SUPORTE DE CURSO

IEC 1131-3 Elementos Comuns

Livro Texto: Programming industrial control systems using IEC 1131-3 – R.W. Lewis

UFMG – Curso de Automação Industrial.
Prof. Constantino Seixas Filho

IEC 1131-3 Elementos comuns

IEC1131-3 define 5 linguagens:

Structured Text (ST)	Textuais
Instruction List (IL)	
Function Block Diagram (FBD)	Gráficas
Ladder Diagram (LD)	
Sequential Function Charts (SFC)	

<table>
<tr><td colspan="5">SFC</td></tr>
<tr><td>ST</td><td>IL</td><td>LD</td><td>FBD</td><td></td></tr>
<tr><td colspan="2">TEXTUAIS</td><td colspan="3">GRÁFICAS</td></tr>
</table>

As linguagens ST, IL, FBD e SFC podem ser usadas dentro de blocos de ação e em transições para construir *Sequential Function Charts*.

Conjunto de caracteres:

- É utilizado o padrão ISO 646 "*Basic code table*".
- Apenas letras maiúsculas são consideradas. Letras minúsculas são convertidas para maiúsculas:
  Heater1 -> HEATER1
- As palavras chaves da linguagem são *case sensitive*.

Identificadores:

- O primeiro caracter não pode ser um dígito.
- Caracter *underline* múltiplos não são permitidos.
- Identificador não pode incluir espaço.
- Embora os identificadores possam ter qualquer tamanho, apenas os 6 primeiros caracteres são considerados para unicidade.

Palavras chaves:

Não usar identificadores que coincidam com as palavras chaves da linguagem.

Comentários:

```
(***********************************)
(* Comentários devem ser envolvidos por *)
(***********************************)
```

Tipos de dados:

Inteiros:

Tipo IEC	Descrição	Bits	Range
SINT	Short integer	8	-127 a +127
INT	Integer	16	-32768 a 32767
DINT	Double Integer	32	-2^{31} a $2^{31}-1$
LINT	Long Integer	64	2^{63} a $2^{63}-1$
USINT	Unsigned short integer	8	0 a 256
UINT	Unsigned integer	16	0 a 65535
UDINT	Unsigned double integer	32	0 a $2^{32}-1$
ULINT	Unsigned long integer	64	0 a $2^{64}-1$

Um CLP não necessita suportar todos estes tipos de dados.

Representação de inteiros:

Decimais: -43 0 +34 45_233_999
Binários: 2#1111_1111
Octal: 8#377
Hexadecimal: 16#A0B

Aumenta legibilidade

Floating Point:

Tipo IEC	Descrição	Bits	Range	Precisão
REAL	Real	32	$\pm 10^{\pm 38}$	$1/2^{23}$
LREAL	Long Real	64	$\pm 10^{\pm 308}$	$1/2^{52}$

Representação de float:

10.123 +12_123.21 -0.001298 -1.65E-10

Duração (tempo)

Tipo IEC	Descrição	Bits	Uso
TIME	Duração do tempo	Dependente de implementação	Armazenar a duração do tempo após evento.

O Uso típico é para medir a duração de intervalos de tempo ou definir timeouts.

Representação de tempo:

T#12d3h3s	define 12 dias, 3 horas e 3 segundos
TIME#6d_5h_3m_4s	define 6 dias, 5 horas, 3 minutos e 4 segundos
T#12d3.5h	define 12 dias 3.5 horas

(com overflow):

T#61m5s	equivalente a T#1h1m5s

Datas e time of day

Tipo IEC	Descrição	Bits	Uso
DATE	Data absoluta	Dependente de implementação	Armazenar datas de calendário.
TIME_OF_DAY ou **TOD**	Time of day	Dependente de implementação	Armazenar real time clock.
DATE_AND_TIME ou **DT**	Data e time of day	Dependente de implementação	Armazenar datas e time of day.

Funções:

- Armazenar a data e tempo de eventos e alarmes.
- Escalonar eventos de controle baseado em data e hora. Por exemplo aquecer automaticamente um reator em determinada hora.
- Armazenar quando um sistema foi desenergizado e recuperou para calcular tempo de *down time*.

Representação:

Tipo de dado	Forma compacta	Forma extendida
DATE	D#	**DATE**#
TIME_OF_DAY ou **TOD**	**TOD**#	**TIME_OF_DAY**#
DATE_AND_TIME ou **DT**	**DT**#	**DATE_AND_TIME** #

Exemplos:

D#1994-06-10	10 de Junho de 1994
d#1995-01-13	13 de Janeiro de 1995
DATE#1993-10-15	15 de Outubro de 1993

TOD#10:10:30	10 horas, 10 minutos e 30 segundos.
TOD#23:59:59	1 segundo para meia noite
TIME_OF_DAY#05:00:00:56	0.56 segundos após as 5

DT#1993-06-12-15:36:55.40	12 de junho de 1993 às 15 horas, 36 minutos e 55.4 segundos
DATE_AND_TIME#1995-02-01-12:00:00	Meio dia de primeiro de fevereiro de 1995

Strings

Tipo IEC	Descrição	Bits	Uso
STRING	String de caracteres	Dependente de implementação	Armazenar texto

Utilização:

- Armazenar nome de bateladas: 'JOB_X32A3'
- Armazenar texto de mensagens para operador
- Armazenar mensagens para outros sistemas

Caracteres não printáveis devem ser inseridos na notação: $Código_hexadecimal_do_caracter

$0D$0A	Alimenta linha e retorna carro

Caracteres de controle pré definidos:

Código	Interpretação
$$	Caracter $
$'	Caracter '
$L ou $l	Alimenta linha
$N ou $n	New line
$P ou $p	Nova página
$R ou $r	Retorno de carro
$T ou $t	Tabulação

Exemplos:

'Fim da batelada AX45_65'
'Fim do relatório $N'
'$01$02$10 - 3 caracteres com código decimal 1, 2 e 16
" - string nulo

Bit strings

Tipo IEC	Descrição	Bits	Uso
BOOL	Bit string de 1 bit	1	Digital, estados lógicos
BYTE	Bit string de 8 bits	8	Informação binária
WORD	Bit string de 16 bits	16	Informação binária
DWORD	Bit string de 32 bits	32	Informação binária
LWORD	Bit string de 64 bits	64	Informação binária

Representação:

- Uma variável do tipo **BOOL** tem dois estados: **TRUE** e **FALSE**.
- Bit strings são definidos da mesma forma que uma variável inteira:
  146
  2#1101
  etc

Valor inicial de variáveis:

Tipo	Valor
Default	0
String	"
Datas	D#0001-01-01

Hierarquia de tipos básicos:

ANY
ANY_NUM
ANY_REAL
LREAL **REAL**

ANY_INT
SINT **INT** **DINT** **LINT** **USINT** **UINT** **ULINT** **UDINT**

ANY_BIT
BOOL **BYTE** **WORD** **DWORD** **LWORD**

STRING

ANY_DATE
DATE_AND_TIME **DATE** **TIME_OF_DAY**

Tipos de dados derivados:

Devemos usar as palavras chaves para construtores de tipos:

TYPE
PRESSURE: **REAL**;
END_TYPE;

Estruturas:

Uma estrutura é construída através de um construtor de estruturas:
STRUCT

END_STRUCT;

Vamos definir um novo tipo para designar um sensor de pressão que deverá conter:

1. O valor corrente da pressão como uma variável analógica.
2. O status do dispositivo: operante ou faltoso
3. A data da calibração
4. O máximo valor seguro de operação
5. O número de alarmes no período de operação corrente.

TYPE PRESSURE_SENSOR:
STRUCT
INPUT: PRESSURE;
STATUS: **BOOL**;
CALIBRATION: **DATE**;

```
        HIGH_LIMIT:         REAL;
        ALARM_COUNT:        INT;
    END_STRUCT;
END_TYPE
```

Enumerações:

Serve para nomear os diferentes status de um valor:

```
TYPE MODO_DO_DISPOSITIVO:
    (INICIALIZANDO, EXECUTANDO, STANDBY, FALHA);
END_TYPE
```

Faixas:

Pode-se definir faixa de valores de segurança para variáveis:

```
TYPE
    MOTOR_VOLTS: INT(-6..+12);
END_TYPE
```

O compilador deve assegurar que apenas valores dentro da faixa designada sejam atribuídos ás variáveis.

