

TM246

# Structured Text



### **Wymagania**

Moduły szkoleniowe	TM210 – Podstawy Automation Studio TM213 – Automation Runtime TM223 – Diagnostyka w Automation Studio
Oprogramowanie	Automation Studio w wersji 3.0.90 lub nowszej
Urządzenie	Brak

## Spis treści

1 Wprowadzenie.....	4
1.1 Zawartość materiałów szkoleniowych.....	4
2 Informacje ogólne.....	5
2.1 Cechy tekstu strukturalnego.....	5
2.2 Funkcje edytora.....	6
3 Podstawowe elementy.....	8
3.1 Wyrażenia.....	8
3.2 Przypisanie.....	8
3.3 Dokumentacja kodu źródłowego: Komentarze.....	8
3.4 Kolejność operacji.....	9
3.5 Zastrzeżone słowa kluczowe.....	10
4 Grupy poleceń.....	11
4.1 Operacje na funkcjach Boole'a.....	11
4.2 Operacje arytmetyczne.....	12
4.3 Konwersja typu danych.....	13
4.4 Operatory porównawcze i decyzje.....	16
4.5 Instrukcja "CASE" - Maszyny stanu.....	19
4.6 Pętle.....	21
5 Funkcje, bloki funkcyjne i akcje.....	27
5.1 Wywoływanie funkcji i bloków funkcyjnych.....	27
5.2 Wywołanie akcji.....	29
6 Dodatkowe i pomocnicze funkcje.....	30
6.1 Wskaźniki i referencje.....	30
6.2 Preprocesor programów IEC.....	30
7 Funkcje diagnostyczne.....	31
8 Ćwiczenia.....	32
8.1 Ćwiczenie - Podnośnik pudeł.....	32
8.2 Ćwiczenie - Mieszadło do farb dyspersyjnych.....	33
9 Podsumowanie.....	35
10 Załącznik.....	36
10.1 Tablice.....	36
10.2 Rozwiązania do ćwiczeń.....	39

## 1 Wprowadzenie

Tekst strukturalny (Structured Text, ST) jest językiem programowania wysokiego poziomu. Jego podstawowa koncepcja opiera się na elementach języka BASIC, Pascal i ANSI C. Najważniejszą cechą tekstu strukturalnego są zrozumiałe konstrukty standardowe, które umożliwiają szybkie i wydajne programowanie aplikacji dla automatyki.



W kolejnych rozdziałach zapoznasz się z poleceniami, słowami kluczowymi i składnią języka tekstu strukturalnego (ST). Na prostych przykładach sprawdzisz znajomość wiedzy teoretycznej dotyczącej omawianego zagadnienia i łatwiej ją utrwalisz.

### 1.1 Zawartość materiałów szkoleniowych

Niniejszy moduł szkoleniowy zawiera ćwiczenia, które pomogą Ci nauczyć się podstaw programowania wysokiego poziomu w tekście strukturalnym.

- Nauczysz się edytora języka wysokiego poziomu i funkcji SmartEdit w Automation Studio.
- Nauczysz się podstawowych elementów wykorzystywanych w programowaniu w językach wysokiego poziomu, a także korzystania z komend ST.
- Nauczysz się korzystania z poszczególnych grup poleceń i funkcji arytmetycznych.
- Nauczysz się korzystać z porównania i operatorów Boole'a.
- Nauczysz się wywoływać elementy używane do kontroli przebiegu programu.
- Nauczysz się pracy z zarezerwowanymi słowami kluczowymi.
- Dowiesz się jakie są różnice pomiędzy akcjami, funkcjami i blokami funkcyjnymi a także w jaki sposób z nich korzystać.
- Dowiesz się jak korzystać z funkcji diagnostycznych dostępnych do programowania w językach wysokiego poziomu.

## 2 Informacje ogólne

### 2.1 Cechy tekstu strukturalnego

#### Informacje ogólne

ST to tekstowy język wysokiego poziomu, przeznaczony do programowania układów automatyki. Dzięki standardowym konstruktom, które zawiera, programowanie w nim przebiega szybko i w nieskomplikowany sposób. W ST wykorzystano wiele tradycyjnie przyjętych cech języków wysokiego poziomu, m.in. zmienne, operatory, funkcje i elementy sterowania przebiegiem programu.

Tekst strukturalny (ST) wchodzi w skład standardu IEC.<sup>1</sup>

#### Cechy:

**Tekst strukturalny odznacza się przede wszystkim następującymi cechami:**

- Język tekstowy wysokiego poziomu
- Programowanie strukturalne
- Łatwe w użyciu standardowe konstrukty
- Szybkie i wydajne programowanie
- Elastyczność i przejrzystość
- Podobieństwo do języka Pascal
- Zgodny z normą IEC 61131-3

#### Możliwości

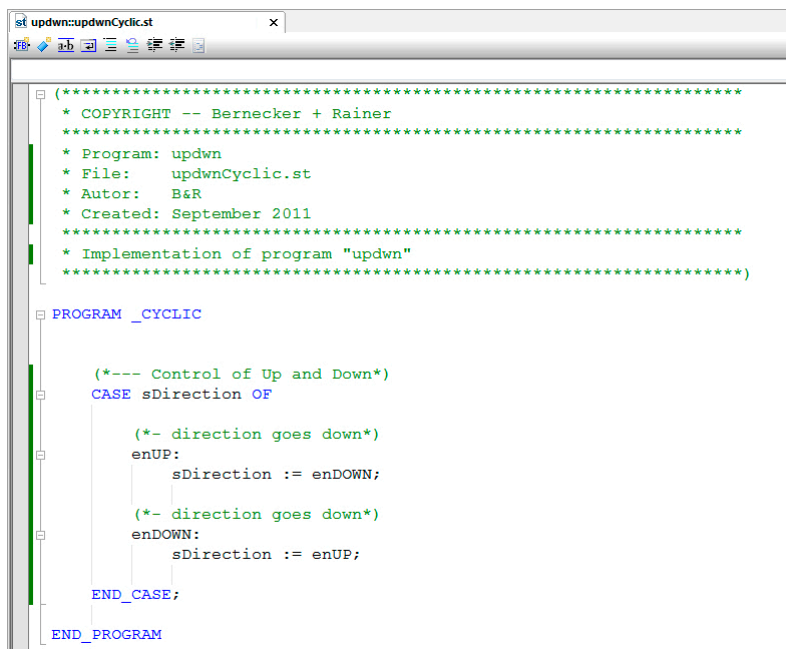
**Automation Studio obsługuje następujące funkcje:**

- Wejścia i wyjścia cyfrowe oraz analogowe
- Operacje logiczne
- Logiczne wyrażenia porównawcze
- Operacje arytmetyczne
- Decyzje
- Sekwensery krokowe
- Pętle
- Bloki funkcyjne
- Zmienne dynamiczne
- Wywołanie akcji
- Zintegrowane funkcje diagnostyczne

<sup>1</sup> Standard IEC 61131-3 jest to jedyny stosowany na całym świecie standard dla języków programowania, wykorzystywany w programowalnych sterownikach logicznych. Oprócz tekstu strukturalnego, również inne języki programowania takie jak język sekwencyjny (SFC), logika drabinkowa (LD), lista instrukcji (IL) i diagram bloków funkcyjnych (FBD) są zdefiniowane w normie.

