はじめてのST言語

―PLCの国際標準プログラミングのすすめ―

PLCopen Japan

はじめに

昨今のプログラマブルコントローラ（ＰＬＣ）は、高速演算や情報処理機能の追加、オープンネットワークへの対応などが進んでおり、高度で複雑な産業用機械・設備への適用から、生活関連機器、環境関連機器などに至るまで用途の拡大を見せています。これに伴い制御ソフトウェア（ソフトウェア）の複雑化、大規模化が進行しており、増大の一途を辿るソフトウェアのライフサイクルコストの抑制と、環境変化に迅速に対応する為の生産性の向上、品質向上の課題を解決する為、異なるベンダの製品を統合し易い「マルチベンダ環境」が望まれてきました。この課題解決への取り組みとして、PLCopen Japan では、PLC プログラミング言語の国際規格である IEC 61131-3 の普及を推進しています。

本書は、近年のトレンドである「システムの見える化」を促進する為、情報処理に適した構造化テキスト言語（ST 言語）に焦点を当て、図表を用いた豊富な例題（サンプル）による実用的な ST 言語の解説書です。また、習得レベルに応じて学習出来るよう、初級・中級・上級編に分類してあります。今までラダー言語しか使ったことのない制御技術者は勿論、C 言語に慣れ親しんだ組み込みソフトウェア技術者及び新卒者にとってもお役に立てるものと期待しております。

本書の発行が実現できましたことは、企画、執筆していただきました会員様、そして株式会社オートメレビュー社の皆様のご尽力、ご協力の賜物です。ここにあらためて深い謝意と敬意を表すると共に、読者の皆様におかれましては、傍らのバイブルとしてご活用いただきますようお願い申上げます。

2013 年 3 月

PLCopen Japan 代表幹事

松隈 隆志

本書に収録しましたサンプルプログラムは、読者様の責任においてご利用いただくことが可能です。但し、必ずしもご利用者様の環境における動作を保証するものではありません。サンプルによっては特定のベンダの PLC に無い命令を使用している場合や、異なるベンダ間で名称や仕様が異なる場合もございます。このような個所には注記を添えてありますので、ご使用の PLC の仕様を確認のうえ、本書を活用して下さい。

また、サンプルプログラムに関するサポートサービスは提供しておりませんので予めご了承願います。その他の本書に関する問い合わせにつきましては、下記まで電子メールでお願いいたします。

PLCopen Japan

info@plcopen-japan.jp

目　次

第6章 サンプルプログラム

付録

第1章　IEC 61131-3 と 5 種類のプログラミング言語

1.1　IEC 61131-3 って何？

IEC 61131 は国際規格団体の一つである IEC(International Electrotechnical Commission: 国際電気標準会議)によって策定された PLC(Programmable Logic Controller)システムに関する国際規格です。
本規格は PLC のハードウェアからプログラミングシステムまでを包括しており(表 1.1)、IEC 61131-3(JIS B 3503)には、プログラミング言語だけではなく、PLC ソフトウェアのプロジェクト作成の為の概念やガイドラインも提示されています。

表 1.1 IEC 61131

規格番号	タイトル	制定,改正,審議状況	規格番号	タイトル
IEC61131-1	Programmable controllers – Part 1 General information	1992年制定 2nd Edition 2003年5月	JIS B 3501	ﾌﾟﾛｸﾞﾗﾏﾌﾞﾙｺﾝﾄﾛｰﾗ 一般情報
IEC61131-2	Programmable controllers – Part 2 Equipment requirement and tests	1992年制定 3rd Edition 2007年7月	JIS B 3502	ﾌﾟﾛｸﾞﾗﾏﾌﾞﾙｺﾝﾄﾛｰﾗ 装置への要求事項及び試験
IEC61131-3	Programmable controllers – Part 3 Programming language	1993年制定 3rd Edition [SC65B]作業中	JIS B 3503	ﾌﾟﾛｸﾞﾗﾏﾌﾞﾙｺﾝﾄﾛｰﾗ ﾌﾟﾛｸﾞﾗﾐﾝｸﾞ言語
IEC61131-4 TR3	Programmable controllers – Part 4 User guideline	1995年制定 2nd Edition 2004年7月		
IEC61131-5	Programmable controllers – Part 5 Messaging service specification	2000年制定 1st Edition 2000年11月		
IEC61131-6	Programmable controllers – Part 6 Functional safety	[SC65B]制定作業中 2011年予定		
IEC61131-7	Programmable controllers – Part 7 Fuzzy control programming	2000年制定 1st Edition 2000年8月		
IEC61131-8 TR	Programmable controllers – Part 8 Guidelines for the application and implementation of programming languages	2000年制定 2nd Edition 2003年9月		