Arrays:

Pode ser constituído de múltiplos valores dos tipos de dados elementares ou derivados.

```
TYPE TANQUE_PRESSURE:
    ARRAY [1..20] OF PRESSURE;
END_TYPE

TYPE PÁTIO_DE_TANQUES:
    ARRAY [1..3, 1..4] OF TANQUE_PRESSURE;
END_TYPE
```

Inicialização de Variáveis:

Definição de valor default para um tipo de dados criado:

```
TYPE PRESSURE:REAL:=1.0;        (* Default 1 bar *)
```

END_TYPE

Reinicialização de valores de variáveis:

```
TYPE PRESSURE_SENSOR:
    STRUCT
        INPUT: PRESSURE:= 2.0;          (* substitui default *)
        STATUS: BOOL := 0;              (* Default 0 *)
        CALIBRATION:
              DATE:= DT#1994-01-10;     (* Data de instalação *)
        HIGH_LIMIT: REAL := 30.0;       (* Limite default *);
        ALARM_COUNT: INT :=0;           (* Sem alarmes *)
    END_STRUCT;
END_TYPE
TYPE MODO_DO_DISPOSITIVO:
      (INICIALIZANDO, EXECUTANDO, STANDBY, FALHA):=
      STANDBY;
END_TYPE
```

Inicialização de arrays:

```
TYPE
     VESSEL_PRESS_DATA: ARRAY[1..20] OF PRESSURE:= 10(1.1),
     5(1.3), 5(1.7);
END_TYPE
```

Redefinindo valores defaults de membros de estruturas:

```
TYPE GAS_PRESSURE_SENSOR:
      PRESSURE_SENSOR(PRESSURE:=4.0, HIGH_LIMIT:=40.0);
END_TYPE
```

Todos os tipos derivados de GAS_PRESSURE_SENSOR herdarão os novos valores default.

Variáveis:

Devem ser definidas no início das ***Program Organisation Units*** (POUs):

- ❑ Programas
- ❑ Blocos de Funções
- ❑ Funções

Uma lista de variáveis é declarada dentro da especificação do tipo de variável:

ESPECIFICAÇÃO_DE_TIPO
    A, B, C: **REAL**;
    IN1, IN2: **INT**;
END_ESPECIFICAÇÃO_DE_TIPO

Variáveis internas ou locais:

São declaradas dentro de uma POU

VAR
    VELOCIDADE_MEDIA: **REAL**;
    INIBE: **BOOL**;
END_VAR

Variáveis de entrada:

São as variáveis de entrada para uma POU e são supridas por uma fonte externa.

VAR_INPUT
    SetPoint: **REAL**;
    Max_Count: **SINT**;
END_VAR

Variáveis de saída:

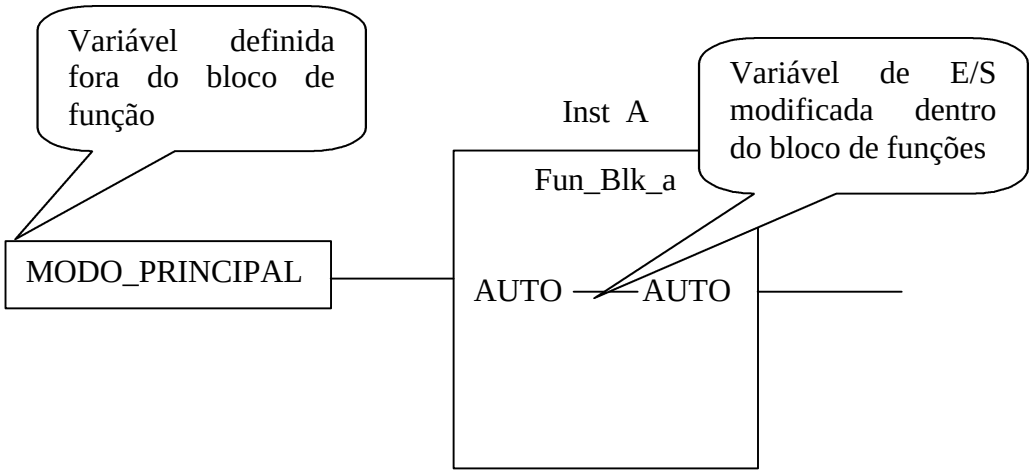
São as variáveis de saída de uma POU e fornecem valores que serão escritos em um dispositivo externo. São requeridas por programas e blocos de função, mas não por funções.

VAR_OUTPUT
    Message: **STRING**(10);
    Status: **BOOL**;
END_VAR

Variáveis de Entrada/Saída:

São as variáveis que agem tanto como entrada como saída podendo ser modificadas em uma POU. O valor de uma variável de entrada e saída é armazenado numa variável que é externa à POU. Funcionam como uma

passagem de parâmetro por referência em PASCAL. Também como em PASCAL são indicados para a passagem de grandes objetos como arrays.



TYPE MODO_DO_DISPOSITIVO:
(INICIALIZANDO, EXECUTANDO, STANDBY, FALHA):= STANDBY;
END_TYPE

VAR_IN_OUT
AUTO: MODO_DO_DISPOSITIVO;
END_VAR

Variáveis Globais:

Podem ser configuradas no nível de configuração, *resource* ou programa e acessadas por POUs por meio de uma declaração de variável externa.

VAR_GLOBAL
Velocidade_Linha: **REAL**;
Numero_Job: **INT**;
END_VAR

Variável Externa:

São declaradas dentro de POUs e propiciam acesso a variáveis globais definidas no nível de configuração, resource ou programa.

VAR_EXTERNAL
Velocidade_Linha: **REAL**;
Numero_Job: **INT**;
END_VAR

Variáveis de representação direta

São variáveis que permitem acessar diretamente as posições de memória dos CLPs. Todas começam com o caracter % seguido de duas letras:

A memória de um CLP pode ser dividida em três regiões lógicas:

Primeira Letra	Interpretação
I	Input: Recebe os valores das variáveis analógicas e discretas dos módulos de entrada.
Q	Output: Para armazenar os valores a serem escritos nos dispositivos externos.
M	Memória interna: armazena valores intermediários.

Segunda Letra	Interpretação
X	Bit
B	Byte (8 bits)
W	Word (16 bits)
D	Double word (32 bits)
L	Long word (64 bits)

Os demais dígitos representam a posição de memória e estabelecem uma hierarquia quando o endereço é constituído por números separados por pontos. O significado de cada campo é dependente de implementação.

Exemplos:

```
%I100          (* Memória de entrada bit 100 *)
%IX100         (* Memória de entrada bit 100 *)
%IW122         (* Memória de entrada palavra 122 *)
%IW10.1.21     (* Poderia representar:
                  Rack 10, Módulo 1, canal 21 *)
%QL100         (* Memória de saída long word 100  *)
%MW132         (* Poisção de memória 132 *)
```

Atributos de variáveis

RETAIN
Indica que as variáveis seguintes serão colocadas em memória retentiva, isto é que mantêm o seu valor em caso de perda de alimentação do CLP.

VAR_OUT RETAIN
PRODUCAO_ACUMULADA: **REAL**;

END_VAR

CONSTANT
Indica que os valores de uma lista de variáveis não podem ser modificados. Não pode ser utilizado com variáveis de saída.

```
(* Define velocidade inicial e relação entre engrenagens como constantes *)
VAR CONSTANT
        StartUp_Speed: REAL := 12.3;     (* m/s *)
        GearRatio: SINT := 12;
END_VAR
```

AT
É usado para fixar a posição de memória do CLP para alguma variável particular que possui um identificador.
Todas as variáveis que não tiverem o atributo AT serão endereçadas automaticamente pelo compilador.

```
(* Define que o array de varredura de dados irá começar na posição  *)
(* de memória word 10, e que as saídas digitais no bit 120          *)

VAR
        SCAN_DATA AT %IW10:          ARRAY[1..8] OF SINT;
        DIG_OUTPUTS AT%QX120:        ARRAY[0..15] OF BOOL;
END_VAR
```

Também é possível usar o atributo AT para declarar que determinados tipos de dados existem em certas posições de memória.

```
(* Declara que a posição de memória 100 é ocupada por um inteiro e que a
posição 210 é ocupada por um real *)

VAR
        AT %IW100: INT;
        AT %QD210: REAL;
END_VAR
```

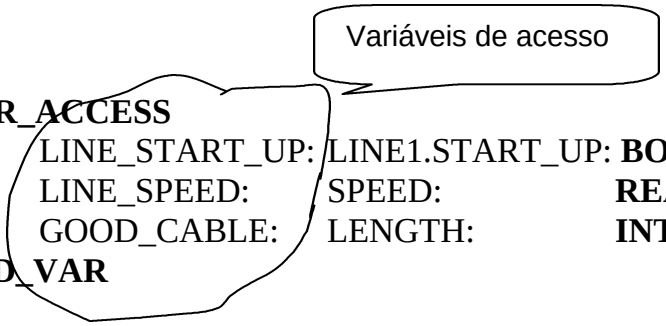
Variáveis de acesso

Podemos definir uma lista de variáveis que proporcionam referências através das quais dispositivos remotos ou outros programas IEC1131-3 remotos podem acessar certas variáveis. Estas variáveis proporcionam um caminho de acesso para variáveis nomeadas. Podem ser referenciadas:

- Variáveis de entrada ou saída de um programa

- Variáveis Globais
- Variáveis de representação direta

Estas variáveis podem ter os atributos: **READ_ONLY** ou **READ_WRITE**.




```
VAR_ACCESS
    LINE_START_UP: LINE1.START_UP: BOOL READ_WRITE;
    LINE_SPEED:    SPEED:          REAL READ_WRITE;
    GOOD_CABLE:    LENGTH:         INT  READ_ONLY;
END_VAR
```

LINE_START_UP, LINE_SPEED e GOOD_CABLE são variáveis de acesso que proporcionam caminhos de acesso remoto para LINE1.START_UP, um parâmetro de entrada de um programa chamado LINE1, e as variáveis globais SPEED e LENGTH.

