NumToStr

JEDNOTKA PRO PŘEVODY ČÍSEL, DATUMU A ČASU A JINÝCH DATOVÝCH STRUKTUR NA ŘETĚZCE A ZPĚT

Příručka uživatele a programátora

***SofCon*®spol. s r.o.**
Střešovická 49
162 00 Praha 6
tel/fax: +420 220 180 454
E-mail: sofcon@sofcon.cz
www: http://www.sofcon.cz

Verze dokumentu 1.20

Informace v tomto dokumentu byly pečlivě zkontrolovány a SofCon věří, že jsou spolehlivé, přesto SofCon nenese odpovědnost za případné nepřesnosti nebo nesprávnosti zde uvedených informací.
SofCon negarantuje bezchybnost tohoto dokumentu ani programového vybavení, které je v tomto dokumentu popsané. Uživatel přebírá informace z tohoto dokumentu a odpovídající programové vybavení ve stavu, jak byly vytvořeny a sám je povinen provést validaci bezchybnosti produktu, který s použitím zde popsaného programového vybavení vytvořil.
SofCon si vyhrazuje právo změny obsahu tohoto dokumentu bez předchozího oznámení a nenese žádnou odpovědnost za důsledky, které z toho mohou vyplynout pro uživatele.

Datum vydání: 20.02.2004
Datum posledního uložení dokumentu: 20.02.2004
(Datum vydání a posledního uložení dokumentu musí být stejné)

Upozornění:
V dokumentu použité názvy výrobků, firem apod. mohou být ochrannými známkami nebo registrovanými ochrannými známkami příslušných vlastníků.

Obsah :

1. O dokumentu

1.1. Revize dokumentu

Verze dokumentu	Verze SW	Autor	Datum vydání	Popis změn
1.00	1.XX	Wil		První vydání
1.100	3.XX	Tum	16.05.2003	Úprava dokumentu dle ISO9000
1.20	4.XX	Wil	20.02.2004	Přidán typ tString24 = String[24] Úprava funkcí ByteStrOct, WordStrOct, LongIntStrOct, OctStrByte, OctStrWord, OctStrLongInt – používání kratších stringů => šetření Stacku. Přidání funkcí NibbleCharHex, HexCharNibble, DwordStrDec, PlainBuffStrHex, BufferStrHex2

1.2. Účel dokumentu

Tento dokument slouží jako popis jednotky pro převod čísel, datumu a času a jiných datových struktur na textové řetězce a zpět.

1.3. Rozsah platnosti

Určen pro programátory a uživatele programového vybavení SofCon.

1.4. Související dokumenty

Pro čtení tohoto dokumentu není potřeba číst žádný další manuál.

Popis formátu verze knihovny a souvisejících funkcí je popsán v manuálu LibVer.

2. Termíny a definice

Používané termíny a definice jsou popsány v samostatném dokumentu „Termíny a definice“.

3. Úvod

Jednotka NumToStr definuje procedury a funkce, které slouží pro převádění hodnot celočíselných a reálných čísel na textové řetězce (dále jen řetězce) a zpět, dále pro převádění datumu a času a dalších datových struktur na řetězce a zpět.

Pro celočíselné převody jednotka obsahuje funkce pro převádění čísel konkrétních typů (byte, word, longint) na řetězce s konkrétní representací v číselné soustavě (dekadická, binární, oktalová, hexadecimální) i univerzální funkce pro převádění obecného čísla (maximální velikosti longint) na řetězce se zvolenou representací v číselné soustavě.

Při převádění reálných typů se používá nejrůznějších formátů výpisu reálného čísla.

Pro převody a práci s datumem a časem jednotka používá funkcí jednotek Dos, WinDos či RtcTime, na základě použité cílové platformy (MCP, PC-DOS či PC-Windows).

Ostatní procedury a funkce převádějící další datové struktury se týkají například převodu ukazatelů na řetězce a zpět, převodu množin bytů či znaků na řetězce a zpět nebo uspořádání (přeskupení) jednotlivých bitů v bytech či wordech.

4. Popis konstant a typů

```
cVerNo = např. $0251; { BCD formát }
cVer   = např. '02.51,07.08.2003';
```

Číslo verze jednotky v BCD tvaru a v textové podobě včetně datumu změny.

```
Nibble     = $0..$F;
```

Intervalový typ **Nibble** určuje typ celého čísla o velikosti 4 bity.

```
tString1  = String[1];
tString2  = String[2];
tString3  = String[3];
tString4  = String[4];
tString5  = String[5];
tString6  = String[6];
tString8  = String[8];
tString9  = String[9];
tString10 = String[10];
tString11 = String[11];
tString12 = String[12];
tString16 = String[16];
tString19 = String[19];
tString24 = String[24];
tString32 = String[32];
tString39 = String[39];
```

Řetězcové typy pro omezení délky používaných řetězců.

```
tIntType = (tpDec, tpBin, tpOkt, tpHex);
```

Výčtový typ **tIntType** určuje číselnou soustavu, do které (nebo z které) se číslo převádí.

tpDec určuje dekadickou soustavu.

tpBin určuje binární soustavu.

tpOkt určuje oktalovou soustavu.

tpHex určuje hexadecimální soustavu.

```
tRealType = (tpMaxDes, tpFixedDes, tpFixedLen);
```

Výčtový typ **tRealType** určuje formát převodu reálného čísla.

tpMaxDes určuje, že převáděné reálné číslo se zaokrouhlí na zadaný počet desetinných míst s oříznutím přebytečných desetinných nul.

tpFixedDes určuje, že převáděné reálné číslo se zaokrouhlí na zadaný počet desetinných míst bez oříznutí přebytečných desetinných nul (pevný počet desetinných míst).

tpFixedLen určuje, že převáděné reálné číslo (včetně desetinných míst) bude mít zadanou délku za předpokladu, že délka celé části čísla je menší než požadovaná celková velikost. V opačném případě se převede jen celá část čísla, která může být v tomto případě delší než požadovaná celková délka.

```
tLanguage = (Us, Cz, CzWin);
```

Výčtový typ **tLanguage** určuje používaný jazyk. Využívají ho procedury a funkce pro převody datumu a času na řetězec pro názvy dní a měsíců.

