Programovací jazyk ST pro PLC Tecomat

TXV 003 21.01
páté vydání
září 2005
změny vyhrazeny

Historie změn

Datum	Vydání	Popis změn
Srpen 2004	1	První verze
Říjen 2004	2	Doplněny popisy knihoven
Leden 2005	3	Provedeny úpravy pro Mosaic Help
Únor 2005	4	Oprava příkladu 3.6.2.5 – doplněno slovo „DO“
Duben 2005	5	Doplnena tab.3.3 Speciální znaky v řetězcích Oprava rozsahu typu SINT v kap.3.2.1 v Tab.3.5 Tab.3.18 doplněna o vzory volání funkcí nad řetězcem znaků Doplněna kap.3.7.2 Knihovna funkcí nad řetězcem znaků

1 ÚVOD

Jazyk ST a norma IEC 61 131

Norma IEC 61 131 pro programovatelné řídící systémy má pět základních částí a představuje souhrn požadavků na moderní řídící systémy. Jednotlivé části normy jsou věnovány jak technickému tak programovénu vybavení těchto systémů.

Jazyk ST je jedním z programovacích jazyků definovaných normou IEC 61 131-3. Norma IEC 61 131-3 je třetí částí z rodiny norem IEC 1131 a představuje první vážný pokus o standardizaci programovacích jazyků pro průmyslovou automatizaci. Je nezávislá na konkrétní organizaci či firmě a má širokou mezinárodní podporu.

V Evropské unii jsou tyto normy přijaty pod číslem EN IEC 61 131.

V ČR byly přijaty jednotlivé části této normy pod následujícími čísly a názvy:

ČSN EN 61 131-1	Programovatelné řídicí jednotky - Část 1: Všeobecné informace
ČSN EN 61 131-2	Programovatelné řídicí jednotky - Část 2: Požadavky na zařízení a zkoušky
ČSN EN 61 131-3	Programovatelné řídicí jednotky - Část 3: Programovací jazyky
ČSN EN 61 131-4	Programovatelné řídicí jednotky - Část 4: Podpora uživatelů
ČSN EN 61 131-5	Programovatelné řídicí jednotky - Část 5: Komunikace
ČSN EN 61 131-7	Programovatelné řídicí jednotky - Část 7: Programování fuzzy řízení

Na normu 61 131-3 je možné pohlížet z různých hledisek, např. tak, že je to výsledek náročné práce sedmi mezinárodních společností, které do vypracování normy vložily svoji desetiletou zkušenost na poli průmyslové automatizace, nebo tak, že ve svém souhrnu obsahuje asi 200 stran textu, a asi 60 tabulek. Na jejím vytváření pracoval tým patřící do pracovní skupiny SC65B WG7 mezinárodní standardizační organizace IEC (International Electrotechnical Comission). Výsledkem je *specifikace syntaxe* a *sémantiky unifikovaného souboru programovacích jazyků, včetně obecného softwarového modelu a strukturujícího jazyka.* Tato norma byla přijata jako směrnice u většiny významných výrobců PLC.

Názvosloví

Soubor norem pro programovatelné řídicí jednotky byl v ČR sice přijat, nikoliv však přeložen do češtiny. Z toho důvodu používá tato příručka názvosloví tak, jak je přednášeno na ČVUT FSI Praha při výuce automatizace. Zároveň je všude v textu uváděno i anglické názvosloví s cílem jednoznačně přiřadit české pojmy k anglickému originálu.

2 ZÁKLADNÍ POJMY

Tato kapitola stručně vysvětluje význam a použití základních pojmů při programování PLC systému Tecomat v jazyce ST. Tyto pojmy budou vysvětleny na jednoduchých příkladech. Detailní popis vysvětlovaných pojmů pak čtenář najde v dalších kapitolách.

2.1 Základní stavební bloky programu v jazyce ST

Základním pojmem při programování v jazyce ST podle normy IEC 61 131 je termín ***Programová Organizační Jednotka*** nebo zkráceně ***POU*** *(Program Organisation Unit)*. Jak vyplývá z názvu, POU je nejmenší nezávislá část uživatelského programu. POU mohou být dodávány od výrobce řídícího systému nebo je může napsat uživatel. Každá POU může volat další POU a při tomto volání může volitelně předávat volané POU jeden nebo více parametrů.

Existují tři základní typy POU :
- ***funkce*** *(function, FUN)*
- ***funkční blok*** *(function block, FB)*
- ***program*** *(program, PROG)*

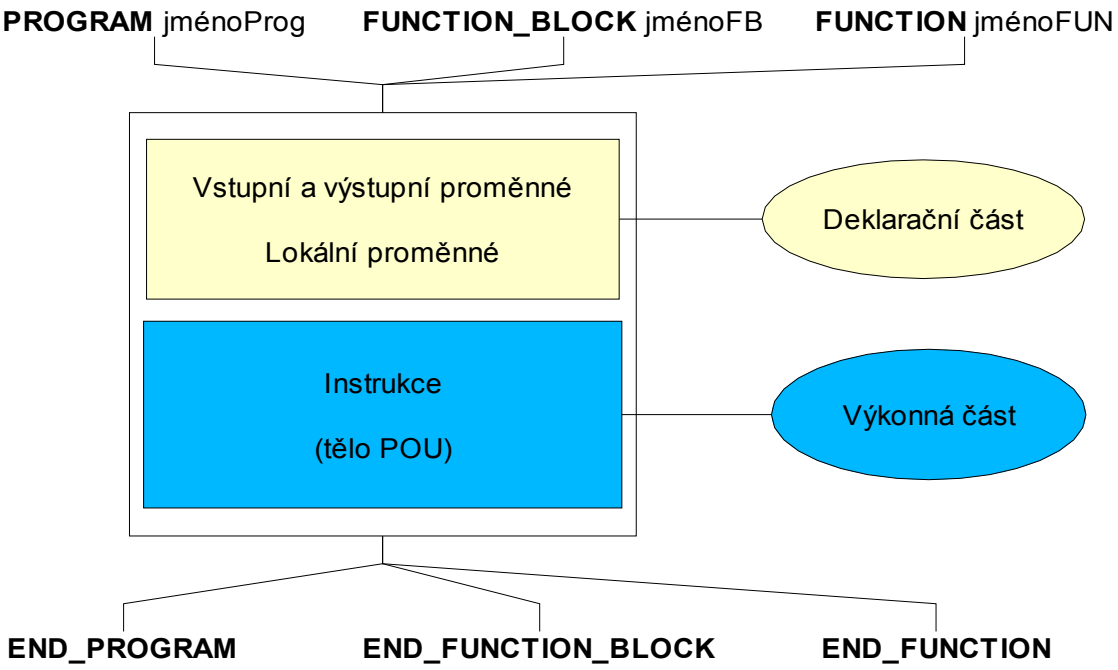
Nejjednodušší POU je ***funkce***, jejíž hlavní charakteristikou je to, že pokud je volána se stejnými vstupními parametry, musí produkovat stejný výsledek (funkční hodnotu). Funkce může vracet pouze jeden výsledek.

Dalším typem POU je ***funkční blok***, který si na rozdíl od funkce, může pamatovat některé hodnoty z předchozího volání (např. stavové informace). Ty pak mohou ovlivňovat výsledek. Hlavním rozdílem mezi funkcí a funkčním blokem je tedy schopnost funkčního bloku vlastnit paměť pro zapamatování hodnot některých proměnných. Tuto schopnost funkce nemají a jejich výsledek je tedy jednoznačně určen vstupními parametry při volání funkce. Funkční blok může také (na rozdíl od funkce) vracet více než jeden výsledek.

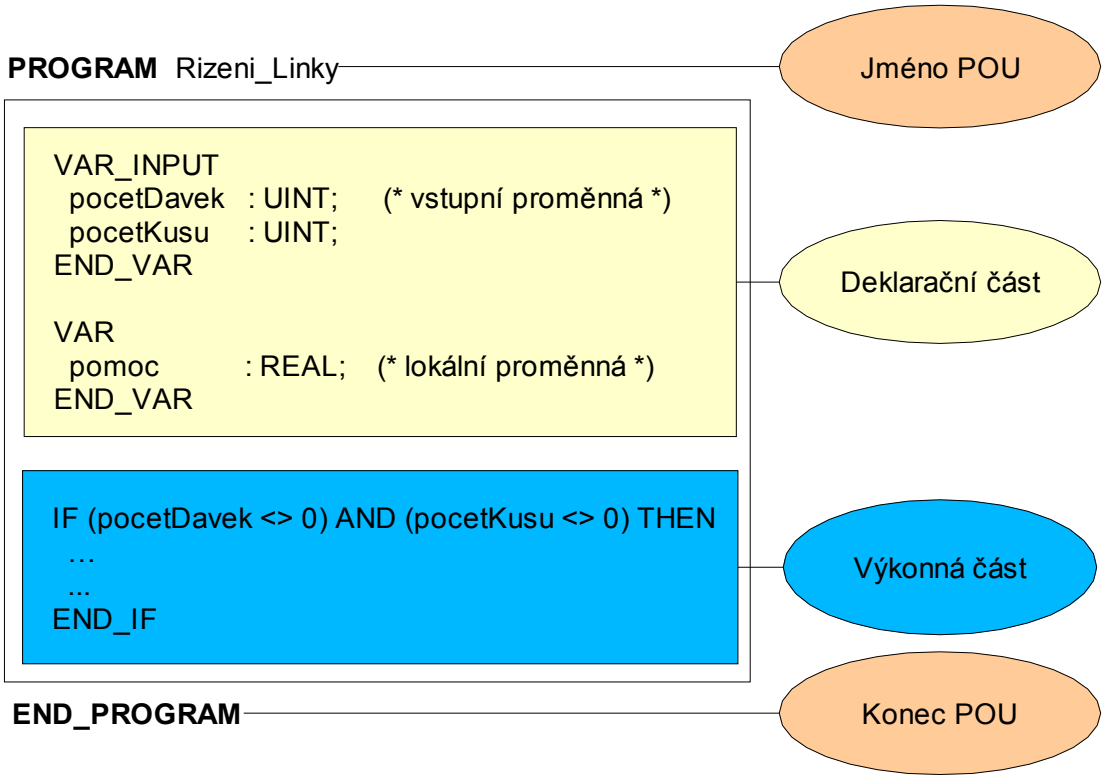
Posledním typem POU je ***program***, který představuje vrcholovou programovou jednotku v uživatelském programu. Centrální jednotka PLC může zpracovávat více programů a programovací jazyk ST obsahuje prostředky pro definice spouštění programů (v jaké periodě vykonávat program, s jakou prioritou, apod.).

Každá POU se skládá ze dvou základních částí : ***deklarační*** a ***výkonné***, jak je vidět na Obr.2.1. V deklarační části POU se definují proměnné potřebné pro činnost POU. Výkonná část pak obsahuje vlastní příkazy pro realizaci požadovaného algoritmu.

Definice POU na Obr.2.2 začíná klíčovým slovem *PROGRAM* a je ukončena klíčovým slovem *END_PROGRAM*. Tato klíčová slova vymezují rozsah POU. Za klíčovým slovem *PROGRAM* je uvedeno jméno POU. Poté následuje deklarační část POU. Ta obsahuje definice proměnných uvedené mezi klíčovými slovy *VAR_INPUT* a *END_VAR* resp. *VAR* a *END_VAR*. Na závěr je uvedena výkonná část POU obsahující příkazy jazyka ST pro zpracování proměnných. Texty uvedené mezi znaky *(** a ***)* jsou poznámky (komentáře).



Obr. 2.1 Základní struktura POU



Obr.2.2 Základní struktura POU PROGRAM

2.2 Deklarační část POU

Deklarační část POU obsahuje definice proměnných potřebných pro činnost POU. Proměnné jsou používány pro ukládání a zpracování informací. Každá ***proměnná*** je definována ***jménem proměnné*** a ***datovým typem***. Datový typ určuje velikost proměnné v paměti a zároveň do značné míry určuje způsob zpracování proměnné. Pro definice proměnných jsou k dispozici standardní datové typy (Bool, Byte, Integer, …). Použití těchto typů závisí na tom, jaká informace bude v proměnné uložena (např. typ Bool pro informace typu ANO-NE, typ Integer pro uložení celých čísel se znaménkem apod.). Uživatel má samozřejmě možnost definovat svoje vlastní datové typy. Umístění proměnných v paměti PLC systému zajišťuje automaticky programovací prostředí. Pokud je to potřeba, může umístění proměnné v paměti definovat i uživatel.

Proměnné můžeme rozdělit podle použití na ***globální*** a ***lokální***. Globální proměnné jsou definovány vně POU a mohou být použity v libovolné POU (jsou viditelné z libovolné POU). Lokální proměnné jsou definovány uvnitř POU a v rámci této POU mohou být používány (z ostatních POU nejsou viditelné).

A konečně proměnné jsou také používány pro předávání parametrů při volání POU. V těchto případech mluvíme o ***vstupních*** resp. ***výstupních*** proměnných.

Příklad 2.1 Deklarace proměnných POU

```
VAR_INPUT                          (* vstupní proměnné *)
  logPodminka         : BOOL;      (* binární hodnota *)
END_VAR
VAR_OUTPUT                         (* výstupní proměnné *)
  vysledek             : INT;      (* celočíselná hodnota se znaménkem *)
END_VAR
VAR                                (* lokální proměnné *)
  kontrolniSoucet      : UINT;     (* celočíselná hodnota *)
  mezivysledek         : REAL;     (* reálná hodnota *)
END_VAR
```

V příkladu 2.1 je uvedena definice vstupní proměnné POU, proměnná se jmenuje *logPodminka* a je typu BOOL, což znamená, že může obsahovat hodnoty TRUE (logická „1“) nebo FALSE (logická „0“). Tato proměnná slouží jako vstupní parametr předávaný při volání POU. Další definovaná proměnná je výstupní, jmenuje se *vysledek* a je typu INT (integer), takže může obsahovat celočíselné hodnoty v rozsahu od –32 768 do +32 767. V této proměnné je předávána hodnota do nadřízené POU. Proměnné definované mezi klíčovými slovy *VAR* a *END_VAR* jsou lokální a lze je tedy používat pouze v rámci POU. Proměnná *kontrolniSoucet* je typu UINT (unsigned integer) a může uchovávat celá čísla v rozsahu od 0 do 65535. Proměnná *mezivýsledek* je typu REAL a je určena pro práci s reálnými čísly.

Tab.2.1 Typy proměnných používaných v jazyce ST

Typ proměnné	Význam	Určení
VAR_INPUT	vstupní	Pro předávání vstupních parametrů do POU Tyto proměnné jsou viditelné z ostatních POU a jsou z nich také nastavovány V rámci POU, ve které jsou proměnné deklarovány, lze provádět pouze jejich čtení
VAR_OUTPUT	výstupní	Pro předávání výstupních parametrů z POU Tyto proměnné jsou viditelné z ostatních POU, kde je možné provádět pouze jejich čtení Změnu hodnoty těchto proměnných lze provádět pouze v rámci POU, ve které byly proměnné deklarovány
VAR_IN_OUT	vstup / výstupní	Pro nepřímý přístup k proměnným ležícím vně POU Proměnné lze číst i měnit jejich hodnotu uvnitř i vně POU
VAR_EXTERNAL	globální	Proměnné definované v mnemokódu PLC
VAR_GLOBAL	globální	Proměnné, které jsou dostupné ze všech POU
VAR	lokální	Pomocné proměnné používané v rámci POU Z ostatních POU nejsou viditelné, to znamená, že je lze číst resp. měnit jejich hodnotu pouze v rámci POU, ve které jsou deklarovány Tyto proměnné mohou uchovávat hodnotu i mezi jednotlivými voláními příslušné POU
VAR_TEMP	lokální	Pomocné proměnné používané v rámci POU Z ostatních POU nejsou viditelné Tyto proměnné vznikají při vstupu do POU a zanikají po ukončení POU – nemohou tedy uchovávat hodnotu mezi dvěma voláními POU

Tab.2.2 Použití proměnných v jednotlivých POU

Typ proměnné	PROGRAM	FUNCTION_BLOCK	FUNCTION	Vně POU
VAR_INPUT	ano	ano	ano	ne
VAR_OUTPUT	ano	ano	ne	ne
VAR_IN_OUT	ano	ano	ano	ne
VAR_EXTERNAL	ano	ano	ne	ne
VAR_GLOBAL	ne	ne	ne	ano
VAR	ano	ano	ano	ne
VAR_TEMP	ano	ano	ano	ne

2.3 Výkonná část POU

Výkonná část POU následuje za částí deklarační a obsahuje příkazy a instrukce, které jsou zpracovány centrální jednotkou PLC. Ve výjimečných případech nemusí definice POU obsahovat žádnou deklarační část a potom je výkonná část uvedena bezprostředně za definicí začátku POU. Příkladem může být POU, která pracuje pouze s globálními proměnnými, což sice není ideální řešení, ale může existovat.

Výkonná část POU může obsahovat volání dalších POU. Při volání mohou být předávány parametry pro volané funkce resp. funkční bloky.

2.4 Ukázka programu v jazyce ST

Příklad 2.2 Ukázka programu v jazyce ST

```
FUNCTION_BLOCK fbStartStop
  VAR_INPUT
    start       : BOOL R_EDGE;
    stop        : BOOL R_EDGE;
  END_VAR
  VAR_OUTPUT
    vystup      : BOOL;
  END_VAR

  vystup := (vystup OR start) AND NOT stop;
END_FUNCTION_BLOCK


FUNCTION_BLOCK fbMotor
  VAR_INPUT
    startMotoru : BOOL;
    stopMotoru  : BOOL;
  END_VAR
  VAR
    startStop   : fbStartStop;
    motorBezi   : BOOL;
    casRozbehu  : TON;
  END_VAR
  VAR_OUTPUT
    hvezda      : BOOL;
    trojuhelnik : BOOL;
  END_VAR

  startStop( start := startMotoru, stop := stopMotoru,
             vystup => motorBezi);
  casRozbehu( in := motorBezi, pv := TIME#12s, q => trojuhelnik);
  hvezda := motorBezi AND NOT trojuhelnik;
END_FUNCTION_BLOCK

VAR_GLOBAL
  // vstupy
  sb1 AT %X0.0,
```

```
  sb2 AT %X0.1,
  sb3 At %X0.2,
  sb4 AT %X0.3   : BOOL;

  // vystupy
  km1 AT %Y0.0,
  km2 AT %Y0.1,
  km3 AT %Y0.2,
  km4 AT %Y0.3   : BOOL;
END_VAR

PROGRAM test
  VAR
    motor1 : fbMotor;
    motor2 : fbMotor;
  END_VAR

  motor1( startMotoru := sb1, stopMotoru := sb2,
          hvezda => km1, trojuhelnik => km2);
  motor2( startMotoru := sb3, stopMotoru := sb4,
          hvezda => km3, trojuhelnik => km4);

END_PROGRAM

CONFIGURATION Plc1
  RESOURCE CPM
    TASK FreeWheeling(Number := 0);
    PROGRAM prg WITH FreeWheeling : test ();
  END_RESOURCE
END_CONFIGURATION
```

3 PROGRAMOVACÍ JAZYK ST PODROBNĚ

Tato kapitola popisuje syntaxi a sémantiku základních prvků programovacího jazyka ST pro PLC systémy Tecomat.

Syntaxe popisuje prvky, které jsou pro programování v jazyce ST k dispozici a způsoby, jakými mohou být vzájemně kombinovány.

Sémantika pak vyjadřuje jejich význam.