Incialização de variáveis

Durante a declaração de uma variável podemos atribuir um valor de inicialização que sobrescreve o valor *default* definido pelo tipo de dados.

```
VAR
    Process_Runs: INT := 10;
    Max_tem: REAL := 350.0;
END_VAR
```

```
VAR
(* Define palavra na posição de entrada 100 para valor binário 1001 *)
    AT %I100: WORD:= 2#0000_1001;
END_VAR
```

```
VAR_OUTPUT
    (* Mensagem inicial para operador *)
    Message: STRING(21) := 'Operacional';
    (* Estabelece perfil de velocidade *)
    Speeds: ARRAY[1..4] OF REAL := 10.1, 2(20.5), 33.1;
END_VAR
```

```
VAR_GLOBAL
      SENSOR1: PRESSSURE_SENSOR( PRESSURE := 4.0,
      HIGH_LIMIT:= 50.0);
END_VAR
```

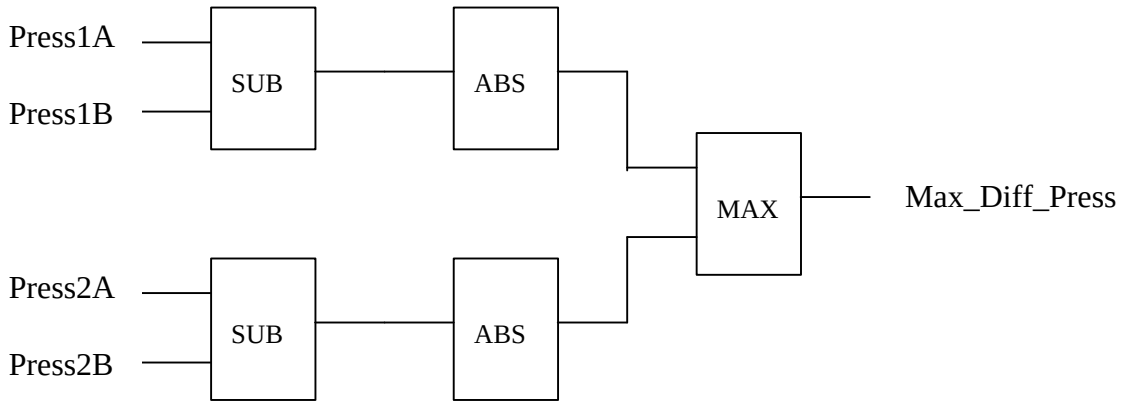
FUNÇÕES

Funções são elementos reutilizáveis de software, que quando executados com um certo conjunto de valores de entrada, sempre produzem um único valor como resultado.

Função em ST:

```
Max_Diff_Press := MAX(ABS (Press1A-Press1B), ABS(Press2A-Press2B));
```

Função em FBD:



- Funções não podem armazenar valores dentro de variáveis internas.
- Blocos de funções podem armazenar valores em variáveis internas e de saída.
- Funções podem ser usadas dentro da declaração de outras funções, blocos de função e programas.

Declaração de função:

```
FUNCTION AVE_REAL:REAL
   VAR_INPUT
        INPUT1, INPUT2: REAL;
   END_VAR
   AVE_REAL:= (INPUT1 + INPUT2) / 2;
END_FUNCTION
```

Funções para tipos de dados sobrecarregados

Na seção 3.7 os tipos de dados genéricos tais como **ANY**_NUM foram apresentados. É possível definir funções aplicáveis a vários tipos de dados. Exemplo: Função raiz quadrada:

```
VAR
        Small_Num, Small_sqrt: REAL;
        Large_num, Large_sqrt: LREAL;
END_VAR;

Small_sqrt := SQRT(Small_num);
Large_sqrt := SQRT(Large_Num);
```

Em compiladores que não suportem esta feature devemos criar funções específicas para cada tipo:
SQRT_REAL
SQRT_LREAL

Evocando Funções em Texto Estruturado:

```
VAR
        IN1: REAL := 10.0; (* Inicializa variáveis *)
        IN2: REAL:= 20.0;
        IN3: REAL := 4.0;
        AVE1, AVE2, AVE3: REAL := 0.0;
END_VAR

AVE1 := AVE_REAL (INPUT1:= IN1, INPUT2:= IN2);
AVE2 := AVE_REAL(INPUT2:= IN3 + 4.0, INPUT1 := 6.0);
AVE3 := AVE_REAL(INPUT1 := 4.0);  (* AVE3 = 2.0 *)
```

Observe que no terceiro exemplo, o parâmetro não utilizado recebeu o valor default 0.0

Muitas funções IEC 1131-3 não têm nomes de parâmetros formais:

```
VAR
        ALARM_MESSAGE: STRING(10);
END_VAR

VAR_EXTERNAL
```

```
        REPORT: STRING(15);
END_VAR

REPORT:= CONCAT('ALARM', ALARM_MESSAGE);
```

Conversão de tipos de dados

As linguagens IEC 1131-3 exigem checagem estrita de tipos de dados.

Quando os dados a serem operados são de tipos diferentes, devem ser utilizadas funções especiais para realizar a conversão de tipos:

<Tipo de dado de entrada>**_TO_**<Tipo de dado de saída>

```
VAR
        COUNT: INT;
        RAMP_RATE: REAL;
        STATUS: WORD;
        DISPLAY_VAL: STRING(16);
END_VAR

(* Converte Inteiro para Real *)
        RAMP_RATE := INT_TO_REAL(COUNT);

(* Converte de inteiro para palavra *)
        STATUS := INT_TO_WORD(COUNT);

(* Converte REAL ponto flutuante para string usando representação *)
(*numérica normal *)
        DISPLAY_VAL := REAL_TO_STRING(RAMP_RATE);
```

Números em ponto flutuante são convertidos para inteiro arredondando-se a parte decimal para o inteiro mais próximo (IEC559).

```
 2.6    → 3
-1.5    → -2
```

Na conversão para string, o valor será convertido para sua representação normal:

```
12      →  '12'
154.001 → '154.001'
```

Convertendo valores BCD

```
VAR
        VALUE: INT;
        BCD_VALUE: WORD := 2#0001_0100_0010; (* BCD 142 *)
END_VAR

(* Convert BCD value to integer *)
VALUE := BCD_TO_INT(BCD_VALUE);      (* value = 142 *)
```

Truncando valores floating point

```
VAR
        Shaft_rpm: LREAL;
        Pulse_count:  INT;
END_VAR

Shaft_rpm:= 3200.45;
(* Truncar para produzir valor inteiro 3200 *)
Pulse_count := TRUC(Shaft_rpm);
```

Detecção de erros

Quando convertendo valores, pode acontecer do tipo destino não suportar o tipo fonte, como por exemplo na conversão de um valor **REAL** muito grande para **SINT**. Neste caso, o PLC deve detectar o problema e fornecer uma mensagem de erro.

Funções Numéricas

Nome da função	Tipo de dado	Descrição
ABS	**ANY_NUM**	Valor absoluto
SQRT	**ANY_REAL**	Raiz quadrada
LN	**ANY_REAL**	Logaritmo neperiano
LOG	**ANY_REAL**	Logaritmo base 10
EXP	**ANY_REAL**	Exponencial
SIN	**ANY_REAL**	Seno
COS	**ANY_REAL**	Coseno
TAN	**ANY_REAL**	Tangente
ASIN	**ANY_REAL**	Arco de seno
ACOS	**ANY_REAL**	Arco coseno
ATAN	**ANY_REAL**	Arco tangente

Funções equivalentes a operadores ST

Funções extensíveis: aceitam número variável de argumentos
Funções não extensíveis: aceitam dois argumentos

Funções Extensíveis			
Nome da Função	**Tipo de Dado**	**operador ST**	**Descrição**
ADD	**ANY_NUM**	+	Adição
MUL	**ANY_NUM**	*	Multiplicação

Exemplo:

Fault_Count := ADD(Dev1, Dev2, AB_34, AB_32, AX_32);
Total_Revs := MUL(Gear1, Gear2, Gear3, 1200);
As funções acima são equivalentes às seguintes instruções em Texto Estruturado:

Fault_Count := Dev1 + Dev2 + AB_34 + AB_32 + AX_32;
Total_Revs := Gear1 * Gear2 * Gear3 * 1200;