Význam jednotlivých jazyků:

Us - americká angličtina

Cz - čeština bez diakritiky

CzWin - čeština pro Windows

```
tByteSet  = Set of Byte;
tCharSet  = Set of Char;
```

Typy množin bytů a znaků. Využívají je procedury a funkce pro převody těchto množin na řetězec a zpět.

```
tConvTabWord = array[0..15] of Byte;
tConvTabByte = array[0.. 7] of Byte;
```

Typy polí bytů pro funkce ConvertByteBits a ConvertWordBits.

```
Language : tLanguage = Cz;
```

Proměnná definující aktuální používaný jazyk s implicitním nastavením na češtinu bez diakritiky.

```
MaxDateTime = $FF9FBF7D; { 31.12.2107 23:59:58 }
MinDateTime = $00210000; { 01.01.1980 00:00:00 }
```

Konstanty definující minimální a maximální hodnotu datumu a času v zapakovaném (komprimovaném) formátu.

5. Procedury a funkce

5.1. Převody celých čísel na řetězce v binárním tvaru a zpět

5.1.1. NibbleStrBin

```
function  NibbleStrBin (N : Nibble) : tString4;
```

Funkce **NibbleStrBin** převádí čtyřbitové číslo N na řetězec o délce čtyři znaky s binární reprezentací parametru N.

```
Př.:  NibbleStrBin(5) = '0101';
```

5.1.2. ByteStrBin

```
function  ByteStrBin (B : Byte) : tString9;
```

Funkce **ByteStrBin** převádí byte B na řetězec o délce 9 znaků s binární reprezentací parametru B. Dolní čtyři bity jsou od horních čtyřech bitů odděleny mezerou.

```
Př.:  ByteStrBin($5A) = '0101 1010';
```

5.1.3. WordStrBin

```
function  WordStrBin (W : Word) : tString19;
```

Funkce **WordStrBin** převádí word W na řetězec o délce 19 znaků s binární reprezentací parametru W. Čtveřice bitů jsou navzájem odděleny mezerou.

```
Př.:  WordStrBin($5AC3) = '0101 1010 1100 0011';
```

5.1.4. LongIntStrBin

```
function  LongIntStrBin (L : LongInt) : tString39;
```

Funkce **LongIntStrBin** převádí longint L na řetězec o délce 39 znaků s binární reprezentací parametru L. Čtveřice bitů jsou navzájem odděleny mezerou.

```
Př.:  LongIntStrBin($5AF0C3E1) =
      = '0101 1010 1111 0000 1100 0011 1110 0001';
```

5.1.5. BinStrNibble

```
function  BinStrNibble (const S : tString8) : Nibble;
```

Funkce **BinStrNibble** převádí řetězec S, který je maximální délky 8 znaků a obsahuje binární reprezentaci čísla, na čtyřbitové číslo (nibble). Jiné znaky než '0' a '1' v řetězci S se přeskakují.

```
Př.:  BinStrNibble('1010')    = 10;
      BinStrNibble('1a10011') = 12;  ☺'a' se vypustí a přebytečné
      znaky se ignorují
```

5.1.6. BinStrByte

```
function  BinStrByte (const S : tString16) : Byte;
```

Funkce **BinStrByte** převádí řetězec S, který je maximální délky 16 znaků a obsahuje binární reprezentaci čísla, na byte. Jiné znaky než '0' a '1' v řetězci S se přeskakují. Příklad je podobný jako pro funkci **BinStrNibble**.

5.1.7. BinStrWord

```
function  BinStrWord (const S : tString32) : Word;
```

Funkce **BinStrWord** převádí řetězec S, který je maximální délky 32 znaků a obsahuje binární reprezentaci čísla, na word. Jiné znaky než '0' a '1' v řetězci S se přeskakují. Příklad je podobný jako pro funkci **BinStrNibble**.

5.1.8. BinStrLongInt

```
function  BinStrLongInt (const S : tString39) : LongInt;
```

Funkce **BinStrLongInt** převádí řetězec S, který je maximální délky 39 znaků a obsahuje binární reprezentaci čísla, na longint. Jiné znaky než ‘0’ a ‘1’ v řetězci S se přeskakují. Příklad je podobný jako pro funkci **BinStrNibble**.

5.1.9. SetStrBin

```
function  SetStrBin (L : LongInt; Len : Byte) : tString32;
```

Funkce **SetStrBin** převádí množinu bitů L na řetězec v binárním zápisu. Z množiny L převádí postupně bity od nultého až do Len-1 (Len tedy určuje kolik bitů se má převádět).

Př.: SetStrBin(10011_{bin},3) = ‘011’;

5.2. Převody celých čísel na řetězce v oktalovém tvaru a zpět

5.2.1. NibbleStrOct

```
function  NibbleStrOct (N : Nibble) : tString2;
```

Funkce **NibbleStrOct** převádí čtyřbitové číslo N na řetězec s oktalovou reprezentací parametru N.

Př.: NibbleStrOct(10) = ‘12’;

5.2.2. ByteStrOct

```
function  ByteStrOct (B : Byte) : tString3;
```

Funkce **ByteStrOct** převádí byte B na řetězec s oktalovou reprezentací parametru B.

