nejprve je však potřeba seznámit se s několika základními potřebami při práci s obrazovkou. Jde o mazání obrazovky, a to buďto celé (na toto má MUMPS řítící znak #), nebo jen výmaz řádku. Dále je nutné nastavení kurzoru na pozici zadanou jako souřadnice řádku a sloupce, změna barvy písma a pozadí, výmaz znaku, změna jasu a několik dalších. Výběr nejdůležitějších z nich je vypsán v následující části programu CURSOR.

```
CURSOR    ;definice funkci cursoru
          S CLS=11*27,""(2J"""
          S CLL=11*27,"[K""
          S BEEP="*27,*7”
          S CUP="*27,*91,DY+1,"";"",DX+1,""H"""
          S COLOR="*27,*91,DY+1,"";"",DX+1,""H"""
          S BS="*27,*8"
          S HI="*27,""[1m"""
          S LI=”*27,””[0m"""
          S BLINK="*27,*91,5,""m"""
END       S DEL="*27,*8,"" "",*27,*8"
COLOR     S BILCERV="S Z=37 W @COLOR S Z=41 W @COLOR"
          S ZELCERV="S Z=32 W @COLOR S Z=41 W @COLOR"
```

Chcete-li ve svých programech využívat možnosti programu CURSOR, pak musíte tento program spustit. Po spuštění na-přiklad příkazem D "CURSOR se Vámi do paměti počítače uloží proměnné, jejichž obsahem jsou výše uvedené řetězce. V následujícím textu se můžete seznámit s jejich používání. Všechny příkazy jsou volány příkazem W. Vyzkoušejte si následující příkazy.

W @CLS<ET>

Příkaz vymaže obrazovku počítače. Poprvé se tu setkáváte se znakem @ (tzv. zavinuté a). Tento znak je používán pro nepřímé volání (tzv. indirekcionální).Pro názornost popišeme interpretaci příkazu W @CLS. Příkaz W vysílá všechny své parametry na obrazovku (obrazovka se používá vždy, když není použito jiné výstupní zařízení). Parametr @CLS se skládá ze znaku @ a proměnné CLS. Proměnná CLS obsahuje řetězec, který lze vypsat pomocí příkazu W CLS. V případě, kdy před proměnnou je znak @ je obsah proměnné vykonán, což lze přirovnat k příkazu,

W *27,*91,*50,*74<ET>

eventuálně i k takto zkrácenému příkazu:

W *27,"[2J"<ET>

Zkrácení je možné z toho důvodu, že *91 je vysláno jako jeden znak, což je znak [. Stejně tak *50 je znak 2 a *74 je znak J.
V proměnných CUP a COLOR jsou použity další proměnné. Hodnoty těchto proměnných musí být zadany před použitím příkazů. Příkaz CUP (CUrsor Position) používa obsah proměnných DX (sloupec 0 - 79) a DY (řádek 0 - 23), příkaz COLOR používá proměnnou Z pro zadáni barvy popředí - znaků (čísla 31 - 37),eventuálně pro zadáni barvy podkladu (čísla 41 - 47). Nové nastavení barvy platí pro každý další výstup na obrazovku (tedy jak pro příkazy W, tak pro příkazy R). Nastavení barvy platí až do její nejbližší změny.
Příklad použití řídícího řetězce v proměnné CUP může být následovný:

S (DX,DY)=10 W @CLS,@CUP,"Tady MUMPS ! "<ET>

Příkaz S přiřadil proměnné DX=10 a DY=10. S (DX,DY)=10 je zkrácená forma příkazu S. Příkaz W provede nejprve výmaz obrazovky (použitím @CLS), pak nastavení kurzoru na řádek číslo 10 a sloupec číslo 10 (použitím @CUP).Nakonec se vykoná poslední argument, tedy výpis textu "Tady MUMPS ! ".Výpis textu bude umístěn na řádku číslo 10 a od sloupce číslo 10.

Řídicí řetězec v proměnné CLL (CLear Line) umožní vymazání řádku od aktuální pozice kurzoru až do konce řádku W @CLL).

Řídicí řetězec v proměnné BEEP vyloudí zvuk z tónového generátoru zvuků (W @BEEP).

Řídicí řetězec v proměnné HOME nastaví kurzor do levého horního rohu obrazovky
(W @HOME).

Řídicí řetězec v proměnné BS umistí kurzor o 1 krok zpět, přitom nevymaže žádný znak (W @BS).

Řídicí řetězec v proměnné HI umožňuje zvýraznění jasu (W @HI), naproti tomu řídicí řetězec v proměnné LI vypíná všechny nastavené barvy a jas. Výsledkem použití řídícího
řetězce v proměnné LI je bílé písmo o snížené intenzitě jasu na černém pozadí. Řídících řetězců v proměnných HI i LI je použito pro každý nově vypsaný znak na obrazovku, a to až do doby dalším změny. Změnou je myšlena změna barev, či změna jasu. Změnou barev se nezruší nastavení HI. Řídicí řetězec v proměnné LI se používá takto - W @LI.

Řídicí řetězec v proměnné BLINK rozblikS vtechny nově vypisované znaky na obrazovce. Platí pro všechny vstupující znaky, a to až do doby nel bude použito LI.

Řídicí řetězec v proměnná DEL provede 1 krok zpět a vy-male znak z pozice, na níž vstoupil.

Používání předdefinovanych barev je stejné jako používání předchozích příkazů. Chcete-li použít například zelené písmo na červeném pozadí, tak použijte příkaz:

W @ZELCERV<ET>

Další vstupující znaky budou již zelené na červeném pozadí. V řetězcích předefinovaných barev jsou dvakrát za sebou použity řetězce stejné jako v proměnné COLOR. V řetězcích však již není požadována proměnná, jsou tam zadány konstanty.

8. Funkce MUMPS

MUMPS, stejně jako ostatní vyšší jazyky obsahuje několik funkcí. V následujícím textu uvedeme třináct funkcí, což představuje všechny funkce potřebné. Dvě funkce záměrně nevysvětlujeme, protože jsou nahrazeny dvěmi výkonnějšími,které jsou mezi uvedenými třinácti.

Funkce lze používat ve všech příkazech MUMPSu. Funkce lze různým způsobem používat jako parametry dalších funkcí a všech příkazů s parametry.
Každá funkce se píše se znakem $ na prvním místě, dále pak následuje název funkce. Název funkce lze zkracovat na
jeden znak, u funkcí, jejichž název začíná na písmeno Z na dva znaky.
Za názvem každé funkce je závorka a v ní se uvádějí parametry dané funkce.

8.1. Funkce $ASC11 pro převod znaku na číslo z ASCII tabulky čísla a funkce $CHAR pro převod čísla na znak z ASCII tabulky

Obě tyto funkce se používají obdobně jako v jiných programovacích jazycích pro převodu znaku na pořadové číslo ASCII tabulky a obráceně. Čísla v nich používaná jsou v dekadickém tvaru (nikoli v hexadecimálním).
Funkce $ASCII se používá například takto:

W $A("B")<ET>

Výsledek (číslo 66) se vypiše na obrazovku. íslo 66 je dekadické pořadové číslo znaku "B" v ASCII tabulce. Viz příloha číslo 2. Parametr funkce je znak (písmeno velké B), což lze poznat z uvozovek uvedených předá a za znakem. Použijete-li B bez uvozovek, pak jde o proměnnou B (o její obsah). Toto pravidlo se používá v celém MUMPSu. Platí též při psaní indexů indexovaných proměnných (viz příklady kapitola 3.3.2.). Jiný možným způsob využití funkce $A je například v příkazu S.

S C=$A("B")<ET>

Tento příkaz uloží číslo 66 do proměnné C. Další způsob použití funkce $A je například v podmiňování příkazů. Běžně lze použít následující podmínění příkazů:

D:$A(B)=67 ZAC<ET>

Tento příklad je možno vyzkoušet i v přímém příkazovém režimu, budete-li mít v paměti počitače program s

návěštím ZAC a bude-li existovat proměnná B. Akce, která pak proběhne se dá popsat takto: obsahuje-li proměnná B jakykoliv řetězec začínající na písmeno velké C, pak MUMPS spustí program v paměti od návěští ZAC; nebude-li řetězec obsahovat na prvním místě velké písmeno C, pak se příkaz neprovede. Další modifikace funkce $A je předvedena na následujícím příkladu:

W $A("ABCD",3)<ET>

Takovýto příkaz vypíše na obrazovce číslo 67, což je dekadické pořadové číslo znaku "C" v ASCII tabulce. Hodnota 3 určuje, který znak zleva má být vyhodnocen. Jistě Vãs napadne velké množství dalších variant použití funkce $A ve spojení s následujícími funkcemi a příkazy. Všechny tyto nápady vyzkoušejte a budete jistě mile překvapeni, jakou variabilitu MUMPS umožňuje. U těch kombinací, které nebudou pracovat, si nejprve ověřte logickou správnost požadované reakce. Neshledáte-li žádnou logickou chybu, pak jde o některou vyjímku, kterých v MUMPSu není mnoho. Od tohoto místa Vašeho studia MUMPSu bude dobré, když si na podobné zkoušení nápadů zvyknete. Není totiž možné vypsat všechny varianty používání příkazů a funkcí - tato obrovská variabilita jednoduchých prostředků MUMPSu je jednou z jeho velmi ceněných vlastnosti.
Funkce $C je obrácenou funkcí k $A. Znamená to tedy, že čísla uvedená jako parametry budou převedena na znaky z ASCII tabulky. Například:

W $C(65,66,67,68)<ET>

vypíše na obrazovku řetězec znaků:

ABCD

Tato funkce může být používána pro posílání skupin řídících znaků na tiskárnu nebo obrazovku, a to se stejným účinkem jako příkazu W * z kapitol 7.1. a 7.2. Následující příkazy:

W $C(27,91,50,74)<ET>
W *27,*91,*50,*74<ET>,

mají naprosto stejné účinky, to znamená, že smažou obrazovku.