## 2.2 Funkcje edytora

Edytor języka jest edytorem tekstowym uwzględniającym szeroki wachlarz funkcji uzupełniających. Polecenia i słowa kluczowe są wyróżniane kolorami. Obszary w polu edycji można rozwijać i zwijać. Dla zmiennych i konstruktów przewidziano funkcję automatycznego uzupełniania treści (SmartEdit).



```
st updown:updownCyclic.st x
{*****
 * COPYRIGHT -- Bernecker + Rainer
 *****/
 * Program: updown
 * File:   updownCyclic.st
 * Autor:  B&R
 * Created: September 2011
 *****/
 * Implementation of program "updown"
 *****/
PROGRAM _CYCLIC

(*--- Control of Up and Down*)
CASE sDirection OF

  (*- direction goes down*)
  enUP:
    sDirection := enDOWN;

  (*- direction goes down*)
  enDOWN:
    sDirection := enUP;

END_CASE;

END_PROGRAM
```

Numer 1: Program użytkownika w edytorze ST

### Edytor odznacza się następującymi funkcjami i cechami:

- Rozróżnianie wielkości znaków (małe i duże litery)
- Autouzupełnianie (SmartEdit, <CTRL> + <SPACE>, <TAB>)
- Wstawianie snippetów kodu i zarządzanie nimi (<CTRL> +q, k)
- Rozpoznawanie odpowiadających sobie par nawiasów
- Rozwijanie i zwijanie konstruktów (zarysy)
- Wstawianie komentarzy do bloków
- Rozpoznawanie adresów URL
- Znaczniki wierszy zmodyfikowanych



[Programming \ Editors \ Text editors](#)

[Programming \ Editors \ General operations \ SmartEdit](#)

[Programming \ Editors \ General operations \ SmartEdit \ Code snippets](#)

[Programming \ Programs \ Structured Text \(ST\)](#)

Edytor deklaracji zmiennych wspiera inicjalizację zmiennych, stałe, typy danych i struktury użytkownika. Ponadto, zastosowane zmienne można udokumentować wykorzystując do tego komentarz. Edytory deklaracji również wspierają funkcję SmartEdit.



Programming \ Editors \ Table editors \ Declaration editors

### 3 Podstawowe elementy

W tej sekcji opisano szczegółowo podstawowe elementy ST. Wyjaśnione są między innymi wyrażenia, przypisania, stosowanie komentarzy w kodzie programu i zarezerwowane słowa kluczowe.

#### 3.1 Wyrażenia

Wyrażenie jest konstruktem, który po obliczeniu zwraca określoną wartość. Wyrażenia składają się z operatorów i argumentów operacji. Argument operacji może być zmienną, stałą lub wywołaniem funkcji. Argumenty operacji łączy się za pomocą operatorów (3.4 "Kolejność operacji"). Każde wyrażenie musi być zakończone średnikiem (";"), niezależnie czy jest to wywołanie funkcji czy przypisanie.



```
b + c
(a - b + c) * COS(b);
SIN(a) + COS(b);
```

Table 1: Przykładowe wyrażenia

#### 3.2 Przypisanie

Przypisanie składa się ze zmiennej, przedstawionej po lewej stronie i do której przydzielony jest wynik obliczenia lub wyrażenia po prawej stronie za pomocą operatora ":=". Każde przypisanie musi kończyć się znakiem średnika, ";".



```
Result := ProcessValue * 2;      (*Result ← (ProcessValue * 2) *)
```

Table 2: Przypisanie przebiega od lewej do prawej

Wykonanie powyższego wiersza kodu powoduje, że wartość "Result" będzie dwa razy większa od wartości zmiennej "ProcessValue".

#### Dostęp do bitów zmiennych

Tworząc przypisania można także operować wyłącznie poszczególnymi bitami. W tym celu wstaw znak kropki (".") za nazwą zmiennej. Wówczas dostęp będzie możliwy według numeru bitu, poczynając od 0. Numery bitów można również zastąpić stałymi.



```
Result := ProcessValue.1;
```

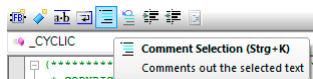
Table 3: Dostęp do drugiego bitu w "ProcessValue"

#### 3.3 Dokumentacja kodu źródłowego: Komentarze

Komentarze są bardzo ważną częścią kodu źródłowego. Opisują one treść kodu, dzięki czemu jest on bardziej czytelny i zrozumiały. Dzięki komentarzom ty i inne osoby możecie zrozumieć czym jest program, nawet gdy został napisany dawno temu. Komentarze nie są kompilowane i nie wpływają w żaden sposób na wykonywanie programów. Komentarze należy bezwzględnie zamykać w nawiasach z gwiazdkami: (\*treść komentarza\*).

Istnieje dodatkowa forma komentarzy, wprowadzana za pomocą "//". W edytorze można zaznaczyć kilka wierszy jednocześnie i dodać do nich komentarz za pomocą ikony na pasku narzędzi. Ta metoda jest rozszerzeniem wskazanej w obowiązującej normie IEC.





Numer 2: Komentowanie bloku tekstu


	<b>Komentarz jednowierszowy</b>	<code>(*This is a single line of comment.*)</code>
	<b>Komentarz wielowierszowy</b>	<code>(*     These     are several     lines of comment. )</code>
	<b>Komentarz znakiem "//"</b>	<code>// This is a general // text block. // It contains comment.</code>

Table 4: Warianty komentarzy




Programming \ Editors \ Text editors \ (Un)commenting selections

Programming \ Structured software development \ Program layout

## 3.4 Kolejność operacji

Istnieje wiele operatorów, a co za tym idzie, zasadniczą kwestią jest priorytet każdego z nich. Priorytet operatorów ma ogromne znaczenie dla rozwiązywania wyrażeń.

Wyrażenia są rozwiązywane rozpoczynając od operatora o najwyższym priorytecie. Operatory o tym samym priorytecie są wykonywane w porządku od lewej do prawej strony wyrażenia.

Operator	Składnia	
Nawiasy	<code>()</code>	Najwyższy priorytet
Wywołanie funkcji	Wywołanie (Argument)	
Wykładnik	<code>**</code>	
Negacja	<code>NOT</code>	
Mnożenie, dzielenie, operator modulo	<code>*</code> , <code>/</code> , <code>MOD</code>	
Dodawanie, odejmowanie	<code>+</code> , <code>-</code>	
Porównanie	<code>&lt;</code> , <code>&gt;</code> , <code>&lt;=</code> , <code>&gt;=</code>	
Porównania równości	<code>=</code> , <code>&lt;&gt;</code>	
Boole'owskie AND	<code>AND</code>	
Boole'owskie XOR	<code>XOR</code>	
Boole'owskie OR	<code>OR</code>	Najniższy priorytet

Wyrażenia są rozwiązywane przez kompilator. Na poniższych przykładach przedstawiono, jak nawiasy wpływają na wynik wyrażenia.



### Rozwiązanie wyrażenia bez nawiasów:

```
Result := 6 + 7 * 5 - 3;          (* Result equals 38 *)
```

Najpierw wykonywane jest mnożenie, a następnie dodawanie. Odejmowanie wykonywane jest na końcu.

### Rozwiązanie wyrażenia z nawiasami:

```
Result := (6 + 7) * (5 - 3);      (* Result equals 26 *)
```

Wyrażenie jest wykonywane w kolejności od lewej do prawej. Operatory w nawiasach są wykonywane przed mnożeniem, ponieważ nawiasy mają wyższy priorytet. Wynik zatem zależy od nawiasów.

## 3.5 Zastrzeżone słowa kluczowe



Programowanie wymaga, aby każda zmienna odpowiadała określonym warunkom umownego nazewnictwa. Ponadto istnieją zastrzeżone słowa kluczowe, które są jako takie rozpoznawane przez edytor i wyświetlane innym kolorem. Nie można ich stosować jako zmiennych.