3.1 Základní prvky

Každý program v jazyce ST se skládá ze základních ***jednoduchých prvků***, určitých nejmenších jednotek, ze kterých se vytvářejí deklarace a příkazy. Tyto jednoduché prvky můžeme rozdělit na :

- *oddělovače* (Delimiters),
- *identifikátory* (Identifiers)
- *literály* (Literals)
- *klíčová slo*va (Keywords)
- *komentáře* (Comments)

Pro větší přehlednost textu jsou klíčová slova psána tučně, aby se dala lépe vyjádřit struktura deklarací a příkazů. Uživatelské identifikátory jsou psány normálním písmem a literály kurzívou.

Oddělovače jsou speciální znaky (např. (,), =, :, mezera, apod.) s různým významem.

Identifikátory jsou alfanumerické řetězce znaků, které slouží pro vyjádření jmen uživatelských funkcí, návěští nebo programových organizačních jednotek (např. Tepl_N1, Spinac_On, Krok4, Pohyb_dopr apod.).

Literály slouží pro přímou reprezentaci hodnot proměnných (např. *0,1*; *84*; *3,79*; *TRUE* ; *zelena* apod.).

Klíčová slova jsou standardní identifikátory (např. FUNCTION, REAL, VAR_OUTPUT, apod.). Jejich přesný tvar a význam odpovídá normě IEC 61 131-3. Klíčová slova se nesmějí používat pro vytváření jakýchkoli uživatelských jmen. Pro zápis klíčových slov mohou být použita jak velká tak malá písmena resp. jejich libovolná kombinace.

K rezervovaným klíčovým slovům patří :

- *jména elementárních datových typů*
- *jména standardních funkcí*
- *jména standardních funkčních bloků*
- *jména vstupních parametrů standardních funkcí*
- *jména vstupních a výstupních parametrů standardních funkčních bloků*
- *prvky jazyka ST*

Všechna rezervovaná klíčová slova jsou uvedena v příloze H normy IEC 61 131-3.

Komentáře nemají syntaktický ani sémantický význam, jsou však důležitou částí dokumentace programu. Komentář je možné v programu zapsat všude tam, kde je možné zapsat znak mezera. Při překladu jsou tyto řetězce ignorovány, takže mohou obsahovat i znaky národních abeced. Překladač jazyka ST rozeznává dva druhy komentářů :

- *obecné komentáře*
- *řádkové komentáře*

Obecné komentáře jsou řetězce znaků začínající dvojicí znaků **(*** a ukončené dvojicí znaků ***)**. To umožňuje zapisovat všechny potřebné typy komentářů, jak ukazuje dále uvedený příklad.

Řádkové komentáře jsou řetězce znaků začínající dvojicí znaků // a jsou ukončeny koncem řádku. Výhodou řádkových komentářů je možnost jejich vnořování do obecných komentářů (viz řádky s definicí proměnných Pomoc1 a Pomoc2 v následujícím příkladu, které budou považovány za komentář a nebudou překladačem překládány).

Příklad 3.1 Komentáře v jazyce ST

```
(************************************************
   toto je ukázka
   víceřádkového komentáře
************************************************)
VAR_GLOBAL
  Start,              (* obecný komentář, např. : tlačítko START *)
  Stop      : BOOL;   (* tlačítko STOP *)
  Pomoc     : INT;    // řádkový komentář
(*
  Pomoc1    : INT;    // vnořený řádkový komentář
  Pomoc2    : INT;
*)
END_VAR
```

3.1.1 Identifikátory

Identifikátor je řetězec písmen (malých nebo velkých), číslic a podtrhovacích znaků, který se používá pro pojmenování následujících prvků jazyka ST :

- *jména konstant*
- *jména proměnných*
- *jména odvozených datových typů*
- *jména funkcí, funkčních bloků a programů*
- *jména úloh*

Identifikátor musí začínat písmenem nebo podtrhovacím znakem a nesmí obsahovat mezery. Znaky národních abeced (písmena s háčky a čárkami) nejsou v identifikátorech povoleny. Umístění podtrhovacího znaku je významné, tedy např. „BF_LM“ a „BFL_M“ jsou dva různé identifikátory. Více podtrhovacích znaků za sebou není povoleno. Velikost písmen nehraje v identifikátoru roli. Například zápis motor_off je rovnocený zápisu MOTOR_OFF resp. Motor_Off. Pokud motor_off bude jméno proměnné, pak všechny uvedené zápisy budou označovat stejnou proměnnou.

Maximální délka identifikátoru je 64 znaků.

Tab.3.1 Příklady platných a neplatných identifikátorů

Platné identifikátory	Neplatné identifikátory
XH2	2XH
MOTOR3F, Motor3F	3FMOTOR
Motor3F_Off, Motor3F_OFF	MOTOR3F__Off
SQ12	SQ$12
Prodleva_12_5	Prodleva_12.5
Rek	Řek
_3KL22	__3KL22
KM10a	KM 10a

3.1.2 Literály

Vnější reprezentace dat v jazyce ST sestává z :

- *numerických literálů*
- *řetězců znaků*
- *časových literálů*

3.1.2.1 Numerické literály

Numerický literál je definován jako číslo (konstanta) v desítkové soustavě nebo v soustavě o jiném základu než deset (např. dvojková, osmičková a šestnáctková čísla). Numerické literály dělíme na integer literály a real literály. Jednoduché podtržítko umístěné mezi číslicemi numerického literálu nemá na jeho hodnotu vliv, je povoleno pro zlepšení čitelnosti. Příklady různých numerických literálů jsou uvedeny v Tab.3.2 .

Tab.3.2 Příklady numerických literálů

Popis	Numerický literál - příklad	Pozn.
Integer literál	14, –9, 125_48756	(12 548 756)
Real literál	–18.0, –8.0, 0.123_4	(0.1234)
Real literál s exponentem	4.47E6, 652E–2	
Literál o základu 2	2#10110111	(183 desítkově)
Literál o základu 8	8#127	(87 desítkově)
Literál o základu 16	16#FF	(255 desítkově)
Boolovské FALSE a TRUE	FALSE, TRUE	

3.1.2.2 Literály řetězce znaků

Řetězec znaků je posloupnost žádného znaku (prázdný řetězec) nebo více znaků, která je uvozena a ukončena jednoduchou uvozovkou (‘). Příklady: ‘‘ (prázdný řetězec), ‘teplota‘ (neprázdný řetězec o délce sedm, obsahující slovo teplota).

Znak dolar, $, se používá jako prefix, který umožňuje uvedení speciálních znaků v řetězci. Speciální znaky, které se netisknou, se používají např. pro formátování textu pro tiskárnu nebo na displej. Pokud je znak dolaru před dvěma šestnáctkovými ciframi, je řetězec interpretován jako šestnáctková reprezentace osmibitového kódu znaku. Např. řetězec ‘$0D$0A‘ je chápán jako reprezentace dvou kódů, a to 00001101 a 00001010. První kód představuje v ASCII tabulce znak Enter, (CR, desítkově 13) a druhý kód odřádkování (LF, desítkově 10).

Literály řetězce znaků, tzv. stringy, se používají např. pro výměnu textů mezi různými PLC nebo mezi PLC a dalšími komponentami automatizačního systému, nebo při programování textů, které se zobrazují na řídicích jednotkách nebo operátorských panelech.

Tab.3.3 Speciální znaky v řetězcích

Zápis	Význam
$$	Znak dolar
$'	Znak jednoduchý apostrof
$L nebo $l	Znak Line feed (16#0A)
$N nebo $n	Znak New line
$P nebo $p	Znak New page
$R nebo $r	Znak Carriage return (16#0D)
$T nebo $t	Znak tabelátor (16#09)

3.1.2.3 Časové literály

Při řízení v podstatě potřebujeme dva různé typy údajů, které nějakým způsobem souvisí s časem. Za prvé je to ***údaj o trvání***, tj. o době, která uplynula nebo má uplynout v souvislosti s nějakou událostí. Za druhé je to údaj o „absolutním čase“, složeném z *data* podle kalendáře (Date) a z *časového údaje v rámci jednoho dne, tzv.* ***denního času*** (Time of Day). Tento časový údaj se může využívat pro synchronizaci začátku nebo konce řízené události vzhledem k absolutnímu časovému rámci. Příklady časových literálů jsou v Tab.3.5.

Doba trvání. Časový literál pro dobu trvání je uvozen některým z klíčových slov T#, t#, TIME#, time#. Vlastní časový údaj je vyjádřen v časových jednotkách: hodinách, minutách, sekundách a milisekundách. Zkratky pro jednotlivé části časového údaje jsou uvedeny v Tab.3.4. Mohou být vyjádřeny malým i velkým písmenem.

Tab.3.4 Zkratky pro časové údaje

Zkratka	Význam
ms, MS	Milisekundy (Miliseconds)
s, S	Sekundy (Seconds)
m, M	Minuty (Minutes)
h, H	Hodiny (Hours)
d, D	Dny (Days)

Denní čas a datum. Reprezentace údaje o datu a čase v rámci dne je stejná jako v ISO 8601. Prefix může být buď krátký nebo dlouhý. Klíčová slova pro datum jsou D# nebo DATE#. Pro časový údaj v rámci jednoho dne se používají klíčová slova TOD# nebo TIME_OF_DAY#. Pro souhrnný údaj o „absolutním čase" pak klíčová slova DT# nebo DATE_AND_TIME#.

Tab.3.5 Příklady různých časových literálů

Popis	Příklady
Doba trvání	T#24ms, t#6m1s, t#8.3s t#7h_24m_5s, TIME#416ms
Datum	D#2003-06-21 DATE#2003-06-21
Denní čas	TOD#06:32:15.08 TIME_OF_DAY#11:38:52.35
Datum a denní čas	DT#2003-06-21-11:38:52.35 DATE_AND_TIME#2003-06-21-11:38:52.35

3.2 Datové typy

Pro jazyk ST jsou v souhlasu s normou IEC 61 131-3 definovány tzv. ***elementární***, předdefinované datové typy, (Elementary data types), dále jsou definovány ***rodové*** datové typy (Generic data type) pro příbuzné skupiny datových typů. A konečně je k dispozici mechanizmus, kterým může uživatel vytvářet vlastní ***uživatelské*** *(odvozené)* datové typy, (Derived data type, Type definition).

3.2.1 Elementární datové typy

Elementární datové typy jsou charakterizované šířkou dat (počtem bitů) a případně i rozsahem hodnot. Přehled podporovaných datových typů je uveden v Tab.3.6.

Tab.3.6 Elementární datové typy

Klíčové slovo	Anglicky	Datový typ	Bitů	Rozsah hodnot
BOOL	Boolean	Boolovské číslo	1	0,1
SINT	Short integer	Krátké celé číslo	8	–128 až 127
INT	Integer	Celé číslo	16	–32 768 až +32 767
DINT	Double integer	Celé číslo, dvojnásobná délka	32	–2 147 483 648 až +2 147 483 647
USINT	Unsigned short integer	Krátké celé číslo bez znaménka	8	0 až 255
UINT	Unsigned integer	Celé číslo bez znaménka	16	0 až 65 535
UDINT	Unsigned double integer	Celé číslo bez znaménka, dvojnásobná délka	32	0 až (2**32-1)
REAL	Real (single precision)	Číslo v pohyblivé řádové čárce (jednoduchá přesnost)	32	±2.9E-39 až ±3.4E+38 Podle IEC 559
LREAL	Long real (double precision)	Číslo v pohyblivé řádové čárce (dvojnásobná přesnost)	64	Podle IEC 559
TIME	Duration	Trvání času	24d 20:31:23.647	
DATE	Date (only)	Datum	Od 1.1.1970 00:00:00	
TIME_OF_DAY nebo TOD	Time of day (only)	Denní čas	24d 20:31:23.647	
DATE_AND_TIME nebo DT	Date and time of day	„Absolutní čas“	Od 1.1.1970 00:00:00	
STRING	String	Řetězec	Max.255 znaků	
BYTE	Byte(bit string of 8 bits)	Sekvence 8 bitů	8	Není deklarován rozsah
WORD	Word (bit string of 16bits)	Sekvence 16 bitů	16	Není deklarován rozsah
DWORD	Double word (bit string of 32 bits)	Sekvence 32 bitů	32	Není deklarován rozsah

3.2.2 Rodové datové typy

Rodové datové typy vyjadřují vždy celou skupinu (rod) datových typů. Jsou uvozeny prefixem ANY. Např. zápisem ANY_BIT se rozumí všechny datové typy uvedené v následujícím výčtu: DWORD, WORD, BYTE, BOOL. Přehled rodových datových typů je uveden v Tab.3.7.

Tab.3.7 Přehled rodových datových typů

<table>
<tr><th colspan="5">ANY</th></tr>
<tr><th>ANY_BIT</th><th colspan="3">ANY_NUM</th><th>ANY_DATE</th><th rowspan="3">TIME
STRING</th></tr>
<tr><td rowspan="2">BOOL
BYTE
WORD
DWORD</td><th colspan="2">ANY_INT</th><th>ANY_REAL</th><td rowspan="2">DATE
DATE_AND_TIME
TIME_OF_DAY</td></tr>
<tr><td>INT
SINT
DINT</td><td>UINT
USINT
UDINT</td><td>REAL
LREAL</td></tr>
</table>

3.2.3 Odvozené datové typy

Deklarace datového typu

Odvozené typy, tzn. typy specifikované buď výrobcem nebo uživatelem, mohou být deklarovány pomocí textové konstrukce **TYPE…END_TYPE**. Jména nových typů, jejich datové typy, případně i s jejich inicializačními hodnotami, jsou uvedena v rámci této konstrukce. Tyto odvozené datové typy se pak mohou dále používat spolu s elementárními datovými typy v deklaracích proměnných. Odvozený datový typ dědí vlastnosti typu, ze kterého byl odvozen.

Deklarace nového datového typu může být provedena např. ***výčtem hodnot*** (Enumerated data type), nebo jako ***dílčí rozsah*** v rozmezí uvedené dolní a horní meze. Novým datovým typem může být také ***pole*** (Array) nebo ***struktura*** (Structure).

Příklad 3.2 Příklad deklarace čtyř různých odvozených datových typů

```
TYPE
  Kvalita_modul  : (Dobry, Zmetek, Oprava);  // výčet hodnot
  Senzor_t       : INT;                      // vlastní typ integer
  Teploty        : ARRAY [1..8] OF Senzor_t; // pole teplot naměřených
                                             // pomocí Senzor_t
  Nadoba_1:
    STRUCT                                 // struktura složená z čísla měření,
      Cislo_mer : INT;                     // pole 8 hodnot teploty a tlaku
      Kvalita   : Kvalita_modul;
      Mereni    : Teploty;
      Tlak      : REAL;
    END_STRUCT;
END_TYPE
```

Inicializace datového typu

Elementární datové typy mají předdefinované (default) počáteční hodnoty. Převážně jsou to nuly, u data je to D#1970-01-01. Pro uživatelské datové typy je to vždy hodnota prvního prvku uvedeného ve výčtu, nebo hodnota prvku z dolní meze rozsahu. V polích a strukturách se počáteční hodnoty přiřazují buď podle předdefinovaných počátečních hodnot typů, nebo podle právě uvedených pravidel v případě výčtu nebo dílčího rozsahu. Počáteční hodnota se dá také přiřadit v deklaraci přímo pomocí přiřazovacího operátoru „:=“.

V Tab.3.8 jsou uvedeny předdefinované počáteční hodnoty pro různé datové typy.

Tab.3.8 Předdefinované počáteční hodnoty pro různé datové typy

Datový typ	Počáteční hodnota (Initial Value)
BOOL, SINT, INT, DINT, LINT	0
USINT, UINT, UDINT, ULINT	0
BYTE, WORD, DWORD, LWORD	0
REAL, LREAL	0.0
TIME	T#0s
DATE	D#1970-01-01
TIME_OF_DAY	TOD#00:00:00
DATE_AND_TIME	DT#1970-01-01-00:00:00
STRING	’ ’ (prázdný string)

Použití datových typů

Jednoduché proměnné, které mají deklarován uživatelský typ, mohou být použity všude, kde může být použita proměnná s „rodičovským“ typem. Tedy např. proměnná Senzor_t z příkladu 3.2 může být použita všude, kde mohou být používány proměnné typu INT. Toto pravidlo může být aplikováno rekurzivně.

Prvek složené proměnné, který má deklarován uživatelský typ, může být použit všude, kde může být použita proměnná s „rodičovským“ typem.

3.3 Proměnné

Podle IEC 61 131-3 jsou ***proměnné*** v podstatě prostředkem pro identifikaci datových objektů, jejichž obsah se může měnit, tzn. dat přiřazených ke vstupům, výstupům nebo paměti PLC. Proměnná může být deklarována některým z elementárních datových typů nebo některým z uživatelských (odvozených) datových typů.

Zde je vhodné si uvědomit, že takto chápaný pojem proměnné nebyl dříve v klasickém světě PLC běžný. Při standardním programování PLC podle normy DIN 19239 se k adresám v paměti PLC a ke vstupům a výstupům přistupovalo *přímo* pomocí *adresových „operandů"*, jako je např. F2.4 (flag nebo paměťový bit číslo 4 ve skupině číslo2), I1.0 (vstup číslo 0 ve skupině číslo 1), IW2 (2.skupina vstupů o šířce 16 bitů) apod. K paměti či vstupům a výstupům se přistupovalo po bitech, bytech, slovech nebo dvojitých slovech. Pokud byla přímá adresa uvedena špatně, mohlo dojít ke zcela neočekávaným a závažným chybám v programu a tím i k velmi nebezpečné situaci vzhledem k řízené technologii.

U některých PLC bylo dalším krokem směrem k větší obecnosti používání tzv. „symbolů", které mohly být používány na místě přímých adresových operandů. Soupis všech používaných přímých adresových operandů a jim přiřazených symbolů byl uveden v tzv. *tabulce symbolů*, (Symbol table, nebo též Assignment list). Tím se program zpřehlednil, ale podstata problému tím vyřešena nebyla, protože uživatel musel např. stále sledovat, kde má jaké údaje uloženy a zajistit, aby nemohlo dojít k jejich překrývání v paměti.

Norma IEC 61 131-3 umožnila udělat zcela novým pojetím proměnné u systémů PLC výrazný krok kupředu. Místo hardwarových adres nebo symbolů jsou zde definovány proměnné tak, jak se používají ve vyšších programovacích jazycích. Proměnné jsou identifikátory (jména) přiřazené programátorem, které slouží v podstatě pro rezervaci místa v paměti a obsahují hodnoty dat programu.

3.3.1 Reprezentace proměnných

Jednoduché proměnné

Jednoduchá proměnná je definována jako proměnná, která reprezentuje jednoduchý datový prvek jednoho z elementárních datových typů nebo uživatelského datového typu (výčet hodnot, dílčí rozsah nebo typ odvozený rekurzivně tak, že se lze zpětně postupně dopracovat opět až k výčtu hodnot nebo dílčím rozsahům nebo elementárním datovým typům). Reprezentace proměnných může být ***symbolická*** nebo i ***přímá*** (přiřazení prvku dat k fyzickým nebo logickým pozicím na vstupech, výstupech a v paměti PLC).

Pro **symbolickou** reprezentaci proměnných se používají identifikátory.

Pro uvození **přímé** reprezentace jednoduchých proměnných se používá speciální znak procento, „**%**", ***prefix umístění*** (Location prefix) a ***prefix šíře dat*** (Size prefix). Za těmito znaky následuje jeden nebo více znaků typu UINT oddělených tečkami.

Příklady přímo reprezentovaných proměnných:

%I2.1 nebo %IX2.1	Vstupní bit číslo 1 ve skupině číslo 2
%IW4	Vstupní slovo číslo 4
%Q3.7 nebo %QX3.7	Výstupní bit číslo 7 ve skupině číslo 3
%QB2	Výstupní byte číslo 2

%M104.5 Paměťový bit na pátém místě ve 104. skupině

Vazbu mezi přímo reprezentovanými proměnnými a jejich fyzickým nebo logickým umístěním v konkrétním PLC musí specifikovat výrobce.