8.2. Funkce EXTRACT pro vyjmutí řetězce z řetězce

Funkce umožňuje vyjmutí libovolné části řetězce a jeho použití prakticky ve všech příkazech a funkcích. Jako příklad mohou sloužit následující příkazy.

W $E("ABCDEFGH")<ET> => vypíše první znak A
Vypíše pouze první znak.

W $E("ABCDEFGH",3)<ET> => vypíše třetí znak C
Vypíše jeden znak, a to ten, který je v pořadí třetí zleva.

W $E("ABCDEFGH",3,5)<ET> => vypíše třetí až pátý znak CDE

W $E("ABCDEFGH",6,3)<ET>
Tato varianta příkazu nevypíše nic, protože první čislo 6) udává pozici od niž začíná vyjímání řetězce z řetězce "ABCDEFGH" a druhé číslo (3) udává pozici, na níž vyjímání končí. Vyjímání začíná zleva, takže obrácený postup je ignorován.

S A=98765,B=$E(A,2,3)<ET> => obsah proměnné B= 87

S A="ABC" D:$E(A,2)="B" ^PRIKLAD1<ET> => spustí ^PRIKLAD1
K dané funkci je možno jen doporučit,abyste ji používali
v kombinaci s příkazy a ostatními funkcemi.

8.3. Funkce $FIND pro prohledání řetězce

Pro jednoduchost začneme příkladem:

W $F("ABCDEFGH","CD")<ET>

Příkaz vypíše 5, což je pátý znak v prvním řetězci "ABCDEFGH" tedy znak "E" a ten je bezprostředně za dvojicí znaků "CD", jež hledá příkaz $F v prvním řetězci. První řetězec je tedy prohledáván na výskyt druhého fetězce. Po jeho nalezeni je vypsáno pořadové číslo znaku následujícího za hledaným řetězcem. Funkci je možno (stejně tak jako všechny ostatní funkce) použit v práci s proměnnými. Toto je možno demonstrovat na dalším příkladu.

S ^AA(1)="ABCDEFGH",B="CD" W $F(^AA(1),B)<ET>

Výsledek je opět číslo 5.

8,4, Funkce PIECE pro vyjmutí řetězce mezi oddělovači a pro vložení řetězce oddělovače

Velmi často používaná funkce, která je schopna vyčleňovat z řetězce skupiny znaků, oddělen oddělovacími znaky. Nejprve si vyzkoušejte následující příkaz.
S A="ABC#DEF#GHI#JKL" W $P(A,"#",2)<ET>

Funkce vypíše na obrazovku řetězec DEF. Popis interpretace funkce je následovný. Vypiš na obrazovku skupinu

znaků (pod-řetězec) z řetězce v proměnné A ("ABC#DEF#GHI#JKL"), jako oddělovač použij znak "#" a vezmi druhý řetězec mezi těmito oddělovači. Vysvětlení je možná trochu krkolomné, takže nezoufejte a pro lepší pochopení čtěte dál. Řetězec v proměnné A se skládá z významových znaků, kterými jsou čtyři trojice znaků "ABC", "DEF", "CHI", "JAL" (tento fakt si budete vždy určovat dle potřeb). Dále je v řetězci opakovaně použit znak "#" jako tzv. oddělovač (tento oddělovací znak si také můžete určit vždy dle potřeby). To, že jej používáte jako oddělovač řečeno ve druhém parametru funkce. Kolikátý podřetězec chcete vypsat je řečeno třetím parametrem funkcí $P. Pro názornost uvedeme několik dalších příkladů.

W $P(A,"G",2)<ET>

Vycházíme-li z řetězce A="ABC#DEF#GHI#JKL", pak výsledkem tohoto příkazu bude řetězec "HI#JKL". 1. parametr funkce určuje zdrojový řetězec (ten je v proměnné A). 2. parametr určuje, že jako oddělovač bude použit znak "G". 3. parametr požaduje nalezení druhého řetězce (první je ABC#DEF#, druhý je HI#JKL). Následující příklad bude použit s příkazem S. (v dalším textu předpokládáme, že v proměnné A budete mít stále uložen řetězec "ABC#DEF#GHI#JKL")

S B=$P("ABC#DEF#GHI#JKL","#",4)<ET>

Výsledkem příkazu je, že obsah proměnné B="JKL". Pro názornost a inspiraci uvádíme ještě jeden přiklad.

D:$P(A,"#",2)="DEF" ^CURSOR<ET

Příkaz zavede z disku a spustí program CURSOR. Druhou variantou příkazu je použití čtyř parametrů. Například:

W $P(A, "#", 2,4)<ET>

Příkaz zobrazí řetězec "DEF#GHI#JKL". Funkce vyjme ze zdrojového řetězce druhý až čtvrtý podřetězec, včetně oddělovacích znaků a příkaz W jej zobrazí.

Třetí varianta funkce $P je vlastně opak dvou předešlých. Jde o vložení řetězce do řetězce opět s použitím oddělovacích znaků. Tato modifikace funkce $P má vztah pouze k příkazu S. V příkazu R není zpracována a hlásí chybu.
S $PEA, "#", 2) ="def "<ET>
Výsledkem je změna řetězce v proměnné A. A="ABC#def#GHI#JKL".

8.5. Funkce $JUSTIFY pro formátování výpisů a zaokrouhlování

Funkce umožňuje formátování výpisů na obrazovku i tiskárnu. Funkce je buďto dvojparametrová nebo třiparametrová. Následující příklad používá dvojparametrovou variantu funkce $J.

W $J("ABCD",20)<ET>

Funkce způsobí výpis řetězce ",ABC" a to tak, že znak "D" je dvacátý znak od předešlého tisku (případně od kraje obrazovky či tiskárny, nebylo-li na daném řádku dosud tisknuto). Další příklad použije funkci $J dvakrát za sebou v tisku na nový řádek.
S A="ABCDEF" W $J(A,20),$J(A,10)<ET>

Výsledek bude vypadat tak, jak je znázorněno, nad výpisem jsou zobrazena čísla sloupců.

12345678**0**12345678**0**1234567890
ABCDEF ABCDEF

Tříparametrová funkce $J je používána u řetězců začínajících číslicí. U ostatních řetězců vypisuje pouze nuly. Například příkazový řádek:

S A=123.456 W $J(A,20,2)<ET>

vypíše na obrazovku číslo 1234.6 , tedy zaokrouhleně a tak, že číslice 6 je umístěna jako dvacátý znak na řádku. Použijete-li funkci $J v příkazu S, pak nesmíte zapomenout to, co
vám ukáže následující příklad.

S A=$J(123.456,20,2)<ET>

Proměnná A bude sice naplněna řetězcem,který obsahuje číslice, avšak na prvním místě řetězce v proměnné A bude znak mezera. Takovýto obsah proměnné A nelze použít v matematických operacích, viz kapitola 3.2.,neboť jeho numerická hodnota je 0. Pozměníte-li druhý parametr ve funkci $J na číslo 0, pak se do proměnné A nedostane žádná mezera. Příkaz může vypadat například takto:

S A=$J(123.456,0,2)<ET>

takže v proměnné A je řetězec s numerickou hodnotou 123.46. Dobře si promyslete následující příklad.

S A=$J(123.456,20,5),B=$J(A,2,3)<ET>

W A,!,B<ET>

kdy se na obrazovce vypíše:
123.45600
0.000
První parametr příkazu S je uložení řetězce do proměnné A tak, že číslo 123.456 bude mít 5 míst za desetinnou tečkou, přičemž poslední číslo bude zapsáno jako 20. znak v pořadí. Druhý parametr příkazu S (B=$J(A,2,3)) pak použije tento dvacetiznakový řetězec a s ohledem na to, že jde o tříparametrovou funkci $J, pracuje s jeho numerickou hodnotou. Numerická hodnota řetězce " 123.45600" je 0 (díky tomu, že řetězec v proměnné A nezačínám numerickým znakem). Funkce má tedy uložit do proměnné B dva znaky, avšak za desetinnou tečkou má být použito tří znaků. Vzniklý rozpor je řešen tak, že druhý parametr funkce $J je ignorován a do proměnné se vloží jen pět znaků (0.000).

8.6. Funkce $LENGTH

Funkce $L má dvě varianty. První je běžně známá i v jiných programovacích jazycich. Tato první varianta zjišťuje délku řetězce (a to opět i řetězce v proměnné). Příkaz:

W $L("ABC")<ET>

vypíše číslo 3, což znamená, že řetězec je dlouhý tři znaky (uvozovky před řetězcem a za ním se nepočítají). Když u je řeč o uvozovkách, tak je dobré vědět, že uvozovky mohou být i součástí řetězce. Tento požadavek lze zajistit

následujícím způsobem:

S A="ABC,""DEF"<ET>

Obsah proměnné A je pak sedm znaků (ABC"DEF). Z uvedeného tedy vyplývá, že zapsání uvozovek do řetězce je možné, avšak je potřeba uvozovky zdvojit. Chcete-li do proměnné A zadat pouze znak uvozovky, pak je potřeba příkaz napsat následovně: S A="""". V proměnné A je pak uložena jedna uvozovka. Použijete-li nyní příkaz W $L na proměnnou A, pak výsledek je číslo jedna.
Druhá varianta přikazu $L souvisí s použitím oddělovacích znaků, o nichž je psáno v kapitole 8.4.
Příkazový řádek:

S A="ABC#DEF#GHI#JKL" W $L(A,"#")<ET>

vypíše číslo 4, což je množství podřetězců v proměnné A při použití znaku # jako oddělovače. Funkce zde vlastně říká, kolik je v řetězci oddělovacích znaků (zde konkrétně znak #) a přičte čislo 1. Pro názornost tohoto tvrzení uvedeme další příklad.