Funções Não Extensíveis			
Nome Função	**Tipo de Dado**	**operador ST**	**Descrição**
SUB	**ANY_NUM**	-	Subtração
DIV	**ANY_NUM**	/	Divisão
MOD	**ANY_INT**	**MOD**	Módulo
EXPT	**ANY_REAL**	**	Exponencial
MOVE	**ANY**	:=	Assignment

Exemplos:

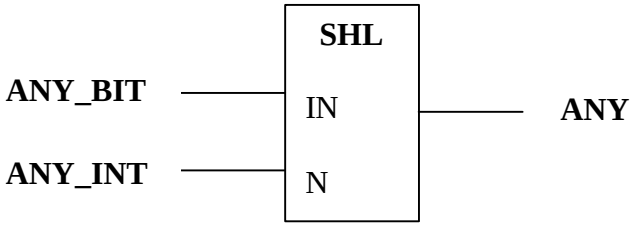
A := **DIV**(12, 3); (* A = 4 *)
A := **DIV**(14, 5); (* A = 2 *)
A := **DIV**(-4, 3); (* A = -1 *)

A operação de módulo só é válida com operandos inteiros.
A := **MOD**(12, 3); (* A = 0 *)
A := **MOD**(14, 5); (* A = 2 *)
A := **MOD**(-4, 3); (* A = -1 *)

Funções de bit strings

Proporcionam operação de shift para bit-strings:

Resultado := **SHL**(IN := Bitstring_De_Entrada, n:= Bits_A_Deslocar);



Nome da função	Tipo de dado	Descrição
SHL	**ANY_BIT**	Shift Left
SHR	**ANY_BIT**	Shift Right
ROR	**ANY_BIT**	Rotate Right
ROL	**ANY_BIT**	RotateLeft

E x e m p l o :

```
VAR
        t_8     : BYTE;
        G_8     : BYTE;
END_VAR


t_8 := 2#0011_0101;
G_8 := SHL (t_8, 4);          (*  G_8 = 2#0101_0000 *)
```

Funções booleanas bitwise

Nome da função	Tipo de dado	Símbolo	Descrição
AND	**ANY_BIT**	&	Result:= I1 & I2 &…
OR	**ANY_BIT**	>=1	Result:= I1 **OR** I2 **OR** …
XOR	**ANY_BIT**	=2k+1	Result := I1 **XOR** I2 **XOR** …
NOT	**BOOL**		Result := **NOT** I1

Exemplos de funções booleanas e expressão equivalente em ST

Funções booleanas:

Confirm:= **AND** (SW1, SW2, AB_1, AB_2, AX_2);
Trip_1 := **OR**(OverTemp1, OverTemp2, UnderPress);

Expressões em ST:

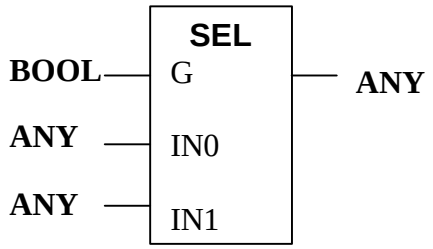
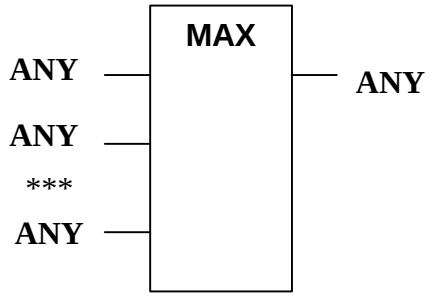
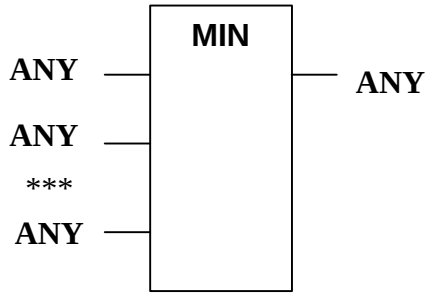
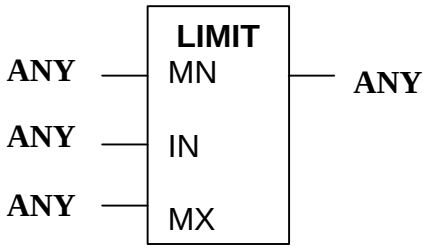
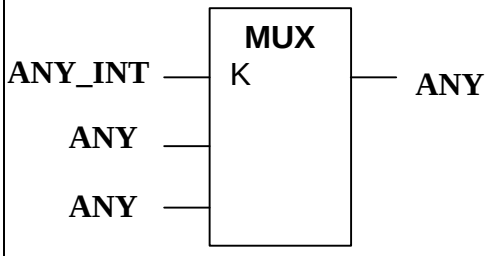
Confirm:= SW1 & SW2 & AB_1 & AB_2 & AX_2;
Trip_1 := OverTemp1 **OR** OverTemp2 **OR** UnderPress;

```
VAR
        Mask: WORD := 16#FF01;
        Status: WORD := 16#0A02;
        X_54: WORD;
END_VAR

X_54 := Mask XOR Status;   (* X_54 = 16#F503 *)
```

Funções de seleção:

Permite selecionar um valor de um conjunto de valores dados de acordo com algum critério.

```
(* Se flag é TRUE A = 230.0 senão A= 120.0 *)
A:= SEL(G:= Flag, IN0 := 120.0, IN1 := 230.0);

(* Selecione a temperatura Máxima *)
TempMax := MAX(TemA, TempB, TempC, TempD);

(* O tempo de enxarque deve ser limitado entre 1 e 4 horas *)
SoakTime := LIMIT(MN:= T#2h, IN := JobTime, MX := T#4h);

(* Selecione valor do próximo canal de entrada *)
value := MUX(K:= ChanNo, IN0:= Chan0, IN1:=Chan1, IN2:=Chan3);
```

Nome da Função	Tipo de dado	Forma Gráfica	Descrição
SEL	**ANY**		**Seleção** Se G= **TRUE** THEN Result:= IN1 ELSE Result:= IN0
MAX	**ANY**		**Maximum** Result = máximo de todos valores de entrada
MIN	**ANY**		**Minimum** Result = mínimo de todos valores de entrada
LIMIT	**ANY**		**Limit** Resultado é o valor de IN limitado pelos valores MN e MX.
MUX	**ANY**		**Multiplexer** Resultado é o valor da entrada selecionada pelo valor de K

Funções de comparação:

Funções de comparação:

- Comparam dados do mesmo tipo
- Sempre retornam **BOOL**
- Podem ser usadas com qualquer tipo de dados

Nome da função	Símbolo	Descrição
GT	>	Maior que Resultado := IN1> IN2
GE	>=	Maior ou igual que Resultado := IN1 >= IN2
EQ	=	Igualdade Resultado := IN1= IN2
LE	<=	Menor que Resultado := IN1 <= IN2
LT	<	Menor que Resultado := IN1 < IN2
NE	<>	Diferente de Resultado := IN1 <> IN2

Exemplos:

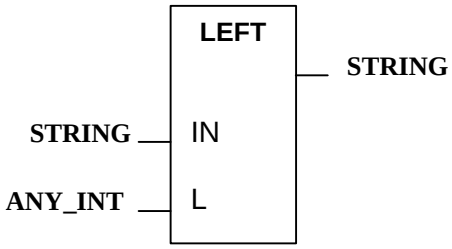
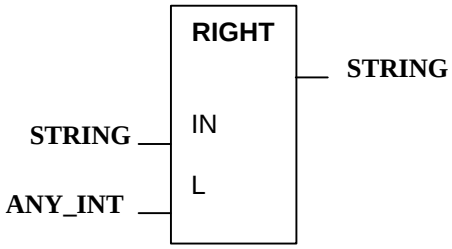
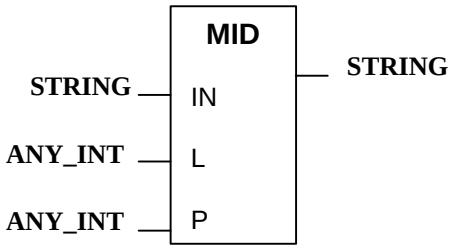
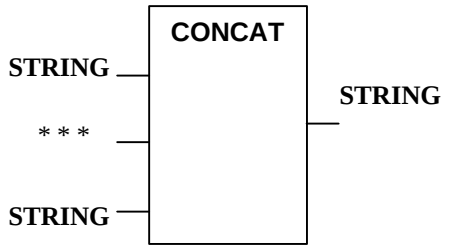
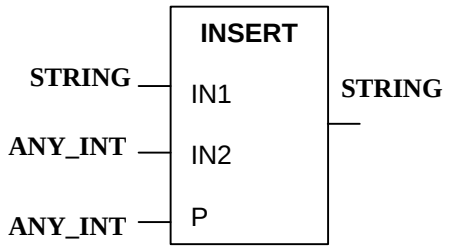
(* Se velocidade1 é menor que velocidade2 então saída = **TRUE** *)
Output_1 := LT(Speed1, Speed2);