Př.: ByteStrOct($5A) = ‘0132’;

5.2.3. WordStrOct

```
function  WordStrOct (W : Word) : tString6;
```

Funkce **WordStrOct** převádí word W na řetězec s oktalovou reprezentací parametru W. Příklad je podobný jako pro funkce **NibbleStrOct** a **ByteStrOct**.

5.2.4. LongIntStrOct

```
function  LongIntStrOct (L : LongInt) : tString11;
```

Funkce **LongIntStrOct** převádí longint L na řetězec s oktalovou reprezentací parametru L. Příklad je podobný jako pro funkce **NibbleStrOct** a **ByteStrOct**.

5.2.5. OctStrNibble

```
function  OctStrNibble (const S : tString4) : Nibble;
```

Funkce **OctStrNibble** převádí řetězec S, který je maximální délky 4 znaky a obsahuje oktalovou reprezentaci čísla, na čtyřbitové číslo (nibble). Jiné znaky než ‘0’ až ‘7’ v řetězci S se přeskakují.

```
Př.:  OctStrNibble(‘10’)   = 8;
      OctStrNibble(‘a 17’) = 15;
```

5.2.6. OctStrByte

```
function  OctStrByte (const S : tString6) : Byte;
```

Funkce **OctStrByte** převádí řetězec S, který je maximální délky 6 znaků a obsahuje oktalovou reprezentaci čísla, na byte. Jiné znaky než ‘0’ až ‘7’ v řetězci S se přeskakují. Příklad je podobný jako pro funkci **OctStrNibble**.

5.2.7. OctStrWord

```
function  OctStrWord (const S : tString12) : Word;
```

Funkce **OctStrWord** převádí řetězec S, který je maximální délky 12 znaků a obsahuje oktalovou reprezentaci čísla, na word. Jiné znaky než ‘0’ až ‘7’ v řetězci S se přeskakují. Příklad je podobný jako pro funkci **OctStrNibble**.

5.2.8. OctStrLongInt

```
function  OctStrLongInt (const S : tString24) : LongInt;
```

Funkce **OctStrLongInt** převádí řetězec S, který je maximální délky 24 znaků a obsahuje oktalovou reprezentaci čísla, na longint. Jiné znaky než ‘0’ až ‘7’ v řetězci S se přeskakují. Příklad je podobný jako pro funkci **OctStrNibble**.

5.3. Převody celých čísel na řetězce v hexadecimálním tvaru a zpět

5.3.1. NibbleStrHex

```
function  NibbleStrHex (N : Nibble) : tString1;
```

Funkce **NibbleStrHex** převádí čtyřbitové číslo N na řetězec o délce jeden znak s hexadecimální reprezentací parametru N.

```
Př.:  NibbleStrHex(14) = ‘E’;
```

5.3.2. ByteStrHex

```
function  ByteStrHex (B : Byte) : tString2;
```

Funkce **ByteStrHex** převádí byte B na řetězec o délce dva znaky s hexadecimální reprezentací parametru B. Příklad je podobný jako pro funkci **NibbleStrHex**.

5.3.3. WordStrHex

```
function  WordStrHex (W : Word) : tString4;
```

Funkce **WordStrHex** převádí word W na řetězec o délce čtyři znaky s hexadecimální reprezentací parametru W.

```
Př.:  WordStrHex($F5A) = '0F5A';
```

5.3.4. LongIntStrHex

```
function  LongIntStrHex (L : LongInt) : tString8;
```

Funkce **LongIntStrHex** převádí longint L na řetězec o délce 8 znaků s hexadecimální reprezentací parametru L. Příklad je podobný jako pro funkci **WordStrHex**.

5.3.5. NibbleCharHex

```
function  NibbleCharHex (N : Nibble) : char;
```

Funkce **NibbleCharHex** převádí čtyřbitové číslo N na znak s hexadecimální reprezentací parametru N. Tato funkce je mnohem efektivnější (co se týče náročnosti na zpracování) než funkce **NibbleStrHex**.

```
Př.:  NibbleCharHex(14) = 'E';
```

5.3.6. HexStrNibble

```
function  HexStrNibble (const S : tString2) : Nibble;
```

Funkce **HexStrNibble** převádí řetězec S, který je maximální délky dva znaky a obsahuje hexadecimální reprezentaci čísla, na čtyřbitové číslo (nibble). Jiné znaky než '0' až '9','A' až 'F' a 'a' až 'f' v řetězci S se přeskakují.

```
Př.:  HexStrNibble('F1') = 15;
```

5.3.7. HexStrByte

```
function  HexStrByte (const S : tString4) : Byte;
```

Funkce **HexStrByte** převádí řetězec S, který je maximální délky 4 znaky a obsahuje hexadecimální reprezentaci čísla, na byte. Jiné znaky než '0' až '9','A' až 'F' a 'a' až 'f' v řetězci S se přeskakují. Příklad je podobný jako pro funkci **HexStrNibble**.

5.3.8. HexStrWord

```
function  HexStrWord (const S : tString8) : Word;
```

Funkce **HexStrWord** převádí řetězec S, který je maximální délky 8 znaků a obsahuje hexadecimální reprezentaci čísla, na word. Jiné znaky než '0' až '9','A' až 'F' a 'a' až 'f' v řetězci S se přeskakují.

```
Př.:  HexStrWord('kAB 45') = 43845;
```

5.3.9. HexStrLongInt

```
function  HexStrLongInt (const S : tString16) : LongInt;
```

Funkce **HexStrLongInt** převádí řetězec S, který je maximální délky 16 znaků a obsahuje hexadecimální reprezentaci čísla, na longint. Jiné znaky než ‘0’ až ‘9’,’A’ až ‘F’ a ‘a’ až ‘f’ v řetězci S se přeskakují. Příklad je podobný jako pro funkci **HexStrWord**.