V Tab.3.9 a Tab.3.10 jsou uvedeny prefixy umístění a prefixy šíře dat pro přímo reprezentované proměnné.

Tab.3.9 Prefixy umístění a prefixy šíře dat pro přímo reprezentované proměnné - IEC

Prefix umístění	Význam	Prefix šíře dat	Význam
%I	Vstup	X	Šíře 1 bit
%Q	Výstup	Žádný	Šíře 1 bit
%M	Paměťové místo	B	Šíře 1 Byte (8 bitů)
		W	Šíře 1 Word (16 bitů)
		D	Šíře 1 Double Word (32 bitů)

Tab.3.10 Prefixy umístění a prefixy šíře dat – tradiční Tecomat

Prefix umístění	Význam	Prefix šíře dat	Význam
%X	Vstup	.n	Šíře 1 bit
%Y	Výstup	Žádný	Šíře 1 Byte (8 bitů)
%S	Systémový registr	W	Šíře 1 Word (16 bitů)
%R	Paměťové místo	L	Šíře 1 Long (32 bitů)

Složené proměnné

Složené proměnné jsou typu ***pole*** nebo typu ***struktury***.

Pole je soubor datových prvků stejného datového typu, na které je možné se odkázat pomocí jednoho nebo více indexů uzavřených v závorkách a oddělených čárkami. Index musí být některým z typů zahrnutých v rodovém typu ANY_INT. Maximální počet indexů (rozměr pole) je 4 a maximální rozsah indexů musí odpovídat typu INT.

Strukturovaná proměnná je proměnná, která je deklarována s typem, který byl předtím specifikován jako struktura dat, tj. datovým typem složeným ze souboru pojmenovaných prvků. Prvek strukturované proměnné je reprezentován dvěma nebo více identifikátory nebo místy v poli, které jsou odděleny tečkou. První identifikátor představuje jméno celé struktury a následující identifikátory představují sekvenci jmen prvků, které vedou ke konkrétnímu prvku struktury.

Maximální počet úrovní ve struktuře není omezen. Praktické omezení představuje pouze celková délka identifikátoru proměnné.

Příklad 3.3 Ukázka složených proměnných

```
VAR
  Hala      :    ARRAY [1..6] OF Nadoba_1;
  Kvalita   :    Kvalita_modul;
END_VAR
```

Složená proměnná se jménem Hala je typu pole se šesti prvky, každý prvek je strukturou typu Nadoba_1, uvedené v příkladu 3.3, např. Hala[3] je třetí prvek tohoto pole.

Složená proměnná pojmenovaná Kvalita je typu struktura s názvem Kvalita_modul z příkladu 3.3. Kvalita.zmetek je pak odkaz na druhý prvek v proměnné Kvalita.

3.3.2 Inicializace proměnných

Po restartu programu může každá z proměnných nabýt jedné z následujících počátečních hodnot:

- Hodnoty, kterou měla proměnná v okamžiku zastavení konfiguračního prvku (retained value)
- Uživatelem specifikované počáteční hodnoty
- Předdefinované (default) počáteční hodnoty

Uživatel může deklarovat, že má být proměnná ***retentivní*** (tzn. že se má uchovat její poslední hodnota) pomocí kvalifikátoru **RETAIN**. Tento kvalifikátor je možné použít pouze pro globální proměnné.

Počáteční hodnota proměnné po restartu systému se určí podle těchto pravidel:

- Pokud je startovací operací tzv. teplý restart, pak počáteční hodnotou retentivních proměnných budou jejich retentivní (poslední zachované) hodnoty
- Pokud je startovací operací tzv. studený restart, pak počáteční hodnotou retentivních proměnných budou uživatelem specifikované počáteční hodnoty
- Neretentivní proměnné budou inicializovány na hodnoty specifikované uživatelem nebo na předdefinované hodnoty pro příslušné datové typy u všech proměnných, kde není počáteční hodnota uživatelem specifikována
- Proměnné, které reprezentují vstupy systému PLC, budou inicializovány podle stavu signálů, připojených na tyto vstupy

3.3.3 Deklarace proměnných

Každá deklarace typu programové organizační jednotky (POU) pro programovatelný automat (tzn. každá deklarace *programu*, *funkce* nebo *funkčního bloku*) má mít na začátku alespoň jednu *deklarační část*, která specifikuje datové typy (a pokud je to nutné i fyzické nebo logické umístění) proměnných používaných v programové organizační jednotce. Tato deklarační část má textovou podobu, používá jedno z klíčových slov VAR, VAR_INPUT, VAR_OUTPUT. Za klíčovým slovem VAR může být volitelně uveden kvalifikátor CONSTANT. Za uvedenými klíčovými slovy následuje jedna nebo více deklarací proměnných oddělených středníkem a ukončených klí-

čovým slovem END_VAR. Součástí deklarace proměnných může být deklarace jejich počátečních hodnot.

Rozsah platnosti deklarací umístěných v deklarační části je ***lokální*** pro tu programovou organizační jednotku, ve které je deklarace uvedena. To znamená, že deklarované proměnné nebudou přístupné ostatním programovým organizačním jednotkám kromě explicitního předávání parametrů přes proměnné, které byly deklarovány jako ***vstupní proměnné*** (VAR_INPUT) resp. ***výstupní proměnné*** (VAR_OUTPUT) této jednotky. Jedinou výjimkou z tohoto pravidla jsou proměnné, které byly deklarovány jako ***globální***. Tyto proměnné jsou definovány vně deklarací všech POU a začínají klíčovým slovem VAR_GLOBAL. Za klíčovým slovem VAR_GLOBAL může být volitelně uveden kvalifikátor RETAIN.

Kvalifikátory v deklaraci proměnných

Kvalifikátory umožňují definovat dodatečné vlastnosti deklarovaných proměnných. Klíčové slovo pro kvalifikátor se uvádí za klíčovým slovem VAR. V deklaraci proměnných lze použít následující kvalifikátory :

- ***RETAIN*** – zálohované proměnné (proměnné, které uchovávají hodnotu i během vypnutí napájení PLC)
- ***CONSTANT*** – konstantní hodnota (hodnota proměnné nemůže být změněna)
- ***R_EDGE*** – náběžná hrana proměnné
- ***F_EDGE*** – sestupná hrana proměnné

Tab.3.11 Použití kvalifikátorů RETAIN, CONSTANT, R_EDGE v deklaraci proměnných

Typ proměnné	Význam	RETAIN	CONSTANT	R_EDGE F_EDGE
VAR	lokální	ne	ano	ne
VAR_INPUT	vstupní	ne	ne	ano
VAR_OUTPUT	výstupní	ne	ne	ne
VAR_IN_OUT	vstup / výstupní	ne	ne	ne
VAR_EXTERNAL	globální	ne	ne	ne
VAR_GLOBAL	globální	ano	ano	ne
VAR_TEMP	lokální	ne	ne	ne

Přiřazení datového typu

Konstrukce VAR…END_VAR se používá pro specifikování datového typu a retentivity pro přímo reprezentované proměnné. Tato konstrukce se používá i pro specifikování datového typu a retentivity a (pokud je to nutné a jen v *programech*), pak i fyzického nebo logického umístění symbolicky reprezentovaných jednoduchých nebo složených proměnných .

Přiřazení fyzického nebo logického umístění symbolicky reprezentovaným proměnným se provádí klíčovým slovem AT. Pokud není takové přiřazení definováno, provede se automatická alokace (umístění) proměnné na vhodné místo v paměti PLC.

Např. deklarací
```
VAR
        Stav  AT %IW2: WORD
END_VAR
```
přiřazujeme vstupní proměnnou Stav typu WORD k fyzické vstupní adrese %IW2.

Přiřazení počáteční hodnoty

Konstrukce VAR…END_VAR se používá pro přiřazení počátečních hodnot přímo reprezentovaných proměnných. Tato konstrukce se používá i pro přiřazení počátečních hodnot symbolicky reprezentovaných jednoduchých nebo složených proměnných.

V deklaracích VAR_EXTERNAL nemohou být přiřazovány počáteční hodnoty.

Při inicializaci polí se při naplňování pole ze seznamu inicializačních proměnných nejrychleji mění index umístěný nejvíce vpravo.

Pokud je proměnná deklarována s uživatelským datovým typem struktura, počáteční hodnoty prvků proměnné mohu být deklarovány seznamem uvedeným v závorce, za kterým následuje identifikátor datového typu.

Např. deklarací
```
VAR RETAIN
                AT %QW2: WORD:=16#00FF;
END_VAR
```
se nastaví při studeném restartu jako počáteční hodnota na výstupním slově číslo dva ze šestnácti bitů osm nejméně významných bitů (Least Significant Bits, LSB) na jedničky a osm nejvíce významných bitů (Most Significant Bits, MSB) na nuly.

3.4 Programové organizační jednotky

Programové organizační jednotky (Program Organization Units, POUs) jsou ***funkce***, ***funkční bloky*** a ***programy***. Mohou být dodány od výrobce nebo je může napsat uživatel.

Programové organizační jednotky ***nejsou rekurzivní***, tzn. že vyvolání jedné programové organizační jednotky nesmí způsobit vyvolání jiné programové organizační jednotky stejného typu!

3.4.1 Funkce

Pro účely programovacích jazyků pro PLC je funkce definována jako programová organizační jednotka, která po provedení vygeneruje vždy jeden datový element (může být složen z více hodnot, jako je např. pole nebo struktura). Volání funkce se může použít v textových jazycích jako operand ve výrazu.

Funkce neobsahují žádnou vnitřní stavovou informaci, tzn. že volání funkce se stejnými argumenty (vstupními parametry) vytvoří vždycky stejné hodnoty (výstup).

Deklarace funkce

Textová deklarace funkce se skládá z těchto prvků:

- Klíčového slova FUNCTION, za kterým je uvedeno jméno deklarované funkce, dvojtečka a datový typ hodnoty, kterou bude funkce vracet
- Konstrukce VAR_INPUT…END_VAR, v jejím rámci jsou uvedeny specifikace jmen a typů vstupních parametrů funkce
- Konstrukce VAR …END_VAR, pokud je požadována, v jejím rámci jsou uvedeny specifikace jmen a typů vnitřních proměnných funkce
- Tělo funkce (Function body), zapsané v jazyce ST. Tělo funkce specifikuje operace, které se mají provádět nad vstupními parametry za účelem přiřazení jedné nebo více hodnot proměnné, která má stejné jméno, jako má funkce, a která reprezentuje návratovou hodnotu funkce
- Závěrečného klíčového slova END_FUNCTION

3.4.1.1 Standardní funkce

Standardní funkce použitelné ve všech programovacích jazycích pro PLC jsou podrobně definovány v normě IEC 61 131-3 v kapitole 2.5.1.5. Souhrn standardních funkcí, které jsou podporovány překladačem jazyka ST pro PLC Tecomat, je uveden v této kapitole.

Přetížení funkce (Overloading)

O funkci nebo operaci říkáme, že je ***přetížená*** (*overloaded)*, pokud může pracovat nad prvky vstupních dat různých typů v rámci rodového jména typu. Např. přetížená funkce sčítání rodového typu ANY_NUM může pracovat nad datovými typy LREAL, REAL, DINT, INT a SINT. Pokud

systém PLC podporuje přetíženou funkci nebo operaci, pak se může daná funkce aplikovat na všechny datové typy daného rodového typu, které jsou systémem podporovány.

Informace o tom, které funkce jsou přetížené, jsou uvedeny dále.Uživatelsky definované funkce nemohou být přetížené.

Pokud jsou všechny formální vstupní parametry standardní funkce stejného rodového typu, potom i všechny aktuální parametry musí být stejného typu. Pokud je to nutné, mohou se použít za tímto účelem funkce pro *konverzi* typu. Výstupní hodnota funkce potom bude stejného typu jako aktuální vstupy.

Rozšiřitelné funkce (Extensible)

Některé standardní funkce jsou ***rozšiřitelné*** *(extensible)*, to znamená mohou mít proměnný počet vstupů. U těchto funkcí se předpokládá, že operace definované funkcí budou prováděny nad všemi aplikovanými vstupy. Pokud je funkce rozšiřitelná, maximální počet vstupů není omezen.

Rozdělení standardních funkcí

Standardní funkce jsou rozděleny do několika základních skupin :

- Funkce pro konverzi typu
- Numerické funkce
  - numerické funkce jedné proměnné
  - aritmetické funkce více proměnných
- Funkce nad řetězcem bitů
  - rotace bitů
  - boolovské funkce
- Funkce výběru
- Funkce porovnávání
- Funkce nad řetězcem znaků
- Funkce s typy datum a čas
- Funkce nad datovými typy „výčet“

Přesná specifikace standardních funkcí viz příloha.

Sloupec s názvem **Ovr** v následujících tabulkách udává, je-li funkce přetížená (overloaded). Sloupec s názvem **Ext** nese informaci o tom, je-li příslušná funkce rozšiřitelná (extensible).

Tab.3.12 Standardní funkce, skupina konverze typu

Standardní funkce, skupina konverze typu					
Jméno funkce	**Datový typ vstupu**	**Datový typ výstupu**	**Popis funkce**	**Ovr**	**Ext**
…_TO_…	ANY	ANY	Konverze datového typu uvedeného na prvním místě na datový typ uvedený na druhém místě	ano	ne
TRUNC	ANY_REAL	ANY_INT	„Ořezávání“	ano	ne
BCD_TO…	ANY_BIT	ANY_INT	Převod z BCD kódu na celé číslo	ano	ne
…TO_BCD	ANY_INT	ANY_BIT	Převod z celého čísla na BCD kód	ano	ne

Tab.3.13 Standardní funkce, skupina numerické funkce jedné proměnné

Standardní funkce, skupina numerické funkce jedné proměnné				
Jméno funkce	**Datový typ vstupu / výstupu**	**Popis funkce**	**Ovr**	**Ext**
ABS	ANY_NUM / ANY_NUM	Absolutní hodnota	ano	ne
SQRT	ANY_REAL / ANY_REAL	Odmocnina	ano	ne
LN	ANY_REAL / ANY_REAL	Přirozený logaritmus	ano	ne
LOG	ANY_REAL / ANY_REAL	Desítkový logaritmus	ano	ne
EXP	ANY_REAL / ANY_REAL	Přirozená exponenciální funkce	ano	ne
SIN	ANY_REAL / ANY_REAL	Sinus vstupního úhlu uvedeného v radiánech	ano	ne
COS	ANY_REAL / ANY_REAL	Kosinus vstupního úhlu uvedeného v radiánech	ano	ne
TAN	ANY_REAL / ANY_REAL	Tangens vstupního úhlu uvedeného v radiánech	ano	ne
ASIN	ANY_REAL / ANY_REAL	Arcus sinus	ano	ne
ACOS	ANY_REAL / ANY_REAL	Arcus kosinus	ano	ne
ATAN	ANY_REAL / ANY_REAL	Arcus tangens	ano	ne

Tab.3.14 Standardní funkce, skupina numerické funkce - aritmetické funkce více proměnných

Standardní funkce, skupina numerické funkce - aritmetické funkce více proměnných					
Jméno funkce	**Datový typ vstupu / výstupu**	**Symbol**	**Popis funkce**	**Ovr**	**Ext**
ADD	ANY_NUM, .. ANY_NUM / ANY_NUM	+	Součet OUT:=IN1+ IN2+…+INn	ano	ano
MUL	ANY_NUM, .. ANY_NUM / ANY_NUM	*	Součin OUT:=IN1* IN2*…*INn	ano	ano
SUB	ANY_NUM, ANY_NUM / ANY_NUM	-	Rozdíl OUT:=IN1-IN2	ano	ne
DIV	ANY_NUM, ANY_NUM / ANY_NUM	/	Podíl OUT:=IN1/IN2	ano	ne
MOD	ANY_NUM, ANY_NUM / ANY_NUM		Modulo OUT:=IN1 modulo IN2	ano	ne
EXPT	ANY_NUM / ANY_NUM	**	Umocnění OUT:=IN1**2	ano	ne
MOVE	ANY_NUM / ANY_NUM	:=	Přesunutí, přiřazení OUT:=IN	ano	ne

Tab.3.15 Standardní funkce, skupina funkce nad řetězcem bitů - rotace bitů

Standardní funkce, skupina funkce nad řetězcem bitů - rotace bitů				
Jméno funkce	**Datový typ vstupu / výstupu**	**Popis funkce**	**Ovr**	**Ext**
SHL	ANY_BIT, N / ANY_BIT	Posun vlevo OUT := IN posunutý vlevo o N bitů, zprava doplněno nulami	ano	ne
SHR	ANY_BIT, N / ANY_BIT	Posun vpravo OUT := IN posunutý vpravo o N bitů, zleva doplněno nulami	ano	ne
ROR	ANY_BIT, N / ANY_BIT	Rotace vpravo OUT := IN odrotovaný vpravo o N bitů, zleva odrotované bity	ano	ne
ROL	ANY_BIT, N / ANY_BIT	Rotace vlevo OUT := IN odrotovaný vlevo o N bitů, zprava odrotované bity	ano	ne

Tab.3.16 Standardní funkce, skupina funkce nad řetězcem bitů - boolovské funkce

Standardní funkce, skupina funkce nad řetězcem bitů - boolovské funkce					
Jméno funkce	**Datový typ vstupu / výstupu**	**Symbol**	**Popis funkce**	**Ovr**	**Ext**
AND	ANY_BIT, .. ANY_BIT / ANY_BIT	&	Log. součin, „a současně“, OUT:=IN1& IN2&…&INn	ano	ano
OR	ANY_BIT, .. ANY_BIT / ANY_BIT		Log. součet, „nebo“, inkluzivní součet, OUT:=IN1 OR IN2 OR … OR INn	ano	ano
XOR	ANY_BIT, .. ANY_BIT / ANY_BIT		Výlučný součet, „buď a nebo“, exkluzivní součet OUT:=IN1 XOR IN2 XOR … XOR INn	ano	ano
NOT	ANY_BIT / ANY_BIT		Negace, „ne“, OUT:=NOT IN1	ano	ne

Tab.3.17 Standardní funkce, skupina funkce výběru

Standardní funkce, skupina funkce výběru				
Jméno funkce	**Datový typ vstupu / výstupu**	**Popis funkce**	**Ovr**	**Ext**
SEL	BOOL, ANY, ANY / ANY	Binární výběr OUT := IN0 if G = 0 OUT := IN1 if G = 1	ano	ne
MAX	ANY, .. ANY / ANY	Maximum OUT := MAX(IN1, IN2, .. INn)	ano	ano
MIN	ANY, .. ANY / ANY	Minimum OUT := MIN(IN1, IN2, .. INn)	ano	ano
LIMIT	MN, ANY, MX / ANY	Omezovač OUT := MIN(MAX(IN, MN), MX)	ano	ne

Tab.3.18 Standardní funkce - funkce porovnávání

Standardní funkce, skupina funkce porovnávání				
Jméno funkce	**Datový typ vstupu / výstupu**	**Popis funkce**	**Ovr**	**Ext**
GT	ANY, .. ANY / BOOL	Klesající sekvence OUT:=(IN1> IN2)& (IN2>IN3)&…& (INn-1>INn)	ano	ano
GE	ANY, .. ANY / BOOL	Monotónní sekvence směrem dolů OUT:=(IN1>= IN2)& (IN2>=IN3) &…&(INn-1>=INn)	ano	ano
EQ	ANY, .. ANY / BOOL	Rovnost OUT:=(IN1= IN2)& (IN2=IN3)&…& (INn-1=INn)	ano	ano
LE	ANY, .. ANY / BOOL	Monotónní sekvence směrem nahoru OUT:=(IN1<= IN2)& (IN2<=IN3) &…&(INn-1<=INn)	ano	ano
LT	ANY, .. ANY / BOOL	Vzrůstající sekvence OUT:=(IN1< IN2)& (IN2<IN3)&…& (INn-1<INn)	ano	ano
NE	ANY, ANY / BOOL	Nerovnost OUT := (IN1<>IN2)	ano	ne

Tab.3.19 Standardní funkce, skupina funkce nad řetězcem znaků

Standardní funkce, skupina funkce nad řetězcem znaků				
Jméno funkce	**Datový typ vstupu / výstupu**	**Popis funkce**	**Ovr**	**Ext**
LEN	STRING / INT	OUT := LEN(IN); Délka řetězce IN	ne	ne
LEFT	STRING, ANY_INT / STRING	OUT := LEFT(IN, L); Ze vstupního řetězce IN přesunout L znaků zleva do výstupního řetězce	ano	ne
RIGHT	STRING, ANY_INT / STRING	OUT := RIGHT(IN, L); Ze vstupního řetězce IN přesunout L znaků zprava do výstupního řetězce	ano	ne
MID	STRING, ANY_INT, ANY_INT / STRING	OUT := MID(IN, L, P); Ze vstupního řetězce IN přesunout od P-tého znaku L znaků do výstupního řetězce	ano	ne
CONCAT	STRING, …. STRING / STRING	OUT := CONCAT(IN1, IN2, ...); Připojení jednotlivých vstupních řetězců do výstupního řetězce	ne	ano
INSERT	STRING, STRING, ANY_INT / STRING	OUT := INSERT(IN1, IN2, P); Vložení řetězce IN2 do řetězce IN1 počínaje od P-té pozice	ano	ano
DELETE	STRING, ANY_INT, ANY_INT / STRING	OUT := DELETE(IN, L, P); Smazání L znaků z řetězce IN počínaje od P-té pozice	ano	ano
REPLACE	STRING, STRING, ANY_INT, ANY_INT / STRING	OUT := REPLACE(IN1, IN2, L, P); Náhrada L znaků řetězce IN1 znaky řetězce IN2, vkládání od P-tého místa	ano	ano
FIND	STRING, STRING / INT	OUT := FIND(IN1, IN2); Nalezení pozice prvního znaku prvního výskytu řetězce IN2 v řetězci IN1	ano	ano

Upozornění !