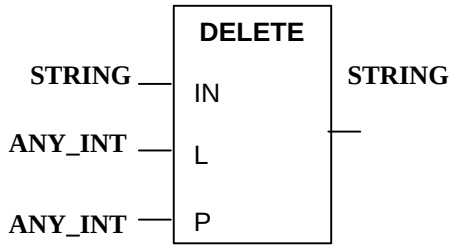
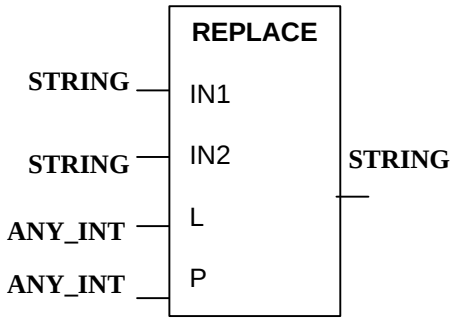
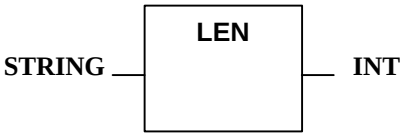
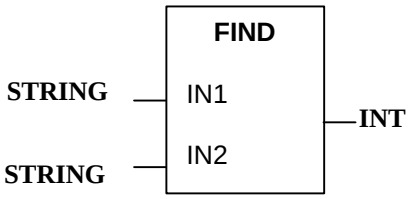
(* Comparação usando Texto estruturado *)
Output_1 := Speed1 > Speed2;

(* Verifique que as pressões da caldeira estão em ordem decrescente *)
(* de P1 para P4 *)
PressOk := GT(P1, P2, P3, P4);

(* Em TE *)
PressOK := P1 > P2 & P2 > P3 & P3 > P4;

Funções de string de caracteres:

Nome da Função	Tipo dado	Forma Gráfica	Descrição
LEFT	**ANY**		**Extrai string à esquerda** Resultado é o string formado pelos L caracteres mais à esquerda do string IN.
RIGHT	**ANY**		**Extrai string à direita** Resultado é o string formado pelos L últimos caracteres mais a direita do string IN.
MID	**ANY**		**Extrai string do meio** Extrai L caracteres do string In começando na posição P.
CONCAT	**ANY**		**Concatena strings** Resultado é o string formado pela concatenação dos strings de entrada. A função é extensível.
INSERT	**ANY**		**Insert string** O resultado é formado pela inserção do string IN2 a P caracteres do inicio do string IN1.

DELETE	**ANY**		**Deleta string** O resultado é formado pela deleção de L caracteres do string IN1 a P caracteres do inicio.
REPLACE	**ANY**		**Substitui string** O resultado é formado substituindo L caracteres do string IN1, começando na posição P, com caracteres do string IN2.
LEN	**ANY**		**Length** Calcula o comprimento do string de entrada.
FIND	**ANY**		**Find string** Resultado é a posição onde o string In2 foi encontrado a partir do inicio de IN1. Se o string não foi encontrado, retorna 0.

Observação:
Pelo padrão IEC, o primeiro caracter do string está na posição 1.

E x e m p l o s :

```
VAR
    Recipe_Spec: STRING(12) := 'Recipe_X_141';
    Recipe: STRING(14);
    Job_Code: STRING(3);
    Batch: STRING(20);
    BatchId: INT := 7;
END_VAR
```

```
(* Extrai código da receita 141 *)
Job_code := RIGTH(IN:= Recipe_Spec, L :=3);

(* Cria nome da receita Recipe_A7X_141'*)
Recipe:= REPLACE(IN1:= Recipe_Spec, IN2:='A7X', L:=1, P:=8);

(* Constrói a descrição do batch como 'Recipe_A7X_141_7' *)
Batch := CONCAT(Recipe, '_', INT_TO_STRING(BatchId));
```

Funções para manipulação de tempo e datas

SOMA: ADD

IN1	IN2	Result
TIME	**TIME**	**TIME**
TIME_OF_DAY	**TIME**	**TIME_OF_DAY**
DATE_AND_TIME	**TIME**	**DATE_AND_TIME**

- Somando uma duração (**TIME**) a um time_of_day produzimos um time_of_day posterior.

SOMA: SUB

IN1	IN2	Result
TIME	**TIME**	**TIME**
DATE	**DATE**	**TIME**
TIME_OF_DAY	**TIME**	**TIME_OF_DAY**
TIME_OF_DAY	**TIME_OF_DAY**	**TIME**
DATE_AND_TIME	**TIME**	**DATE_AND_TIME**
DATE_AND_TIME	**DATE_AND_TIME**	**TIME**

- Subtraindo uma data de outra data produzimos um duração (**TIME**).
- Subtraindo uma duração de um time_of_day produzimos um time_of_day anterior.

SOMA: MUL e **DIV**

IN1	IN2	Result
TIME	**ANY_NUM**	**TIME**

E x a m p l o :

```
VAR
        ProcessTime: TIME := T#2h;
        JobTime:  TIME;
        scale: REAL:= 1.5;
END_VAR

(* Define a hora do job para 3 horas *)
JobTime := MUL(processTime, scale);

(* Equivalente em TE *)
JobTime := processTime * scale;
```

CONCAT

IN1	IN2	Result
DATE	**TIME_OF_DAY**	**DATE_AND_TIME**

E x e m p l o :

```
VAR
        startDate:    DATE := DATE#1994-03-19;
        alarmTime: TIME := TIME_OF_DAY#13:15:00;
        timestamp: DATE_AND_TIME;
END_VAR

(* define time stamp como sendo 1994-03-19-12:15:00 *)
timeStamp := CONCAT(startDate, alarmTime);
```

Funções de conversão:

DATE_AND_TIME_TO_TIME_OF_DAY
Extrai time_of_day de um valor composto: date e time_of_day.

DATE_AND_TIME_TO_DATE
Extrai date de um valor composto: date e time_of_day.

E x e m p l o :

```
VAR
      event: DATE_AND_TIME:= DT#1995-03020-12-:15:00;
      eventDate: DATE;
      eventTime: TIME_OF_DAY;
END_VAR
```

(* Extrai time_of_day como 12:15: 00 *)
eventTime:= **DATE_AND_TIME_TO_TIME_OF_DAY**(event);
(* Extrai a data como 19995-03-20 *)
eventDate := **DATE_AND_TIME**R**_TO_DATE**(event);

Funções para tipo enumeração:

Nome da função	Símbolo ST	Descrição
SEL		Seleciona um de dois valores enumerados dependendo do valor de uma variável booleana.
MUX		Seleciona um de vários valores enumerados dependendo do valor de uma variável inteira.
EQ	=	Testa a condição de igualdade de dois valores enumerados do mesmo tipo.
NE	<>	Testa de dois valores enumerados são diferentes.

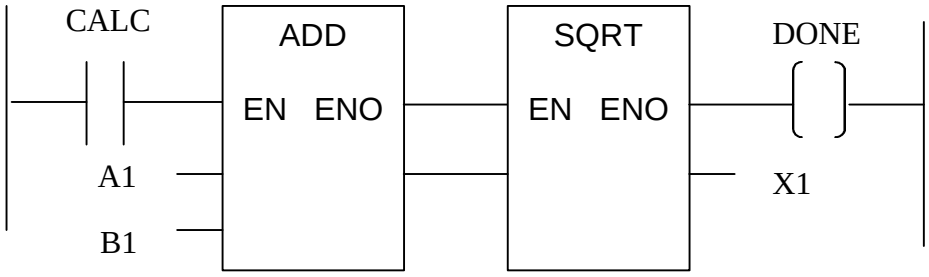
Controle de execução de funções:

Se a função é usada nas linguagens Ladder Diagram (LD) ou Diagrama de Bloco de Função (FBD), podemos controlar quando uma função será executada pela entrada EN (*Enable*).
Cada função nestas linguagens também tem uma saída digital extra denominada ENO (*Enable Output*) que é definida como **TRUE** quando a função é completada com sucesso.

Nas demais linguagens, a entrada EN e a saída ENO também existem, mas não são visíveis. Os seus valores podem entretanto serem testados dentro do corpo das funções.

É comum se encadear a saída ENO de uma função com a entrada EN da outra para garantir que a cadeia só produzirá um resultado correto, quando todas as etapas estiverem corretas.

A figura seguinte ilustra este encadeamento. Quando CALC=**TRUE**, ADD é executado usando os valores correntes de A1 e B1. Se ADD completa corretamente, ENO é feito igual a **TRUE**. Isto habilita o bloco de raiz quadrada. Se SQRT executa corretamente, EN0 de SQRT vai para **TRUE** e o resultado correto é produzido em X1.



Não é necessário declarar as variáveis EN e ENO dentro de uma função.