S A="A#B#####C#" W $L(A,"#")<ET>

Na obrazovku se vypíše číslo 8. Tuto variantu funkce lze použít například při rozebírání řetězců uložených v proměnných, kdy bude použit cyklus v tolika opakováních, kolik je podřetězcú v proměnné. Například bude-li A="AB#C##D#", pak následujicí příkaz:

F I=1:1:$L(A,"#") S B(I)=$P(A,"#",I)<ET>

naplní indexované proměnné B(1)="AB", B(2)="C", B(3)="" (tj. prázdny znak), B(4)="D" a B(5)="".
Obě varianty příkazů lze opět použit ve velkém množství obměn s příkazy a funkcemi (což platí skoro u všech funkcí MUMPS).

8.7. Funkce $RANDOM pro generování náhodných čísel

Funkce umožňuje generovat náhodná čísla, avšak v předem stanoveném rozsahu. Příkaz:

F I=1:1 W !,$R(7) R *KONEC Q:KONEC=75<ET>

umožní nahradit kostku při hře člověče nezlob se. Vždy po zobrazení čísla na obrazovce počítač čeká na další stlačení klávesy. Stlačíte-li klávesu <K> (má hodnotu 75 z ASCII tabulky), tak se příkaz ukončí. Příkaz R s hvězdičkou viz kapitola 3.3.3.
Příklad, tak jak byl uveden pro hru člověče nezlob se, generuje i nulu. Funkce totiž generuje z čísla (z jeho celočíselné části) čísla v rozmezí 0 až číslo o 1 menší než je parametr funkce (parametr je nyní číslo 7). Znamená to, že příklad funkce $R generuje čísla od 0 do 6. Další příklad již řeší problém nuly, takže generátor je opravdu použitelny jako hrací kostka.

F I=1:1 W !,$R(6)+1 R *KONEC Q:KONEC=75<ET>

Funkce generuje náhodná čísla od 0 do 5, tisk se však provádí v rozmezí čísel 1 až 6.
Parametrem funkce nesmí byt číslo 0. Použijete-li jako parametr číslo 1, tak se dočkáte vždy jen výsledku 0. Funkce totiž generuje pouze celočíselné hodnoty od nuly až do hodnoty parametru sníženého o číslo 1. Pak horní hranice je dle následujícího propočtu 1 - 1 = 0 .... rovna nule.

8.8. Funkce $TEXT pro výpis řádku programu

Funkce umožní výpis řádku programu, který se právě nalézá v paměti počítače. Proveďte následující příkazy:

ZL POKUSD<ET>
W $T(ZAC)<ET>

pak se Vám na obrazovce zobrazí:

ZAC ;POKUSD

Příkazem ZL je načten program POKUSD, který jste uložili na disk při práci v kapitole 5.5. Druhy příkaz zobrazí na obrazovce první řádek programu. Funkce $T obsahuje návěstí řádku (psané bez uvozovek). Funkce $T pracuje tak, že nejprve program v paměti prohledává a v případě, že nalezne řádek s uvedeným návěštím, tak jej vypíše. Návěstí je možno psát i následovně:

W $T(ZAC+2)<ET>

což vypíše obsah třetího řádku programu, tedy
R !,vstup 1,2,3 ",KAM#l

Číslo za návěštím (často se mu říká offset) je možné zadavat i prostřednictvím proměnné, takže předcházející příkaz by mohl vypadat takto:

S A=2 W $T(ZAC+A)<ET>

8.9. Funkce $DATA pro určení existence a stavu proměnné

Pokud jste ve výuce MUMPSu dospěli úspěšně až sem, tak je nutné pro zdárné vysvětleni funkce $D vysvětlit další ze základních filosofických myšlenek MUMPSu. Zadejte následující příkazy:

K ^B S ^B(1,1)=”ANO”<ET>
V 2<ET>

Příkaz postupně vzruší databázi ^B, dále pak zadají prvek ^B(1,1) a nakonec jsou vypsány názvy všech globálů (databází). Mezi globály najdete i global ^B. Global ^B však nebyl zadáván, pouze jeden jeho prvek. MUMPS si sám vytvořil fiktivní spojovací uzly, tedy uzel ^B a ^B(1). Tyto uzly jsou použity pro uložení prvku ^B(1,1) a dále jsou využívány pro přímý přístup k prvku ^B(1,1). Jde vlastně o fiktivní prvky, bez kterých by prvek ^B(1,1) nemohl existovat. Struktura databáze vypadá takto:

^B	0. úroveň databáze ,fiktivní prvek
^B (1)	1.úroveň fiktivní prvek
^B (1,1)	reálný prvek obsahující řetězec "ANO"

MUMPS ukládá indexované proměnné tak, že zjistí zda existují spojovací prvky k zadávanému prvku. Jestliže kterýkoli spojovací prvek neexistuje, tak je nahrazen prvkem fiktivním. Takto vzniklý fiktivní prvek nemá žádný obsah (neobsahuje řetězec). Kterýkoliv fiktivní prvek však může být kdykoli naplněn řetězcem. Byl-li tedy v předchozím příkladě zadán prvek ^B(1,1), pak prvky ^B a ^B(1) neobsahují žádné řetězce. Zadáte-li příkaz:

K ^B(1,1)<ET>

tak zrušíte nejen prvek ^B(1,1), ale také celou databázi ^B, protože v MUMPSu nemohou existovat prázdné prvky, na nějž není napojen alespoň jediný prvek s minimálním řetězcem (minimální řetězec - prázdný znak- vznikne takto S ^B(1,1)=""). Zrušíte-li prvek ^B(1 , 1), tak prvky ^B a ^B(1) nemohou existovat samostatně, protože neobsahují minimální řetězce. Pro jednoznačnost pochopení zadejte následující příkazy:

K ^B S ^B(1, 1)="ANO" l^B="NE"<ET>
V 2<ET>
K ^B(1,1) V 2<ET>

Nyní se v obou případech databăze ^B zobrazí. Při prvém zobrazení bází příkazem (V 2) existuji prvky ^B, ^B(1) a prvek ^B(1,1), o čemž se můžete sami jednodute přesvědčit výpisem. Při druhém zobrazen1 (V 2) již v databázi existuje pouze prvek ^B.
Teprve nyní lze přistoupit k vysvětlování funkce $D. Zadejte příkaz:

K A W $D(A)<ET>

a na obrazovce se vypíše číslo 0. Funkce $D(A) takto oznamuje, že proměnná A neexistuje. Po zadání následujícího příkazového řádku:

S A="ANO" W $D(A)<ET>

se na obrazovce zobrazí čislo 1. Funkce $D(A) takto oznamuje, že proměnná A existuje, tzn. že je naplněna alespoň minimálním řetězcem (prázdný znak). V obou dosud uvedenych případech se jednalo o jednociferné číslo (jeho hodnota může byt pouze 0 nebo 1). Číslo 1 navíc informuje o tom, že za danou proměnnou již neexistuje žádný prvek na nižší úrovni (tj. v tomto případě skutečnost, že neexistuje prvek A s indexy). Použijete-li funkci $D na dříve nadefinovanou databázi ^B (raději si databázi ^B zadejte znovu příkazem K ^B S ^B(1,1)="ANO"), pak příkaz:

W $D(^B(1,1))<ET>

zobrazí na obrazovce číslo 1, což znamená , že proměnnA ^B(1,1) existuje a dále, že jde o poslední prvek dané větve. To znamená, že za prvkem ^B(1,1) již nexistuje prvek na nižší úrovni.
Následující řádky by měly přiblížit chováni fuňkce $D u indexovanych lokálnich či globálních proměnných, které nejsou posledními prvky v databázi. K vysvětleni použijeme opět databázi ^B. Nyní zadejte:

W $D(^B(1))<ET>

pak se na obrazovce zobrazí číslo 10, stejně tak výsledkem příkazu:

W $D(^B)<ET>

je číslo 10. V obou případech funkce oznamuje tiskem dvouciferné položky, že jde o prvky, za nimiž existují další prvky na nižší úrovni. 0 na druhém místě oznamuje, že prvky samy neobsahují řetězce.

		0. úroveň databáze ^B.
$D(^B)=10	B	Prvek má následovníka.
		Prvek je bez minimálního řetězce.
		1. roveň databáze ^B.

$D(^B(1))=10 B(1) Prvek má následovníka.
Prvek je bez minimálního řetězce.

2. roveň databáze ^B $D(^B(1,1))=1
B(1,1) Prvek bez následovníka.
poslední prvek větve -> Reálný prvek obsahující řetězec "ANO".

Nyní zadejte:

S ^B(1)="NE",^B="" W $D(^B)," ",$D(^B(1))<ET>

a na obrazovce se zobrazí 11 11

0. úroveň databáze ^B.
$D(^B)=11 B Prvek má následovníka.
Prvek obsahuje "", tedy minimálně řetězec.

1. úroveň database ^B.
$D(^B(1))=11 B(1) Prvek má následovníka
Prvek obsahuje řetězec "NE".

2. úroveh databáze ^B
$D(^B(1,1))=1 B(1,1) Prvek bez následovníka.
poslední prvek větve -> Prvek obsahuje řetězec “ANO”

Čísla jsou opět dvojciferná, cože znamená, že jde o prvky za nimiž existují další prvky na nižší úrovni. 1 na druhém místě oznamuje, že prvky samy obsahují řetězce.
Pro rekapitulaci uvádíme následující tabulku.