【IEC61131-3の誕生まで】
1977　GRAFCET(フランス)
　　DIN 40719, Function Charts(ドイツ)
1978　NEMA ICS-3-304,
　　Programmable Controllers (アメリカ)
1980　DIN 19239,
　　Programmable Controller (ドイツ)
1983　IEC65A(Sec)38,　Programmable Controllers
1985　IEC SC65A(Sec)49, PC Languages
1987　IEC848, Function Charts
1993　IEC1131-3

1.2　ユーザにとってのメリットは？

PLC システムエンジニアは、機械・装置の種類に応じて異なるベンダの PLC を同時に使う場合があります。その為、従来のエンジニアは幾つもの異なるプログラミングツールについて個々にトレーニングを受ける必要がありました。このような場合、IEC 61131-3 準拠の PLC(図 1.1)を使うことにより、仮に PLC ベンダが異なっていても、言語要素やプログラムの構造など“ルック・アンド・フィール”は似たような形に統一されているので、(ベンダ固有の特殊機能を除けば)プログラミング習得時間は大幅に削減され、プログラムの流用も容易になります。

また、IEC 61131-3 にはプログラミング言語として、従来のラダーダイアグラムを含む3種類のグラフィック言語(LD/FBD/SFC)と2種類のテキスト言語(IL/ST)が用意されており、エンジニアのスキルや適用用途に応じて最適な言語を選ぶことが出来るため、より効率的な開発が可能になります。