Biblioteki "OPERATOR" i "AslecCon" są standardowymi elementami nowego projektu. Funkcje, które zawierają, są funkcjami IEC interpretowanymi jako słowa kluczowe.

Norma IEC określa również literały liczb i ciągów znaków. Umożliwia to przedstawianie liczb w różnych formatach.



[Programming \ Structured software development \ Naming conventions](#)

[Programming \ Standards \ Literals in IEC languages](#)

[Programming \ Programs \ Structured Text \(ST\) \ Keywords](#)

[Programming \ Libraries \ IEC 61131-3 functions](#)

## 4 Grupy poleceń

Niżej przedstawione grupy poleceń są podstawowymi konstruktami programowania wysokiego poziomu. Konstrukty te mogą być elastycznie łączone i zagnieżdżane jeden w drugim.

**Można je zaklasyfikować według poniższych kategorii grup poleceń:**

- [4.1 "Operacje na funkcjach Boole'a"](#)
- [4.2 "Operacje arytmetyczne"](#)
- [4.4 "Operatory porównawcze i decyzje"](#)
- [4.5 "Instrukcja "CASE" - Maszyny stanu"](#)
- [4.6 "Pętle"](#)

### 4.1 Operacje na funkcjach Boole'a

Operatory Boole'owskie mogą być stosowane do łączenia wartości zmiennych na poziomie binarnym. Rozróżnia się operatory NOT, AND, OR i XOR. Argumenty nie muszą mieć typu danych BOOL. Jednak kolejność operacji powinna być brana pod uwagę. Nawiasy mogą być używane.

Symbol	Operacja logiczna	Przykłady
NOT	Negacja binarna	<code>a := NOT b;</code>
AND	Logiczne AND	<code>a := b AND c;</code>
OR	Logiczne OR	<code>a := b OR c;</code>
XOR	Wyłączne OR	<code>a := b XOR c;</code>

Table 5: Przegląd spójników boole'owskich

Tablica prawdy dla tych operacji przedstawia się następująco:

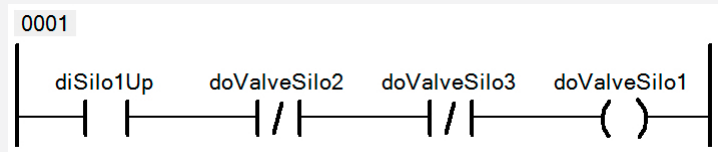
Wejście		AND	OR	XOR
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Table 6: Tablica prawdy dla spójników boole'owskich

Operacje na funkcjach Boole'a można łączyć ze sobą w dowolny sposób. Kolejne zestawy nawiasów poprawiają czytelność programu, a także gwarantują prawidłowe rozwiązywanie wyrażenia. Jedynymi możliwymi wynikami wyrażenia są: "TRUE" (logiczne 1) lub "FALSE" (logiczne 0).



## Spójnik boole'owski - Porównanie logiki drabinkowej z tekstem strukturalnym



Numer 3: Łączenie styków normalnie rozwartych z normalnie zwartymi

```
doValveSilo1 := diSilo1Up AND NOT doValveSilo2 AND NOT doValveSilo3;
```

Table 7: Implementacja ze spójnikami boole'owskimi

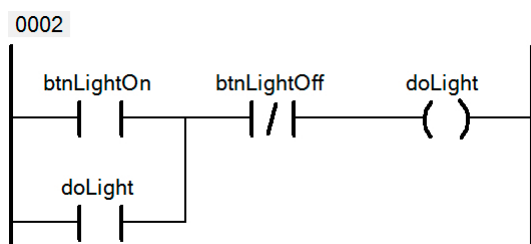
Nawiasy są tu zbędne, ponieważ "NOT" ma wyższy priorytet niż "AND". Niemniej powinno się używać nawiasów, aby program był bardziej przejrzysty.

```
doValveSilo1 := (diSilo1Up) AND (NOT doValveSilo2) AND (NOT doValveSilo3);
```

Table 8: Implementacja ze spójnikami boole'owskimi i z nawiasami

### Ćwiczenie: Sterowanie oświetleniem

Wyjście "DoLight" powinno być WŁĄCZONE po naciśnięciu przycisku "BtnLightOn" i pozostawać w tym stanie do naciśnięcia "BtnLightOff". Rozwiąż to zadanie operacjami na funkcjach Boole'a.



Numer 4: Przelącznik dwupozycyjny (on/off), przekaźnik z zatraskiem

## 4.2 Operacje arytmetyczne

Kluczową zaletą języków programowania wysokiego poziomu jest to, iż łatwo radzą sobie z operacjami arytmetycznymi.

### Przegląd operacji arytmetycznych

Tekst strukturalny uwzględnia podstawowe operacje arytmetyczne stosowane w aplikacjach. Również tutaj kolejność operacji powinna być brana pod uwagę, gdy są one używane. Na przykład mnożenie jest wykonywane przed dodawaniem. Pożądane zachowanie może zostać osiągnięte poprzez użycie nawiasów.

Symbol	Operacja arytmetyczna	Przykład
:=	Przypisanie	a := b;

Table 9: Przegląd operacji arytmetycznych

Symbol	Operacja arytmetyczna	Przykład
+	Dodawanie	<code>a := b + c;</code>
-	Odejmowanie	<code>a := b - c;</code>
*	Mnożenie	<code>a := b * c;</code>
/	Dzielenie	<code>a := b / c;</code>
MOD	Modulo, reszta dzielenia całkowitoliczbowego	<code>a := b MOD c;</code>

Table 9: Przegląd operacji arytmetycznych

Typ danych zmiennych i wartości zawsze ma krytyczne znaczenie dla obliczeń. Wynik obliczany jest po prawej stronie wyrażenia i dopiero następnie przydzielany do odpowiadającej mu zmiennej. Wynik zależy od przyjętych typów danych i składni (notacji). Przedstawia to poniższa tabela.

Wyrażenie / Składnia	Typy danych			Wynik
	Wynik	Argument operacji 1	Argument operacji 2	
<code>Result := 8 / 3;</code>	INT	INT	INT	2
<code>Result := 8 / 3;</code>	REAL	INT	INT	2.0
<code>Result := 8.0 / 3;</code>	REAL	REAL	INT	2.66667
<code>Result := 8.0 / 3;</code>	INT	REAL	INT	*Błąd

Table 10: Wykonana przez kompilator, konwersja niejawna

Wartość wynikowa "\*Błąd" odpowiada następującej wiadomości o błędzie kompilatora: "Error 1140: Incompatible data types: Cannot convert REAL to INT.". Błąd ten występuje, ponieważ nie można przydzielić wyrażenia do tego typu danych.

### 4.3 Konwersja typu danych

Podczas programowania napotykamy różne typy danych. W zasadzie można używać różnych typów danych w jednym programie. Możliwe jest również przypisanie różnych typów danych. Niemniej należy czynić to z rozważą.

#### Niejawna konwersja typu danych

Za każdym razem, gdy w kodzie programu realizowane jest przypisanie, kompilator sprawdza typy danych. Przypisania w instrukcji są wykonywane w porządku od prawej do lewej. Zmienna zatem musi mieć dość miejsca, by zachować wartość. Konwersja mniejszego typu danych w większy wykonywana jest niejawnie przez kompilator i nie wymaga od użytkownika żadnych czynności.

Próba przypisania wartości o większym typie danych do zmiennej o mniejszym typie danych skutkuje błędem kompilatora. W takich przypadkach konieczna jest tzw. jawna konwersja typu danych.