Estas variáveis são definidas implicitamente como:

```
VAR_INPUT
      EN: BOOL := 1;
END_VAR
VAR_OUTPUT
      ENO: BOOL;
END_VAR
```

Exemplo: Definindo EN0 em outras linguagens

Usando Lista de instruções:

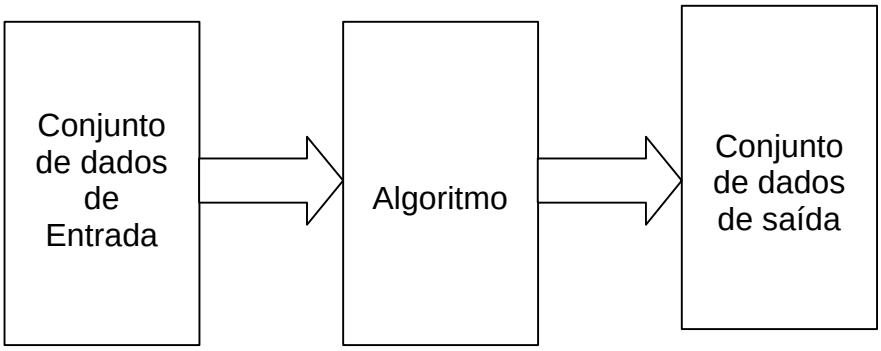
```
LD    Count
GT    100    (* Se é  maior do que 100 ?      *)
ST    ENO    (* Então define ENO               *)
```

Usando Texto Estruturado:

```
ENO := (Count > 100);
```

Blocos de Funções

É uma categoria de Unidade de Organização de Programa (POU).. Permite um algoritmo específico ou conjunto de ações a serem aplicados em um conjunto de dados para produzir um novo conjunto de dados de saída.



São usados para: algoritmos PID, Contadores, Geradores de rampa, Filtros, etc.

O bloco de funções possui variáveis:

- De entrada
- De saída
- Internas ou temporárias.

Uso de blocos de função

- Bloco de funções permite a persistência de dados.
- Apenas as variáveis de entrada e saída de um bloco de função são acessíveis externamente.
- Uma instância de bloco de função só é invocada de uma rede de blocos conectados ou por uma chamada de um programa em ST ou IL.
- Uma instância de bloco de função pode ser acessada por outro FBD ou por um diagrama Ladder.

A norma IEC 1131-3 define uma série de blocos de função padrões.

Declarando instâncias de blocos de função:

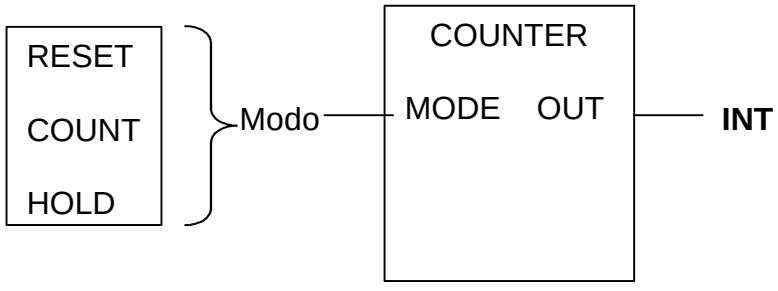
Instâncias de blocos de função podem ser declaradas usando a mesma construção de variáveis. Instâncias de blocos de funções só podem, ser definidas dentro de um programa ou definição de blocos de função.

Vamos supor que SPEED_RAMP E PRESS_MONITOR são funções previamente definidas. Vamos dar um exemplo de declaração de instâncias para estes blocos.

```
VAR
    Line1_ramp, Line2_ramp: SPEED_RAMP;
END_VAR

(* Dados com persistência retidos na partida à quente *)
VAR RETAIN
    Press_X32, Press_X54: PRESS_MONITOR;
END_VAR

(* Bloco de função global *)
VAR_GLOBAL
    Main_Press: PRESS_MONITOR;
END_VAR

(* Referência a uma instância de bloco de função *)
(* Passado em outra POU *)
VAR_INPUT
    PressMon: PRESS_MONITOR;
END_VAR
```

Exemplo de bloco de função:

Vamos criar um bloco de função para implementar um contador.



O contador possui três modos de funcionamento:
RESET: saída é zero
COUNT: Saída é incrementada de 1 a cada execução.
HOLD: a saída é congelada no último valor.

O bloco se divide em:

- Definições (var de entrada, saída e internas) e
- Corpo do bloco (algoritmo)

Não é permitido referência a variáveis que façam referência direta a endereços absolutos de CLPs (variáveis diretamente representadas).

As definições dentro de um bloco de função podem conter instâncias de outros blocos de função e variáveis externas.

```
TYPE
      ModeType: (RESET, COUNT, HOLD) := RESET;
END_TYPE

FUNCTION_BLOCK COUNTER
      (* Define referência externa *)
      VAR_INPUT
            MODE: ModeType := RESET;
      END_VAR
      VAR_OUTPUT
            OUT: INT := 0;
      END_VAR

      (* Define o Algoritmo *)
      IF MODE = RESET THEN
        OUT := 0;
      ELSIF MODE = COUNT THEN
            OUT := OUT + 1;
      END_IF;

END_FUNCTION_BLOCK
```

Introduz definição de bloco de função

Definições: Lista de entradas, saída e declarações de var internas.

Corpo do bloco: Algoritmo expresso em: ST, FBD, LD, SFC ou IL

Encerra definição de bloco de função

Uma instância da função contador pode ser declarada dentro de outra POU como por exemplo outro bloco de funções.

Exemplo: Texto Estruturado

```
PROGRAM COUNT1
      VAR_INPUT
        InputMode: ModeType;
      END_VAR

      VAR_OUTPUT
```

```
  Max_Count: INT;
END_VAR

VAR
  C1: COUNTER;
END_VAR

(* Chama bloco de funções C1 com o modo definido pelo valor *)
(* de InputMode, um parâmetro de entrada para o programa *)

C1(MODE:= InputMode);

(* Atribui a saída para uma variável *)
Max_count := C1.OUT;

END_PROGRAM
```

Parâmetros de entrada/saída de um bloco são acessados como se pertencessem a uma estrutura.

PROGRAMS

Um programa é a maior forma de POU e pode ser declarada no nível de resource.

Programas podem conter instâncias de blocos de função, mas não de outros programas.

Definindo o tipo programa

Lay out:

PROGRAM

<lista de entradas/ saídas e variáveis internas>
<corpo do programa> em qualquer linguagem IEC

END_PROGRAM

Exemplo:

```
PROGRAM Fermentador
    VAR_INPUT  (* Entradas do programa *)
        Codigo_Reagente: INT;
        Esteriliza: BOOL;
        Periodo_fermentacao: TIME;
    END_VAR

    VAR_OUTPUT (* Saídas do programa *)
        Producao: REAL:
        Status:WORD;
    END_VAR

    VAR  (* variáveis internas e blocos de função *)
        pG_Loop, Temp_Loop: PID;
        Fase: INT := 1;
    END_VAR

    (* Corpo não mostrado no programa, mas
     *pode ser ST, FBD, LD, SFC ou IL
     *)

END_PROGRAM
```

Declarando instâncias de programas

Programas podem ser declarados em **RESOURCES**.

E x e m p l o s :

PROGRAM Line1: Fermentador;
PROGRAM CT1, CT2: COUNT1;

A declaração também pode conter conexões entre entradas e saídas do programa e variáveis declaradas fora do programa.

E x e m p l o :

PROGRAM Line1: Fermentador (Codigo_Reagente := A1,
Esteriliza : = A2,
Periodo_Fermentacao := FTIME,
Producao => AJ_43, Status => KX56);

A instância de programa Line1 irá ser executada:

- As variáveis de resource ou globais A1, A2 e FTIME suprirão os valores para as entradas Codigo_Reagente, Estereliza e Periodo_Fermentacao respectivamente.
- As variáveis de saída Producao e Status são escritas nas variáveis AJ_43 e KX56.

Um programa pode ser colocado para ser executado sob o controle de uma task determinada.

PROGRAM Line2 **WITH** Slow_Task: Linha_de_Embalagem(
Velocidade := %IW21_10,
VelocidadeProduto => %QW33_81,
Amostrador1 **WITH** Fast_Task,
Seladora **WITH** Fast_Task);

- Line2 é uma instância de um programa do tipo Linha_de_Embalagem que roda sob o controle da task Slow_Task.
- A variável de entrada, velocidade, recebe um valor que vem da uma variável diretamente representada, que pode ser associada a uma entrada de um PLC.
- A variável de saída VelocidadeProduto é escrita em uma memória de referência do CLP que pode ser associada a um canal de saída do CLP.
- As instâncias de blocos de função Amostrador1 e Seladora1 são associadas à task Fast_Task.

RESOURCES

Resources são unidades de processamento capazes de executar programas IEC.

RESOURCE ResourceName **ON** Processador

< Declarações de variáveis Globais >
< Caminhos de acesso >
< Declaração de variáveis >
< Declaração de Programas >
< Definição de Tasks >

END_RESOURCE

TASKS

É um mecanismo de escalonamento que executa periodicamente ou em resposta a mudança de estado de alguma variável booleana.
A cada task podemos atribuir um período de execução e uma prioridade. 0 é a maior prioridade possível.