5.3.10. HexCharNibble

```
function  HexCharNibble (Zn : char) : Nibble;
```

Funkce **HexCharNibble** převádí znak Zn, který obsahuje hexadecimální reprezentaci čísla, na čtyřbitové číslo (nibble). Pokud parametr Zn bude mít jinou hodnotu než znaky ‘0’ až ‘9’,’A’ až ‘F’ a ‘a’ až ‘f’, vrátí funkce číslo 0.

```
Př.:  HexCharNibble(‘F’) = 15;
```

5.4. Převody celých čísel na řetězce v dekadickém tvaru

5.4.1. NibbleStrDec

```
function  NibbleStrDec (N : Nibble; Len : Byte) : tString2;
```

Funkce **NibbleStrDec** převádí čtyřbitové číslo N na řetězec s dekadickou reprezentací parametru N. Parametr Len určuje, na kolik míst se má výsledný řetězec zarovnat (zepředu doplnit mezerami). Pokud je parametr Len menší než délka převedeného řetězce, zarovnání se neprovede. Pokud je parametr Len větší než maximální délka převedeného řetězce, zarovná se řetězec na maximum (v tomto případě 2).

```
Př.:  NibbleStrDec(10,1) = ‘10’;
```

5.4.2. ByteStrDec

```
function  ByteStrDec (B : Byte; Len : Byte) : tString3;
```

Funkce **ByteStrDec** převádí byte B na řetězec s dekadickou reprezentací parametru B. Použití parametru Len je stejné jako pro funkci **NibbleStrDec**.

```
Př.:  ByteStrDec($3F,3) = ‘ 63’;
```

5.4.3. WordStrDec

```
function  WordStrDec (W : Word; Len : Byte) : tString5;
```

Funkce **WordStrDec** převádí word W na řetězec s dekadickou reprezentací parametru W. Použití parametru Len je stejné jako pro funkci **NibbleStrDec**. Příklad je podobný jako pro funkci **ByteStrDec**.

5.4.4. DWordStrDec

```
function  DWordStrDec (D : Longint; Len : Byte) : tString10;
```

Funkce **DWordStrDec** převádí doubleword D (jelikož Pascal verze 7.0 nezná typ DWORD, je použit typ LONGINT) na řetězec s dekadickou reprezentací parametru D. Použití parametru Len je stejné jako pro funkci **NibbleStrDec**. Příklad je podobný jako pro funkci **ByteStrDec**.

5.4.5. IntegerStrDec

```
function  IntegerStrDec (I : Integer; Len : Byte) : tString6;
```

Funkce **IntegerStrDec** převádí integer na řetězec s dekadickou reprezentací parametru I. Použití parametru Len je stejné jako pro funkci **NibbleStrDec**.

```
Př.:  IntegerStrDec(-145,5) = ' -145';
```

5.4.6. LongIntStrDec

```
function  LongIntStrDec (L : LongInt; Len : Byte) : tString11;
```

Funkce **LongIntStrDec** převádí longint L na řetězec s dekadickou reprezentací parametru L. Použití parametru Len je stejné jako pro funkci **NibbleStrDec**. Příklad je podobný jako pro funkci **IntegerStrDec**.

5.5. Převody celých čísel na řetězce se zvolenou číselnou soustavou

5.5.1. IntToStr

```
function IntToStr(L : LongInt; Typ : tIntType; Len : Byte):tString32;
```

Funkce **IntToStr** převádí číslo L, které je maximální velikosti longint, na řetězec o maximální délce 32 znaků s reprezentací v číselné soustavě zadané parametrem Typ. Použití parametru Len je podobné jako u funkcí pro převod celých čísel na řetězce v dekadické reprezentaci s tím rozdílem, že výsledný řetězec je zepředu doplňován místo mezer nulami.

```
Př.:  IntToStr($5A  ,tpBin,8) = '01011010';
       IntToStr(-1   ,tpHex,4) = 'FFFFFFFF';
       IntToStr($5A  ,tpDec,0) = '90';
       IntToStr($5A3C,tpOkt,7) = '0055074';
```

Pozn: Funkce **IntToStr** nahrazuje všechny tyto funkce: NibbleStrBin až LongIntStrBin, NibbleStrOct až LongIntStrOct, NibbleStrHex až LongIntStrHex a částečně i nahrazuje funkce NibbleStrDec až LongIntStrDec, které ale pro zarovnání používají mezeru, kdežto tato funkce používá '0'.

5.5.2. StrToInt

```
function StrToInt (S : tString39; Typ : tIntType;
                   var ErrCode : Integer) : longint;
```

Funkce **StrToInt** je inverzní funkcí k funkci **IntToStr**. Funkce převádí číslo zadané v řetězci S, který je délky maximálně 39 znaků, s reprezentací v číselné soustavě zadané parametrem Typ na číslo maximální velikosti longint. Případné mezery a počáteční nuly v řetězci S se přeskakují. Pokud funkce při dekódování řetězce S narazí na jiný znak, než který přísluší požadované číselné soustavě, vrátí index tohoto znaku v proměnné ErrCode a převádění dále neprovádí.

```
Př.:  StrToInt('01101', tpBin,ErrCode) = 13;    ErrCode = 0;
      StrToInt('1F0A C',tpHex,ErrCode) = 7946;  ErrCode = 5;
      StrToInt('14.3',  tpDec,ErrCode) = 14;    ErrCode = 3;
      StrToInt('77779', tpOkt,ErrCode) = $FFF;  ErrCode = 5;
```

Pozn: Funkce **StrToInt** částečně nahrazuje všechny tyto funkce: BinStrNibble až BinStrLongInt, OctStrNibble až OctStrLongInt a HexStrNibble až HexStrLongInt. Liší se od nich jen v tom, že znaky, které nepatří do dané číselné soustavy, bere jako chybné, tj. je schopna zdetekovat nesprávné formáty. Funkce je navíc velice univerzální, ale oproti výše zmíněným funkcím má o něco větší režii procesorového času.