Nawet jeśli wykonano niejawną konwersję typu danych, nadal istnieje prawdopodobieństwo wystąpienia nieprawidłowego wyniku, które zależy od kompilatora i przedmiotu konwersji. Istnieje, na przykład, ryzyko przypisania wartości bez znaku do typu danych ze znakiem.

Grozi to błędem przekroczenia zakresu podczas dodawania lub mnożenia. Zależy to jednak od wykorzystywanej platformy. Kompilator w takim przypadku nie sygnalizuje ostrzeżenia.



Wyrażenie	Typy danych	Uwaga
Result := Value; (*UINT USINT*)	UINT, USINT	Niejawne przekształcenie nie wymaga dalszych działań
Result := Value; (*INT USINT*)	INT, UINT	Zachowaj ostrożność z liczbami ujemnymi!
Wynik := wartość1 + wartość2; (*UINT USINT USINT*)	UINT, USINT, USINT	Niebezpieczeństwo przekroczenia zakresu podczas dodawania

Table 11: Przykłady niejawnej konwersji typu danych

### Jawna konwersja typu danych

Mimo, że niejawna konwersja typu danych jest metodą prostszą w użyciu od konwersji jawnej, nie zawsze należy używać jej w pierwszej kolejności. Schłodność programu wymaga prawidłowego postępowania z typami danych poprzez używanie jawnej konwersji typu danych. Poniższe przykłady przedstawiają przypadki, w których konieczne jest jawne przekształcenie typu danych.



Wszystkie deklaracje zmiennych wymienione są w standardzie IEC. Deklaracje zmiennych można wprowadzać w tym formacie w podglądzie tekstowym pliku .var danego programu.



W omawianym przypadku występuje ryzyko błędu przekroczenia zakresu podczas dodawania:

<b>Deklaracja:</b>	<pre>VAR     TotalWeight: INT;     Weight1: INT;     Weight2: INT; END_VAR</pre>
<b>Kod programu:</b>	<pre>TotalWeight := Weight1 + Weight2;</pre>

Table 12: Błąd przekroczenia zakresu może wystąpić bezpośrednio na prawo od operatora przypisania.

W tym przypadku typ danych dla wyniku powinien mieć dość wolnego miejsca dla sumy z operacji dodawania. Wynikiem tego konieczny jest większy typ danych. Podczas dodawania należy przekształcić co najmniej jeden z argumentów operacji w większy typ danych. Kompilator wykonuje niejawną konwersję drugiego argumentu operacji.

<b>Deklaracja:</b>	<pre>VAR     TotalWeight: DINT;     Weight1: INT;     Weight2: INT; END_VAR</pre>
<b>Kod programu:</b>	<pre>TotalWeight := INT_TO_DINT (Weight1) + Weight2;</pre>

Table 13: Za pomocą konwersji jawnej można zapobiec błędom przekroczenia zakresu.

Argumenty operacji obliczane na platformach 32-bitowych są przekształcane w wartości 32-bitowe. W tym przypadku dodawanie nie skutkuje błędem przekroczenia zakresu.



[Programming \ Variable and data types \ Data types \ Basic data types](#)

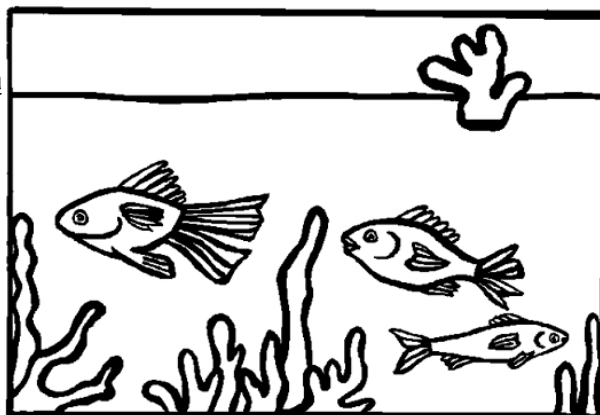
[Programming \ Libraries \ IEC 61131-3 functions \ CONVERT](#)

[Programming \ Editors \ Text editors](#)

[Programming \ Editors \ Table editors \ Declaration editors \ Variable declaration](#)

## Ćwiczenie: Akwarium

Dane jest akwarium, którego temperatura jest mierzona w dwóch punktach. Stwórz program, który oblicza średnią mierzonych temperatur i przesyła ją na wyjście analogowe. Pamiętaj, że typ danych dla wejść i wyjść analogowych to "INT".



Numer 5: Akwarium

**Deklaracja:**

```
VAR
    aiTemp1: INT;
    aiTemp2: INT;
    aoAvgTemp: INT;
END_VAR
```

Table 14: Sugerowana deklaracja zmiennych

## 4.4 Operatory porównawcze i decyzje

Tekst strukturalny uwzględnia proste konstrukcje służące porównywaniu zmiennych. Zwracają one wartość "TRUE" lub "FALSE". Operatory porównawcze i operacje logiczne są zwykle używane jako warunki w instrukcjach takich, jak np. IF, ELSIF, WHILE oraz UNTIL.

Symbol	Typ porównania	Przykład
=	Równość	IF a = b THEN
<>	Nierówność	IF a <> b THEN
>	Większe od	IF a > b THEN
>=	Równe lub większe od	IF a >= b THEN
<	Mniejsze od	IF a < b THEN
<=	Równe lub mniejsze od	IF a <= b THEN

Table 15: Przegląd logicznych operatorów porównawczych

### Decyzje

Instrukcja "IF" jest jednym ze sposobów podejmowania decyzji w programie. Znasz już operatory porównawcze. Możesz ich tu użyć.

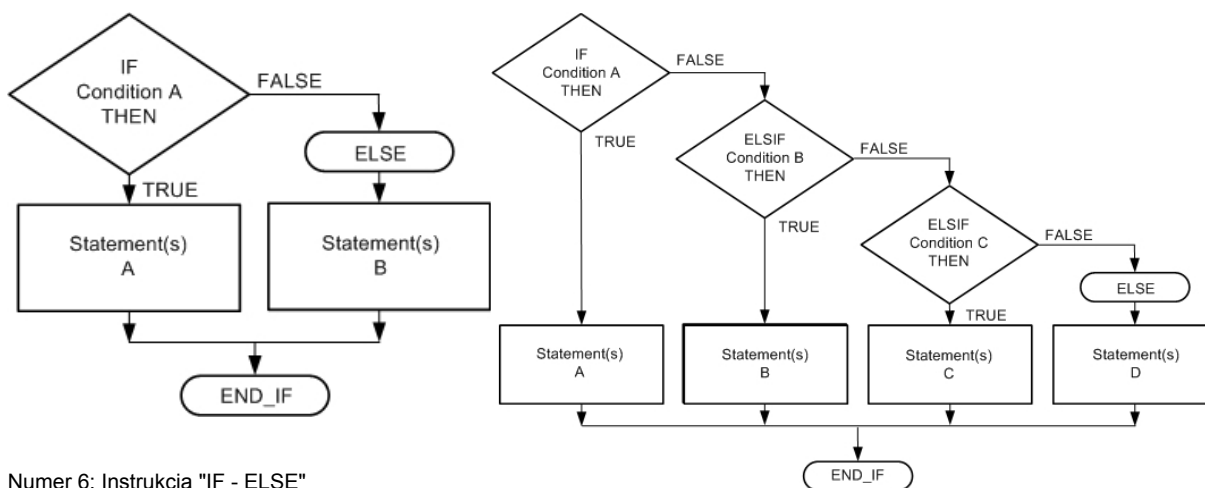
Słowa kluczowe	Składnia	Opis
IF .. THEN	IF a > b THEN	1. Porównanie

Table 16: Składnia instrukcji "IF"



Słowa kluczowe	Składnia	Opis
	<code>Result := 1;</code>	Instrukcja "jeżeli 1. Porównanie" jest "TRUE"
<code>ELSIF .. THEN</code>	<code>ELSIF a &gt; c THEN</code>	2. Porównanie
	<code>Result := 2;</code>	Instrukcja "jeżeli 2. Porównanie" jest "TRUE"
<code>ELSE</code>	<code>ELSE</code>	Gałąź alternatywna, bez porównań "TRUE"
	<code>Result := 3;</code>	Instrukcja gałęzi alternatywnej
<code>END_IF</code>	<code>END_IF</code>	Koniec decyzji

Table 16: Składnia instrukcji "IF"



Numer 6: Instrukcja "IF - ELSE"

Numer 7: Instrukcja "IF - ELSIF - ELSE"

Table 17: Instrukcje "IF" w skrócie

Wyrażenia porównawcze można łączyć ze spójnikami boole'owskimi, umożliwiając jednocześnie sprawdzanie wielu warunków.