Standardní funkce s řetězci znaků jsou implementovány v cetrálních jednotkách CP-7001 resp. CP-7002 od verze software č.4.0.

Tab.3.20 Standardní funkce, skupina funkce s typy datum a čas

Standardní funkce, skupina funkce s typy datum a čas						
Jméno funkce	**Symbol**	**IN1**	**IN2**	**OUT**	**Ovr**	**Ext**
ADD	+	TIME TOD DT	TIME TIME TIME	TIME TOD DT	ano	ne
SUB	-	TIME DATE TOD TOD DT DT	TIME DATE TIME TOD TIME DT	TIME TIME TOD TIME DT TIME	ano	ne
MUL	*	TIME	ANY_NUM	TIME	ano	ne
DIV	/	TIME	ANY_NUM	TIME	ano	ne
Funkce konverze typu						
DATE_AND_TIME_TO_TIME_OF_DAY, DAT_TO_TIME **DATE_AND_TIME_TO_DATE, DAT_TO_DATE**						

TOD … TIME_OF_DATE
DT … DATE_AND_TIME

Tab.3.21 Standardní funkce, skupina funkce nad datovými typy „výčet“

Standardní funkce, skupina funkce nad datovými typy „výčet“		
Jméno funkce	**Symbol**	**Poznámka**
SEL		Viz Tab. 3.14, pozice 1, funkce výběru
EQ	=	Viz Tab. 3.15, pozice 3, funkce porovnávání
NE	<>	Viz Tab. 3.15, pozice 6, funkce porovnávání

3.4.2 Funkční bloky

Při programování v jazyce ST je ***funkční blok*** taková organizační jednotka programu, která po provedení vygeneruje jednu nebo více hodnot. Z funkčních bloků se dají vytvářet násobné pojmenované ***instance*** (kopie). Každá instance má přiřazený identifikátor (*jméno instance*) a datovou strukturu, která obsahuje její vstupní, vnitřní a výstupní proměnné. Všechny hodnoty proměnných v této datové struktuře se uchovávají od jednoho provedení funkčního bloku k dalšímu jeho provedení. Vyvolání jednoho funkčního bloku se stejnými argumenty (vstupními parametry), tedy nemusí vždy vést ke stejným výstupním hodnotám.Instance funkčního bloku se vytváří použitím deklarovaného typu funkčního bloku v rámci konstrukce VAR…END_VAR.

Jakýkoli funkční blok, který již byl deklarován, může být znovu použit v deklaraci jiného funkčního bloku nebo programu.

Rozsah působnosti instance funkčního bloku je lokální pro tu programovou organizační jednotku, v níž je *instanciován*, (tj. kde je vytvořena jeho pojmenovaná kopie), pokud ovšem není deklarován jako globální.

Jméno instance, které označuje instanci funkčního bloku, se může použít jako vstup pro funkci nebo funkční blok, pokud je deklarován jako vstupní proměnná v deklaraci VAR_INPUT nebo jako vstupní/výstupní proměnná v deklaraci VAR_IN_OUT.

Následující příklad ukazuje postup při deklaraci funkčního bloku, vytvoření jeho instance v programu a konečně jeho vyvolání (provedení).

Příklad 3.4 Funkční blok v jazyce ST

```
//
// deklarace funkcniho bloku fbStartStop
//
FUNCTION_BLOCK fbStartStop
  VAR_INPUT
    start       : BOOL R_EDGE;
    stop        : BOOL R_EDGE;
  END_VAR
  VAR_OUTPUT
    vystup      : BOOL;
  END_VAR

  vystup := (vystup OR start) AND not stop;
END_FUNCTION_BLOCK

PROGRAM Test
  //
  // instance funkcniho bloku fbStartStop
  //
  VAR
    StartStopObvod : fbStartStop;
  END_VAR

  // vyvolani instance funkcniho bloku StartStopObvod
  StartStopObvod( start := TRUE, stop := FALSE);
END_PROGRAM
```

Vstupní a výstupní proměnné instance funkčního bloku mohou být reprezentovány jako prvky datového typu struktura.

Pokud je instance funkčního bloku globální, tak může být také deklarována jako retentivní.V tomto případě platí pouze pro vnitřní a výstupní parametry funkčního bloku.

Zvenku instance jsou přístupné pouze vstupní a výstupní parametry funkčního bloku, tzn. že vnitřní proměnné funkčního bloku zůstávají uživateli funkčního bloku skryté. Přiřazení hodnoty zvenku do výstupní proměnné funkčního bloku není dovoleno, tuto hodnotu přiřazuje jenom zvnitřku sám funkční blok. Přiřazení hodnoty vstupu funkčního bloku je dovoleno kdekoliv v nadřízené POU (typicky je to součást volání funkčního bloku).

Deklarace funkčního bloku

- Oddělovací klíčová slova pro deklaraci funkčních bloků jsou FUNCTION_BLOCK… END_FUNCTION_BLOCK
- Funkční blok může mít více než jeden výstupní parametr, deklarovaný textově konstrukcí VAR_OUTPUT…END_VAR
- Hodnoty proměnných, které jsou předávány funkčnímu bloku pomocí konstrukce VAR_IN_OUT nebo VAR_EXTERNAL mohou být modifikovány zvnitřku funkčního bloku.
- V deklaraci vstupních proměnných funkčního bloku mohou být použity kvalifikátory R_EDGE a F_EDGE. Tyto kvalifikátory označují funkci detekce hran na Boolovských vstupech. Tím je vyvolána implicitní deklarace funkčního bloku R_TRIG nebo F_TRIG.
- Konstrukce definovaná pro inicializaci funkcí se používá i pro deklaraci defaultních hodnot vstupů funkčního bloku a pro počáteční hodnoty jeho vnitřních a výstupních proměnných

Pomocí konstrukce VAR_IN_OUT mohou být do funkčního bloku předávány pouze proměnné (předávání instancí funkčních bloků není podporováno). Kaskádování konstrukcí VAR_IN_OUT je dovoleno.

3.4.2.1 Standardní funkční bloky

Standardní funkční bloky jsou podrobně definovány v normě IEC 61 131-3 v kapitole 2.5.2.3.

Standardní funkční bloky jsou rozděleny do následujících skupin (viz Tab.3.22)

- Bistabilní prvky
- Detekce hrany
- Čítače
- Časovače

Standardní funkční bloky jsou uloženy v knihovně StdLib_Vxx_*.mlb, kde Vxx je verze knihovny.

Tab.3.22 Přehled standardních funkčních bloků

Jméno standardního funkčního bloku	Jméno vstupního parametru	Jméno výstupního parametru	Popis
Bistabilní prvky (klopné obvody)			
SR	S1, R	Q1	dominantní nastavení (sepnutí)
RS	S, R1	Q1	dominantní mazání (vypnutí)
Detekce hrany			
R_TRIG	CLK	Q	detekce náběžné hrany
F_TRIG	CLK	Q	detekce sestupné hrany
Čítače			
CTU	CU, R, PV	Q, CV	dopředný čítač
CTD	CD, LD, PV	Q, CV	zpětný čítač
CTUD	CU, CD, R, LD, PV	QU, QD, CV	oboustranný (reverzibilní) čítač
Časovače			
TP	IN, PT	Q, ET	pulzní časovač
TON (T--0)	IN, PT	Q, ET	zpoždění náběžné hrany
TOF (0--T)	IN, PT	Q, ET	zpoždění sestupné hrany
RTC	EN, PDT	Q, CDT	hodiny reálného času

Názvy, významy a datové typy proměnných používané u standardních funkčních bloků :

Název vstupu / výstupu	**Význam**	**Datový typ**
R	Mazací (resetovací) vstup	BOOL
S	Nastavovací (setovací) vstup	BOOL
R1	Dominantní mazací vstup	BOOL
S1	Dominantní nastavovací vstup	BOOL
Q	Výstup (standardní)	BOOL
Q1	Výstup (pouze u klopných obvodů)	BOOL
CLK	Hodinový (synchronizační) signál	BOOL
CU	Vstup pro dopředné čítání	BOOL
CD	Vstup pro zpětné čítání	BOOL
LD	Nastavení předvolby čítače	BOOL
PV	Předvolba čítače	INT
QD	Výstup (zpětného čítače)	BOOL
QU	Výstup (dopředného čítače)	BOOL
CV	Aktuální hodnota (čítače)	INT
IN	Vstup (časovače)	BOOL
PT	Předvolba časovače	TIME
ET	Aktuální hodnota časovače	TIME
PDT	Předvolba - datum a čas	DT
CDT	Aktuální hodnota - datum a čas	DT

3.4.3 Programy

Program je v normě IEC 61 131-1 definován jako „logický souhrn prvků programovacích jazyků a konstrukcí nutných pro zamýšlené zpracování signálů, které je vyžadováno pro řízení stroje nebo procesu systémem programovatelného automatu“.

Jinak řečeno funkce a funkční bloky lze přirovnat k podprogramům (subroutines) zatímco POU program je hlavní program (main program). Deklarace a používání *programů* je identické s deklarací a používáním funkčních bloků.

Odlišnosti v přístupu k programům oproti funkčním blokům:

- Klíčová slova vymezující deklaraci programu jsou PROGRAM….END_PROGRAM
- *Programy* mohou být instanciovány pouze v rámci *zdrojů (Resources)*, jak je uvedeno v kap. 3.5. Naproti tomu, *funkční bloky* mohou být instanciovány pouze v rámci *programů* nebo jiných *funkčních bloků*
- Programy mohou volat funkce a funkční bloky. Naopak volání programů z funkcí nebo fuknčních bloků není možné

Příklad 3.5 POU Program v jazyce ST

```
PROGRAM test
  VAR
    motor1 : fbMotor;
    motor2 : fbMotor;
  END_VAR

  motor1( startMotoru := sb1, stopMotoru := sb2,
          hvezda => km1, trojuhelnik => km2);
  motor2( startMotoru := sb3, stopMotoru := sb4,
          hvezda => km3, trojuhelnik => km4);

END_PROGRAM
```

Při psaní programu v jazyce ST je důležité si uvědomit, že POU *program* je stejně jako funkční blok pouze „předpisem“, ve kterém je definována struktura dat a algoritmy, prováděné nad touto datovou strukturou. Pro vykonávání definovaného programu je potřebné založit jeho instanci a přiřadit (asociovat) program k některé ze stadradních úloh, ve které pak bude prováděn. Tyto úkony popisuje následující kapitola.

3.5 Konfigurační prvky

Konfigurační prvky popisují run-time vlastnosti programů a přiřazují provádění programů ke konkrétnímu hardwaru v PLC. Představují tak vrcholový předpis celého programu pro PLC.

Při programování systémů Tecomat v jazyce ST se používají následující konfigurační prvky :

- ***Konfigurace*** *(Configuration)* – označuje konkrétní PLC systém, který bude provádět všechny naprogramované POU
- ***Zdroj*** *(Resource)* – označuje konkrétní procesorový modul v PLC, který zajistí provádění programu
- ***Úloha*** *(Task)* – přiřazuje úlohu (proces), v rámci kterého bude příslušná POU *PROGRAM* prováděná

Příklad 3.6

```
CONFIGURATION Plc1
  RESOURCE CPM
    TASK FreeWheeling(Number := 0);
    PROGRAM prg WITH FreeWheeling : test ();
  END_RESOURCE
END_CONFIGURATION
```

V programovacím prostředí Mosaic jsou všechny konfigurační prvky generovány automaticky po vyplnění konfiguračních dialogů.

3.5.1 Konfigurace

Konfigurace označuje PLC systém, který poskytne zdroje pro provádění uživatelského programu. Jinými slovy konfigurace označuje řídící systém, pro který je uživatelský program určen.

Deklarace konfigurace

- Klíčová slova vymezující *konfiguraci* jsou CONFIGURATION….END_ CONFIGURATION
- Za klíčovým slovem CONFIGURATION je uvedeno pojmenování konfigurace, v programovacím prostředí Mosaic jméno konfigurace odpovídá jménu projektu
- *Konfigurace* slouží jako rámec pro definici *Zdroje (Resource)*

Příklad 3.7

```
CONFIGURATION jméno_konfigurace

  // deklarace zdroje

END_CONFIGURATION
```

3.5.2 Zdroje

Zdroj definuje, který modul v rámci PLC poskytne výpočetní výkon pro vykonávání uživatelského programu. V PLC řady TC700 je to vždy procesorový modul systému.

Deklarace zdroje

- Klíčová slova vymezující *zdroj* jsou RESOURCE….END_ RESOURCE
- Za klíčovým slovem RESOURCE je uvedeno pojmenování zdroje, v programovacím prostředí Mosaic je toto jméno implicitně „CPM“
- *Zdroje (Resources)* mohou být deklarovány pouze v rámci *konfigurace*

Příklad 3.8

```
CONFIGURATION Plc1
  RESOURCE CPM

    // deklarace úloh

    // přiřazení programů do deklarovaných úloh

  END_RESOURCE
END_CONFIGURATION
```

3.5.3 Úlohy

Pro účely normy IEC 61 131-3 je *úloha* definována jako výkonný řídicí prvek, který je schopen vyvolávat buď periodicky nebo na základě výskytu vzestupné hrany specifikované boolovské proměnné provádění souboru programových organizačních jednotek (POUs). Těmito programovými organizačními jednotkami mohou být *programy* a v nich deklarované *funkční bloky*.

V PLC Tecomat je pojem úloha totožný s tradičně používaným pojmem proces.

Deklarace úloh

- Klíčové slovo pro označení *úlohy* je TASK
- Za klíčovým slovem TASK je uvedeno jméno úlohy
- Za jménem úlohy jsou uvedeny vlastnosti úlohy, konkrétně číslo odpovídajícího procesu
- *Úlohy (Tasks)* mohou být deklarovány pouze v rámci deklarace *zdroje (Resource)*

Přiřazení programů k úlohám

- Přiřazení programu ke konkrétní úloze je uvozeno klíčovým slovem PROGRAM, kterým se zároveň automaticky zakládá instance uvedeného programu
- Za klíčovým slovem PROGRAM následuje jméno instance programu
- Klíčové slovo WITH uvozuje jméno úlohy, ke které bude program přiřazen
- Na závěr je za dvojtečkou uvedeno jméno asociovaného programu včetně specifikace vstupních a výstupních parametrů
- S jednou úlohou může být asociováno několik programů, pořadí jejich vykonávání v rámci úlohy pak odpovídá pořadí, v jakém byly asociovány
- Přiřazení programů může být deklarováno pouze v rámci deklarace *zdroje (Resource)*

Příklad 3.9

```
CONFIGURATION Plc1
  RESOURCE CPM
    TASK FreeWheeling(Number := 0);
    PROGRAM prg WITH FreeWheeling : test ();
  END_RESOURCE
END_CONFIGURATION
```

3.6 Příkazy v jazyce ST

Jazyk strukturovaného textu je jedním z jazyků definovaných normou IEC 61 131-3. Je to velmi výkonný vyšší programovací jazyk, který má kořeny ve známých jazycích Ada, Pascal a C. Je objektově orientován a obsahuje všechny podstatné prvky moderního programovacího jazyka, včetně větvení (IF-THEN-ELSE a CASE OF) a iterační smyčky (FOR, WHILE a REPEAT). Tyto prvky mohou být vnořovány. Tento jazyk je vynikajícím nástrojem pro definování komplexních funkčních bloků.

Algoritmus zapsaný v jazyce ST lze rozdělit na jednotlivé ***příkazy*** *(statements)*. Příkazy se používají pro výpočet a přiřazení hodnot, řízení toku vykonávání programu a pro volání resp. ukončení POU. Část příkazu, která vypočítává hodnotu, je nazývána ***výraz***. Výrazy produkují hodnoty nezbytné pro provádění příkazů.

3.6.1 Výrazy

Výraz je konstrukce, ze které se po vyhodnocení vygeneruje hodnota odpovídající některému z datových typů, které byly definovány v kapitole 3.2.

Výraz se skládá z ***operátorů a operandů***. Operandem může být literál, proměnná, volání funkce nebo jiný výraz.

Operátory jazyka strukturovaného textu ST jsou přehledně uspořádány v Tab.3.23.

Tab.3.23 Operátory v jazyce strukturovaného textu ST

Operátor	Operace	Priorita
( )	Závorky	Nejvyšší
******	Umocňování	
- **NOT**	Znaménko Doplněk	
***** **/** **MOD**	Násobení Dělení Modulo	
+ **-**	Sčítání Odčítání	
<, >, <= ,>=	Porovnávání	
= **<>**	Rovnost Nerovnost	
&, AND	Boolovské AND	
XOR	Boolovské exkluzivní OR	
OR	Boolovské OR	Nejnižší

Pro operandy operátorů platí stejná omezení jako pro vstupy odpovídajících funkcí definovaných v kapitole 3.4.1.1. Např. výsledek vyhodnocení výrazu A**B je stejný jako výsledek vyhodnocení funkce EXP(A, B), jak je definována v Tab.3.13.

Vyhodnocení výrazu spočívá v aplikaci operátorů na operandy a to s ohledem na prioritu vyjádřenou v Tab.3.23. Operátory s nejvyšší prioritou ve výrazu jsou aplikovány nejdříve, pak následují další operátory směrem k nižší prioritě dokud není vyhodnocování dokončeno.Operátory se stejnou prioritou se vyhodnocují tak jak jsou zapsány ve výrazu směrem odleva doprava.