5.6. Převody reálných čísel na řetězce

5.6.1. RealStrDec

```
function  RealStrDec (R : Real; Len : Byte; Flt : Byte) : tString32;
```

Funkce **RealStrDec** převádí reálné číslo na řetězec dvěma způsoby, které závisí na zadaných parametrech Len a Flt.

1. Pokud Len = Flt, převede funkce číslo R na řetězec, který potom ořeže odzadu tak, aby jeho výsledná délka byla Len.

```
Př.:  RealStrDec(123.56, 7, 7) = '123.560';
      RealStrDec(123.56, 2, 2) = '12';
```

2. Pokud Len ≠ Flt, převede funkce číslo R na řetězec tak, že Flt bude pevný počet desetinných míst, a pokud celková délka čísla bude menší než Len, zarovná se řetězec zepředu mezerami na délku Len.

```
Př.:  RealStrDec(123.56, 8, 3) = ' 123.560';
      RealStrDec(123.56, 5, 4) = '123.5600';
```

5.6.2. RealStrExp

```
function  RealStrExp (R : Real; Len : Byte) : tString16;
```

Funkce **RealStrExp** převádí reálné číslo R na řetězec v exponenciálním tvaru. Minimální výsledná délka řetězce je vždy 8 (první znak je vždy '-' pro záporné číslo nebo ' ' pro kladné číslo, poté následují minimálně tři znaky (např. '1.2') a nakonec exponent (např. 'E-05')). Pokud parametr Len bude větší než 8, převede se tolik desetinných míst, aby výsledná délka řetězce byla Len.

```
Př.:  RealStrExp(123.56,  6) = ‘ 1.2E+02’;
      RealStrExp(-123.56,10) = ‘-1.236E+02’;
```

5.6.3. RealToStr

```
function RealToStr(R : Real; Len : Byte; Typ : tRealType) :tString32;
```

Funkce **RealToStr** převádí reálné číslo R na řetězec. Parametr Typ určuje formát a způsob převádění reálného čísla:

Typ = tpMaxDes - Reálné číslo se zaokrouhlí na Len desetinných míst a přebytečné koncové nuly za desetinnou tečkou se oříznou.

Typ = tpFixedDes - Reálné číslo se zaokrouhlí na Len pevných desetinných míst.

Typ = tpFixedLen - Reálné číslo se zaokrouhlí tak, aby jeho celkový počet míst byl Len.

```
Př.:  RealToStr(-25.50149, 2, tpMaxDes)   = ‘-25.5’;
      RealToStr(-25.50149, 4, tpFixedDes) = ‘-25.5015’;
      RealToStr(-25.50149, 7, tpFixedLen) = ‘-25.501’;
      RealToStr(-25.50149, 2, tpFixedLen) = ‘-25’;
```

5.7. Převody datumu a času na řetězce a zpět

5.7.1. TimeToStr

```
function TimeToStr (DT: DateTime; const Param: tString39): tString39;
```

Funkce **TimeToStr** převede čas z parametru DT, který obsahuje datum a čas v záznamu typu DateTime, na řetězec podle formátu zadaném parametrem Param. Typ DateTime je definován v jednotce Dos.

Parametr Param určuje formát času ve výsledku funkce.
Funkční znaky parametru Param:

`'H'` - údaj hodin (ve výsledku např.: '12','9')

`'M'` - údaj minut (ve výsledku např.: '7','45')

`'S'` - údaj sekund (ve výsledku např.: '8','32')

`'HH'` - údaj hodin zarovnaný na 2 místa (ve výsledku např.: '02','12')

`'MM'` - údaj minut zarovnaný na 2 místa (ve výsledku např.: '45','09')

`'SS'` - údaj sekund zarovnaný na 2 místa (ve výsledku např.: '30','03')

Ostatní znaky se striktně opisují do výsledného řetězce, čehož lze využít při použití libovolných oddělovacích znaků. Pozn: Funkční znaky musí být zadány velkými písmeny, jedná-li se o malý znak, není to znak funkční a do výsledku se pouze opíše.

```
Př.:  DT.Hour = 9; DT.Min = 24; DT.Sec = 5;
      TimeToStr(DT, 'HH:MM:SS')       = '09:24:05';
      TimeToStr(DT, 'H:MM')           = '9:24';
      TimeToStr(DT, 'H min S sek')    = '24 min 5 sek';
      TimeToStr(DT, 'čas H/M/S')      = 'čas 9/24/5';
      Pozor! TimeToStr(DT, 'M Minut') = '24 24inut';
```

5.7.2. TimeToStrPT

```
function TimeToStrPT (Time : Longint; const Param : tString39)
                    : tString39;
```

Funkce **TimeToStrPT** převede čas na řetězec stejně jako funkce **TimeToStr**, ale jako parametr s časem se jí zadá datum a čas v zapakovaném formátu, který je definován v operačním systému MS-DOS. Pokud parametr Time = 0, bude se převádět aktuální systémový čas.

5.7.3. DateToStr

```
function DateToStr (DT: DateTime; const Param: tString39): tString39;
```

Funkce **DateToStr** převede datum z parametru DT na řetězec podle formátu zadaném parametrem Param.