Tabulka vtech alternativ funkce-$D

$D=	obsah proměnné	následovníci
0	proměnná neexistuje	
1	proměnná existuje	nemá následovníky
10	proměnná neobsahuje řetězec	má následovniky
11	proměnná obsahuje řetězec	má následovníky

Při používání funkce $D je potřeba nezapomínat na to, že zjišťujete vždy existenci proměnné. Z toho plyne důležitý závěr: uvnitř závorek ve funkci $D se nikdy nepoužívaji uvozovky. Nelze tedy napsat příkaz:

W $D("A")<ET>

A, jako parametr funkce $D může být pouze lokální či globální proměnná, ale nikdy ne řetězec. Chcete-li použít jako parametr funkce $D obsah proměnné, pak toto proveďte následovně:

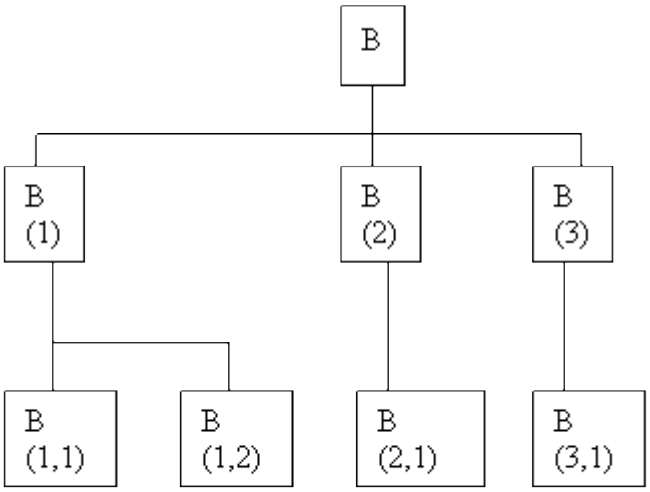
S A="^C" W $D(@A)<ET>

Je tedy nutné nejprve do nějaké pomocné proměnné (zde použita proměnná A) uložit celý název hledané proměnné a pak použít nepřímého svolání za pomoci znaku @. Znak @ řídká, že parametr funkce (tedy jméno testované proměnné) je uložen v proměnné A. Název hledané proměnné můžete vytvořit například použitím zřetězováni, viz příkaz:

S B="C”,A="^"_B_"(1)"<ET>

Příkaz uloží do proměnné A řetězec "^C(1)", který pak můžete použít jako název globální indexované proměnné.

Na závěr funkce $D:



Pochopení funkce $D je velice důležité pro jakoukoli dalším práci v MUMPSu, pročež ještě na obrázku upřesňujeme co je následovník a co není následovnik prvku. Přímí následovníci prvku B jsou B(1), B(2) a B(3), další následovníci prvku B jsou prvky B(1,1), B(1,2), B(2,1) a B(3,1). Následovníci prvku B(1) jsou pouze prvky B(1,1) a B(1,2). Prvek B(2) má jediného následovníka a tím je B(2,1). Stejně tak prvek B(3)má jediného následovníka a to prvek B(3,1). Posledními prvky větvi jsou prvky B(1,1) , B(1,2), 2), B(2,1) a B(3,1).

8.10. Program PRIKLAD2.MMP

Program, který nyní následuje, vznikl úpravou programu z kapitoly 5.6.
Příkazy a funkce v něm použité jimž byly vysvětleny, takže si sami můžete otestovat míru pochopení dosud vysvětlených příkazů a funkci. Na disketě je program uložen pod názvem

PRIKLAD2.MMP

```
ZAC             ;ZALOZENI DATABAZE,MP,
                ;založeni pracovniku do baze A
CURSOR          D ^CUR$OR;volani podprogramu pro rizeni kurzoru
DALSI           R #!!!!,"ZALOZIT DALSIHO PRACOVNIKA A/N ",DALSI#l
Q:(DALSI="N")l(DALSI="n")
                G:(DALSI'="A")&(DALSI'="a") DALSI
```

```
OSCIS       R !!!,"zadejte osobni cis.pracovnika ",OSCIS#4,!!!!
            G:$D(^A(OSCIS))=0 NOVY
            S STARY=$P(,"'A(OSCIS),"i",1),ZBYTEK=$P(^A(OSCIS),"#",2,3)
            W "jmeno : ",$P(STARY," ",1)
            W !,"prijmeni   ",$PASTRY," ",2)
            W !,"narozen :",$P(ZBYTEK,"#",1)
            W !,"stav : ",$P(ZBYTEK,"#",2)
ZMEN        S DX=O,DY=18 W @CUP,"budete menit ? AM " R ZMEN#l
            G DALSI.:(ZMEN="N")I(ZMEN="n"),ZMEN:(ZMEN'="A")&(ZMEN'="a")          S
DX=8,DY=11 W @CUP R JMENO#20
            S DX=11,DY=12 W KUP R PRIJMENI#20
            S DX=10,DY=13 W @CUP R DATNAR#8
            S DX=7,DY=14 W KUP R STAV#10
            S:JMENO'="" $P(STARY," ",1)=JMENO
            S:PRIJMENI="" $P(STARY," ",PRIJMENI
            S:DATNAR="" $P(ZBYTEK,"#",1)=DATNAR
            S:STAV'="" $P(ZBYTEK,"#",2)=STAV
            S ^A(OSCIS)=STARY_"#"_ZBYTEK G DALSI
NOVY        R "jmeno : ",JMENO#20
            R !,"prijmeni   ",PRIJMENI#20
            R !,"narozen    ",DATNAR#8,1,"stav ".,STAV#10
ULOZ        S "A(OSCIS)=JMENO=" "_PRIJMENI_"#"_DATNAR_"#"_STAV
            G DALSI
```

Nevýhodou původního programu z kapitoly 5.6 je skutečnost, že jednou založené číslo pracovníka můžete přepsat novými údaji, aniž byste to zjistili. Tento vylepšený program Vás upozorní na skutečnost, že pracovník je již zadán. Jednotlivé údaje o pracovníku pak vypíše na obrazovku a můžete se rozhodnout, zda pracovníka zadat znovu nebo zda není potřeba provádět změny.

Popis funkce programu. Program ihned po svém zavedeni volá program s názvem CURSOR. Program CURSOR nadefinuje proměnné pro usnadněni prace s obrazovkou. Až po návěští OSCIS je program shodný. Na řádku OSCIS zjišťuje funkce $D zda není v databázi A pod indexem, který je v proměnné OSCIS, již záznam. Funkce je použita pro podmíněné vykonání příkazu G. Zjisti-1i funkce $D, že právě zadaný klíč neexistuje, tak pošle program na návěští NOVY. Tato část programu je opět shodná s původním programem (v původním programu od návěští OSCIS+1).

Pokud program neskočí na návěstí NOVY, vypíší se na obrazovku data nalezeného pracovníka. Výpisu předchází naplnění dvou proměnných. Proměnná STARY je naplněna jménem a příjmením pracovníka, datum narození a stav jsou naplněny do proměnné ZBYTEK. Rozdělení záznamu tak bylo provedeno proto, že jméno a příjmení mají mezi sebou jako oddělovač mezeru, kdežto ostatní záznamy jsou odděleny znakem #. Takovéto rozdělení v programu dále zjednodušuje práci. Po vypsání pracovníka se program táže, zda budete provádět změny, či nikoli. V případě, že změny provádět nebudete, tak se vracíte na návěští DALSI. Stlačíte-li cokoli kromě znaků A, a, N, n, tak se program vrací znovu na otázku, zda budete provádět změnu. Rozhodnete-li se pro změny, tak Vás program žádá o jejich jednotlivé provedení. Změny jsou zadávány do proměnných JMENO, PRIJMENI, DATNAR a STAV. V další části programu (od řádku ZMEN+6) jsou změny ukládány do proměnných STARY a ZBYTEK. Změny se však provádí jen z naplněných proměnných JMENO až STAV. Jestliže kterákoli z těchto proměnných obsahuje prázdný řetězec, tak není použita k aktualizaci a ponechává se původní obsah dané části proměnné STARY či ZBYTEK. Jako příklad je opsán řádek z programu.

```
S:JMENO'="" $P(STARY," ",1)=JMENO
```

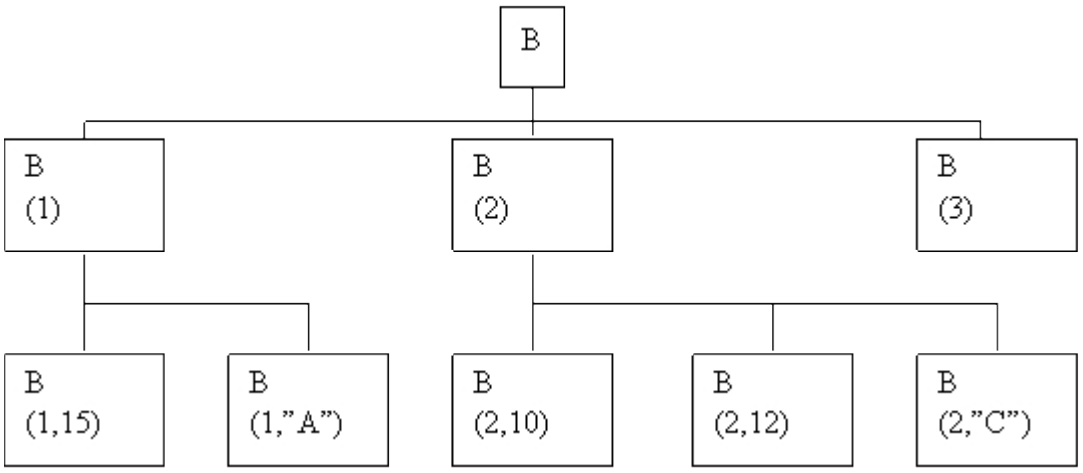
Přikaz uloží obsah proměnné JMENO do prvé části proměnné STARY, ale jen za podmínky, že v proměnné JMENO existuje alespoň jednoznakový řetězec. Po ukončení aktualizace proměnných STARY a ZBYTEK se

provede uloženi do databáze A. V podstatě jde o přepis původního záznamu v databázi A.
V programu jsou použity příkazy pro nastavení místa na obrazovce, od nějž proběhnou dalším tisky. Vždy je nejdříve naplněna proměnná DX (souřadnice sloupce) a DY (souřadnice řádku) Po jejich naplněni je použit příkaz W @CUP, který nastaví kurzor do požadovaného místa obrazovky. Od tohoto místa jsou pak tištěny texty nebo je používán vstup v příkazech R.