Na prática, um programa de CLP deve poder ser executado a diferentes velocidades, dependendo das necessidades temporais de cada processo controlado. Um forno por exemplo, que possui uma capacidade térmica muito grande, pode ser controlado por um algoritmo que executa uma vez a cada minuto. Já as funções de intertravamento de segurança de uma máquina ferramenta devem ser executadas a cada 5 ms.

Escalonamento premptivo e não preemptivo

Escalonamento não preemptivo

Neste tipo de escalonamento uma task sempre completa seu processamento, uma vez iniciado. Quando a task termina, a task de maior prioridade à espera do processador é escalonada. Caso haja empate na prioridade, a task que está esperando há mais tempo é escalonada. Após a sua execução, uma task só será escalonada, quando o seu período de execução se esgotar.

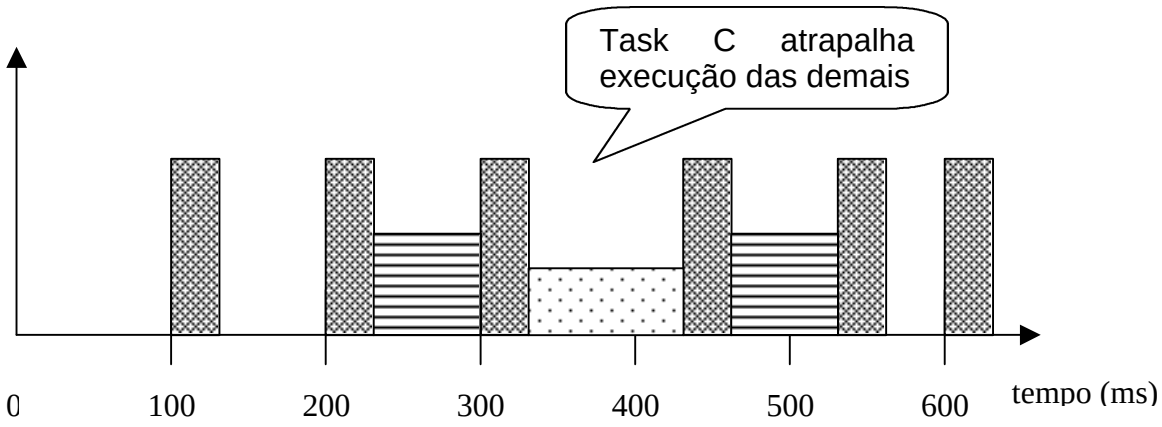
O intervalo entre a execução de tasks pode variar muito neste tipo de escalonamento. Uma task que demore um pouco mais em um loop, irá atrasar todas as demais tasks. Isto torna impossível prever com exatidão

quando uma determinada task será executada e caracteriza o sistema como não determinístico.

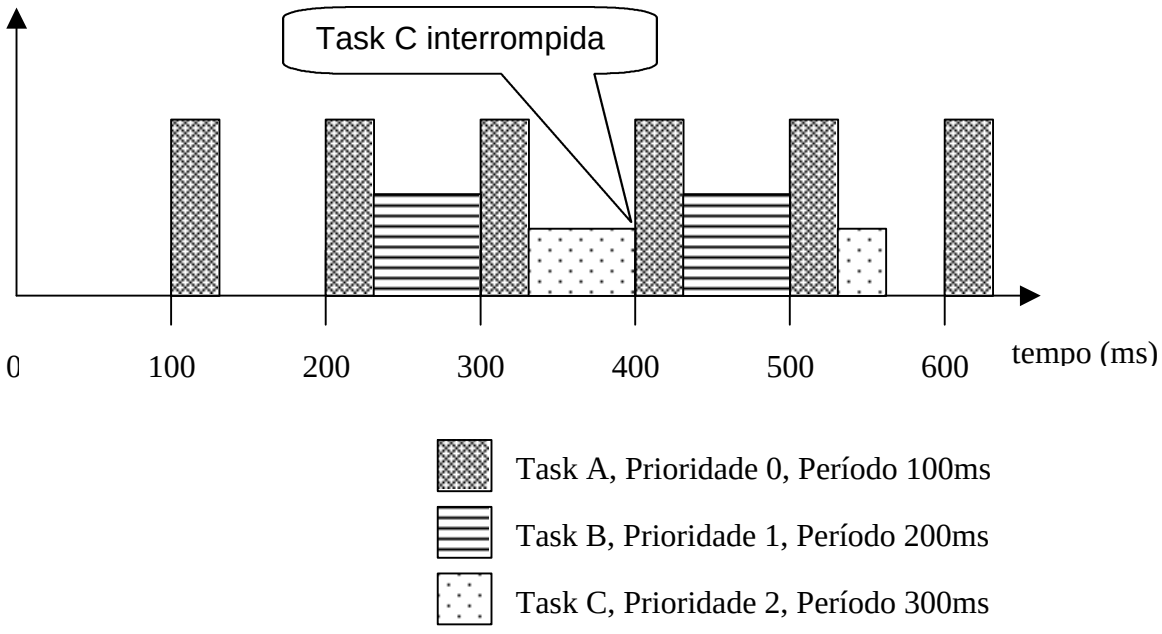
Escalonamento preemptivo

É recomendado para sistema que devem apresentar comportamento determinístico no tempo. Neste sistema quando o período de uma tarefa de maior prioridade vence, a tarefa em execução sofre preempção e a nova tarefa passa a executar imediatamente. Quando a tarefa de maior prioridade termina, a tarefa anterior volta a executar.

Escalonamento não preemptivo



Escalonamento preemptivo



Embora a norma não especifique como, os sistemas devem oferecer maneiras de dizer quando não será possível cumprir uma designação de períodos e prioridades e se possível ajudar o programador a otimizar a designação de prioridades.

Declaração de tasks

TASK NomeDaTask < (parâmetro = valor, …)>

Parâmetro da Task	Tipo de dados	Descrição
SINGLE	**BOOL**	variável booleana. A subida do pulso desta variável fará com que a task seja escalonada uma vez.
INTERVAL	**TIME**	Período de tempo entre execução da task. Se este parâmetro for definido e se **SINGLE** não estiver definido, a task será periódica.
PRIORITY	**UINT**	Prioridade da task. 0 é o maior nível de prioridade.

Exemplos:

TASK FAST-INTERLOCKS(**INTERVAL** := t#30ms, **PRIORITY** := 0);
TASK LOG**_TASK**(**SINGLE** := LogFlag, **PRIORITY** := 3);
TASK CONTROL**_TASK**(**INTERVAL** := t#500ms, **PRIORITY** :=1);

Associando programas e blocos de função a tasks

Programas e blocos de funções são associados a tasks através da palavra chave **WITH**:

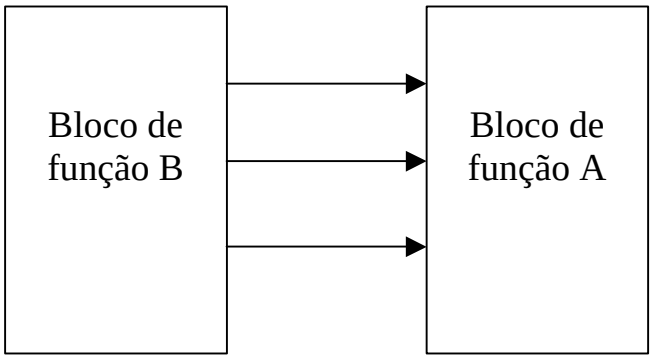
PROGRAM logger1 **WITH** LOG**_TASK**: log();

Esta associação deve ser feita com critério, para que a saída do processo sempre seja modificada dentro de uma janela de tempo, em resposta à modificações na entrada.

Blocos de função não associados a tasks, executam na mesma task do programa pai.

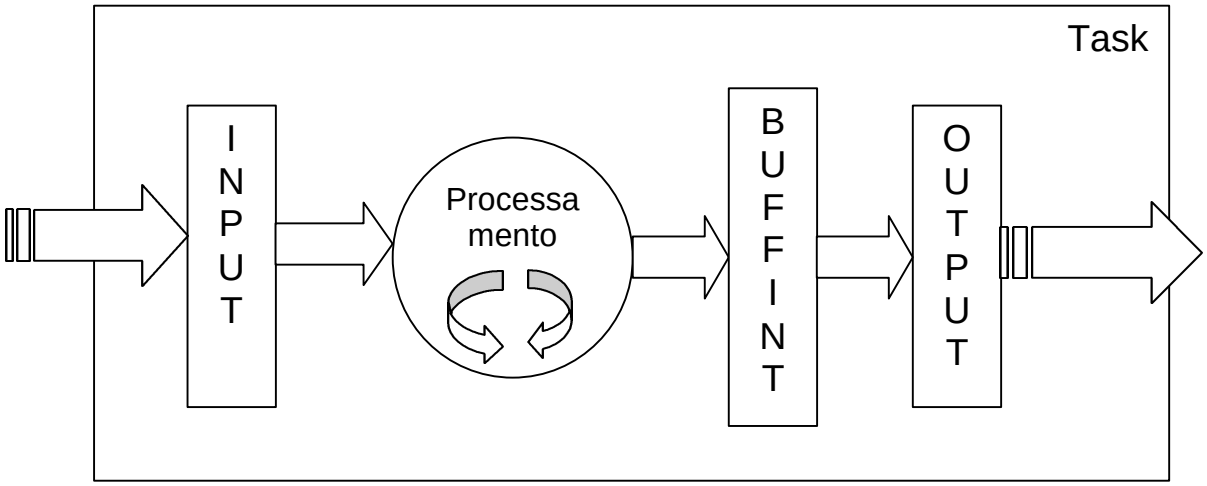
Qualquer programa que não estiver alocado a uma task será executado na prioridade mínima e será reescalonado assim que completar, isto é, ficará executando livremente em background.