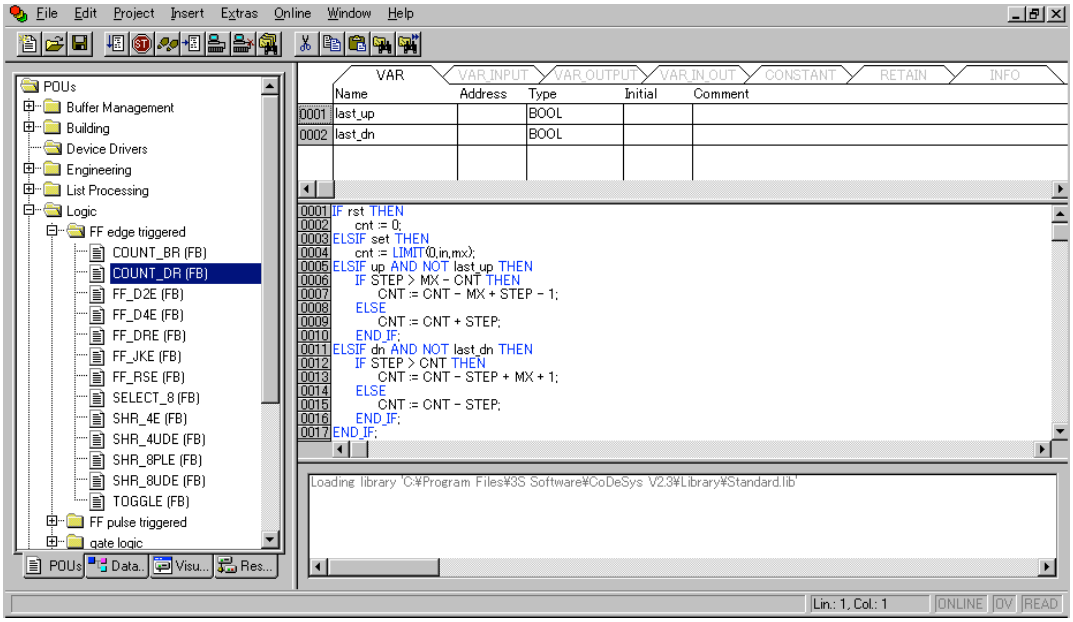

図 1.1 IEC 61131-3 準拠のツール例

1.3 プログラミング言語の特徴は？

1.3.1 LD（ラダー ダイアグラム：Ladder Diagram）

リレーシーケンス回路の置換えや従来の PLC に慣れているエンジニア向けのグラフィック言語であり、日本の制御システム開発において最も普及しているプログラミング言語です（図 1.2）。

LD はリレーシーケンス回路と同等なロジックを使用しており、I/O のインターロック処理など、ビットレベルの処理には向いていますが、システムが大規模且つ複雑になるほど機能単位での区分が難しく一本の巻物スタイルになってしまいます。その為、第3者にとっては解読が難しく、他のシステムへの流用や将来発生するであろう改造要求に対応し難いところが欠点です。

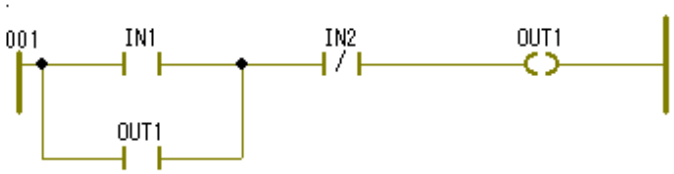

図 1.2 Ladder Diagram

1.3.2 FBD(ファンクション ブロック ダイアグラム:Function Block Diagram)

DCS(Distributed Control System)に慣れているエンジニア向けのグラフィック言語であり、計装分野を中心に使われています。最近では制御適用範囲の拡大や、プログラムの可読性が向上する為、PLC でも使われることが多くなりました。

図 1.3 のように、FBD 言語は電子部品(ファンクションと呼ばれる箱)とそれらを接続する配線により、あたかも電子回路を設計するようにプログラムを記述出来る為、「データの流れが一目見てわかる」という利点があります。ファンクションの左側にある変数は入力パラメータで、右側の変数が演算結果である出力パラメータです。

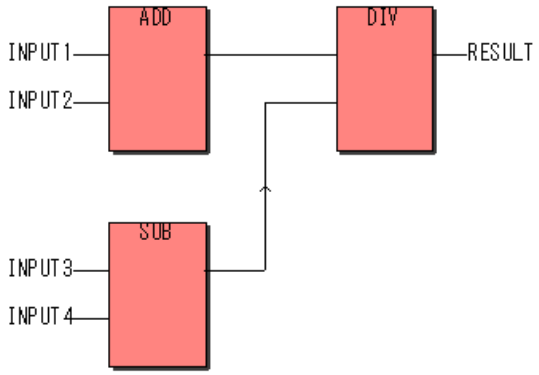

図 1.3 Function Block Diagram

1.3.3 SFC(シーケンシャル ファンクション チャート:Sequential Function Chart)

製造ラインなどの状態遷移を記述することに適したグラフィック言語です。SFC は演算機能や入出力機能を持たない為、厳密には言語と規定されていませんが、言語要素を併せ持つため本書では言語として扱います。