Příklad 3.10 Priorita operátorů při vyhodnocování výrazů

```
PROGRAM PRIKLAD
  VAR                                  // lokální proměnné
    A       : INT := 2;
    B       : INT := 4;
    C       : INT := 5;
    D       : INT := 8;
    X,Y     : INT;
    Z       : REAL;
  END_VAR

  X := A + B - C * ABS(D);             // X = -34
  Y := (A + B - C) * ABS(D);           // Y = 8
  Z := INT_TO_REAL( Y);
END_PROGRAM
```

Vyhodnocením výrazu A + B- C * ABS(D) v příkladu 3.10 dostaneme hodnotu –34. Pokud požadujeme jiné pořadí vyhodnocování než je uvedeno, musíme použít závorky. Pro stejné hodnoty proměnných pak vyhodnocením výrazu (A + B - C) * ABS(D) dostaneme hodnotu 8.

Funkce se volají jako prvky výrazů, které se skládají ze jména funkce, za nímž následuje seznam argumentů v závorce.

Pokud má operátor dva operandy, operátor, který je nejvíce vlevo se bude vyhodnocovat jako první. Například výraz součinu dvou goniometrických funkcí COS(Y) * SIN(X) bude proto vyhodnocen v tomto pořadí: výpočet výrazu COS(Y), výpočet výrazu SIN(X) a poté teprve výpočet součinu COS(Y) a SIN(X).

Boolovské výrazy mohou být vyhodnocovány pouze v rozsahu nutném pro získání jednoznačné výsledné hodnoty. Například pokud platí, že C <= D, pak může být z výrazu (C > D) & (F < A) vyhodnocena pouze první závorka (C > D). Její hodnota je vzhledem k předpokladu nulová a to již postačuje k tomu, aby byl nulový celý logický součin. Druhý výraz (F < A) se tedy už vyhodnocovat nemusí.

Pokud operátor ve výrazu může být reprezentován jako jedna z přetížených funkcí definovaných v kapitole 3.4.1.1, probíhá konverze operandů a výsledků podle pravidel a příkladů uvedených v této kapitole.

3.6.2 Souhrn příkazů v jazyce ST

Seznam příkazů jazyka strukturovaného textu ST je souhrnně uveden v Tab.3.24. Příkazy jsou ukončeny středníkem.Se znakem konec řádku se v tomto jazyku zachází stejně jako se znakem mezery (space).

Tab.3.24 Seznam příkazů jazyka strukturovaného textu ST

Příkaz	Popis	Příklad	Poznámka
:=	Přiřazení	A **:=** 22;	Přiřazení hodnoty vypočtené na pravé straně do identifikátoru na levé straně
	Volání funkčního bloku	**InstanceFB(** par1 := 10, par2 := 20);	Volání funkčního bloku s předáváním parametrů
IF	Příkaz výběru	**IF** A > 0 **THEN** B := 100; **ELSE** B := 0; **END_IF;**	Výběr alternativy v podmíněný výrazem BOOL
CASE	Příkaz výběru	**CASE** kod **OF** 1 : A := 11; 2 : A := 22; **ELSE** A := 99; **END_CASE;**	Výběr bloku příkazů podmíněný hodnotou výrazu „kod“

FOR	Iterační příkaz smyčka FOR	**FOR** i := 0 **TO** 10 **BY** 2 **DO**
		j := j + i;
		END_FOR;
			Vícenásobná smyčka bloku příkazů s počáteční a koncovou podmínkou a hodnotou inkrementu

WHILE	Iterační příkaz smyčka WHILE	**WHILE** i > 0 **DO**
		n := n * 2;
		END_WHILE;
			Vícenásobná smyčka bloku příkazů s podmínkou ukončení smyčky na začátku

REPEAT	Iterační příkaz smyčka REPEAT	**REPEAT**
		k := k + i;
		UNTIL i < 20;
		END_REPEAT;
			Vícenásobná smyčka bloku příkazů s podmínkou ukončení smyčky na konci

EXIT	Ukončení smyčky	**EXIT;**	Předčasné ukončení iteračního příkazu
RETURN	Návrat	**RETURN**;	Opuštění právě vykonávané POU a návrat do volající POU
;	Prázdný příkaz	**;;**	

3.6.2.1 Příkaz přiřazení

Přiřazovací příkaz nahrazuje aktuální hodnotu jednoduché nebo složené proměnné výsledkem, který vznikne po vyhodnocení výrazu. Přiřazovací příkaz se skládá z odkazu na proměnnou na levé straně, za ním následuje operátor přiřazení „:=“, za kterým je uveden výraz, který se má vyhodnotit.

Příkaz přiřazení je velmi mocný. Může přiřadit jednoduchou proměnnou ale i celou datovou strukturu. Jak ukazuje příklad 3.11, kde přiřazovací příkaz A := B je použit pro nahrazení hodnoty jednoduché proměnné A aktuální hodnotou jednoduché proměnné B (obě proměnné jsou základního typu INT). Ovšem přiřazení lze s úspěchem použít i pro složené proměnné AA := BB a pak se tímto přiřazovacím příkazem přepíší všechny položky složené proměnné AA položkami složené proměnné BB. Proměnné musí být samozřejmě stejného datového typu.

Příklad 3.11 Přiřazení jednoduché a složené proměnné

```
TYPE
  tyZAZNAM : STRUCT
    vyrobniCislo     : UDINT;
    barva            : (cervena, zelena, bila, modra);
    jakost           : USINT;
  END_STRUCT;
END_TYPE

PROGRAM PRIKLAD
  VAR                        // lokální proměnné
    A, B   : INT;
    AA, BB : tyZAZNAM;
  END_VAR

  A  := B;                        // přiřazení jednoduché proměnné
  AA := BB;                       // přiřazení složené proměnné
END_PROGRAM
```

Přiřazovací příkaz se může použít také pro přiřazení návratové hodnoty funkce, a to umístěním jména funkce na levé straně přiřazovacího operátoru v těle deklarace funkce. Návratová hodnota funkce bude výsledkem posledního vyhodnocení tohoto přiřazovacího příkazu.

Příklad 3.12 Přiřazení návratové hodnoty funkce

```
FUNCTION PRIKLAD : REAL
  VAR_INPUT                   // vstupní proměnné
    F, G   : REAL;
    S      : REAL  := 3.0;
  END_VAR

  PRIKLAD := F * G / S;       // návratová hodnota funkce
END_FUNCTION
```

3.6.2.2 Příkaz volání funkčního bloku

Funkce může být volána jako součást vyhodnocení výrazu, jak bylo uvedeno v této kapitole, odstavec výrazy.

Funkční bloky se volají příkazem, který se skládá ze jména instance funkčního bloku, za kterým následuje seznam pojmenovaných vstupních parametrů s přiřazenými hodnotami.Na pořadí, v němž jsou parametry v seznamu při volání funkčního bloku uvedeny, nezáleží. Při každém volání funkčního bloku nemusí být přiřazeny všechny vstupní parametry. Pokud nějakému parametru není přiřazena hodnota před voláním funkčního bloku, pak se použije hodnota naposledy přiřazená (nebo hodnota počáteční, pokud nebylo ještě provedeno žádné přiřazení).

Příklad 3.13 Příkaz volání funkčního bloku

```
// deklarace funkčního bloku
FUNCTION_BLOCK fb_OBDELNIK
  VAR_INPUT
    A,B               : REAL;          // vstupní proměnné
  END_VAR
  VAR_OUTPUT
    obvod, plocha     : REAL;          // výstupní proměnné
  END_VAR

  obvod := 2.0 * (A + B); plocha := A * B;
END_FUNCTION_BLOCK

// globální proměnné
VAR_GLOBAL
  OBDELNIK : fb_OBDELNIK;             // globální instance FB
END_VAR

// deklarace programu
PROGRAM main
  VAR
    o,s     : REAL;                   // lokální proměnné
  END_VAR

  // volání FB s úplným seznamem parametrů
  OBDELNIK( A := 2.0, B := 3.0, obvod => o , plocha => s);
  IF  o > 20.0 THEN
    ....
  END_IF;

  // volání FB s neúplným seznamem parametrů
  OBDELNIK( B := 4.0, A := 2.5);
  IF  OBDELNIK.obvod > 20.0 THEN
    ....
  END_IF;
END_PROGRAM
```

3.6.2.3 Příkaz IF

Příkaz IF specifikuje, že se má provádět skupina příkazů jedině v případě, že se přiřazený boolovský výraz vyhodnotí jako pravdivý (TRUE). Pokud je podmínka nepravdivá, pak se neprovádí buď žádný příkaz nebo se provádí skupina příkazů, které jsou uvedeny za klíčovým slovem ELSE (nebo za klíčovým slovem ELSIF, pokud jemu přiřazená podmínka je pravdivá).

Příklad 3.14 Příkaz IF

```
FUNCTION PRIKLAD : INT
  VAR_INPUT
    kod     : INT;                    // vstupní proměnná
  END_VAR

  IF kod < 10 THEN PRIKLAD := 0;      // při kod < 10 fce vrátí 0
  ELSIF kod < 100 THEN PRIKLAD := 1;  // při 9 < kod < 100 fce vrátí 1
  ELSE PRIKLAD := 2;                  // při kod > 99 fce vrátí 2
  END_IF;
END_FUNCTION
```

3.6.2.4 Příkaz CASE

Příkaz CASE obsahuje výraz, který se vyhodnotí do proměnné typu INT (to je tzv. „selektor“), a dále seznam skupin příkazů, kde každá skupina je označena jedním nebo více přirozenými čísly nebo rozsahem přirozených čísel. Tím je vyjádřeno, že se bude provádět první skupina příkazů, do jejíchž mezí patří vypočítaná hodnota selektoru. Pokud se vypočítaná hodnota nehodí ani do jedné skupiny příkazů, provede se sekvence příkazů, které jsou uvedeny za klíčovým slovem ELSE (pokud se v příkazu CASE vyskytuje). Jinak se neprovede žádná sekvence příkazů.

Příklad 3.15 Příkaz CASE

```
FUNCTION PRIKLAD : INT
  VAR_INPUT
    kod     : INT;                  // vstupní proměnná
  END_VAR

  CASE kod OF
    10      : PRIKLAD := 0;         // při kod = 10 fce vrátí 0
    20      : PRIKLAD := 1;         // při kod = 20 fce vrátí 1
    21..55  : PRIKLAD := 2;         // při 20 < kod < 56 fce vrátí 2
    100     : PRIKLAD := 3;         // při kod = 100 fce vrátí 3
  ELSE
              PRIKLAD := 4;         // jinak fce vrátí 4
  END_CASE;
END_FUNCTION
```

3.6.2.5 Příkaz FOR

Příkaz FOR se používá, pokud počet iterací může být určen předem, jinak se používají konstrukce WHILE nebo REPEAT.

Příkaz FOR indikuje, že sekvence příkazů se má provádět opakovaně až do výskytu klíčového slova END_FOR, přičemž se zvyšují hodnoty řídicí proměnné smyčky FOR. Řídicí proměnná, počáteční hodnota a koncová hodnota jsou výrazy stejného typu.integer (SINT, INT nebo DINT) a nesmí se měnit vlivem jakéhokoli z opakovaných příkazů. Příkaz FOR zvyšuje nebo snižuje hodnotu řídicí proměnné cyklu od počáteční do koncové hodnoty, a to po přírůstcích určených hodnotou výrazu (defaultně je tento přírůstek roven jedné). Test ukončovací podmínky se provádí na začátku každé iterace, takže pokud počáteční hodnota řídicí proměnné cyklu překročí hodnotu koncovou, sekvence příkazů se neprovede.

Příklad 3.16 Příkaz FOR

```
FUNCTION FAKTORIAL : UDINT
  VAR_INPUT
    kod     : USINT;                    // vstupní proměnná
  END_VAR
  VAR_TEMP
    i       : USINT;                    // pomocná proměnná
    tmp     : UDINT := 1;               // pomocná proměnná
  END_VAR

  FOR i := 1 TO kod DO
    tmp := tmp * USINT_TO_UDINT( i);
  END_FOR;
  FAKTORIAL := tmp;
END_FUNCTION
```

3.6.2.6 Příkaz WHILE

Příkaz WHILE způsobí, že se sekvence příkazů až do klíčového slova END_WHILE bude provádět opakovaně, až do té doby, dokud není přiřazený boolovský výraz nepravdivý. Pokud je přiřazený boolovský výraz na začátku nepravdivý, pak se sekvence příkazů neprovede vůbec. Smyčka FOR ... END_FOR se dá přepsat použitím konstrukce WHILE … END_WHILE. Příklad 3.17 lze s použitím příkazu WHILE přepsat následovně:

Příklad 3.17 Příkaz WHILE

```
FUNCTION FAKTORIAL : UDINT
  VAR_INPUT
    kod     : USINT;                    // vstupní proměnná
  END_VAR
  VAR_TEMP
    i       : USINT;                    // pomocná proměnná
    tmp     : UDINT := 1;               // pomocná proměnná
  END_VAR

  i := kod;
  WHILE i <> 0 DO
    tmp := tmp * USINT_TO_UDINT( i);  i := i - 1;
  END_WHILE;
  FAKTORIAL := tmp;
END_FUNCTION
```

Pokud, je příkaz WHILE použit v algoritmu, pro který není zaručeno splnění podmínky ukončení nebo provedení příkazu EXIT, pak řídící systém vyhlásí chybu cyklu.

3.6.2.7 Příkaz REPEAT

Příkaz REPEAT způsobí, že se sekvence příkazů až do klíčového slova UNTIL bude provádět opakovaně (a alespoň jednou) až do té doby, dokud není přiřazený boolovský výraz pravdivý. Smyčka WHILE…END_WHILE se dá přepsat použitím konstrukce REPEAT … END_REPEAT, což opět ukážeme na stejném příkladu :

Příklad 3.18 Příkaz REPEAT

```
FUNCTION FAKTORIAL : UDINT
  VAR_INPUT
    kod     : USINT;                    // vstupní proměnná
  END_VAR
  VAR_TEMP
    i       : USINT := 1;               // pomocná proměnná
    tmp     : UDINT := 1;               // pomocná proměnná
  END_VAR

  REPEAT
    tmp := tmp * USINT_TO_UDINT( i);  i := i + 1;
  UNTIL i > kod
  END_REPEAT;
  FAKTORIAL := tmp;
END_FUNCTION
```

Pokud, je příkaz REPEAT použit v algoritmu, pro který není zaručeno splnění podmínky ukončení nebo provedení příkazu EXIT, pak řídící systém vyhlásí chybu cyklu.

3.6.2.8 Příkaz EXIT

Příkaz EXIT se používá pro ukončení iterací před splněním ukončovací podmínky.

Pokud je příkaz EXIT umístěn uvnitř vnořené iterační konstrukce (příkazy FOR, WHILE, REPEAT), odchod nastane z nejhlubší smyčky, ve které je EXIT umístěn, tzn. že se řízení předá na další příkaz za prvním ukončením smyčky (END_FOR, END_WHILE, END_REPEAT), který následuje za příkazem EXIT.

Příklad 3.19 Příkaz EXIT

```
FUNCTION FAKTORIAL : UDINT
  VAR_INPUT
    kod     : USINT;                    // vstupní proměnná
  END_VAR
  VAR_TEMP
    i       : USINT;                    // pomocná proměnná
    tmp     : UDINT := 1;               // pomocná proměnná
  END_VAR

  FOR i := 1 TO kod
    IF i > 13 THEN
      tmp := 16#FFFF_FFFF; EXIT;
    END_IF;
    tmp := tmp * USINT_TO_UDINT( i);
  END_FOR;
  FAKTORIAL := tmp;
END_FUNCTION
```

Při výpočtu faktoriálu pro čísla větší než 13 bude výsledek větší než maximální číslo, které lze uložit do proměnné typu UDINT. Tato situace je v příkladu 3.19 ošetřena pomocí příkazu EXIT.

3.6.2.9 Příkaz RETURN

Příkaz RETURN se používá k opuštění funkce, funkčního bloku nebo programu před jeho dokončením.

V případě použití příkazu RETURN ve funkci je potřebné nastavit výstup funkce (proměnnou, která se jmenuje stejně jako funkce) před provedením příkazu RETURN. V opačném případě nebude výstupní hodnota funkce definována.

Pokud bude příkaz RETURN použit ve funkčním bloku, měl by programátor zajistit nastavení výstupních proměnných funkčního bloku před provedením příkazu. Nenastavené výstupní proměnné budou mít hodnotu odpovídající inicializační hodnotě pro příslušný datový typ nebo hodnotu nastavenou v předchozím voláním funkčního bloku.

Příklad 3.20 Příkaz RETURN

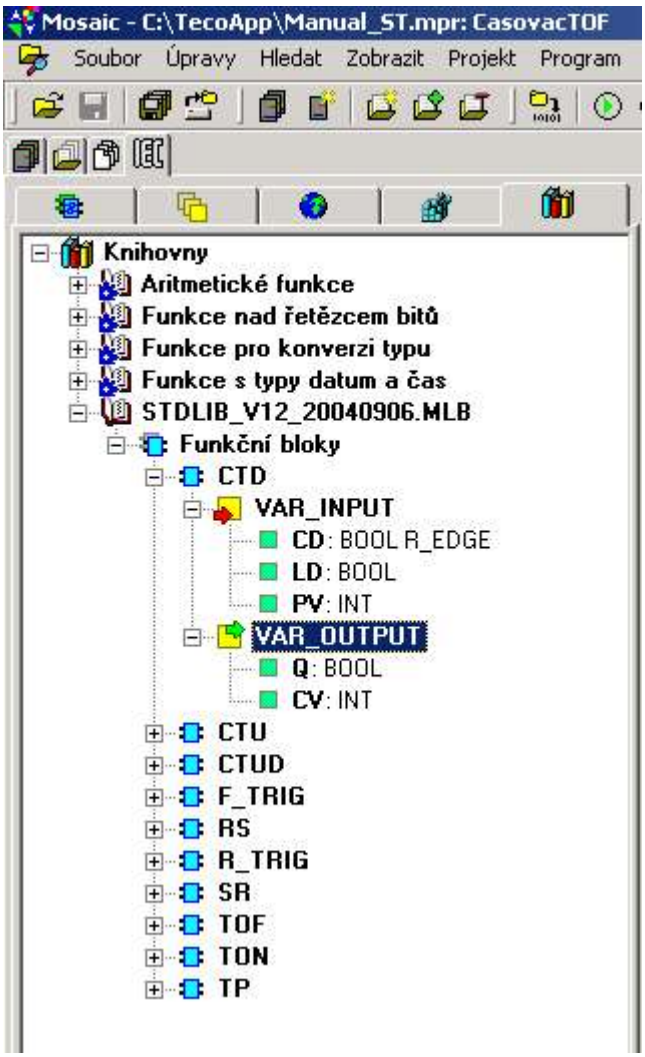
```
FUNCTION FAKTORIAL : UDINT
  VAR_INPUT
    kod     : USINT;                    // vstupní proměnná
  END_VAR
  VAR_TEMP
    i       : USINT;                    // pomocná proměnná
    tmp     : UDINT := 1;               // pomocná proměnná
  END_VAR

  IF kod > 13 THEN FAKTORIAL := 16#FFFF_FFFF; RETURN; END_IF;
  i := kod;
  WHILE i <> 0 DO
    tmp := tmp * USINT_TO_UDINT( i);  i := i - 1;
  END_WHILE;
  FAKTORIAL := tmp;
END_FUNCTION
```

Při výpočtu faktoriálu pro čísla větší než 13 bude výsledek větší než maximální číslo, které lze uložit do proměnné typu UDINT. Tato situace je v příkladu 3.20 ošetřena pomocí příkazu RETURN.

3.7 Knihovny

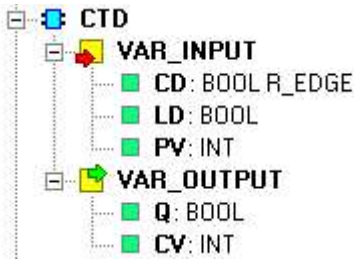
3.7.1 Standardní knihovna funkčních bloků

Standardní knihovna obsahuje funkční bloky čítačů CTD, CTU a CTUD, časovačů TON, TOF a TP, klopných obvodů RS a SR a bloky detekce hran R_TRIG a F_TRIG. Následují obrázek ukazuje začlenění funkčních bloků do knihovny v prostředí Mosaic.