8.11. Funkce $ORDER

Pro zjištění adresy části adresy sousedícího prvku v lokální nebo globální indexované proměnné

Funkce $ORDER a dále popisovaná $ZORDER mají mnoho společného. Před studováním jejich činnosti si založte následující globální databázi. Globální proto, že funkce $ZO nepracuje s lokálními databázemi.



Všechny části globálu ^B (mimo ^B(2)) naplňte libovolnymi řetězci.

Zadejte příkaz:

W $O(^B(""))<ET>

Funkce $O zobrazí číslo 1. Dále zadejte příkaz:

W $O(^B(1))<ET>

Funkce zobrazí číslo 2. Příkaz znóvu zopakujte, avšak s číslem 2. Funkce zobrazí číslo 3. Poslední příkaz bude:

W $O(^B(3))<ET>

a funkce nevypíše nic (vypíše prázdný znak). Co jsme Vám tímto vlastně chtěli říci ? Pouze to, že šlo o Vaši první "procházku " po jedné úrovni globálních proměnných. Příkazem W $O(^B("")) jste přikázali funkci $O, aby nalezla první prvek v databázi AB, a to na jejím první úrovni (to je na úrovni prvních indexů). Tímto prvkem je ^B(1), z čehož funkce $O zjišťuje pouze index dané úrovně, což je číslo 1. To, kterou indexní úroveň má funkce prohledávat, je dáno tím, kde se vyskytují za sebou dvě uvozovky. V daném případě se vyskytují jako jediné znaky v závorkách indexu báze ^B, jde tedy o první indexní úrovéň. Příkaz

W $O(^B,""))<ET>

pak znamená, aby funkce $O nalezla první prvek v databázi ^B, a to na její druhé úrovni (uvozovky jsou až za první čárkou, jde tedy o druhou roveň), konkrétně pod prvkem ^B(l).'Funkce $0 nalezne číslo 15, což vytvoří celý index hledaného prvku ^B(1,15). Pak příkaz:

W $0(^B(1,15))<ET>

nalezne znak "A" ,z čehož plyne index dalšího prvku v pořadí, a to prvku ^B(1, "A"). Za tímto prvkem není již žádný prvek patřící pod ^B(1), takžé příkaz:

W $0(^B(1,"A"))<ET>

zobrazí prázdný znak. Prázdný znak je takto symbolem počátku prohledávání dané úrovně viz ^B(1,"") a též koncem prohledávání. Obdobná anabáze pod prvkem ^B(2) dá výsledky 10, 12, "C" a prázdný znak. Anabáze pod prvkem ^B(3) dává ihned prázdny znak, protože pod prvkem již není nižším úroveň (prvek ^B(3) by na příkaz W $D(^B(3)) odpověděl číslem 1). Stejně jako prvek ^B(3) se zachovají i prvky ^B(1,15), ^B(1,"A"), ^B(2,10), ^B(2,12) a ^B(2,10), protože nemají pod sebou prvky na nižší úrovni.
Funkce $O vrací poslední části indexu prvku následujícího tomu prvku, který je v ní uveden jako parametr. Pomocí funkce $O lze procházet napříč v databázích. Tato procházka je však vždy vázána pod jeden prvek na nejbližší vyšší úrovni (prvky ^B(1,15) a ^B(1,"A") patří pod prvek ^B(1), kterýžto je na nejbližší vyšší úrovni,- stejně tak prvky ^B(1), ^B(2) a ^B(3) patří pod prvek na nejbližší vyšší úrovni, což je prvek ^B).
V předcházejících příkladech byl pro demonstraci použit příkaz W. V reálných situacích se však používají příkazové řetězce jako například tento.

```
S X="" F I=1:1 S X=$O(^B(1,X)) Q:X=""  W !,X,"* ",^B(1,X)<ET>
```

Do proměnné X je uložen prázdný řetězec. Dále je pak prováděn nekonečný cyklus, v němž se do proměnné X uloží vždy aktuální hodnota indexu následujícího prvku na dané úrovni. Konec cyklu nastává tehdy, když neexistuje další prvek dané úrovně. Průběžně jsou prováděny tisky aktuální hodnoty indexu a jemu odpovídající řetězec v proměnné.
Předchozí příklad vypíše na obrazovku.

```
15* řetězec z ^B(1,15)
A* řetězec z ^B(1,"A")
```

Změníte-li příkaz následovným způsobem:

```
S X="" F I=1:1 S X=$O(^B(X)) Q:X=""   W !,X,"* ",^B(X)<ET>
```

pak se na obrazovce objeví:

```
1* řetězec z ^B(1)
2* řetězec z ^B(2)
3* řetězec z ^B(3)
```

Obdobně pak příkaz:

```
S X="" F I=1:1 S X=$O(^B(2,X)) Q:X="" W !,X,"* ",^B(2,X)<ET>
```

vypíše na obrazovku:

```
10* řetězec z ^B(2,10)
12* řetězec z ^B(2,12)
```

C* řetězec z ^B(2,'V')

8.12. Funkce $ZORDER pro zjištění adresy následujícího prvku v globální indexované proměnné

Funkce $ZO pracuje tak, že zjišťuje vždy celou adresu následujícího prvku (funkce pracuje pouze s globály). Adresy fiktivních prvků (prvků bez minimálního řetězce) ignoruje. Adresy jsou však zjišťovány jinak než u funkce $O. Pro ukázku si zadejte následujíc příkaz:

S X="^B("""")" F I=1:1 S X=$ZO(@X) Q:X="" W !,X,"* ",@X<ET>

a funkce vypíše na obrazovku:

B(1)* řetězec z ^B(1)
B(1,15)* řetězec z ^B(1,15)
B(1,15")* řetězec z ^B(1,"A")
B(2,10)* řetězec z ^B(2,10)
B(2,12)* řetězec z ^B(2,12)
B(2 "C")* řetězec z ^B(2, "C")
B(3)* řetězec z ^B(3)

Z uvedeného je patrné, že funkce $ZO prochází databázi vertikálně a horizontálně zároveň. Postup prohledávání databáze je demonstrován na posledním uvedeném výpisu. Porovnejte si to s obrázkem. Funkce postupuje v databázi zleva doprava. Stejně jako u funkce $O platí, že prázdny znak je symbolem počátku prohledávání (avšak celého zbytku globálu). Nalezení konce globálu je opět signalizováno prázdným znakem.

Funkce $ZO vrací celou adresu prvku následujícího tomu prvku, který je v ní uveden jako parametr. Za posledním prvkem databáze vrátí prázdný znak.

8.13. funkce $SELECT pro alternativní vykonání zadaného příkazu

Funkce $S umožňuje alternativně vykonat prakticky kterýkoli příkaz. Pro vysvětleni jejího používání si zapište následující příkazy.

S A=10<ET>
W $S(A=5:"A je 5,A=10: "A je 10",1:"A není, 5 ani 10")<ET>

Na obrazovce počítače se objeví text:

A je 10

Druhý příkazový řádek obsahuje funkci $S a interpretace je následující. Napiš první variantu tisku, jehož podmínka je splněna. Je-li A=5, pak napiš text "A je 5. Je-li A=10, pak napiš text "A je 10". V případě, že není splněna žádná z uvedených podmínek, použij poslední alternativu. Tato alternativa je uvozena číslem 1 a vypíše text "A není 5 ani 10". Poslední alternativa musí být vždy splněna, protože musí být použita ve chvíli, kdy,předchozí alternativy nejsou platné. Není-li platná ani,poslední alternativa, tak dojde k chybě v programu. Aby poslední alternativa byla vždy platná,používá se pro její uvedení vždy číslo větším než 0 (v našemu případě je to číslo 1 ve výrazu 1:"A není 5 ani

10" Dále následuje příklad pro inspiraci.

S A=$S(A=5:50,A=10:100,1:1000)<ET>

H $S(A=5:5,A=10:20,1:100)<ET>

G @$S(A=S:"^CURSOR",1:"^POKUS")<ET>

R @$S(A=5:"B",A=10:"C",1:"D")<ET>

Příklady S A=$5... a H $S... jsou shodné s příkazem W. Pouze u příkazu SET je potřeba si uvědomit, že proměnná do níž se bude alternativně ukládat je uvedena před funkci $S.

Zato další dvě varianty (příkazy G a R) si zaslouží drobný výklad. Tyto příkazy a stejně tak další příkazy, jako napříiklad K a D,je potřeba psát se znakem @ a dále je potřeba návěští, názvy proměnných a programů psát do uvozovek. Toto přijměte spíše jako konvenci, než aby tento fakt byl vysvětlován. Lehce lze vyzkoušet, že například nebude fungovat příkaz:

K $S(A=5:B,A=10:C,1:D)<ET>

ale příkaz:

K @$S(A=5:"B",A=10:"C",1:"D")<ET>

bude fungovat.

Jaké jsou rozdíly mezi použitim příkazu I, podmíněním libovonného přikazu a funkci $S ? Příkaz I má platnost pro celý zbytek řádku, který se buďto bude, nebo nebude vykonávat. Podmínění libovolného příkazu se vztahuje k vykonání či nevykonání daného příkazu. Funkcí $S si můžete alternativně měnit průběh vykonání jednoho příkazu. Příkaz bude vykonán vždy dle stavu nastavených podmínek. Je samozřejmé, že se funkce $S bude minimálně používat u příkazu G a D, které mají samy možnost alternativního rozhodováni. častěji bude používaná například s příkazem W, HANG, R nebo S.

9. Zbývající operátory, systémové proměnné,funkce $ZMATH, příkaz ZOPTION, používání holého odkazu a nepřímé adresace

V průběhu předchozích kapitol bylo záměrně vynecháno několik operátorů a systémových proměnných MUMPSu. Nyní se k některým důležitým pro kompletnost vracíme. Předesíláme, že tato příručka neobsahuje stoprocentně všechny prvky jazyka, ale obsahuje vše potřebné pro pochopení filosofie jednouživatelského MUMPSu. Prvky, které v tomto manuálu nejsou popsány se stejně používají jen v minimální míře.