Parametr Param určuje formát datumu ve výsledku funkce.
Funkční znaky parametru Param:

`'Y'` nebo `'YY'`	- údaj posledního dvojčíslí roku (např.: '97')
`'YYY'` nebo `'YYYY'`	- údaj celého roku (např.: '1997','2000')
`'M'`	- údaj měsíce (např.: '8','11')
`'MM'`	- údaj měsíce zarovnaný na 2 místa (např.: '08','12')
`'MMMM'`	- údaj názvu měsíce podle nastaveného jazyka (např.: 'September','prosinec')
`'D'`	- údaj dne (např.: '9','31')
`'DD'`	- údaj dne zarovnaný na 2 místa (např.: '30','03')
`'W'` nebo `'WW'`	- údaj zkratky názvu dne v týdnu podle nastaveného jazyka (např.: 'Su','ne')
`'WWW'` nebo `'WWWW'`	- údaj celého názvu dne v týdnu podle nastaveného jazyka (např.: 'Sunday','neděle')

Ostatní znaky se striktně opisují do výsledného řetězce, čehož lze využít při používání libovolných oddělovacích znaků. Pozn: Funkční znaky musí být zadány velkými písmeny, jedná-li se o malý znak, není to znak funkční a do výsledku se pouze opíše.

```
Př.:  DT.Year = 1997; DT.Month = 3; DT.Day = 6;
      DateToStr(DT, 'DD.MM.YYYY')      = '06.03.1997';
      DateToStr(DT, 'DD/MM/YY')        = '06/03/97';
      DateToStr(DT, 'Y_M')             = '97_3';
      DateToStr(DT, 'datum D.M.YYYY')  = 'datum 6.3.1997';
      DateToStr(DT, 'D.MMMM YYYY')     = '6.březen 1997';
      DateToStr(DT, 'D.MMMM YYYY')     = '6.March 1997';
      DateToStr(DT, 'D.MMMM WW')       = '6.brezen ct';
      DateToStr(DT, 'WWWW D.MM.YY')    = 'Thursday 6.03.97';
      Pozor! DateToStr(DT, 'Datum DD') = '6atum 06';
```

5.7.4. DateToStrPT

```
function DateToStrPT (Time : Longint; const Param : tString39)
                    : tString39;
```

Funkce **DateToStrPT** převede datum na řetězec stejně jako funkce **DateToStr**, ale jako parametr se jí zadá datum a čas v zapakovaném formátu. Pokud parametr Time = 0, bude se převádět aktuální systémové datum.

5.7.5. StrToDate

```
function StrToDate (const S : tString39; var DT : DateTime): boolean;
```

Funkce **StrToDate** převede datum zadané v řetězci S do zadaného "var" parametru DT typu DateTime a vrátí příznak úspěšnosti převodu. Formát parametru S by měl být zhruba následující: "DD.MM.YYYY" nebo "D.M.YY", tj. pořadí den, měsír, rok. Přeházené pořadí (např. měsíc před dnem) není přípustné. Pro správné dekódování letopočtu i v případě roku 2000 a vyšším, platí pro jejich zadávání jistá pravidla (zejména při zadávání jen posledního dvojčíslí letopočtu):

1. Je zadán rok 1980 a více, výsledný rok bude shodný se zadaným.
2. Je zadáno poslední dvojčíslí roku v rozsahu 0 až 79, výsledný rok bude 2000 + zadané poslední dvojčíslí.
3. Je zadáno poslední dvojčíslí roku v rozsahu 80 až 99, výsledný rok bude 1900 + zadané poslední dvojčíslí.

V jiných případech (např. "120") bude funkce vracet chybu.

```
Př.:  StrToDate('07.8.97',DT) = true;
            DT.Year  = 1997;
            DT.Month = 8;
            DT.Day   = 7;
            DT.Hour,DT.Min,DT.Sec nezměněny
```

5.7.6. StrToTime

```
function StrToTime (const S : tString39; var DT : DateTime): boolean;
```

Funkce **StrToTime** převede čas zadaný v řetězci S do zadaného "var" parametru DT typu DateTime a vrátí příznak úspěšnosti převodu. Formát parametru S by měl být zhruba následující: "HH:MM:SS" nebo "H:M:S", tj. pořadí hodina, minuta, sekunda. Přeházené pořadí není přípustné, ale parametr nemusí obsahovat údaj sekund – v DT.Sec bude hodnota 0.

```
Př.:  StrToTime('07.8.57',DT) = true;
            DT.Hour = 7;
            DT.Min  = 8;
            DT.Sec  = 57;
            DT.Year,DT.Month,DT.Day nezměněny
```

5.7.7. StrToDatePT

```
function StrToDatePT (const S: tString39; var DT: Longint ): boolean;
```

Funkce **StrToDatePT** převede datum zadané v řetězci S do zadaného "var" parametru DT, který je typu zapakovaného formátu datumu a času, a vrátí příznak úspěšnosti převodu. Příklad je podobný jako pro funkci **StrToDate**.

5.7.8. StrToTimePT

```
function StrToTimePT (const S: tString39; var DT: Longint ): boolean;
```

Funkce **StrToTimePT** převede čas zadaný v řetězci S do zadaného "var" parametru DT, který je typu zapakovaného formátu datumu a času, a vrátí příznak úspěšnosti převodu. Příklad je podobný jako pro funkci **StrToTime**.

5.8. Funkce pro práci s datumem a časem

5.8.1. GetPackTime

```
function GetPackTime : Longint;
```

Funkce **GetPackTime** vrátí aktuální hodnotu systémového datumu a času v zapakovaném formátu.