3.7.1.1 Funkční blok čítače dolů CTD

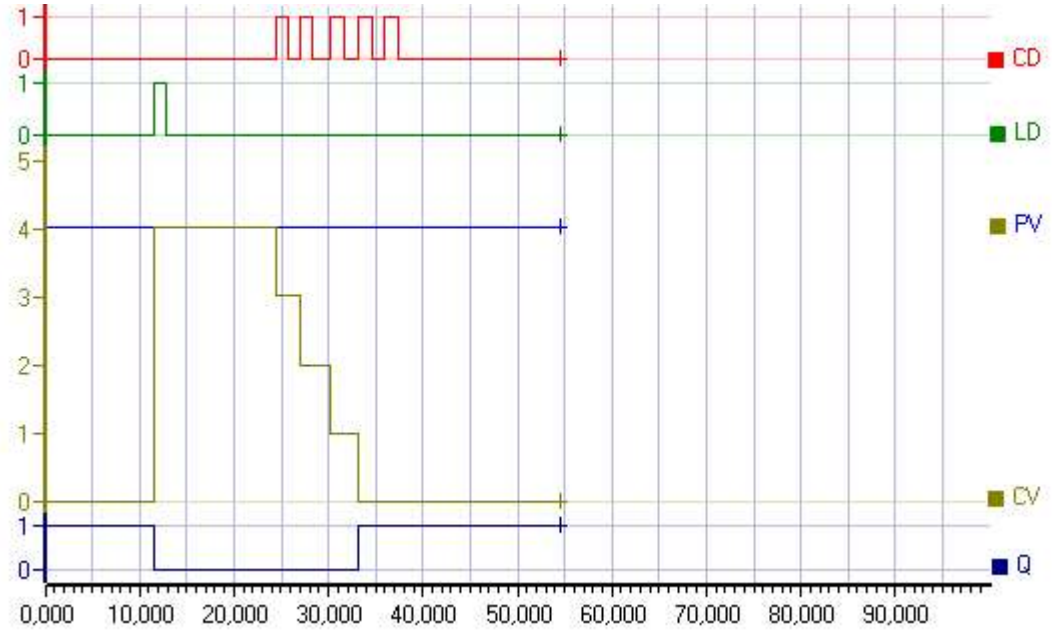
Funkční blok pro čítání dolů.



Vstupní proměnné :
CD vstup pro čítání dolů
LD vstup pro nastavení předvolby
PV předvolba čítače
Výstupní proměnné :
Q výstup čítače
CV hodnota čítače

Pokud má vstupní proměnná LD hodnotu TRUE, pak se hodnota předvolby PV naplní do hodnoty čítače CV. Pokud má vstupní proměnná LD hodnotu FALSE a vstupní proměnná CD přejde ze stavu FALSE do stavu TRUE (náběžná hrana), hodnota čítače CV je snížena o 1. Je-li hodnota čítače CV rovna nule, výstup čítače Q je nastaven na hodnotu TRUE. Jinak má hodnotu FALSE.

Chování čítače CTD vysvětluje následující obrázek.

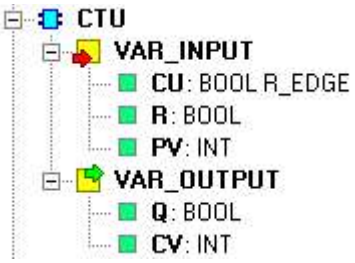


Příklad programu s voláním funkčního bloku CTD :

```
PROGRAM Citac
  VAR
    pulzy          : BOOL;
    setCTD         : BOOL;
    counterCTD     : CTD;              // instance čítače
    output         : BOOL;
  END_VAR

  counterCTD( CD := pulzy, LD := setCTD, PV := 4, Q => output);
END_PROGRAM
```

3.7.1.2 Funkční blok čítače nahoru CTU

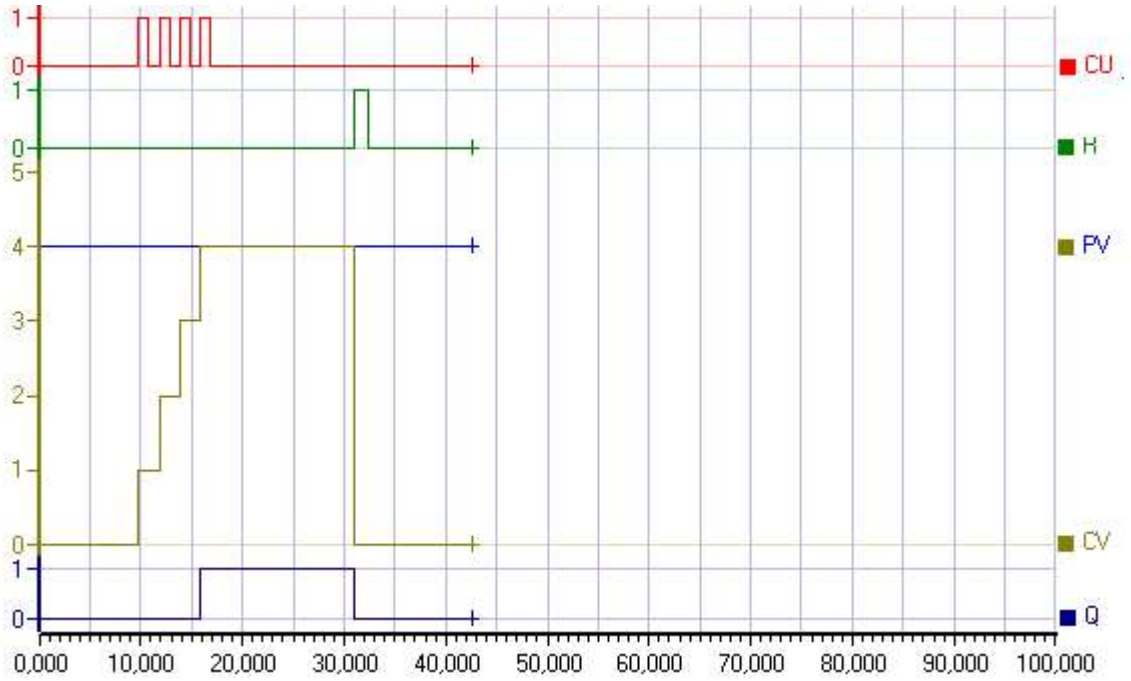
Funkční blok pro čítání nahoru.



Vstupní proměnné :
CU vstup pro čítání nahoru
R reset čítače
PV předvolba čítače
Výstupní proměnné :
Q výstup čítače
CV hodnota čítače

Pokud má vstupní proměnná R hodnotu TRUE, hodnota čítače CV je vynulovaná. Pokud má vstupní proměnná R hodnotu FALSE a vstupní proměnná CU přejde ze stavu FALSE do stavu TRUE (náběžná hrana), hodnota čítače CV je zvýšena o 1. Je-li hodnota čítače CV větší nebo rovná předvolbě PV, výstup čítače Q je nastaven na hodnotu TRUE. Jinak má hodnotu FALSE.

Chování čítače CTU vysvětluje následující obrázek.

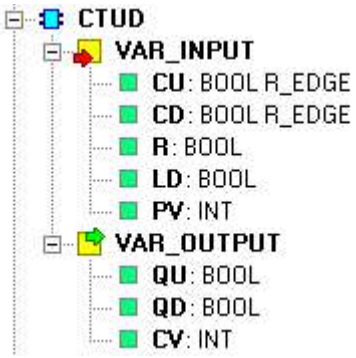


Příklad programu s voláním funkčního bloku CTU :

```
PROGRAM Citac
  VAR
    pulzy          : BOOL;
    resetCTU       : BOOL;
    counterCTU     : CTU;                  // instance čítače
    output         : BOOL;
  END_VAR

  counterCTU( CU := pulzy, R := resetCTU, PV := 4, Q => output);
END_PROGRAM
```

3.7.1.3 Funkční blok obousměrného čítače CTUD

Funkční blok pro obousměrné čítání.



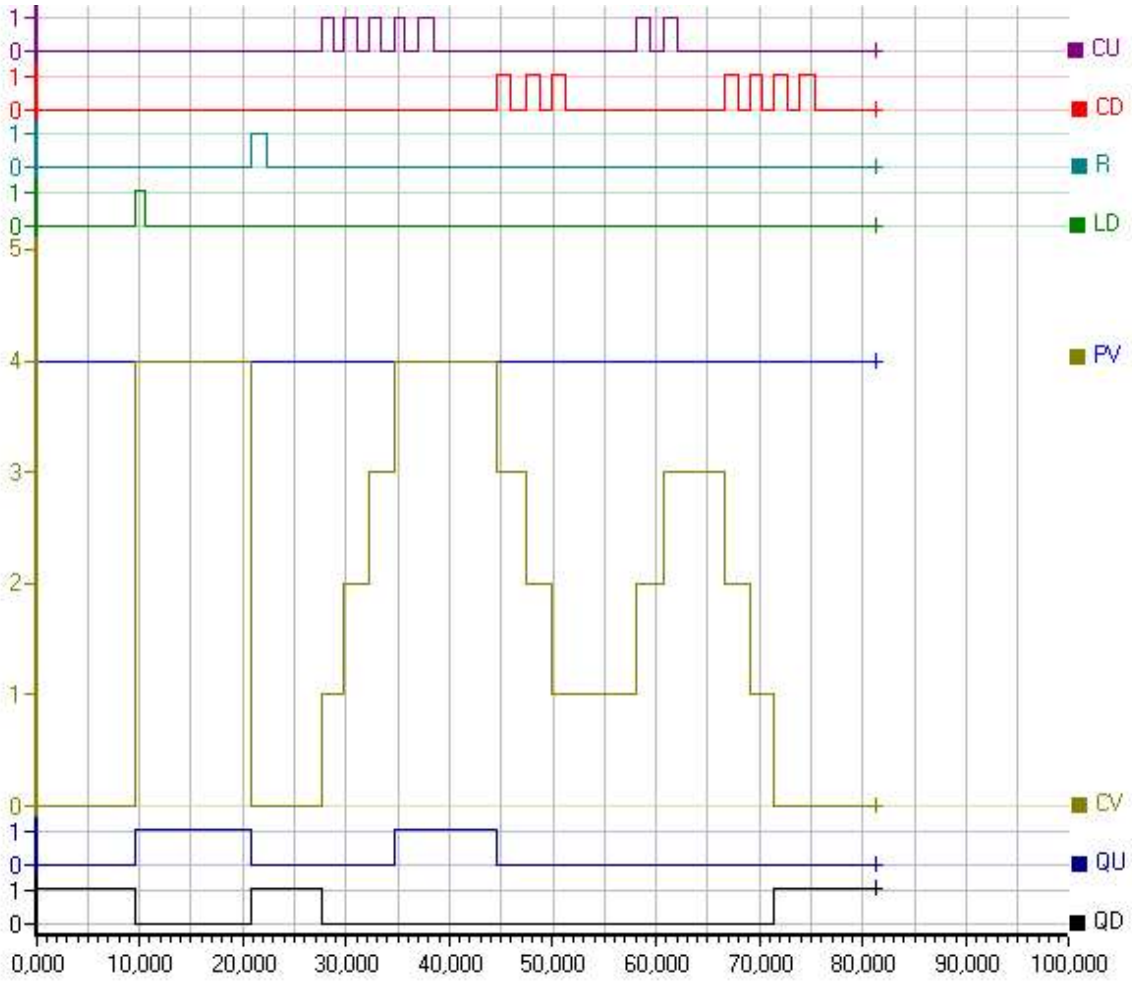
Vstupní proměnné :

CU	vstup pro čítání nahoru
CD	vstup pro čítání dolů
R	reset čítače
LD	vstup pro nastavení předvolby
PV	předvolba čítače

Výstupní proměnné :

QU	výstup čítače nahoru
QD	výstup čítače dolů
CV	hodnota čítače

Chování čítače CTUD vysvětluje následující obrázek.



Jestliže má vstupní proměnná R hodnotu TRUE, hodnota čítače CV bude vynulovaná. Jestliže má vstupní proměnná LD hodnotu TRUE, hodnota čítače CV bude nastavena na hodnotu předvolby PV.

Když vstupní proměnná CU přejde ze stavu FALSE do stavu TRUE (náběžná hrana), hodnota čítače CV je zvýšena o 1, čítač čítá nahoru. Podobně pokud vstupní proměnná CD přejde ze stavu FALSE do stavu TRUE (náběžná hrana), hodnota čítače CV je snížena o 1, čítač čítá dolů.

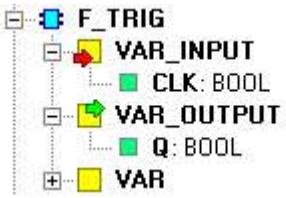
Je-li hodnota čítače CV větší nebo rovná předvolbě PV, výstup čítače QU je nastaven na hodnotu TRUE, jinak má hodnotu FALSE. Je-li hodnota čítače CV rovna nule, výstup čítače QD je nastaven na hodnotu TRUE, jinak má hodnotu FALSE.

Příklad programu s voláním funkčního bloku CTUD :

```
PROGRAM Citac
  VAR
    pulzyUP       : BOOL;
    pulzyDOWN     : BOOL;
    resetCTUD     : BOOL;
    setCTUD       : BOOL;
    counterCTUD   : CTUD;                    // instance čítače
    limitUP       : BOOL;
    limitDOWN     : BOOL;
  END_VAR

  counterCTUD( CU := pulzyUP,   CD := pulzyDOWN,
               R  :=  resetCTUD, LD := setCTUD,
               PV := 4,
               QU => limitUP, QD => limitDOWN );
END_PROGRAM
```

3.7.1.4 Funkční blok F_TRIG

Funkční blok pro pro detekci sestupné hrany.

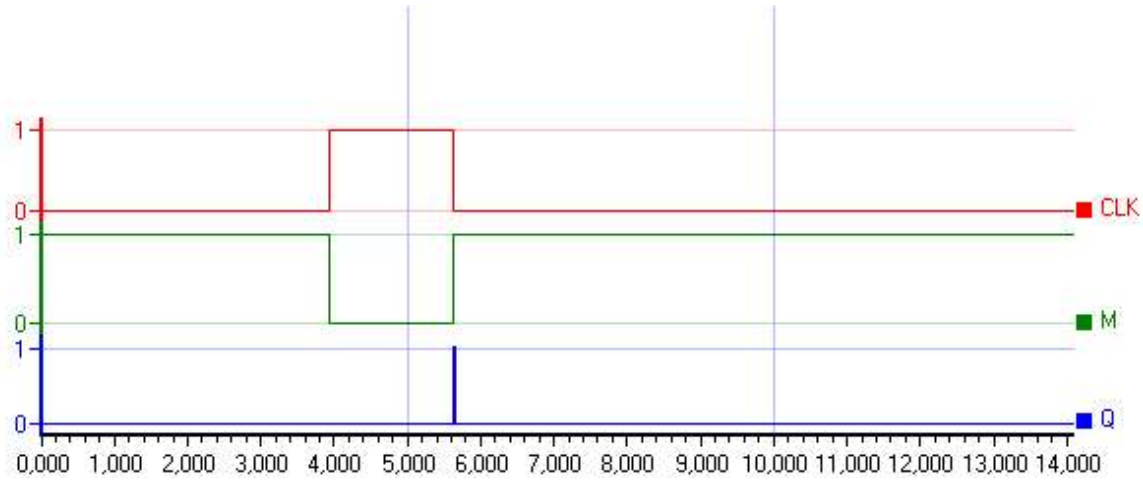


Vstupní proměnné :
CLK

Výstupní proměnné :
Q

Chování funkčního bloku F_TRIG odpovídá následujícímu programu v jazyce ST.

```
function_block F_TRIG
//--------------------------------------
//  Falling Edge Detector
//
  var_input  CLK : BOOL;          end_var
  var_output Q   : BOOL;          end_var
  var        M   : BOOL := TRUE; end_var

  Q := not CLK and not M; M := not CLK;
end_function_block
```

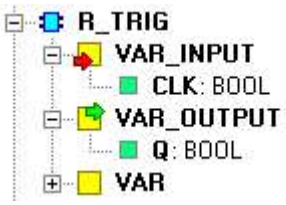


Příklad programu s voláním funkčního bloku F_TRIG :

```
PROGRAM EdgeDetect
  VAR
    input          : BOOL;
    inst_FTRIG     : F_TRIG;               // instance FB F_TRIG
    output         : BOOL;
  END_VAR

  inst_FTRIG( CLK := input, Q => output);
END_PROGRAM
```

3.7.1.5 Funkční blok R_TRIG

Funkční blok pro pro detekci náběžné hrany.

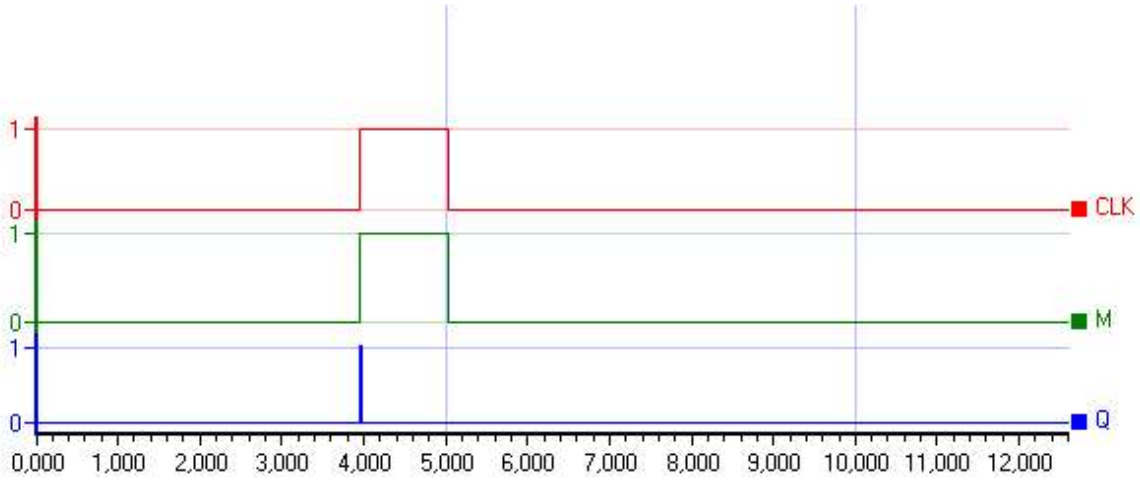


Vstupní proměnné :
CLK

Výstupní proměnné :
Q

Chování funkčního bloku R_TRIG odpovídá následujícímu programu v jazyce ST.

```
function_block R_TRIG
//------------------------------
//  Rising Edge Detector
//
  var_input  CLK : BOOL; end_var
  var_output Q   : BOOL; end_var
  var        M   : BOOL; end_var

  Q := clk and not M; M := CLK;
end_function_block
```

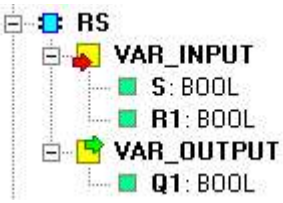


Příklad programu s voláním funkčního bloku R_TRIG :

```
PROGRAM EdgeDetect
  VAR
    input          : BOOL;
    inst_RTRIG     : R_TRIG;                 // instance FB R_TRIG
    output         : BOOL;
  END_VAR

  inst_RTRIG( CLK := input, Q => output);
END_PROGRAM
```

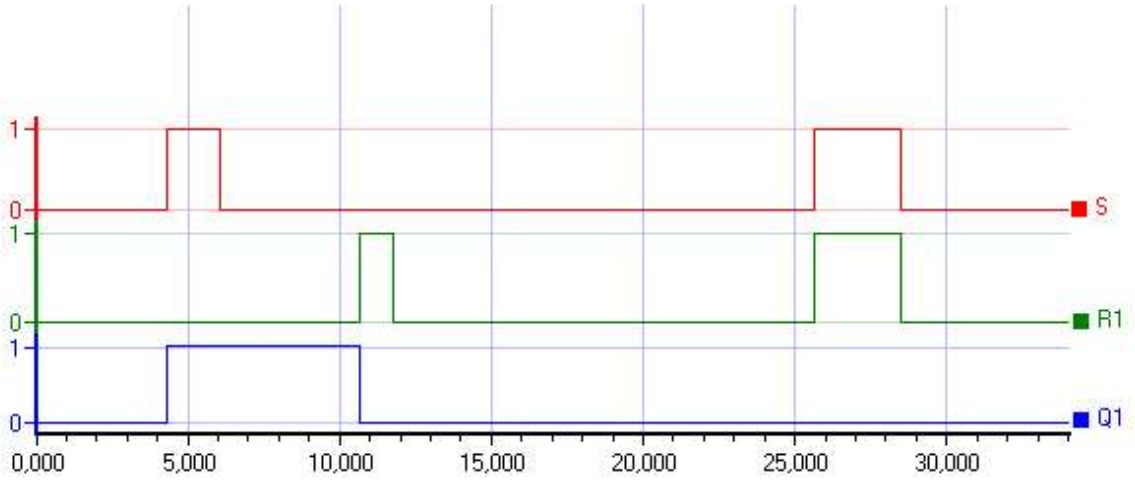
3.7.1.6 Funkční blok RS

Funkční blok pro realizaci dvoustavého klopného obodu s dominantní funkcí RESET.