	<b>Objaśnienia:</b>	Jeżeli "a" jest większe od "b" zaś "a" mniejsze od "c", to "Result" jest równy 100.
	<b>Kod programu:</b>	<pre>IF (a &gt; b) AND (a &lt; c) THEN     Result := 100; END_IF;</pre>

Table 18: Używanie wielu wyrażen porównawczych

Instrukcja "IF" może uwzględniać dodatkowe zagnieżdżone instrukcje "IF". Należy pamiętać, aby nie używać zbyt wielu poziomów zagnieżdżeń, ponieważ zwykle czyni to program za mało przejrzystym.



Funkcja SmartEdit w edytorze ułatwia wprowadzanie informacji. Jeżeli chcesz wstawić instrukcję "IF", wpisz "IF" i wciśnij przycisk **<TAB>**. Polecenie automatycznie doda podstawową strukturę instrukcji "IF" w edytorze.



Programming \ Programs \ Structured Text (ST) \ IF statement

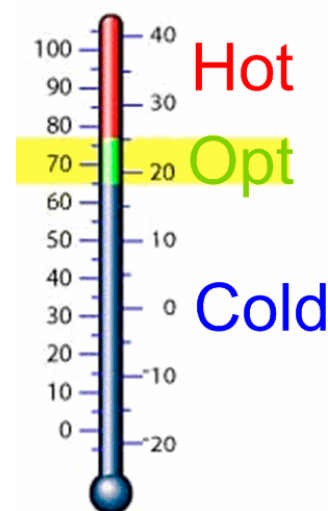
Programming \ Editors \ General operations \ SmartEdit

### Ćwiczenie: Stacja meteorologiczna – Część I

Czujnik temperatury mierzy temperaturę zewnętrzną. Temperatura jest odczytywana na wejściu analogowym i powinna wewnątrz domu zostać przedstawiona na wyjściu w postaci informacji tekstowej.

- 1) Jeżeli temperatura jest poniżej 18°C, wskazanie powinno mieć treść "Cold" (zimno).
- 2) Jeżeli temperatura zawiera się między 18°C i 25°C, wskazanie powinno mieć treść "Opt" (optymalne).
- 3) Jeżeli temperatura jest powyżej 25°C, wskazanie powinno mieć treść "Hot" (upalnie).

Utwórz rozwiązanie za pomocą instrukcji "IF", "ELSIF" i "ELSE".



Numer 8: Termometr



Aby wyprowadzić wyjście tekstowe, należy użyć zmiennej o typie danych "STRING". Przypisanie przedstawia się następująco: `sShowText := 'COLD';`

### Ćwiczenie: Stacja meteorologiczna – Część II

Oprócz temperatury należy podać wskazania wilgotności.

Treść "Opt" powinna wyświetlać się dla wilgotności w przedziale od 40 do 75% i gdy temperatura mieści się w przedziale od 18 do 25°C. W przeciwnym wypadku wskazanie powinno mieć treść "Temp. OK"

Rozwiąż zadanie za pomocą zagnieżdżonej instrukcji "IF".



Jeżeli kilka instrukcji "IF" sprawdza tą samą zmienną, warto zastanowić się, czy nie będzie lepiej zastąpić je lepszą i bardziej przejrzystą instrukcją "CASE".

Instrukcja "CASE" ma taką zaletę w stosunku do instrukcji "IF", że wykonuje porównania tylko jeden raz, a tym samym kod programu jest skuteczniejszy.

#### 4.5 Instrukcja "CASE" - Maszyny stanu

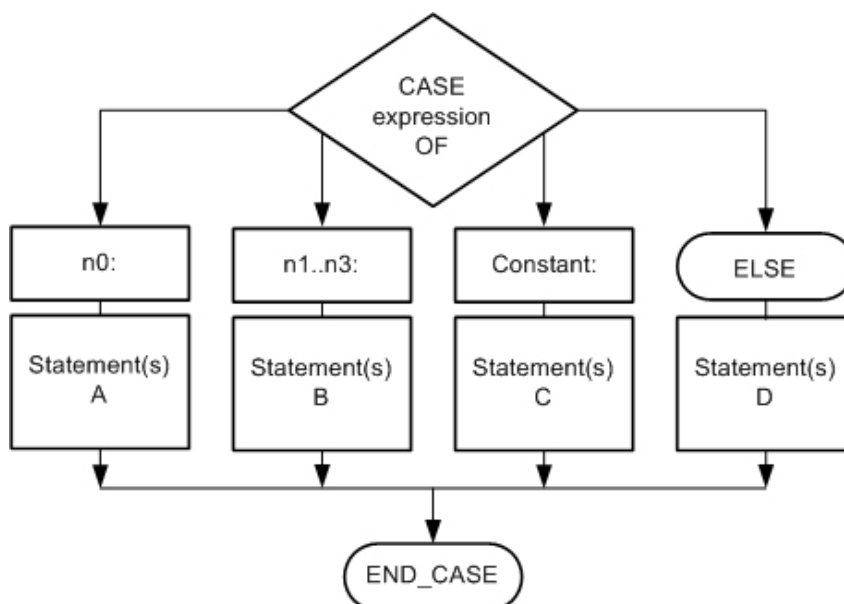
Instrukcja "CASE" porównuje zmienną krokową z wieloma wartościami. Jeżeli jedno z porównań jest zgodne, wówczas zostaną wykonane instrukcje powiązane z tym krokiem. Jeżeli żadne z porównań nie jest zgodne, wówczas zostanie wykonany kod programu pod klauzulą "ELSE", tak, jak w przypadku instrukcji "IF".

W zależności od konkretnej aplikacji instrukcję "CASE" można wprowadzić celem skonfigurowania tzw. maszyny stanu lub automatów.

Słowa kluczowe	Składnia	Opis
<code>CASE .. OF</code>	<code>CASE sStep OF</code>	Początek "CASE"
	<pre> 1, 5:     Show := MATERIAL; </pre>	Dla 1 and 5
	<pre> 2:     Show := TEMP; </pre>	Dla 2
	<pre> 3, 4, 6..10:     Show := OPERATION; </pre>	Dla 3, 4, 6, 7, 8, 9 i 10
<code>ELSE</code>	<code>ELSE</code>	Gałąź alternatywna
	<code>(*...*)</code>	
<code>END_CASE</code>	<code>END_CASE</code>	Koniec "CASE"

Co cykl programu wykonywany jest jeden krok instrukcji "CASE".