9.1. Operátory _ . ? [ ]# ~

Operátor zřetězení (_) je použit v programu PRIKLAD1 v kapitole 5.6. Zřetězení se používá na fyzické spojení dvou řetězců v jeden. Řetězec je možno použít přímo, nebo řetězec může být uložený v proměnné. Na řetězec v proměnné lze odkázat pomocí nepřímého odkazu na proměnnou, vníž je teprve název proměnné a teprve v takto adresované proměnné je zadávaný řetězec. Následující příkazy toto demonstrují.

```
S A="ABC"_"DEF"<ET>              =>     A="ABCDEF"
S B=”GHI”,A=A_B_” JKL"<ET>  =>     A="ABCDEFGHI JKL"
S C=”B”,A=@C_@C<ET>           =>     A="GHIGHI”
S A="123 “_ B_” “_@C<ET>       =>     A="123 GHI GHI"
```

První řádek spojí řetězce "ABC" a "DEF” do jednoho a ten je uložen do proměnné A. Ve druhém řádku je k takto vzniklému řetězci "ABCDEF" připojen obsah proměnné B a řetězec " JKL Třetím řádek ukazuje použit nepřímé adresace. Do proměnné C je uložen název proměnné B a teprve v B je vlastní řetězec, jenž je použit v následujícím příkazu
S A=@C _@C. @C znamená, že obsahem proměnné C je teprve název proměnné, jejíž obsah je použit k řetězení.
Operátor ? je používán na kontrolu obsahu řetězce dle vzoru. Zadejte následující příkazový řetězec.

```
R A W:A?3N "správně zadáno"<ET>
```

Jestliže nyní zadáte tříciferné číslo a <ET>, tak se na obrazovce objeví text "správně zadáno". V jakémkoli jiném případě nebude vypsáno nic. Interpretace je následující. Vypiš řetězec "správně zadáno",ale jen v případě, že obsah proměnné A odpovídá podmínce U.N. ? znamená operátor kontroly na obsah, 3N znamená, že následující znaky musí být 3 a všechny musí být číselné (tj. 0,1...8,9). Změníte-li příkaz takto:

```
R A W:A?2N1"."1N "spravně"<ET>
```

pak jedině správný řetězec se skládá ze dvou čísel (2N), desetinné tečky (1".") a jednoho čísla za desetinnou tečkou (1N). Text "správně" se vytiskne například po zadání čísla 23.8 . V případě, že chcete aby číslo bylo zadáno například ve tvaru: libovolné množství cifer, desetinná tečka a znovu libovolné množství cifer, pak příkaz vypadá takto.

```
R A W:A?.N1".".N "správně"<ET>
```

Znak tečky (.) vyjadřuje, že na tomto místě v řetězci může být libovolné množství cifer (.N). Pak jsou správná všechna čísla jako: 2222.1111 .2 0.123 111. a další.
V následující tabulce uvádíme seznam všech možných kontrolních znaků. Zatím jste se seznámili se znakem N pro kontrolu na numerické znaky a s možnosti uvedení libovolného znaku v uvozovkách.

znak	význam	
N	libovolný číselný znak	0,1,2.....9
U	libovolné velké písmeno	A,B,C....Z
L	libovolné malé písmeno	a,b,c .....z
A	libovolné písmeno	A,a,B,b .. Z,z
P	libovolný interpunkční znak včetně mezery	
C	libovolný řídící znak včetně DEL	

E libovolný znak

Uvedené znaky je možno kombinovat jako například.

R A W:A?l0NU ",správnę"<ET>

10NU znamená, že kontrolovaný řetězec musí byt desetiznakový, přičemž každý ze znaků může byt buďto číslice, nebo velké písmeno.

Operátor [je používán na testováni obsahu řetězce v jiném řetězci. Například příkazový řádek:

S A="ABCDEF"["BCD" W A<ET>

uloží do proměnné A číslo 1 a zobrazí ho na obrazovce. Číslo 1 jako výsledek porovnání obou řetězců říká, že řetězec "BCD" se nachází v řetězci "ABCDEF". Zadáte-li například:

S A="ABCDEF",B=A["BCF" W B<ET>

tak v proměnné B bude číslo 0, což říká, že řetězec "BCF" se v řetězci "ABCDEF" nenachází. Číslo 1 tedy znamená pravdu a číslo 0 znamená neúspěch (nepravdu).
Operátor] je používán na testování zda 1. řetězec následuje za druhým řetězcem dle tabulky ASCII. Například:

S A="EVA"]"ADAM"<ET>

Znak "E" v řetězci "EVA" následuje v ASCII tabulce za znakem "A" z řetězce "ADAM". Do proměnné A bude uloženo číslo 1. Zadáte-li příkaz obráceně:

S A="ADAM"]"EVA"<ET>

pak v proměnné A bude 0, protože "ADAM" nenásleduje abecedně (a tedy ani dle ASCII tabulky) za řetězcem "EVA". Opět tedy číslo 1 znamená (stejně jako v celém MUMPSu) pravda a číslo 0 znamená nepravda (neúspěch).

Operátor # je používán pro výpočet zbytku po dělení. Takže příkaz:

W 10#7<ET>

vypíše na obrazovku číslo 3, což je zbytek po dělení čísla 10 číslem 7. Výsledek příkazu:

S A=1000.2#512<ET>

bude uloženo číslo 488.2 do proměnné A, protože 488.2 je zbytek po dělení čísla 1000.2 číslem 512.

Dalším operátorem je \. Tento operátor se používá pro celočíselné dělení. Například příkaz:
W 25.3\5<ET>
.zobrazí na obrazovce číslo 5.
Vysledek dalšího příkazu:

W 24\5<ET>

bude číslo 4. Budete-li používat celočíselné dělení nebo operand pro zbytek po dělení, pak se dobře seznamte v

druhé části příručky s tím, jaké výsledky dávají tyto operátory, když se do výpočtů začne plést záporné znaménko !

Poslední poznámka patři k používání operátoru NOT, tedy (‘) . Operátor lze používat opět v různě propletených variantách. Jako příklady k inspiraci Vám mohou sloužit následující řádky.

R A W:A'?5N "ano, nejde o 5 cifer"<ET>

Text bude vytištěn pouze tehdy, když obsáh proměnné A nebude pěticiferné číslo.

S A=100,B='A<ET>

Proměnná A má hodnotu 100. Negace hodnoty 100 je 0. Protó bude obsah proměnné B=0. Negace je zde prováděna s numerickou interpretací proměnné A. Je-li numerická hodnota řetězce v proměnné různá od nuly, pak je její logická interpretace rovna číslu 1 a negace je tedy 0. Numerická interpretace řetězců je popsána v kapitole 3.2.
Příkaz:

S A="ABCDEF",B='A<ET>

uloží číslo.1 do proměnné B. Numerická interpretace řetězce "ABCDEF" je 0 a negace čísla 0 je číslo 1. Jako další příklad pro inspiraci může sloužit operátor NOT v příkazu I.

S A=10 I A+5’=18 W "ano, A+5= ",A+5<ET>

Protože A+5 je 15 a nikoli 18,-je příkaz W za příkazem I vykonán.

9.2. Systémové proměnné

Proměnná $TEST byla vysvětlena v kapitole 5.2.1. Další systémové proměnné o nichž byste měli byt informováni jsou proměnné $HOROLOG a $STORAGE. Systémové proměnné, které nepatří do standardu jazyka jsou $ZCOUNT, $ZNAME, $ZGLOBAL, $ZROUTINE a $ZERROR. Systémové proměnné nejsou zobrazitelné pomocí příkazu V 0 ani V 1.

Systémová proměnná $H obsahuje zašifrované datum a čas. Datum je zaznamenáno před čárkou jako množství dnů ode dne 31. 12. 1840 (toto je den 0). čas v sekundách od půlnoci je zaznamenán za čárkou. Pro rychlé rozluštění času a datumu existují systémové programy. Datum zjistíte v proměnné %DT po spuštění programu %DATE. čas bude uložen v proměnné %T po spuštění programu %MTIME.

Systémová proměnná $S obsahuje vždy aktuální velikost nepoužité paměti pro data a programy. Spustíte-li MUMPS, tak příkaz:

W $S<ET>

vypíše velikost takzvané PARTITION v bytech. PARTITION je vlastně pamět pro uložení programu a lokálních proměnných. Nejde tu o takzvané BUFFERY. Zavedete-li do paměti například program CURSOR, tak se číslo v proměnné $S zmenší o velikost programu. Spustíte-li tento program, tak po jeho ukončení si opět nechejte vypsat hodnotu proměnné $S. Hodnota bude opět nižší, protože část paměti si obsadily lokální proměnné. Zadáte-li bezparametrovy příkaz K, tak zrušíte lokální proměnné a obsah systémové proměnné (na tuto žádné K neplatí)

bude opět vyšší. Zadáte-li příkaz ZR, tak vymažete i program z paměti počítače (z PARTITION) a hodnota proměnné $S bude opět jako při spustění MUMPSu.

Systémová proměnná $ZN obsahuje vždy jméno posledního aktuálního programu. Jde o program, který se právě nachází v paměti a lze jej vypsat příkazem ZP.

Systémová proměnná $ZG obsahuje název mechaniky, na níž se nachází globály s nimiž se právě pracuje. V následující kapitole bude popsáno, jak je možno tyto mechaniky programově měnit. Jde o příkaz ZO.

Systémová proměnná $ZR obsahuje název mechaniky, na níž se nachází programy. Tato mechanika může byt opět programově měnitelná, viz následující kapitola.

Systémová proměnná $ZE je použita tam, kde chcete zajistit 100 procentní bezchybnost programu. Chcete-li v případě jakékoli možné chyby programu zavolat program pro ošetření havárie, pak je potřeba do této proměnné uložit název takovéhoto programu včetně znaku ^ . V případě programové chyby mu bude předáno řízení a v proměnné $ZE bude číslo chyby a označení řádku, kde chyba vznikla. Uložíte-li do proměnné např. $ZE=^HAVARIE, tak v případě chyby právě pracujícího programu bude zavolán program HAVÁRIE a do proměnné $ZE MUMPS uloží např. 3;ZAC+1^POKUS. číslo 3 znamená číslo chyby z chybovníku viz příloha č.4 (nedefinovaná lok. proměnná), za středníkem je označeno místo vzniku chyby.