Chování funkčního bloku RS odpovídá následujícímu programu v jazyce ST.

```
function_block RS
//-------------------------------------
//  Flip-Flop (Reset Dominant)
//
  var_input   S, R1      : BOOL; end_var
  var_output Q1          : BOOL; end_var

  Q1 := not R1 and (S or Q1);
end_function_block
```

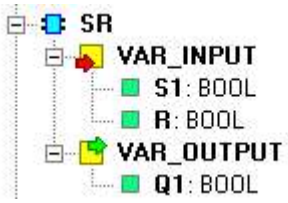


Příklad programu s voláním funkčního bloku RS :

```
PROGRAM BistableBlock
  VAR
    inputSET       : BOOL;
    inputRESET     : BOOL;
    inst_RS        : RS;
    output         : BOOL;
  END_VAR

  inst_RS( S := inputSET, R1 := inputRESET, Q1 => output);
END_PROGRAM
```

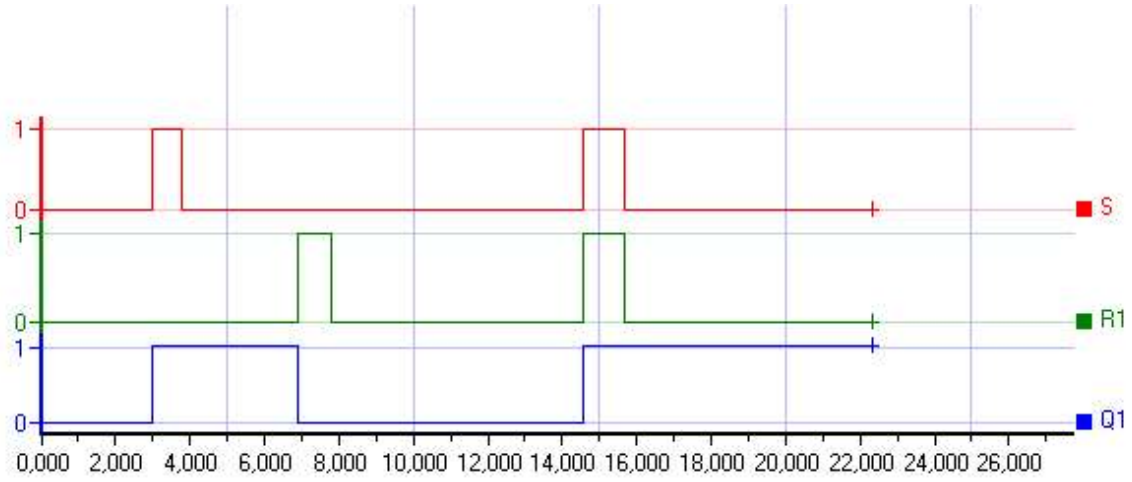
3.7.1.7 Funkční blok SR

Funkční blok pro realizaci dvoustavého klopného obodu s dominantní funkcí SET.



Chování funkčního bloku SR odpovídá následujícímu programu v jazyce ST.

```
function_block SR
//--------------------------------------
//  Flip-Flop (Set Dominant)
//
  var_input  S1, R      : BOOL; end_var
  var_output Q1         : BOOL; end_var

  Q1 := S1 or (not R and Q1);
end_function_block
```

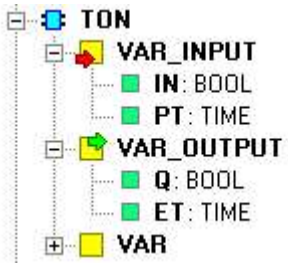


Příklad programu s voláním funkčního bloku SR :

```
PROGRAM BistableBlock
  VAR
    inputSET      : BOOL;
    inputRESET    : BOOL;
    inst_SR       : SR;
    output        : BOOL;
  END_VAR

  inst_SR( S1 := inputSET, R := inputRESET, Q1 => output);
END_PROGRAM
```

3.7.1.8 Funkční blok časovače TON

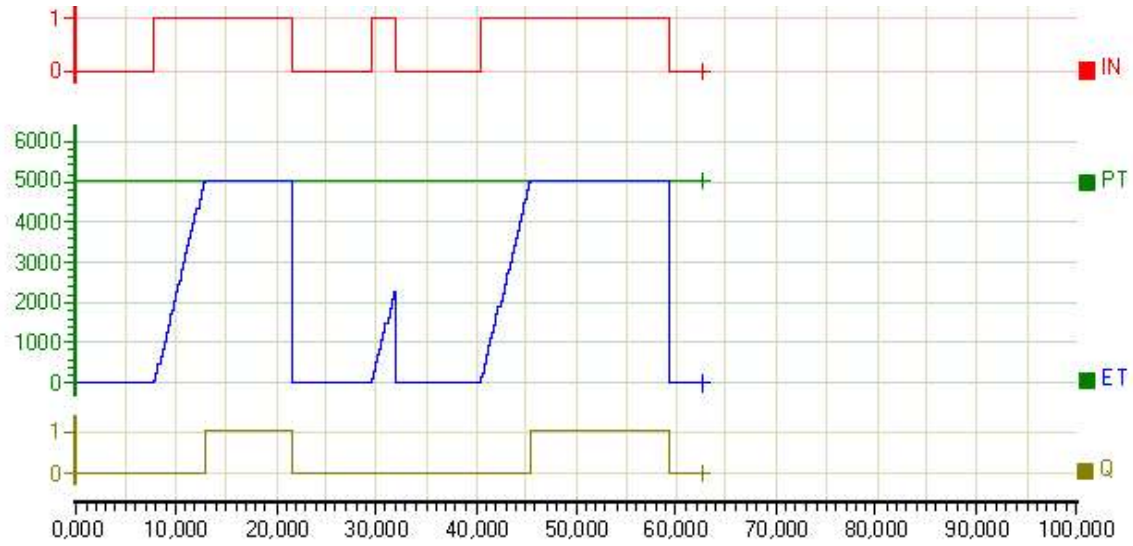
Funkční blok TON (Timer On Delay) realizuje prodlevu na náběžnou hranu, tj. vlastně relé se zpožděným přítahem.



Vstupní proměnné :
IN vstup časovače
PT předvolba časovače

Výstupní proměnné :
Q výstup časovače
ET aktuální hodnota časovače

Jestliže vstup IN je FALSE, výstup Q je FALSE a ET má hodnotu 0. Jakmile vstup IN přejde do stavu TRUE, aktuální hodnota časovače ET se začne zvětšovat a ve chvíli kdy dosáhne předvolby PT výstup Q se nastaví na hodnotu TRUE. Aktuální hodnota časovače se dále nezvětšuje. Chování časovače TON vysvětluje následující obrázek.

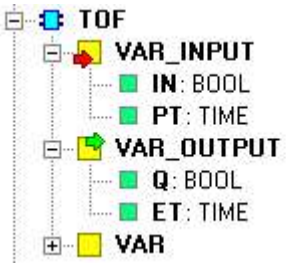


Příklad programu s voláním funkčního bloku TON :

```
PROGRAM Timer
  VAR
    start          : BOOL;
    timerTON       : TON;
    output         : BOOL;
  END_VAR

  timerTON( IN := start, PT :=T#5s, Q => output );
END_PROGRAM
```

3.7.1.9 Funkční blok časovače TOF

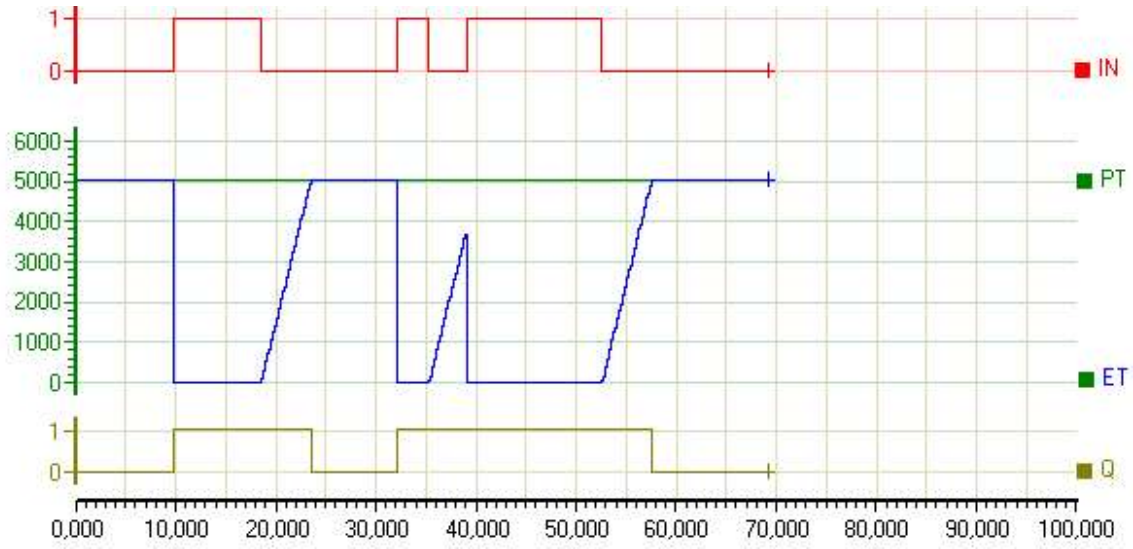
Funkční blok TOF (Timer Off Delay) realizuje prodlevu na sestupnou hranu, tj. vlastně relé se zpožděným odpadem.



Vstupní proměnné :
IN vstup časovače
PT předvolba časovače

Výstupní proměnné :
Q výstup časovače
ET aktuální hodnota časovače

Jakmile vstup IN přejde do stavu FALSE, aktuální hodnota časovače ET se začne zvětšovat a ve chvíli kdy dosáhne předvolby PT výstup Q se nastaví na hodnotu TRUE. Aktuální hodnota časovače se pak dále nezvětšuje. Výstup Q je ve stavu FALSE vždy, když je vstup IN FALSE a zároveň aktuální hodnota ET je rovna předvolbě PT. Chování časovače TOF popisuje následující obrázek.

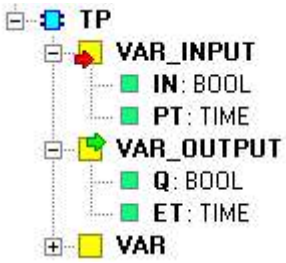


Příklad programu s voláním funkčního bloku TOF :

```
PROGRAM Timer
  VAR
    start          : BOOL;
    timerTOF       : TOF;
    output         : BOOL;
  END_VAR

  timerTOF( IN := start, PT :=T#5s, Q => output );
END_PROGRAM
```

3.7.1.10 Funkční blok časovače TP

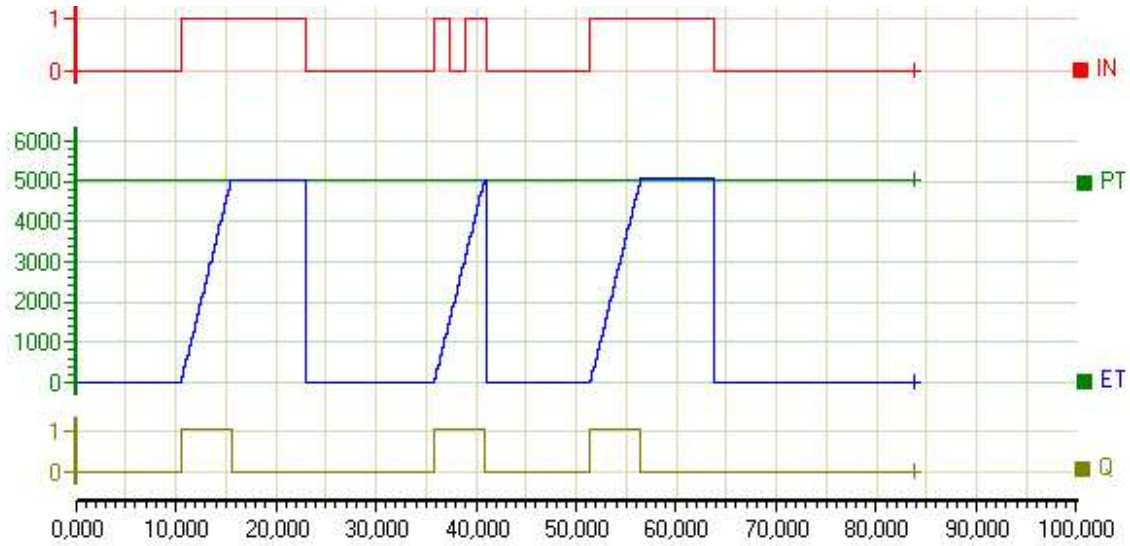
Funkční blok TP (Timer Pulse) generuje pulz dané šířky na náběžnounou hranu.



Vstupní proměnné :
IN vstup časovače
PT předvolba časovače

Výstupní proměnné :
Q výstup časovače
ET aktuální hodnota časovače

Jakmile vstup IN přejde do stavu TRUE, aktuální hodnota časovače ET se začne zvětšovat až do chvíle kdy dosáhne předvolby PT. Aktuální hodnota časovače se pak dále nezvětšuje. Výstup Q je nastaven na hodnotu TRUE jestliže byla detekovaná náběžná hrana na vstupu IN a zároveň aktuální hodnota ET je menší než předvolba PT. Jinak má výstup Q hodnotu FALSE. Chování časovače TP popisuje následující obrázek



Příklad programu s voláním funkčního bloku TP :

```
PROGRAM Timer
  VAR
    start          : BOOL;
    timerTP        : TP;
    output         : BOOL;
  END_VAR

  timerTP( IN := start, PT :=T#5s, Q => output );
END_PROGRAM
```

3.7.2 Knihovna funkcí nad řetězcem znaků

Tato knihovna obsahuje funkce pro práci s řetězci znaků (datový typ STRING). Obsahuje funkci pro zjištění délky řetězce LEN, funkce pro vybírání části řetězce LEFT, RIGHT a MID, funkci pro spojování řetězců CONCAT, funkci pro vkládání řetězce do řetězce INSERT, funkci pro vypouštění části řetězce DELETE, funkci pro náhradu části řetězce jiným řetězcem REPLACE a konečně funkci pro nalezení pozice řetězce v jiném řetězci FIND. Následují obrázek ukazuje začlenění funkcí nad řetězcem znaků do knihovny v prostředí Mosaic.

Implementace datového typu STRING v systémech Tecomat odpovídá řetězcům v jazyce C.

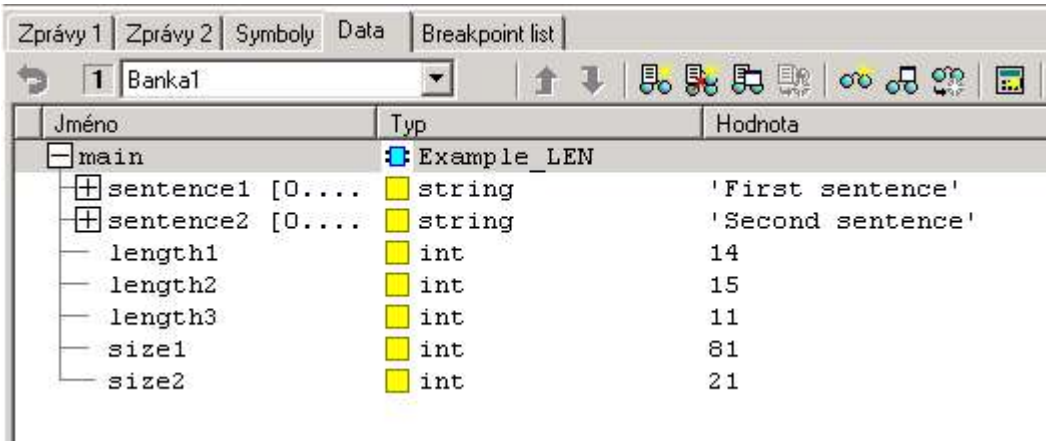
3.7.2.1 Funkce LEN

Funkce vrátí délku vstupního řetězce IN. Délka řetězce odpovídá aktuálnímu počtu znaků v řetězci. Návratová hodnota je typu INT.

Vstupní proměnné :
IN znakový řetězec
Návratová hodnota : délka řetězce

Použití funkce LEN ukazuje následující příklad. V souvislosti s délkou řetězce je dobré si uvědomit, že aktuální délka řetězce ve většině případů neodpovídá velikosti proměnné, ve které je řetězec uložen. Velikost proměnné typu STRING je daná deklarací proměnné a lze jí zjistit pomocí funkce SIZEOF zatímco délka řetězce v proměnné je daná počtem ASCII znaků v řetězci a zjišťuje se pomocí funkce LEN.

```
PROGRAM Example_LEN
  VAR
    sentence1     : STRING     := 'First sentence';
    sentence2     : STRING[20] := 'Second sentence';
    length1,
    length2,
    length3 : INT;
    size1,
    size2   : INT;
  END_VAR

  // length of string
  length1 := LEN( sentence1);
  length2 := LEN( IN := sentence2);
  length3 := LEN( 'Hallo word!');

  // size of variable
  size1 := sizeof( sentence1);
  size2 := sizeof( sentence2);

END_PROGRAM
```



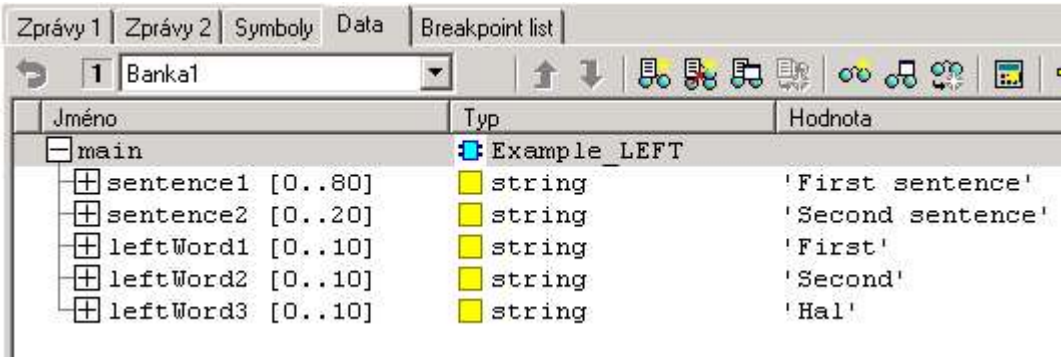
3.7.2.2 Funkce LEFT

Funkce LEFT vrátí L znaků ze vstupního řetězce IN. Znaky jsou vráceny zleva, tj. od začátku vstupního řetězce.