Bloco de função e regras de execução de programas



Todos os valores de entrada de A, produzidos por B, devem ser produzidos pela mesma execução de B.

Se um certo número de blocos de função estiverem associados a uma task e todos recebem valores de saídas de um bloco B de uma outra task, os valores de entrada devem ser provenientes de uma mesma execução de B.



As entrada colhidas de outros blocos de função, são armazenadas num buffer de entrada. A tarefa executa seus programas e blocos de função. Os novos valores de saída são armazenados num buffer intermediário. Quando a tarefa termina, os valores são então copiados, do buffer intermediário para as saídas. Outras tasks não podem interromper o processamento durante a cópia de dados do buffer intermediário para a saídas.

Configurations

Uma *configuration* define o software para um PLC completo e irá incluir no mínimo um resource.
A *configuration* depende da configuração de hardware do CLP e do tipo de CPU empregada.

A configuration inclui:

- Recursos de processamento: CPUs.
- Endereços de memória para entradas e saídas.
- Capacidades do sistema (número máximo de tasks e velocidades de execução).

Observações:

- PLC tem dois processadores: PROC_386 e PROC_8044.
- O Resource Res1 contém uma instância de programa turbine1 que tem conexões de I/O para variáveis globais e memória de CLP.
- As instâncias de blocos de função loop1, ramp1 e io_scanner1 estão configuradas para rodar sob o controle de tasks periódicas.

- O Resource Res2 contém dois programas logger1 e diagnóstico 1 . logger1 executa sob o controle da task dirigida a eventos LOG_TASK. Ela só será executada quando a variável mudar de 0 para 1.
- O programa diagnose1 roda continuamente em *background*, em baixa prioridade.

- Várias variáveis de acesso estão disponíveis para que dispositivos remotos possam acessar certas variáveis.
  - UNIT1_ALARM é uma referência externa a ALRM_FLAG.
  - UNIT1_START proporciona acesso remoto para uma variável de entrada do programa Turbine1.
  - Usando mecanismos de comunicação, uma estação de operação remota pode por exemplo, escrever em UNIT1_START para iniciar a partida da turbina no programa turbine1 e monitorar alarmes através da variável UNIT1_ALARM.
  - Escrevendo em UNIT1_LOG logger1 será executado uma vez para atualizar dados históricos. Este dado pode ser lido pela estação de operação através da variável de acesso UNIT1_DATA.
- Variáveis dentro de um Resource são referenciadas por um nome hierárquico: Res1.Turbine1.Start.

Exemplo de configuração;

```
CONFIGURATION unit_l_config
    VAR_GLOBAL
        G_speed_setpoint            : REAL;
        G_runUp_Time                : TIME;
        G_Log_Event AT %M100        : BOOL;
        G_Log_data                  :ARRAY[1..100] OF INT;
    END_VAR

    RESOURCE Res1 ON Proc_386
        VAR_GLOBAL
            ALARM_FLAG      : BOOL;
        END_VAR
        TASK IO_SCAN_TASK
                    (INTERVAL := t#100ms, PRIORITY:= 0);
        TASK CONTROL_TASK
                    (INTERVAL := t#200ms, PRIORITY := 1);
        TASK PROG_TASK
                    (INTERVAL := t#400ms, PRIORITY := 2);

        PROGRAM turbine1 WITH PROG_TASK: turbine (
            speed_setpoint      := G_speed_setpoint,
            runUp_time          := g_runUp_Time,
            speed_pv            := %ID200,
            actuator_output     => %QW310,
            loop1               WITH CONTROL_TASK,
            ramp1               WITH CONTROL_TASK,
            io_scanner1         WITH IO_SCAN_TASK);
    END_RESOURCE
    RESOURCE Res2 ON Proc_8044
        TASK LOG_TASK
                    (SINGLE := G_Log_Event, INTERVAL := T#0ms);
        PROGRAM logger1 WITH LOG_TASK: log (
                                data => G_LOG_DATA);
        PROGRAM diagnose1: diagnostics;
    END_RESOURCE
    VAR_ACCESS
        UNIT1_START:    Res1.Turbine1.speed_pv: BOOL
                        READ_WRITE;
        UNIT1_ALARM:    Res1.alarm_flag: BOOL READ_ONLY;
        UNIT1_LOG:      G_Log_Event: BOOL READ_WRITE;
        UNIT1_DATA:     G_Log_Data: ARRAY [1..100] OF INT
                        READ_ONLY;
    END_VAR
END_CONFIGURATION
```

Leitura Complementar:

- The History of the PLC, Dick Morley, http://www.barn.org/FILES/historyofplc.html
- Your Personal PLC Tutor- Get the Book!, http://www.plcs.net/book.shtml
- Bonfatti, Monari, Sampieri, IEC1131-3 Programming Methodology, CJ International, 1997.

Exercícios:

1. Crie um tipo para enumerar os estados de um equipamento de controle.

2. Defina uma estrutura para conter todos os parâmetros de um:
   a) Equipamento
   b) Controlador PID

3. Marque **V**erdadeiro ou **F**also:

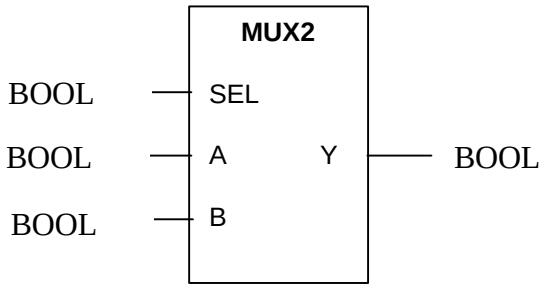
   ( ) Todas as funções bit string são aplicáveis nos seguintes tipos de dados: **BOOL**, **BYTE**, **WORD**, **DWORD**, **LWORD**.
   ( ) Blocos de função permitem a definição de variáveis com persistência.
   ( ) A criação ou cópia de um bloco de funções à partir de um tipo bloco de funções é chamado instanciação.
   ( ) Quando conjuntos de valores são passados de blocos de função sendo executados em uma task, para blocos de outra task, os valores devem ser produzidos na mesma execução da task.
   ( ) Em uma task, a cópia dos dados do buffer intermediário para as saídas, deve ser efetuada como uma instrução atômica.

4. Defina:
   - Função
   - Bloco de Função
   - Programa
   - Resource
   - Task
   - Configuration

5. Complete o programa abaixo para substituir os bits de 4 a 7 de X1 pelos bits de 4 a7 de X2:

```
VAR
    X1: INT := 2#1011_1100_1011_1101;
    X2: INT := 2#1100_1010_1110_0010;
    Mask: INT;
END_VAR
```

6. Desenvolver a função de bloco para sintetizar a função MUX2:



FUNCTION_BLOCK MUX
VAR_INPUT
SEL: BOOL;
A: INT;
B: INT;
END_VAR

VAR_OUTPUT

END_VAR

END_FUNCTION_BLOCK

7. Complete as colunas faltantes:

BatchName: **STRING**(20) := “ACIDO_ADIPICO_X34”;
Empresa: **STRING**(9) := “ÄCME”;
Name: **STRING**(20);
N: **INT**;

	Operação	**Name**	**N**
A	Name:=**LEFT**(IN:=BatchName, L:=4);		
B	Name:=**RIGHT**(IN:=BatchName, L:=5);		
C	Name:=**MID**(IN:=BatchName, P:=2, L:=5);		
D	Nam*e:*=**CONCAT**(Empresa, ‘_’, BatchName);		
E	N:= **LEN**(Empresa);		
F	Name:=**REPLACE**(IN1:=BatchName, IN2:=”OPS”, L:=2, P:=3);		
G	Name:=**INSERT**(IN1:=BatchName, IN2:=“DEF“, P:=3);		
H	Name:=**DELETE**(IN:=BatchName, L:=4, P:=3);		
I	N:=**FIND**(IN:=BatchName, IN2:=“Nam“);		