Poslední zajímavou systémovou proměnnou je $ZC. Tato proměnná obsahuje informaci o počtu volných bloků pro ukládání globálů na dané mechanice (blok je 768 bytů). Takže zjistíte-li, že $ZC má hodnotu 100, pak v právě používaném souboru GLOBALS.DAT je 100*768 volnych bytů. Tento soubor obsahuje všechny Vaše globální proměnné.

9.3. Příkaz ZOPTION

MUMPS, který se Vám spolu s touto příručkou dostal do rukou neumí spojitě využívat disková média (implementace MUMPSU na větších počítačích toto dovedou). Toto pro uživatele znamená, že přímý přístup má vždy jen na maximálně dvě mechaniky. Na jedné z nich jsou programy (tzv. ROUTINES) a na druhé jsou globály (soubor GLOBALS.DAT). Oboje může být rovněž na jednom disku. Chcete-li však mít globály na několika mechanikách současně a všechny je používat, pak je k tomuto zapotřebí znalost příkazu ZO.

Příkaz:

ZO<ET>

Vás informuje o aktuálním nastavení mechaniky s programy a mechaniky s globály. Následující text:

ROUTINES ON C
GLOBALS ON C

Vás informuje o tom, že programy jsou volány z mechaniky C a globály jsou v mechanice C. Chcete-li přiřazení změnit, a máte-li zájem pracovat s programy v mechanice A a s globály v mechanice B, pak napište příkaz:

ZO ("A":"B")<ET>

Příkaz:

ZO ("B")<ET>

přepne MUMPS pro práci s programy v mechanice B a přiřazení mechaniky pro práci s globály se nezmění.

Příkaz:

ZO (:"A")<ET>

přepne MUMPS pro práci s globály v mechanice A, přičemž nezmění přiřazení mechaniky z níž jsou volány programy.
MUMPS používá pro zrychlení vlastní činnosti rozsáhlé buffery. Jde o oblast v paměti počítače, kde se uchovává mnoho informací před tím, než jsou fyzicky uloženy na disk. Může se Vám poštěstit, že při práci bude mít MUMPS všechny údaje, jež má ukládat na disku, v bufferech. Pak se všechen obsah těchto bufrů může ztratit například vypadkem elektrického proudu. Z tohoto důvodu Vám chceme doporučit abyste ve svych programech občas používali příkaz:

ZO ($ZR:$ZG)<ET>

ktery zajistí, že všechna data budou fyzicky uložena na disků. Tento příkaz musíte použít též v případě, že chcete vyjmout datovou disketu z mechaniky a vložit tam jinou. Příkaz v tomto případě musíte použít před vyjmutím púvodní diskety a pak ihned po vložení nové diskety. Pouze tento postup Vás uchrání od ztráty dat (obdobu tohoto musíte dělat u všech programovacích systémů, nejde o zvláštnost MUMPSu).
Vyprázdnění vyrovnávacích bufferů a uložení dat na disk se provede i v případě, že použijete příkaz HALT v přímém příkazovém režimu (disketu vyjmete až po ohlášení systému MS-DOS).

9.4. Funkce $ZMATH

Pro jednoduché matematické propočty,pracující přes přirozený logaritmus,je použitelná matematická funkce $ZM. Tato funkce umožňuje vypočítat druhou odmocninu čísla, přirozený logaritmus čísla a inverzi přirozeného logaritmu na číslo. Příkaz se používá například takto:

W $ZM(1,100)<ET>

Výsledkem funkce je druhá odmocnina čísla 100. To, že jde o druhou odmocninu je indikováno prvním parametrem (číslo 1). Změníte-li první parametr na číslo 2, pak budete počítat přirozený logaritmus čísla 100. Příkaz bude vypadat takto:

W $ZM(2,100)<ET>

Výsledkem je číslo 4.60517019. Obráceným postupem při použití čísla 3 jako prvního parametru) se dojde k výsledku 100. Příkaz vypadá takto:

W $ZM(3,4.60517019)<ET>

9.5. Používání holého odkazu

Tato kapitola obsahuje doplňkové informace k tomu, jak je možno úsporněji přistupovat ke globálním proměnným. Tradiční přistup je takový, že se vypisuje vždy celá adresa proměnné, a to v příkazech i funkcích MUMPSu. Zjednodušením je metoda přístupu pomoci holého odkazu (též nahé reference). Pro vysvětlení uvádíme následující příklad.

S ^A(1)=100,^(2)=150,^(5)=300<ET>

Příkaz S má uložit do proměnné ^A(1) řetězec 100, do proměnné ^A(2) řetězec 150 a do proměnné ^A(5) řetězec 300.
Proč je tomu právě tak ?
MUMPS si totiž vždy pamatuje celou poslední použitou adresu globální proměnné. To znamená, že adresa ^A(1) byla poslední použitou adresou globální proměnné a tuto si zapamatoval. Dále si ještě navíc pamatuje poslední použitou úroveň indexů. V případě proměnné ^A(1) šlo o první úroveň. Z těchto předpokladů si pak vytváří celou novou adresu. To, že se znak ^ objeví přímo před závorkou (normální je to, že za znakem ^ je název proměnné a teprve pak závorka s indexy) říká, že adresu následující globální proměnné si musí MUMPS vytvořit sám. MUMPS tedy použije z původní adresy vše, až na index v nejnižší použité úrovni. V našem případě tedy použije ^A( ) a na první úroveb indexů dosadí číslo 2, takže vznikne adresa ^A(2). Tuto adresu použije k vykonání příkazu (S ^A(2)=150) a navíc si ji zapamatuje jak již bylo řečeno (jako poslední použitou adresu globální proměnné . Narazí-li dále na ^(5), tak dle stejných pravidel vytvoří adresu ^A(5) a tuto použije v příkazu S ^A(5)=300. Následující příkaz uloží řetězec "NE" do proměnné ^A(2, "JAN",7) .

S ^A(2,"JAN" , "TM") ="ANO", ^(7) ="NE"<ET>

Adresa (^A(2,"JAN",7)) může být vytvořena proto, že si MUMPS pamatuje poslední použitou adresu globální proměnné (^A(2,"JAN","TM")) a to, že poslední použitá roveň je úroveň třetí. Třetí proto, že proměnná ^A(2,"JAN,"TM") má tři indexy (první je číslo 2, druhá je řetězec "JAN" a tře- tí je řetězec "TM"). Na místo třetího indexu uloží MUMPS číslo 7 a tak vznikne nová adresa.
Holý odkaz může způsobit i zahloubení do dalších úrovní. Následující příklad toto zobrazuje.

S C="ABC",^A(2,"JAN",7)="NE",^(7,C)<ET>

Nově vzniklá adresa pak vypadá takto:

^A(2,"JAN",7,C) => ^A (2,"JAN" , 7,"ABC")

Nová adresa má čtyři úrovně. Na čtvrté úrovni je použita proměnná C.

9.6. Používání nepřímé adresace, tj. znaku @

Pojem nepřímá adresace zahrnuje velmi důležitou a silnou stránku MUMPSu. Několikrát již byla použita v textu bez dalšího zevšeobecňujícího vysvětlení. Nyní bychom Vám ji rádi přiblížili.
První a nejdůležitější podmínkou použití nepřímé adresace je znalost základního tvaru příkazu či funkce, v nichž chcete nepřímou adresaci použít. Například příkaz S:

S B=10,B=B*2 W B<ET>

To stejné lze napsat pomocí nepřímé adresace např. takto:

S A="B",B=10,@A=B*2 W B<ET

Výsledkem je zobrazení čísla 20 na obrazovce. Příklad se dá popsat takto. Do proměnné A je uložen jednoznakový řetězec B. Do proměněné B je uloženo číslo 10. @A=B*2 znamená, aby do proměnné, jejíž název je uložen v proměnné A (tedy do proměnné B) byl uložen obsah proměnné B vynásobeny číslem 2. Možná se Vám tento příklad zdá samoúčelný. Jde však o vysvětlení filozofie a použití v reálných situacích záleží na Vás. Další příklad ukazuje skok na návěští programu na disku.

G ZAC^PRIKLAD1<ET>

Ekvivalentní příkaz, za použití nepřímé adresace, je:

S A="ZAC^PRIKLAD1" G @A<ET>

Příkaz S uloží do proměnné A název programu a návěští z daného programu. Příkaz G @A provede spuštění programu od daného návěští, tedy spuštění programu PRIKLAD1 od návěští ZAC.
V tuto chvíli je potřebné pochopit to, že nepřímá adresace je vždy jakoby posunutí o jednu příkazovou úroveň dále. To znamená, je-li v základním příkazu použit název proměnné, návěští nebo například kontrola dle vzorku, pak je pro použití nepřímé adresace potřeba uložit proměnnou, návěští či řetězec kontroly dle vzorku do zprostředkující proměnné. Na tuto zprostředkující proměnnou se pak odvoláte pomocí znaku @ v místě, kde byste normálně uvedli název proměnné, návěští či řetězec kontroly dle vzorku. Tuto zásadu je potřebné mnohokráte vyzkoušet, až posléze budete naprosto jednoznačně chápat tvrzení o posunutí o jednu příkazovou úroveň. Poslední příkaz Vám ukáže možnost několikanásobného vnoření nepřímé adresace.

S A="@B",B="@C",C="D",@A="to C="D",@A="to je ale zmatek"<ET>

Výsledný řetězec je uložen do proměnné D.
Používání nepřímé adresace dovolí spoustu velice krkolomných výrazů, které často umožní takřka nemožné. Na druhé straně je si však třeba uvědomit, že každý program je někdy potřeba upravit. Bude-li tedy Váš i relativně jednoduchý program hýřit nepřímým adresováním, pak neshledáte patrně jinou možnost než napsat program nový.