Vstupní proměnné :
IN vstupní řetězec
L počet znaků
Návratová hodnota : řetězec

Použití funkce LEFT ukazuje následující příklad.

```
PROGRAM Example_LEFT
  VAR
    sentence1     : STRING     := 'First sentence';
    sentence2     : STRING[20] := 'Second sentence';
    leftWord1,
    leftWord2,
    leftWord3     : STRING[10];
  END_VAR

  leftWord1 := LEFT( sentence1, 5);
  leftWord2 := LEFT( IN := sentence2, L := 6);
  leftWord3 := LEFT( 'Hallo word!', 3);

END_PROGRAM
```



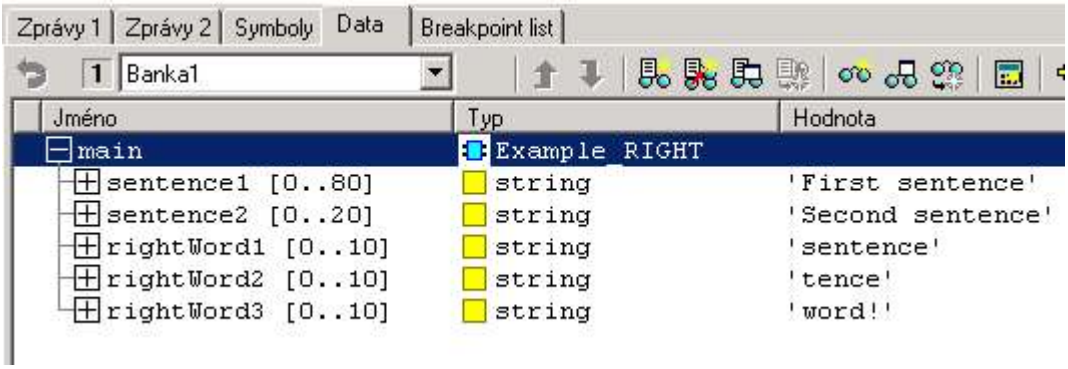
3.7.2.3 Funkce RIGHT

Funkce RIGHT vrátí L znaků ze vstupního řetězce IN. Znaky jsou vráceny zprava, tj. od konce vstupního řetězce.

Vstupní proměnné :
IN vstupní řetězec
L pocet znaků
Návratová hodnota : řetězec

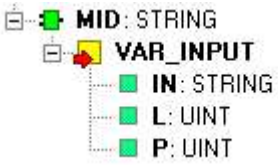
Použití funkce RIGHT ukazuje následující příklad.

```
PROGRAM Example_RIGHT
  VAR
    sentence1     : STRING     := 'First sentence';
    sentence2     : STRING[20] := 'Second sentence';
    rightWord1,
    rightWord2,
    rightWord3    : STRING[10];
  END_VAR

  rightWord1 := RIGHT( sentence1, 8);
  rightWord2 := RIGHT( IN := sentence2, L := 5);
  rightWord3 := RIGHT( 'Hallo word!', 5);

END_PROGRAM
```



3.7.2.4 Funkce MID

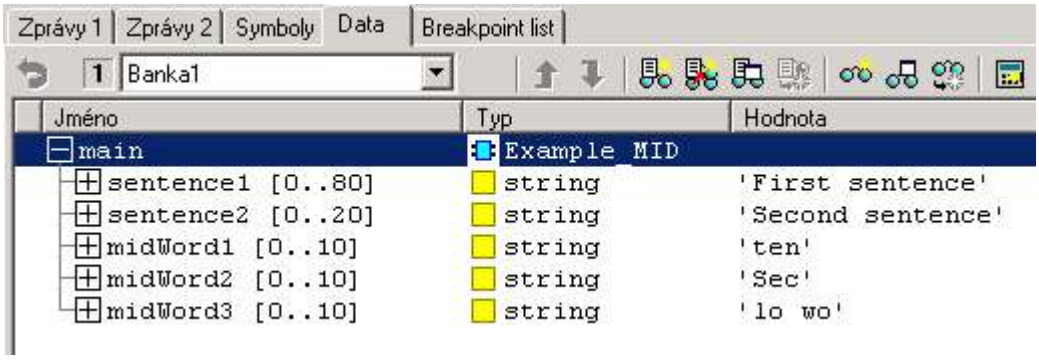
Funkce MID vrátí L znaků ze vstupního řetězce IN. Znaky jsou vráceny od pozice P ve vstupním řetězci. První znak v řetězci má pozici 1.



Vstupní proměnné :
IN znakový řetězec
L pocet znaků
P pozice ve vstupním řetězci
Návratová hodnota : řetězec

Použití funkce MID ukazuje následující příklad.

```
PROGRAM Example_MID
  VAR
    sentence1     : STRING := 'First sentence';
    sentence2     : STRING[20];
    midWord1,
    midWord2,
    midWord3      : STRING[10];
  END_VAR

  midWord1  := mid( sentence1, 3, 10);
  sentence2 := 'Second sentence';
  midWord2  := mid( IN := sentence2, L := 3, P := 1);
  midWord3  := mid( 'Hallo word!', 5, 4);

END_PROGRAM
```



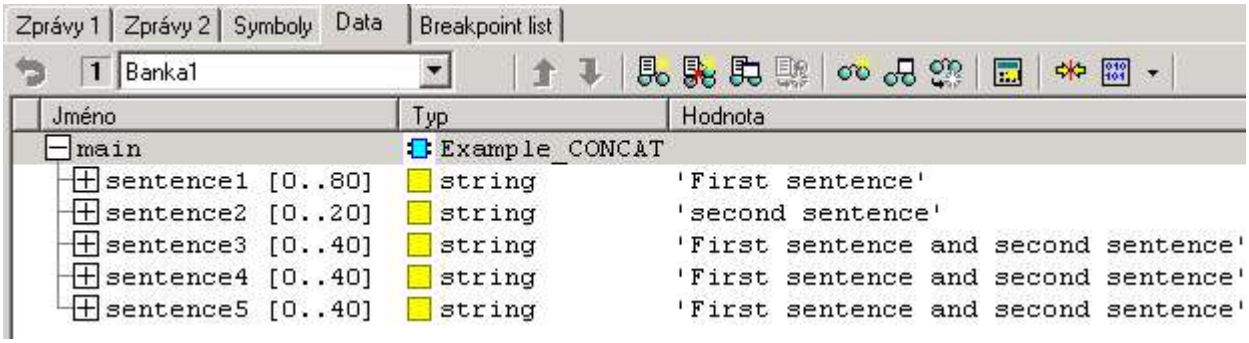
3.7.2.5 Funkce CONCAT

Funkce CONCAT sloučí několik řetězců do výstupního řetězce.

Vstupní proměnné :
IN1 znakový řetězec
IN2 znakový řetězec
Návratová hodnota : řetězec

Použití funkce CONCAT ukazuje následující příklad. Volání funkce je možné provádět buď klasicky jménem funkce nebo pomocí přetíženého operátoru “+”. Funkce CONCAT je rozšiřitelná, takže může mít různý počet vstupních řetězců.

```
PROGRAM Example_CONCAT
  VAR
    sentence1     : STRING     := 'First sentence';
    sentence2     : STRING[20] := 'second sentence';
    sentence3,
    sentence4,
    sentence5     : STRING[40];
  END_VAR

  sentence3 := CONCAT( sentence1, ' and ', sentence2);
  sentence4 := CONCAT( IN1 := sentence1, IN2 := ' and ', IN3 := sentence2);
  sentence5 := sentence1 + ' and ' + sentence2;
END_PROGRAM
```



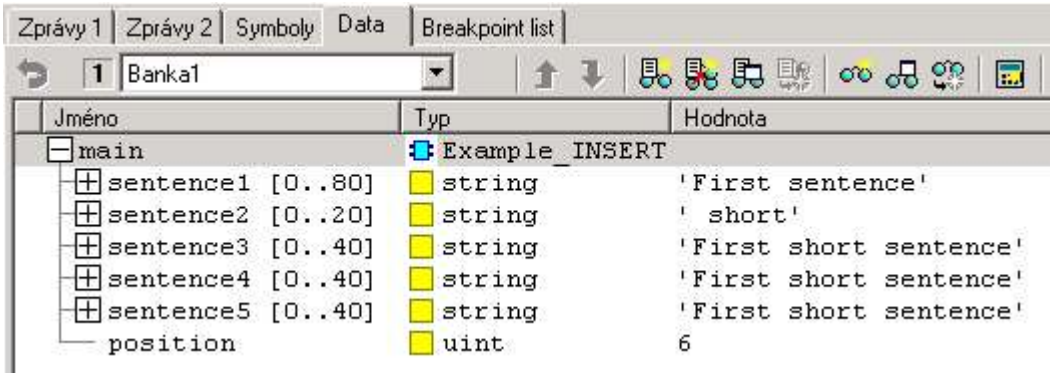
3.7.2.6 Funkce INSERT

Funkce INSERT vloží řetězec IN2 do řetězce IN1 na pozici P. Takto vytvořený řetězec je vrácen jako návratová hodnota funkce.

Vstupní proměnné :
IN1 vstupní řetězec
IN2 vkládaný řetězec
P pozice ve vstupním řetězci
Návratová hodnota : řetězec

Použití funkce INSERT ukazuje následující příklad.

```
PROGRAM Example_INSERT
  VAR
    sentence1     : STRING     := 'First sentence';
    sentence2     : STRING[20] := ' short';
    sentence3,
    sentence4,
    sentence5     : STRING[40];
    position      : UINT := 6;
  END_VAR

  sentence3 := INSERT( sentence1, ' short', 6);
  sentence4 := INSERT( IN1 := sentence1, IN2 := ' short', P := 6);
  sentence5 := INSERT( sentence1, sentence2, position);
END_PROGRAM
```



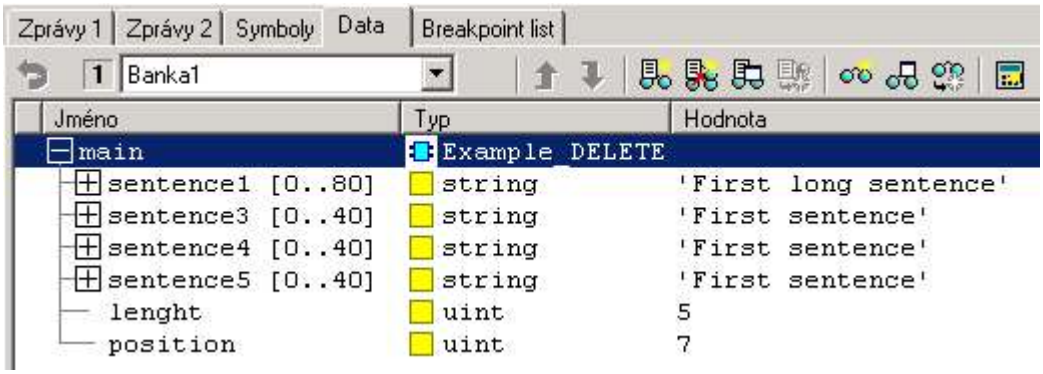
3.7.2.7 Funkce DELETE

Funkce DELETE vypustí počet znaků L ze vstupního řetězce IN. Znaky jsou vypuštěny od pozice P. Takto vytvořený řetězec je vrácen jako návratová hodnota funkce.

Vstupní proměnné :
IN vstupní řetězec
L pocet vypouštěných znaků
P pozice, od které se znaky vypustí
Návratová hodnota : řetězec

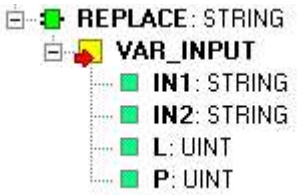
Použití funkce DELETE ukazuje následující příklad.

```
PROGRAM Example_DELETE
  VAR
    sentence1     : STRING     := 'First long sentence';
    sentence3,
    sentence4,
    sentence5     : STRING[40];
    lenght        : UINT := 5;
    position      : UINT := 7;
  END_VAR

  sentence3 := DELETE( sentence1, 5, 7);
  sentence4 := DELETE( IN := sentence1, L := lenght, P := 7);
  sentence5 := DELETE( sentence1, lenght, position);
END_PROGRAM
```



3.7.2.8 Funkce REPLACE

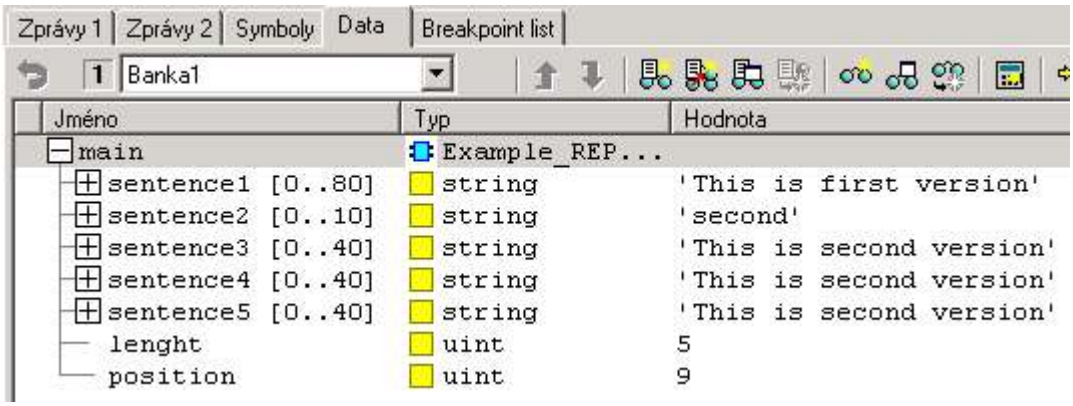
Funkce REPLACE nejprve vypustí L znaků ze vstupního řetězce IN1. Znaky jsou vypuštěny od pozice P. Poté se na stejnou pozici P vloží řetězec IN2. Takto vytvořený řetězec je vrácen jako návratová hodnota funkce.



Vstupní proměnné :
IN1 vstupní řetězec
IN2 vkládaný řetězec
L pocet vypouštěných znaků
P pozice ve vstupním řetězci
Návratová hodnota : řetězec

Použití funkce REPLACE ukazuje následující příklad.

```
PROGRAM Example_REPLACE
  VAR
    sentence1     : STRING     := 'This is first version';
    sentence2     : STRING[10] := 'second';
    sentence3,
    sentence4,
    sentence5     : STRING[40];
    lenght        : UINT := 5;
    position      : UINT := 9;
  END_VAR

  sentence3 := REPLACE( sentence1, 'second', 5, 9);
  sentence4 := REPLACE( IN1 := sentence1, IN2 := sentence2, L := 5, P := 9);
  sentence5 := REPLACE( sentence1, sentence2, lenght, position);
END_PROGRAM
```



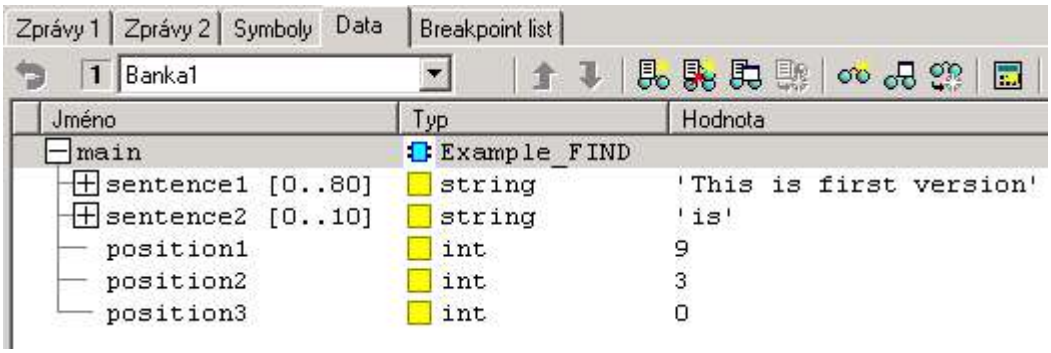
3.7.2.9 Funkce FIND

Funkce FIND vrátí pozici řetězce IN2 ve vstupním řetězci IN1. Pokud pokud řetězec IN2 není ve vstupním řetězci IN1 obsažen, funkce vrátí 0. Pokud je řetězec IN2 obsažen v řetězci IN1 vícekrát, funkce FIND vrátí pozici prvního výskytu. Při vyhledávání se berou v úvahu velká a malá písmena. Návratová hodnota funkce je typu INT.

Vstupní proměnné :
IN1 vstupní řetězec
IN2 hledaný řetězec
Návratová hodnota : pozice IN2 v řetězci IN1

Použití funkce FIND ukazuje následující příklad.

```
PROGRAM Example_FIND
  VAR
    sentence1     : STRING     := 'This is first version';
    sentence2     : STRING[10] := 'is';
    position1,
    position2,
    position3     : INT;
  END_VAR

  position1 := FIND( sentence1, 'first');
  position2 := FIND( IN1 := sentence1, IN2 := sentence2);
  position3 := FIND( sentence1, 'First');
END_PROGRAM
```

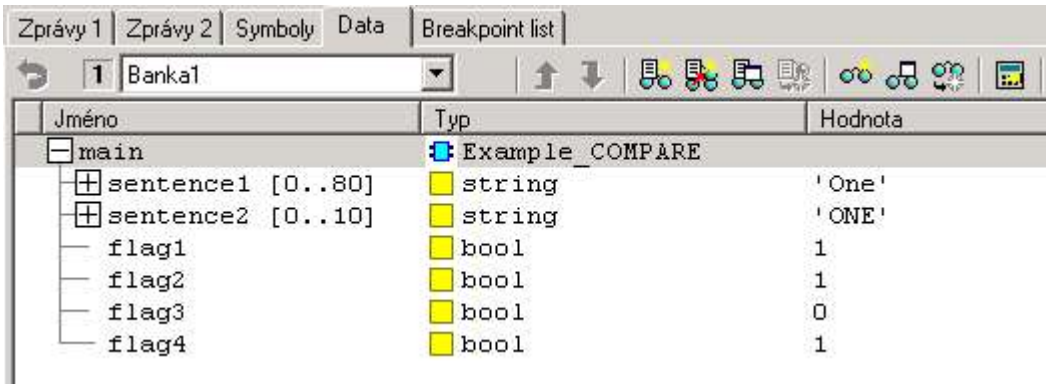


3.7.2.10 Funkce porovnání řetězců

Standardní funkce pro porovnání (větší, menší, větší nebo rovno, ...) jsou přetížené i pro typ STRING, takže řetězce lze mezi sebou porovnávat. Řetězce jsou porovnávány znak po znaku, velká a malá písmena se rozlišují. Z pohledu porovnání mají malá písmena "větší hodnotu", neboť ASCII kód malých písmen je větší než velkých písmen.

Použití funkcí pro porovnání řetězců ukazuje následující příklad.

```
PROGRAM Example_COMPARE
  VAR
    sentence1     : STRING     := 'One';
    sentence2     : STRING[10] := 'ONE';
    flag1,
    flag2,
    flag3,
    flag4    : BOOL;
  END_VAR

  flag1 := sentence1 = 'One';
  flag2 := sentence1 <> sentence2;
  flag3 := sentence1 = sentence2;
  flag4 := sentence1 > sentence2;
END_PROGRAM
```



OBSAH