Zmienna krokowa powinna mieć typ danych opisujący liczbę całkowitą dodatnią (np. USINT, UINT, UDINT).



Numer 9: Omówienie instrukcji "CASE"



W kodzie programu należy używać stałych lub elementów typów danych wyliczeniowych, zamiast stałych wartości liczbowych. Zastąpienie wartości tekstem czyni kod programu o wiele czytelniejszym. Ponadto jeżeli wartości te trzeba zmienić w programie, to konieczna zmiana wprowadzana jest tylko w deklaracji, a nie w kodzie programu.



Programming \ Programs \ Structured Text (ST) \ CASE statement

Programming \ Variables and data types \ Variables \ Constants

Programming \ Variables and data types \ Data types \ Derived data types \ Enumerations

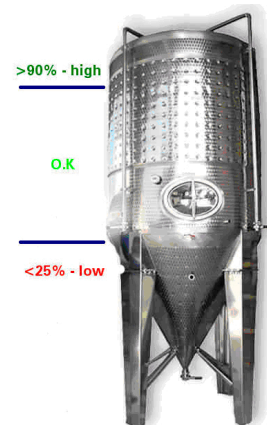
### Ćwiczenie: Kontrolowanie poziomu

Dany jest zbiornik, a poziom cieczy w nim zawartej należy monitorować w trzech obszarach: "niski poziom", "wysoki poziom" i "OK".

Dla każdego z poziomów: "niski poziom", "wysoki poziom" i "OK" przyjmij po jednym wyjściu.

Poziom cieczy w zbiorniku jest odczytywany jako wartość analogowa (0 - 32767) i jest wewnętrznie przekształcany w wartość procentową (0-100%).

Jeżeli zawartość spadnie poniżej 1%, powinno pojawić się ostrzeżenie dźwiękowe. Stwórz rozwiązanie przy pomocy instrukcji "CASE".



Numer 10: Ocena poziomu cieczy w zbiorniku

#### Deklaracja:

```
VAR
    aiLevel : INT;
    PercentLevel : UINT;
    doLow : BOOL;
    doOk : BOOL;
    doHigh : BOOL;
    doAlarm : BOOL;
END_VAR
```

Table 19: Sugerowana deklaracja zmiennych

## 4.6 Pętle

W wielu aplikacjach konieczne jest, aby sekcje kodu były wykonywane wielokrotnie w tym samym cyklu. Ten rodzaj przetwarzania danych nazywamy pętlą. Kod w pętli jest wykonywany aż do wystąpienia określonego warunku zakończenia pętli.

Pętle służą do skracania programów i nadania przejrzystości ich kodowi. Kolejnym zagadnieniem w tym przypadku jest możliwość poszerzenia funkcjonalności programu.

W zależności od struktury programu może wystąpić błąd uniemożliwiający programowi wyjście z pętli, dopóki nie zainterweniuje mechanizm monitorowania czasu w sterowniku. Aby uniknąć występowania nieskończonych pętli, należy zawsze zadbać o możliwość przerwania pętli po określonej liczbie jej powtórzeń.

Istnieją dwa podstawowe typy pętli: takie, gdzie sterowanie w pętli rozpoczyna się od góry, i takie, w którym rozpoczyna się od dołu.

Pętle, w których sterowanie rozpoczyna się od góry (FOR, WHILE) sprawdzają warunek zakończenia przed wejściem w pętlę. Pętle, w których sterowanie rozpoczyna się od dołu (REPEAT) sprawdzają warunek zakończenia na końcu pętli. Taka pętla zawsze wykona przynajmniej jeden cykl.

### 4.6.1 Instrukcja "FOR"

Instrukcja "FOR" służy do wykonywania sekcji programu z ograniczoną liczbą powtórzeń. Pętle "WHILE" i "REPEAT" przeznaczone są dla aplikacji, w których nie jest odgórnie znana liczba cykli.

Słowa kluczowe	Składnia
FOR .. TO .. BY <sup>2</sup> .. DO	FOR i:= StartVal TO StopVal BY Step DO
	Res := Res + 1;
END_FOR	END_FOR

Table 20: Elementy instrukcji "FOR"

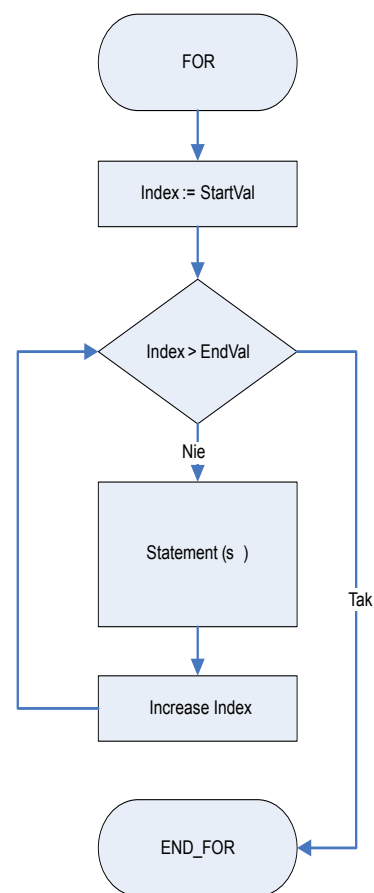
<sup>2</sup> Stosowanie słowa kluczowego "BY" jest opcjonalne.

Licznik pętli "Index" jest inicjalizowany wartością początkową "StartVal". Pętla powtarza się aż ilość jej powtórzeń osiągnie wartość zmiennej końcowej "StopVal". W tym przypadku licznik pętli przyrasta o 1, co zwane jest "BY step" (o krok). Jeżeli dla przyrostu "step" przyjmimy wartość ujemną, wówczas pętla będzie odliczała wstecz.

Typ danych licznika pętli, wartości początkowej i wartości końcowej musi być taki sam. Warunek ten można spełnić stosując jawną konwersję danych (4.3 "Konwersja typu danych").



Jeżeli wartości początkowe i końcowe są identyczne na początku, to ten typ pętli wykona co najmniej jeden cykl! (np. gdy wartość początkowa i końcowa to 0)



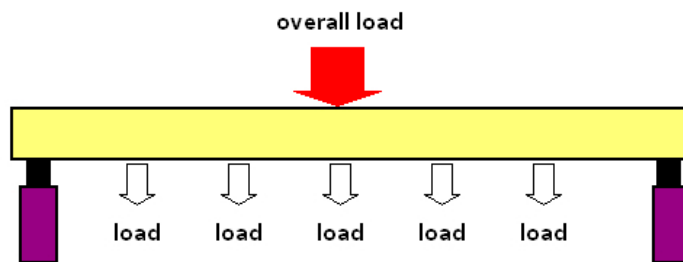
Numer 11: Przegląd instrukcji "FOR"



Programming \ Programs \ Structured Text (ST) \ FOR statement

### Ćwiczenie: Całkowite obciążenie dźwigu

Dany dźwig może podnieść pięć ładunków naraz. Czujniki obciążenia są podłączone do wejść analogowych i zwracają wartości w zakresie od 0 do 32767. W celu obliczenia całkowitego oraz średniego obciążenia, poszczególne wartości obciążenia muszą najpierw zostać zsumowane, a otrzymana suma podzielona przez liczbę czujników obciążenia. Stwórz rozwiązanie tego zadania za pomocą pętli "FOR".



Numer 12: Dźwig z pięcioma ładunkami

**Deklaracja:**

```

VAR
  aWeights : ARRAY [0..4] OF INT;
  iCnt : USINT;
  sumWeight : DINT;
  avgWeight : INT;
END_VAR

```

Table 21: Sugerowana deklaracja zmiennych



Tablice są potrzebne do rozwiązania tego ćwiczenia. Dalsze informacje o tablicach można znaleźć w załączniku lub w Automation Studio.