Bude to znamenat, že jste velmi dobře pochopili nepřímé adresování, avšak je potřeba, abyste se naučili je ještě využít !

Příloha č.1 - tabulka příkazů

Příkazy z ANSI standardu:

BREAK	laděni programů
CLOSE	ukončení používání tiskárny, vnějších souborů a rozhraní V24 (RS232)
DO	skok se zachováním návratové adresy
ELSE	provede zbytek řádku, když $T=0
FOR	cyklus v rámci řádku
GOTO	skok bez zachování návrátové adresy
HALT	ukončení programu, ukončení práce v MUMPSu
HANG	pozastavení vykonávání programu
IF	proved zbytek řádku, když $T=1
KILL	ruší lokální a globální proměnné
OPEN	otevření vnějších zařízení (tiskárna, vnější soubory a rozhraní V24
PRINT	*vypsání programu z paměti na obrazovku
QUIT	ukončení cyklů a podprogramů
READ	načítání řetězců do proměnných z klávesnice, vnějších souborů či z rozhraní V24
SET	uložení řetězce do proměnné
USE	označení zařízení, na němž bude probíhat následující komunikace (tiskárna, vnější soubory či V24)
VIEW	zobrazení adresářů a proměnných
WRITE	vypsání řetězce na zvolené zařízení (obrazovka, tiskárna, vnější soubor či rozhraní V24
XECUTE	provedení příkazů, uložených v proměnných

Rozšiřující příkazy:

ZDELETE	příkaz pro zrušení programu na disku
ZGO	pokračování programu po přerušení příkazem BREAK
ZINSERT	vložení řádku do programu
ZLOAD	zavedení programu z disku do paměti
ZOPTION	mění přiřazení mechanik programům a globálům
ZPRINT	vypsání programu z paměti na obrazovku
ZREMOVE	mazání řádků programu či celého programu
ZSAVE	uložení programu z paměti na disk

*Příkaz nebude v nové ANSI normě. Používejte příkaz ZP.

Funkce z ANSI standardu:

$ASCII	zjistí pořadové číslo znaku v ASCII tabulce
$CHAR	převede pořadové číslo znaku z ASCII tabulky na znak
$DATA	zjistí existenci proměnné
$EXTRACT	vyjme podřetězec z řetězce
$FIND	zjistí zda v řetězci existuje daný podřetězců
$JUSTIFY	formátuje výpisy a zaokrouhluje

$LENGTH zjistí délku řetězce a množství podřetězců
$ZNEXT* zjistí část adresy dalšího prvku na dané úrovni
$ORDER zjistí část adresy dalšího prvku na dané úrovni
$PIECE vyjme (uloží) řetězec mezi oddělovači
$RANDOM generuje náhodné číslo v zadaném rozsahu
$SELHAT umožní větvit většinu příkazů
$TEXT vypíše 1 řádek z programu v paměti

Rozšiřující funkce:

$ZMAT provádí druhou odmocninu a konverze přirozeného logaritmu v obou směrech
$ZNEXT *zjistí celou adresu dalšího, reálného glob.prvku
$ORDER zjistí celou adresu dalšího reálného glob.prvku
$ZSORT *zjistí část adresy dalšího prvku na dané úrovni avšak jinak uspořádanou než v $ORDER

*Funkce jsou nahrazeny výkonnějšími funkcemi (ty jsou uvedeny vždy ihned pod nimi). V nové ANSII normě nebudou.

Systémové proměnné z ANSI standardu:

$ HOROLOG obsahuje datum a čas
$ IO obsahuje číslo právě používaného vstupně/výstupního (V / V) zařízeni
$STORAGE udává velikost nepoužitě uživatelské paměti
$TEST proměnná využívaná v příkazech IF a ELSE
$X obsahuje aktuální souřadnici sloupce obrazovky nebo tiskárny
$Y obsahuje aktuální sóuřadnici řádku obrazovky nebo tiskárny

Rozšiřující systémové proměnné:

$ZCOUNT obsahuje počet volných bloků v GLOBALS.DAT
$ZERROR použitelná pro řešení programových havarijních situací
$ZGLOBAL obsahuje název mechaniky, v níž jsou aktuální globály (soubor GLOBALS.DAT)
$ZNAME obsahuje název programu, který je právě v uživatelské paměti
$ZROUTINE obsahuje název mechaniky, v níž jsou volané programy

ASCII tabulka

10	16	znak	10	16	znak	10	16	znak	10	16	znak
0	00	NULL	32	20	SP	64	40	@	96	60	
1	01	SOH	33	21	!	65	41	A	97	61	a
2	02	STX	34	22	"	66	42	B	98	62	b
3	03	ETX	35	23	#	67	43	C	99	63	c
4	04	EOT	36	24	$	68	44	D	100	64	d
5	05	ENQ	37	25	%	69	45	E	101	65	e
6	06	ACK	38	26	&	70	46	F	102	66	f
7	07	BEL	39	27	'	71	47	G	103	67	g
8	08	BS	40	28	(	72	48	H	104	68	h
9	09	HT	41	29	)	73	49	I	105	69	i
10	OA	LF	42	2A	*	74	4A	J	106	6A	j
.11	OB	VT	43	2B	+	75	4B	K	107	6B	k
12	OC	FF	44	2C	,	76	4C	L	108	6C	l
13	OD	CR	45	2D	-	77	4D	M	109	6D	m
14	OE	SO	46	2E	.	78	4E	N	110	6E	n
15	OF	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	-49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DO	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	-53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	lE	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F		127	7F	DEL

Drobné programy :

NASOB.MMP
ZAC ;tisk male nasobilky; 10 JAN 1989 at 10:25
W "Tisk male nasobilky",!!
F B=1:5:10 W !! F J=1:1:20 W ! S SL=-16 F I=B:1:B+4 D TISK
Q
TISK S SL=SL+16 W ?SL,I," * ",J," = ",I*J Q

CURSOR.MMP

```
CURSOR      ;definice funkci cursoru; 10 JAN 1989 at 22:50
            s CLS="*27,” ”[2J “ ” ”
            s CLL="*27, “ “ [K” ”
            s BEEP="*27,*7”
            s CUU="*27,*91,DX+1,""A"""
            s CUD="*27,*91,DX-1,""B"""
            s HOME="*27, " "[H"""
```

```
                s CUP="*27,*91,DY+1,"";"",DX+1,” “H” “ “
                s COLOR="*27,*91,Z,""m"""
                s MOD=11*27,*91,””=11””,Z,””””h"""”
                s BS="*27,*8"
                s CR="*27,*13"
                s LF="*27,*10"
                s HI=”*27,””[1m”””
                s L I = " *27, “”[Om”””
                s BLINK=”*27,*91,5,””m”””
                s DEL=”*27,*8,”” "",*27,*8"
                s BLANK="                                                    “
SETCOL          S BILCERV=11*27,*91,37,””m"",*27,*91,41,””m”””
                S ZELCERV="*27,*91,32,""m"",*27,*91,41,””m”
                S ZELMOD=11*27,*91,32,""m"",*27,*91,44,””m”””
                S BILMOD="*27,*91,37,""m"",*27,*91,44,""m"""”
                S MODZEL= 11 *27, *91, 34, " " m" " , *27, *91, 42,””m”””
                S CERNZEL=11*27,*91,30,””m"",*27,*91,42,"”m"""
                S CERNMOD="*27,*91,30,"llm"",*27,*91,44,""m"""
                S B I ICERN= "*27, *91, 37, " "m" " , *27, *91, 40,””m”””
                S ZELCERN=”*27,*91,32,""m"",*27,*91,40,””m”””
                S CERVMOD="*27,*91,31,""m"",*27,*91,44,””m”””
```

Příloha č.2 chybové zprávy MUMPSu

Číslo	Text chyby
0	Chybí znak začátku řádku
1	Přeplnění zásobníku
2	Aritmetické přeplnění
3	Nedefinovaná lokální proměnná
4	Nesprávný název funkce
5	Nesprávný název příkazu
6	Nedefinovany název programu
7	Nedefinovaná globální proměnná
8	Globál a hodnota jsou příliš dlouhé
9	Plný adresář
10	Program větší než uživatelská oblast
11	Řetězec je příliš dlouhý
12	Nesprávný počet závorek
13	Nesprávně použitý operátor negace
14	Nesprávně ukončený příkaz

15 Příliš mnoho nahých odkazů
16 Nedovolený přístup ke globálu
17 Dělení nulou
18 Nesprávný znak
19 Přeplnění syntaktického zásobníku
20 Nesprávný výraz
21 Nesprávný vzorek
22 Chybí čárka
23 Nesprávné jméno proměnné
24 Nesprávné použití nepřímé adresace
25 Nedefinované číslo programového řádku
26 Nesprávný numerický literál
27 Chybí rovnítko
28 Nesprávný název programu či návěští
29 Nesprávná syntax názvu
30 Neimplementovaná operace
31 Přeplnění tabulky symbolů
32 Systémová chyba
33 Duplikované návěští
34 Nesprávný odkaz na řádek
35 V $SELECT není pravdivá hodnota
36 Nesprávná nahá reference globálu
37 Soubor globálů není na disku
38 Program není na disku
39 Disková V/V chyba
40 Program není pojmenován
41 Program je už v knihovně
42 Zruš nebo schovej program
43 Příkaz používaný jen v nepřímém režimu
44 Nesprávný výraz v průběhu BREAKu
45 Příkaz používaný jen v průběhu BREAKu
46 Chybí index
47 Nesprávný index
48 Nesprávný parametr OPEN
49 Zařízení není otevřeno
50 Nesprávná mechanika
51 Čtení ze zařízení, na něž lze jen zapisovat
52 Nesprávné číslo zařízení
53 Není použito (rezerva)
54 Chyba kontrolní sumy při opakování
55 Chyba kontrolní sumy při čtení