図 1.4 に図 1.2 の自己保持回路と等価な SFC プログラム例を示します。

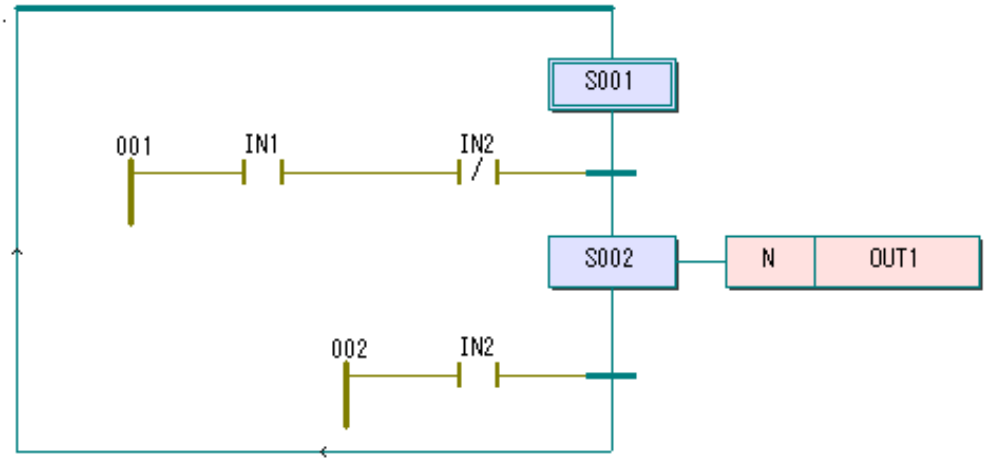

図 1.4 Sequential Function Chart

四角の箱の S001、S002 は“ステップ”と呼ばれ、工程の状態を示します。特に, 2 重四角の S001 から始まり、これを“初期ステップ”と呼びます。SFC 内では同時に1つだけのステップが実行され、実行状態にあることを“活性化”と呼びます。

ステップの下に位置する横棒は“トランジション”と呼ばれ、次のステップへ移行する為の遷移条件となっています。S001 から S002 への遷移条件は“IN1”が TRUE 且つ“IN2”が FALSE となることです。この条件が成立したときに S001 が非活性化されると同時に S002 が活性化されます。

S002 の右横の四角は“アクション”と呼ばれ、S002 で行う処理を記述します。S002 が活性化状態の時、“OUT1”に TRUE が出力されます。“OUT1”への TRUE 出力は、S002 の下に記述されたトランジション“IN2”が TRUE になるまで継続します。

“IN2”が TRUE になると、S002 は非活性化状態になり、最初の S001 が活性化状態となります。

このように、SFC では工程間の遷移条件や工程内の処理を明確に分けて記載出来るところが利点です。

1.3.4 IL(インストラクション リスト:Instruction List)

従来の PLC にあるニーモニックに相当する言語で、マイコンで言えばアセンブラのようなテキスト言語です。

アプリケーションの小型化や高速化には有効ですが、プログラミングの生産性やメンテナンス性に劣る為、使用する機会は減ってきていると思われます。

図 1.2 の自己保持回路を IL で記述すると図 1.5 のようになります。

```
LD    IN1
OR    OUT1
ANDN  IN2
ST    OUT1
```

図 1.5 Instruction List

1.3.5 ST(ストラクチャード テキスト:Structured Text)

PASCAL をベースに設計された構造化テキスト言語で、パソコンの高級言語に慣れ親しんだマイコンボードの開発者や C、C++などの教育を受けてきた新卒のエンジニアに向いています。

数値演算式の表現、ネスティングした条件分岐など、LD が苦手としている用途で効果を発揮します。仮に LD で多項式あるいは括弧付きの計算式を処理しようとした場合は、初めに二項式に分解し、次に分解された式の演算結果をそれぞれ一時的にメモリに格納してから、最後に繋げるといった処理が必要になってしまいます。このことは処理内容の可読性を著しく阻害しています。

図 1.6 は図 1.3 の FBD と同じ処理です。“INPUT1”と“INPUT2”を加算した値を“INPUT3”から“INPUT4”を引いた値で除算し、結果を“RESULT”に代入しています。

```
RESULT:= (INPUT1 + INPUT2 ) / (INPUT3 - INPUT4);
```

図 1.6 Structured Text

図 1.7 は欧州、アジア、米国における上記 5 言語の使用状況(普及度)を示しています。

欧州では全ての言語が普及しているのに対し、日本を含むアジア地域では LD、FBD が中心で、ST 言語はごく少数になっています。

PLCopen Japan では、IEC 61131-3 普及の鍵は、「ST 言語の解説、技術情報の少なさ」を解消することと考え、次章以降の全般にわたり、ST 言語の有効性について事例を混じえて解説することにいたしました。