Tam, gdzie to możliwe, należy używać deklaracji tablicowych dla ładunków oraz stałych, aby ograniczyć wartości końcowe pętli. Poprawia to znacznie czytelność deklaracji i programów. Ponadto będzie łatwiej wprowadzać ewentualne późniejsze zmiany.



Programming \ Programs \ Variables and data types \ Data types \ Composite data types \ Arrays

Patrz [10.1 "Tablice" na stronie 36](#).

### Ćwiczenie: Całkowite obciążenie dźwigu - ulepszenie kodu programu

Pochodząca z poprzedniego zadania suma poszczególnych obciążeń może być generowana za pomocą pętli. Aż do teraz, określone wartości liczbowe znajdują się w deklaracji zmiennej i w kodzie programu. Celem tego zadania jest zastąpienie, wszędzie tam gdzie jest to możliwe, wartości liczbowych z deklaracji i kodu programu stałymi.

**Deklaracja:**

```

VAR CONSTANT
  MAX_INDEX : INT := 4;
END_VAR

```

Table 22: Sugerowana deklaracja zmiennych

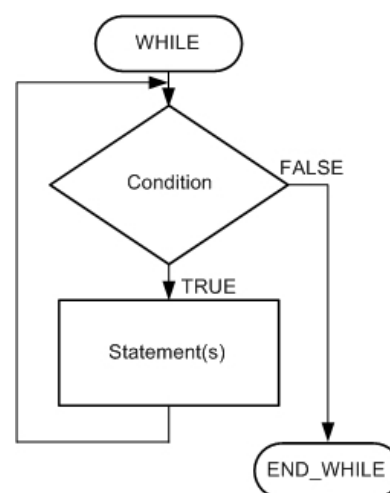
## 4.6.2 Instrukcja "WHILE"

W przeciwieństwie do instrukcji "FOR", pętle "WHILE" nie zależą od licznika pętli. Ten typ pętli jest wykonywany dopóty, dopóki warunek lub wyrażenie ma wartość "TRUE". Pętla powinna mieć określony koniec, aby podczas wykonywania programu nie wystąpiło przekroczenie czasu cyklu.

Słowa kluczowe	Składnia
WHILE .. DO	WHILE i < 4 DO
	Res := value + 1;
	i := i + 1;
END_WHILE	END_WHILE

Table 23: Wykonywanie instrukcji "WHILE"

Wykonanie instrukcji powtarza się dopóty, dopóki warunek ma wartość "TRUE". Jeżeli warunek zwraca wartość "FALSE" podczas pierwszej jego oceny, wówczas instrukcja nie zostanie wykonana w ogóle.



Numer 13: Przegląd instrukcji "WHILE"



Programming \ Programs \ Structured Text (ST) \ WHILE statement

## 4.6.3 Instrukcja "REPEAT"

Pętla "REPEAT" różni się od pętli "WHILE" tym, że warunek zakończenia sprawdzany jest dopiero po wykonaniu pętli. Oznacza to, że pętla zostanie wykonana co najmniej raz – bez względu na warunek zakończenia.

Słowa kluczowe	Składnia
REPEAT	REPEAT
	(*program code*)
	i := i + 1;
UNTIL	UNTIL i > 4
END_REPEAT	END_REPEAT

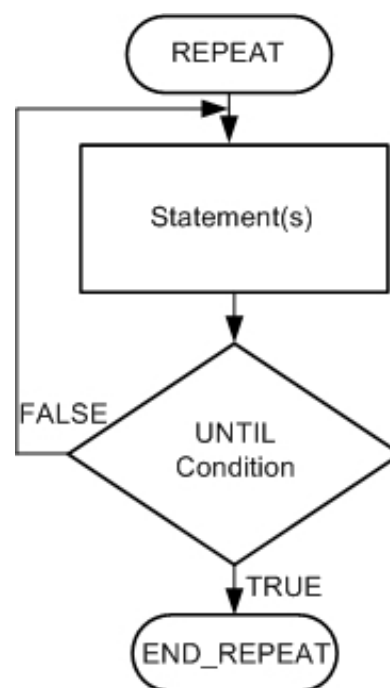
Table 24: Wykonywanie instrukcji "REPEAT"



Wykonanie instrukcji powtarza się dopóki warunek "UNTIL" nie osiągnie wartości "TRUE". Jeżeli warunek "UNTIL" jest prawdziwy od samego początku, to instrukcje zostaną wykonane jeden raz.



Jeżeli warunek "UNTIL" nigdy nie przyjmuje wartości "TRUE", to instrukcje są powtarzane bez końca, skutkując błędem czasu wykonania programu.



Numer 14: Przegląd instrukcji "REPEAT"



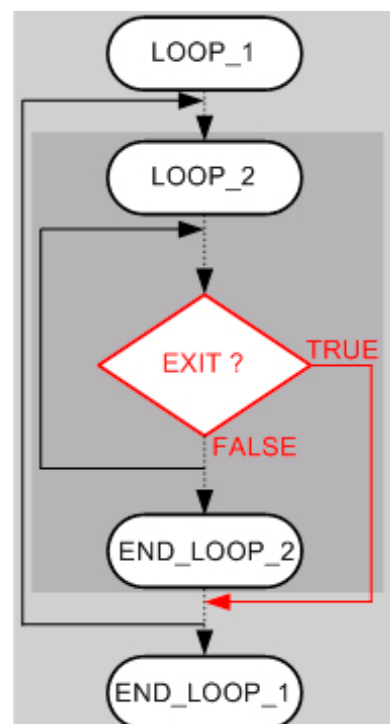
Programming \ Programs \ Structured Text (ST) \ REPEAT statement

#### 4.6.4 Instrukcja "EXIT"

Instrukcja "EXIT" może być używana w każdym typie pętli przed spełnieniem warunku zakończenia. Po wykonaniu instrukcji "EXIT" pętla zostanie zakończona.

Słowa kluczowe	Składnia
	REPEAT
	IF setExit = TRUE THEN
EXIT	EXIT;
	END_IF
	UNTIL i > 5
	END_REPEAT

Po wykonaniu instrukcji "EXIT" pętla zostanie zakończona bez względu na to, czy osiągnięto warunek zakończenia, czy też wartość końcową pętli. W przypadku pętli zagnieżdżonych kończone są wyłącznie pętle zawierające instrukcję "EXIT".



Numer 15: Instrukcja "EXIT" kończy tylko pętlę wewnętrzną.



Programming \ Programs \ Structured Text (ST) \ EXIT statement

### Ćwiczenie: Wyszukiwanie z zakończeniem

Należy znaleźć jedną, ściśle określoną liczbę na liście 100 liczb. Lista zawiera liczby losowe. Jeżeli znaleziono liczbę 10, proces wyszukiwania powinien zostać zakończony. Istnieje jednak możliwość, że szukanej liczby nie ma na liście.

W rozwiązaniu użyj instrukcji "REPEAT" i "EXIT". Pamiętaj o obu warunkach zakończenia.

**Deklaracja:**

```
VAR
    aValues : ARRAY[0..99] OF INT;
END_VAR
```



Poszczególne elementy tablic można zainicjalizować wartościami w kodzie programu lub w oknie deklaracji zmiennych.



Programming \ Editors \ Table editors \ Declaration editors \ Variable declaration

## 5 Funkcje, bloki funkcyjne i akcje

Funkcje i bloki funkcyjne rozszerzają funkcjonalność języka programowania. Akcje służą do nadania skuteczniejszej struktury programom. Funkcje i bloki funkcyjne można wstawiać za pomocą paska narzędzi.