5.8.2. SetPackTime

```
procedure SetPackTime(Time : longint);
```

Procedura **SetPackTime** nastaví hodnotu systémového datumu a času podle parametru Time, který obsahuje datum a čas v zapakovaném tvaru.

5.8.3. fPackTime

```
function fPackTime (DT : DateTime) : Longint;
```

Funkce **fPackTime** vrátí datum a čas v parametru DT v zapakovaném formátu. Tato funkce je podobná proceduře **PackTime** implementované v jednotce Dos, s tím rozdílem, že výsledek vrací jako svoji funkční hodnotu, nikoli jako "var" parametr.

5.8.4. GetDayOfWeek

```
function GetDayOfWeek (DT : DateTime) : Byte;
```

Funkce **GetDayOfWeek** vypočte ze zadaného data v parametru DT příslušný den v týdnu a vrátí jeho index. Dny v týdnu se číslují ve stylu: 0 = neděle, 1 = pondělí ... 6 = sobota.

5.8.5. GetDayOfWeekPT

```
function GetDayOfWeekPT (Time : Longint) : Byte;
```

Funkce **GetDayOfWeekPT** vypočte den v týdnu ze zadaného data stejně jako funkce **GetDayOfWeek**, ale jako parametr se jí zadá datum a čas v zapakovaném formátu.

5.8.6. CheckTime

```
function CheckTime (DT : DateTime) : Byte;
```

Funkce **CheckTime** otestuje správnost datumu a času v parametru DT a vrátí jednu z následujících hodnot:

návratová hodnota	význam
0	datum a čas je v pořádku
1	špatné sekundy
2	špatné minuty
3	špatné hodiny
5	špatný den
6	špatný měsíc
7	špatný rok

Funkce provádí kontroly i na přestupné roky.
Zde je několik příkladů výsledků této funkce pro následující nastavení parametru DT:

DT.Year	DT.Month	DT.Day	DT.Hour	DT.Min	DT.Sec	výsledek funkce
0	0	0	0	0	0	7
1900	1	1	0	0	0	7
1980	1	1	0	0	0	0
2002	10	5	23	60	0	2
2000	2	30	10	24	30	5

5.8.7. CheckTimePT

```
function CheckTimePT (Time : longint) : Byte;
```

Funkce **CheckTimePT** otestuje správnost datumu a času stejně jako funkce **CheckTime**, ale datum a čas je v zapakovaném tvaru v parametru Time.

5.9. Převody množin bytů a znaků na řetězce a zpět

5.9.1. StrToByteSet

```
procedure StrToByteSet (const Str : String; var ByteSet : tByteSet;
                        var ErrCode : Integer);
```

Procedura **StrToByteSet** převede řetězec s výčtem či intervalem množiny bytů do zadaného "var" parametru typu množiny bytů. Pokud operace proběhla úspěšně, vrátí v proměnné ErrCode hodnotu 0, jinak vrátí v ErrCode pozici chyby v dekódovaném řetězci.

```
Př.:  StrToByteSet('1,2..4,10',ByteSet,ErrCode);
            ByteSet = [1,2..4,10];
            ErrCode = 0;
      StrToByteSet('10,2..4,3,8..9',ByteSet,ErrCode);
            ByteSet = [2..4,8..10];
            ErrCode = 0;
```

```
      StrToByteSet('1,2..,10',ByteSet,ErrCode);
            ByteSet = [1,2];
            ErrCode = 6;
```

5.9.2. StrToCharSet

```
procedure StrToCharSet (const Str : String; var CharSet : tCharSet;
                        var ErrCode : Integer);
```

Procedura **StrToCharSet** převede řetězec s výčtem či intervalem množiny znaků do zadaného "var" parametru typu množiny znaků. Pokud operace proběhla úspěšně, vrátí v proměnné ErrCode hodnotu 0, jinak vrátí v ErrCode pozici chyby v dekódovaném řetězci.

```
Př.:  StrToCharSet('''A'',''D''..''Z''',CharSet,ErrCode);
            CharSet = ['A','D'..'Z'];
            ErrCode = 0;
      StrToCharSet('''A'''+#40+'..''0''',CharSet,ErrCode);
            CharSet = ['('..'0','A'];
            ErrCode = 0;
      StrToCharSet('''A''..',CharSet,ErrCode);
            CharSet = ['A'];
            ErrCode = 6;
```

5.9.3. ByteSetToStr

```
function ByteSetToStr (ByteSet: tByteSet; HexOrDec: boolean): String;
```

Funkce **ByteSetToStr** převede zadanou množinu bytů na řetězec. Parametr HexOrDec určuje, zda se mají jednotlivé byty vypisovat v hexadecimálním či dekadickém tvaru.

```
Př.:  ByteSetToStr([2,4..5,$FF],false) = '2,4..5,255';
      ByteSetToStr([4..6,5,$FE],true)  = '$04..$06,$FE';
```

5.9.4. CharSetToStr

```
function CharSetToStr (CharSet: tCharSet; HexOrDec: boolean): String;
```

Funkce **CharSetToStr** převede zadanou množinu znaků na řetězec. Parametr HexOrDec určuje, zda se mají jednotlivé znaky vypisovat v hexadecimálním či dekadickém tvaru. Pokud má některý převáděný znak hodnotu #32 až #255, převede se do vypisovatelné (znakové) podoby.

```
Př.:  CharSetToStr([#3,#$10,#65],false) = '#3,#16,''A''';
      CharSetToStr([#3..#16,#65],true)  = '#$03..#$10,''A''';
```

5.10. Převody různých datových struktur na řetězce a zpět

5.10.1. PlainBuffStrHex

```
function  PlainBuffStrHex  (P : Pointer; Len : Byte) : String;
```

Funkce **PlainBuffStrHex** převede byty z paměti délky Len, na kterou odkazuje ukazatel P, na řetězec, který je tvaru 'xxyy..', kde xx je hexadecimální representace bytu na adrese P, yy je hexadecimální representace bytu na adrese P+1 adt.