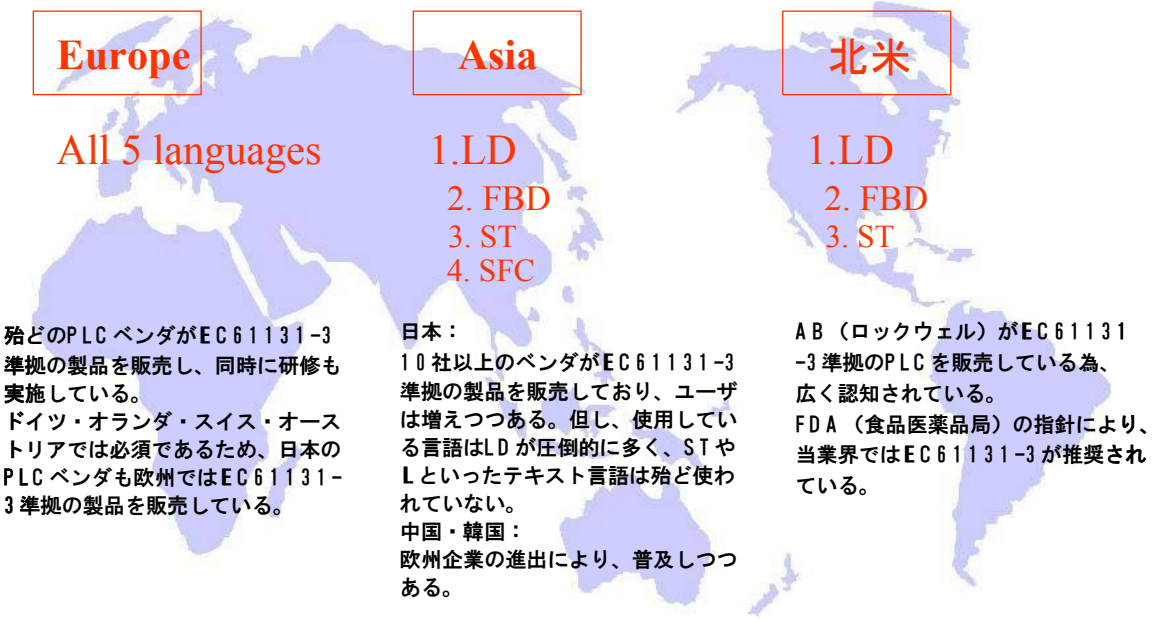

図 1.7 プログラミング言語の使用状況

第2章　初級編 「ST 言語で演算や文字列処理をしてみよう」

2.1 まず、計算式を書いてみよう

これまでのラダーでは、四則演算、浮動小数点演算を記述する場合には、PLC ベンダ独自の応用命令を組み合わせたプログラムが必要で、計算式の全体を直感的に把握しにくく、またデバッグも大変でした。これに対して ST で記述すると非常にかんたんに分かりやすくプログラミングすることができるようになります。

簡単な計算式を記述してみましょう。

```
Y := 2 * X + 10;
```

中学校で学習した式と同じで、X を 2 倍し、10 を加えたものを Y に代入するという処理になります。ただし、記述のルールとして以下の点を注意してください。

記述のルール）
・代入文は、「:=」　（イコールの記号の前にコロンが必要）
・計算式の最後はセミコロン「;」が必要
・掛け算は「*」、割り算は「/」記号で表します。
※　上記はすべて半角の記号です。

また、計算する順序（優先順位）を変更する場合は半角のカッコ“(”、“)”を使用します。以下の例では X に 10 を加えたものを 2 倍し、Y に代入することになります。

```
Y := 2 * (X + 10);
```

これも通常の計算式と同じルールです。ただし、記述する際に全角のカッコ“(”や、全角スペースはエラーになりますので注意が必要です。また、この例では X と Y という記号を使用していますが、これらは変数と呼ばれ、値や計算結果を一時的に記憶する場所になります。

記述のルール）
・変数名は、半角アルファベットの大文字、小文字を区別しません。
・　次の例は同じ変数を示します。 例）　abc，aBc，ABC

2.2 整数の切り上げ、切り下げ、四捨五入

それでは次に、四則演算についてもう少し見てゆきましょう。

```
Y := X / 10 * 10;
```

上記の式の意味は X を 10 で割り、10 掛けたものとなります。しかしながら、整数の計算と、少数点付きの実数の計算では結果が異なります。整数ならば、X が 10 より小さい場合は商が 0 になるため Y は 0 となります。また、X が実数の場合は Y=X が成立します。

X が整数の場合）	X が実数の場合）
Y := (2 / 10) * 10; ↓ Y := (0) * 10; ↓ Y := 0;	Y := (2 / 10) * 10; ↓ Y := (0.2) * 10; ↓ Y := 2;

整数と実数とで計算結果が異なるのであれば、これらの変数が整数か、実数かをあらかじめ宣言しておく必要がありそうです。とりあえず、本書では以下の表記で宣言し、明確にしておきましょう。INT は INTEGER の略で、整数を意味します。

変数名	データ型
X	INT
Y	INT

X と Y を整数と宣言した上で、再度、先ほどの式を見てみましょう。

```
Y := X / 10 * 10;
```

この式は、常に整数 X の 1 桁目を切り捨てた値が、Y に代入されることになります。では、次の式の結果はどうでしょうか？

```
Y := (X + 9) / 10 * 10;  (*整数の切り上げ*)
Z := (X + 5) / 10 * 10;  (*整数の四捨五入*)
```

Y には X の 1 の位を切り上げした結果が入り、Z は四捨五入の結果が入ることになります。わかりやすいように式の後ろにコメントをつけています。コメントは“(*”と“*)”の間に記述でき、自由な位置に挿入することができます。

2.3 積算時間(秒)から時･分･秒を求める

次に、設備の稼動時間など秒単位で計測した積算時間（秒）から、時・分・秒の表記に変換する例を見てみましょう。考え方は以下のようになります。

式①：時間の位は、積算時間（秒）を 3600 秒（=1 時間)の値で割った商となります。
式②：分の位は、積算時間（秒）を 3600 秒で割った余りを、再度 60 秒(=1 分)で割った商となります。
式③：秒の位は、式②の余りです。

これらを数式で表現すると、以下となります。入力された積算時間（秒）を時間、分、秒に変換します。式中の MOD（Modulo の略）は整数の割り算で“余り”を求めるための演算子です。

変数名	データ型
積算時間_秒	INT
時間	INT
分	INT
秒	INT

```
時間 := 積算時間_秒 / 3600;
分 := (積算時間_秒 MOD 3600) / 60;
秒 := (積算時間_秒 MOD 3600) MOD 60;
```

上記の例では、”積算時間_秒”が 100,000 秒であるとすると、計算式としては 27 時間 46 分 40 秒に変換されるはずです。さて、上記の式で本当に想定どおりの結果が得られるでしょうか？

まず、プログラムに以下の式を追加しただけではエラーになります。

```
積算時間_秒 := 100000; (* ← 追加した式 *)

時間 := 積算時間_秒 / 3600;
分 := (積算時間_秒 MOD 3600) / 60;
秒 := (積算時間_秒 MOD 3600) MOD 60;
```

なぜでしょう？

変数に代入できる数字の範囲に関係があります。変数はそれぞれ記憶領域のサイズがあり、そのサイズにより表現できる数字の範囲が決まっています。

ここでは整数を INT としましたが、もう少し厳密には 16 ビット（2 バイト）の記憶領域に格納される符号付きの整数を示しています。したがって、INT のデータ範囲は-32,768 から 32,767 となりますので、”積算時間_秒”には、±32,767 秒（約-9～9 時間の範囲）までしか表現できないことになります。
もう少し長い時間を表現するために、4 バイト整数を使ってみましょう。INT の 2 倍のデータサイズがあるので、Double INT (DINT)と表記します。DINT のデータ範囲は、-2^{31}～$+2^{31}-1$（-2,147,483,648～+2,147,483,647）となります。符合なしの 4 バイト整数は Unsigned DINT(UDINT)と表記し、0～2^{32}(4,294,967,296)までの値を表現することができます。2^{32} 秒を時間で換算すると約 1,193,046 時間（約 136 年）まで表現することが可能です。