Př.:
Obsah paměti :

Adresa	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1A0B:002x	12	4A	F5	1B	89	00	EE	25	AC	FD	8E	5B	6C	B2	00	00
1A0B:003x	AA	B0	DF	BC	54	21	03	F0	FF	08	9A	5B	3C	05	F2	6A

```
PlainBuffStrHex(Ptr($1A0B,$0020), 3) = '124AF5'
PlainBuffStrHex(Ptr($1A0B,$0026), 5) = 'EE25ACFD8E'
```

5.10.2. BufferStrHex

```
function  BufferStrHex  (P : Pointer; Len : Byte; S1,S2 : tString4) :
                        : String;
```

Funkce **BufferStrHex** převede byty z paměti délky Len, na kterou odkazuje ukazatel P, na řetězec, který je tvaru S1+'xx'+S2+S1+'yy..'. Tj. před každý převáděný byte se vloží řetězec S1 a za každý převáděný byte se vloží řetězec S2.

```
Př.:  BufferStrHex(P,3, '', '')   = '124AF5'
      BufferStrHex(P,3, '$', '')  = '$12$4A$F5'
      BufferStrHex(P,3, '$', ' ') = '$12 $4A $F5 '
```

5.10.3. BufferStrHex2

```
function  BufferStrHex2 (P : Pointer; Len : Byte; Pref,Suf : char) :
                        : String;
```

Funkce **BufferStrHex2** převede byty z paměti délky Len, na kterou odkazuje ukazatel P, na řetězec, který je tvaru Pref+'xx'+Suf+Pref+'yy..'+Suff. Tj. před každý převáděný byte se vloží znak Pref a za každý převáděný byte se vloží znak Suff. Pokud některý z parametrů Pref, Suf bude mít hodnotu #0, do výsledného řetězce se nevloží.

```
Př.:  BufferStrHex2(P,3, #0, #0)   = '124AF5'
      BufferStrHex2(P,3, '$', #0)  = '$12$4A$F5'
      BufferStrHex2(P,3, '$', ' ') = '$12 $4A $F5 '
```

5.10.4. BufferStrASCII

```
function  BufferStrASCII (P : Pointer; Len : Byte) : String;
```

Funkce **BufferStrASCII** převede byty z paměti délky Len, na kterou odkazuje ukazatel P, na řetězec s ASCII representací jednotlivých bytů. Z maximální délky řetězců vyplývá, že maximální hodnota parametru Len musí být 255.

5.10.5. StrHexBuffer

```
function  StrHexBuffer (const S : String; P : Pointer) : Byte;
```

Funkce **StrHexBuffer** převede pole bytů zadané hexadecimálně v řetězci S do paměti, na kterou odkazuje ukazatel P. Funkce vrátí jako svou funkční hodnotu počet dekódovaných bytů.

```
Př.:  StrHexBuffer('$40 $20,10a0', P) = 4   P^ = ($40,$20,$10,$a0...)
```

5.10.6. StrASCIIBuffer

```
function  StrASCIIBuffer(const S : String; P : Pointer) : Byte;
```

Funkce **StrASCIIBuffer** převede pole bytů zadané ASCII v řetězci S do paměti, na kterou odkazuje ukazatel P. Funkce vrátí jako svou funkční hodnotu počet dekódovaných bytů.

5.10.7. PtrToStr

```
function  PtrToStr(P : Pointer  ) : tString11;
```

Funkce **PtrToStr** převede ukazatel P na řetězec ve tvaru '$SEGM:$OFFS'.

5.10.8. StrToPtr

```
function  StrToPtr(S : tString11) : Pointer;
```

Funkce **StrToPtr** převede řetězec tvaru '$SEGM:$OFFS' na ukazatel.

5.10.9. PtrToLongInt

```
function  PtrToLongInt(P : Pointer) : Longint;
```

Funkce **PtrToLongInt** převede ukazatel na longint.

5.10.10. LongIntToPtr

```
function  LongIntToPtr(L : Longint) : Pointer;
```

Funkce **LongIntToPtr** převede longint na ukazatel.

5.10.11. ConvertByteBits

```
function ConvertByteBits(B: Byte; const ConvTabB: tConvTabByte):Byte;
```

Funkce **ConvertByteBits** přeskupí podle tabulky ConvTabB bity vstupního bytu B. V tabulce ConvTabB je 8 bytů, jejichž hodnoty určují uspořádání jednotlivých bitů ve výsledku funkce. Jednotlivé byty tabulky ConvTabB mohou nabývat hodnot 0 až 7. Pro přeskupování bitů používá funkce násobení vektoru maticí.

Př.: ConvertByteBits(00110101_{bin},[0,2,1,3,4,6,5,7]) = 01010011_{bin};

5.10.12. ConvertWordBits

```
function ConvertWordBits(W: Word; const ConvTabW: tConvTabWord):Word;
```

Funkce **ConvertWordBits** přeskupí podle tabulky ConvTabW bity vstupního wordu W. V tabulce ConvTabW je 16 bytů, jejichž hodnoty určují uspořádání jednotlivých bitů ve výsledku funkce. Jednotlivé byty tabulky ConvTabW mohou nabývat hodnot 0 až 16. Pro přeskupování bitů používá funkce násobení vektoru maticí.

```
Př.:  ConvertWordBits(0000000000110101bin,[0,2,1,3,4,6,5,7,8,
                      9,10,11,12,13,14,15]) = 0000000001010011 bin;
```