ST では、データのサイズを変更する場合は、データ型を変更するだけでよい[1]ので、先ほどの整数を定義したテーブルを以下のように書き直せば、期待する計算結果が得られます。

変数名	データ型
積算時間_秒	UDINT
時間	UDINT
分	UDINT
秒	UDINT

データ型の表記ルールと、表現できる値の範囲について詳細な確認が必要な場合は、「02.11.1　データ型について」をご参照ください。

[1] 各 PLC ベンダの仕様により、該当変数のデータ型を変更しただけではエラーとなる場合があります。その場合は、プログラムに記述された各数値（定数）の前にデータ型の指定を挿入してください。
例）　UDINT#3600

第6章 サンプルプログラム

6.1 初級レベル

6.2 中級レベル

6.3 上級レベル

本章のサンプルプログラムは、必ずしもご利用者様の環境における動作を保証するものではありません。ご利用の際には、環境に合わせて修正いただき、十分なテストを実施してください。

数値の切り上げ、切り下げ、四捨五入 ～四則演算を使う①～

■サンプルプログラムの概要

入力された数値を1の位で切り下げ、切り上げ、四捨五入します。
下記の例では、“45”を入力すると、切り下げ値“40”、切り上げ値“50”、四捨五入値“50”を出力します。

■STサンプル

変数種別	変数名	データ型
入力変数	数値	DINT

変数種別	変数名	データ型
出力変数	切り下げ	DINT
	切り上げ	DINT
	四捨五入	DINT

```
(*
*********************************************************************************
STサンプルプログラム：切り下げ、切り上げ、四捨五入
********************************************************************************* *)

(* 切り下げ後の値の算出 *)
切り下げ := 数値 / 10 * 10;

(* 切り上げ後の値の算出 *)
切り上げ := ( 数値 + 9 ) / 10 * 10;

(* 四捨五入後の値の算出 *)
四捨五入 := ( 数値 + 5 ) / 10 * 10;
```

データ型指定が必要な場合の例）

```
(* 切り下げ後の値の算出 *)
切り下げ := 数値 / DINT#10 * DINT#10;

(* 切り上げ後の値の算出 *)
切り上げ := ( 数値 + DINT#9 ) / DINT#10 * DINT#10;

(* 四捨五入後の値の算出 *)
四捨五入 := ( 数値 + DINT#5 ) / DINT#10 * DINT#10;
```

トータル時間(秒)から時・分・秒を求める ～MOD関数を使う～

■サンプルプログラムの概要

入力された積算時間(秒)を時間、分、秒に変換します。
下記の例では、積算秒に100,000秒を入力すると、27時間46分40秒に変換されます。
ここでは、入力のデータ範囲を倍長符号なし10進数(UDINT)で指定していますので、
0～4,294,967,295の範囲で扱えます。

■STサンプル

変数種別	変数名	データ型
入力変数	積算時間_秒	UDINT

変数種別	変数名	データ型
出力変数	時間	UDINT
	分	UDINT
	秒	UDINT

```
(* ********************************************************************
STサンプルプログラム: 積算時間(秒)を時間・分・秒に変換する
******************************************************************** *)

時間 := 積算時間_秒 / 3600;
分 := (積算時間_秒 MOD 3600 ) / 60;
秒 := (積算時間_秒 MOD 3600 ) MOD 60;
```

データ型指定が必要な場合の例)

```
時間 := 積算時間_秒 / UDINT#3600;
分 := (積算時間_秒 MOD UDINT#3600 ) / UDINT#60;
秒 := (積算時間_秒 MOD UDINT#3600 ) MOD UDINT#60;
```