Numer 16: Wstawianie funkcji i bloków funkcyjnych z paska narzędzi

### 5.1 Wywoływanie funkcji i bloków funkcyjnych

#### Funkcje

Funkcje są podprogramami, które po wywołaniu zwracają określoną wartość. Funkcję można wywołać m.in. za pomocą wyrażenia. Argumenty funkcji (parametry wejściowe), są wartościami przekazywanymi do funkcji. Są one oddzielane przecinkami.

<b>Deklaracja:</b>	<pre>VAR     sinResult : REAL;     xValue : REAL; END_VAR</pre>
<b>Kod programu:</b>	<pre>xValue := 3.14159265; sinResult := SIN(xValue);</pre>

Table 25: Wywoływanie funkcji z jednym argumentem

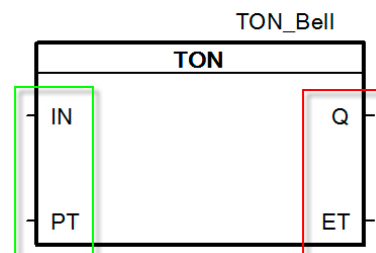


Programming \ Programs \ Structured Text (ST) \ Calling functions

#### Bloki funkcyjne

Blok funkcyjny wyróżnia się tym, że może przyjąć kilka argumentów i zwracać wiele wyników.

W przeciwieństwie do funkcji należy zadeklarować zmienną (instancję), której typ danych jest taki sam jak bloku funkcyjnego. Zaletą bloku funkcyjnego jest to, że może on obsługiwać zadania złożone, obliczając wynik w ciągu kilku cykli. Za pomocą różnych instancji można wywołać kilka bloków funkcyjnych tego samego typu, z różnymi argumentami.



Numer 17: Argumenty (kolor zielony) i wyniki (kolor czerwony) dla bloku funkcyjnego "TON()"

Wywołując blok funkcyjny można przekazać wszystkie argumenty lub tylko niektóre z nich. Parametry wejściowe (argumenty) i wyniki są dostępne w kodzie programu za pomocą elementów instancji bloku funkcyjnego.

<b>Deklaracja:</b>	<pre> VAR     diButton : BOOL;     doBell : BOOL;     TON_Bell : TON; END_VAR </pre>
<b>Wariant wywołania 1:</b>	<pre> TON_bell(IN := diButton, PT := T#1s); doBell := TON_bell.Q; </pre>
<b>Wariant wywołania 2:</b>	<pre> (*-- parameters*) TON_bell.IN := diButton; TON_bell.PT := T#1s (*-- call functionblock*) TON_bell(); (*-- read results*) doBell := TON_bell.Q; </pre>

Table 26: Odbicie przycisku blokiem funkcyjnym "TON()"

W wariantcie 1 wywołania wszystkie parametry są bezpośrednio przekazywane w chwili wywołania bloku funkcyjnego. W wariantcie 2 wywołania parametry są przydzielane elementom zmiennej (instancji). W obu przypadkach żądany wynik musi zostać odczytany ze zmiennej (instancji) po wystąpieniu wywołania.



Programming \ Programs \ Structured Text (ST) \ Calling function blocks

## Ćwiczenie: Bloki funkcyjne

Wywołaj niektóre bloki funkcyjne z biblioteki "STANDARD". Najpierw zapoznaj się z opisami funkcji i parametrów w systemie pomocy oprogramowania Automation Studio.

- 1) Wywołaj blok funkcyjny zwłoki włączania "TON".
- 2) Wywołaj blok funkcyjny licznika zliczającego w górę "CTU".



Szczegółowy opis funkcji w bibliotekach znajduje się w systemie pomocy Automation Studio. Wciśnięcie <F1> otwiera dokumentację pomocy dla wybranego bloku funkcyjnego.

Wiele bibliotek zawiera przykłady różnych aplikacji, które można importować bezpośrednio do projektów w Automation Studio.




Programming \ Libraries \ IEC 61131-3 functions \ STANDARD

Programming \ Examples


5.2 Wywołanie akcji

Akcja jest kodem programu, który można dodawać do programów i bibliotek. Akcje to kolejny element umożliwiający strukturyzację programów, a ponadto można je pisać w językach programowania innych niż język programu je wywołującego. Akcje są identyfikowane poprzez swoje nazwy.

Wywoływanie akcji przebiega bardzo podobnie do wywoływania funkcji. Jediną różnicą jest brak parametrów wejściowych (argumentów) oraz fakt, że akcja nie zwraca żadnej wartości.



Jeżeli przyjmimy instrukcję "CASE" do sterowania złożoną sekwencją działań, to treści poszczególnych kroków "CASE" można przenieść do akcji. Pozwala to zminimalizować wielkość programu. Jeżeli taka możliwość jest potrzebna w innej części programu, wystarczy ponownie przywołać omawianą akcję.



**Program:**

```
CASE Sequence OF
  WAIT:
    IF cmdStartProc = 1 THEN
      Sequence := STARTPROCESS;
    END_IF

    STARTPROCESS:
      acStartProc; (*machine startup*)

    IF Process_finished = 1 THEN
      Sequence := ENDPROCESS;
    END_IF

    ENDPROCESS:
      acEndProc;  (*machine shutdown*)
      (*...*)
    END_CASE
```

**Akcja:**

```
ACTION acStartProc:
  (*add your sequence code here*)
  Process_finished := 1;
END_ACTION
```

Table 27: Wywoływanie akcji w programie głównym



Actions

### 6 Dodatkowe i pomocnicze funkcje

#### 6.1 Wskaźniki i referencje

B&R umożliwia rozszerzenie funkcjonalności istniejącej normy IEC o wskaźniki dostępne w ST. Wskaźniki umożliwiają przydzielanie do zmiennej dynamicznej określonego adresu pamięci podczas wykonywania programu. Procedura ta nazywana jest referencją lub inaczej inicjalizacją zmiennej dynamicznej.

Po zainicjalizowaniu zmiennej dynamicznej można jej użyć, aby uzyskać dostęp do zawartości pod adresem pamięci, który zmienna wskazuje. Operacja ta wymaga słowa kluczowego "ACCESS".

<b>Deklaracja:</b>	<pre>VAR     iSource : INT;     pDynamic : REFERENCE TO INT; END_VAR</pre>
<b>Kod programu:</b>	<pre>pDynamic ACCESS ADR(iSource); (*pDynamic references to iSource*)</pre>

Table 28: Odnoszenie wskaźnika



Rozszerzone funkcje standardu IEC można włączyć w ustawieniach projektu w Automation Studio.

#### 6.2 Preprocesor programów IEC

W tekstowych językach programowania można używać tzw. dyrektyw preprocesora. Zaimplementowane instrukcje pod względem składni są bardzo zbliżone do instrukcji preprocesora ANSI C.

Polecenia preprocesora są rozszerzeniem normy IEC. Należy je włączyć w ustawieniach projektu.

Dyrektywy preprocesora służą do warunkowego kompilowania programów lub całych konfiguracji. Ich włączenie wpływa na proces kompilowania kodu programu przez kompilator.

Opis i pełen wykaz dostępnych poleceń znajduje się w systemie pomocy Automation Studio.



[Project management \ The workspace \ General project settings \ Settings for IEC compliance Programming \ Programs \ Preprocessor for IEC programs](#)

## 7 Funkcje diagnostyczne

Kompleksowe narzędzia diagnostyczne czynią programowanie znacznie wydajniejszym zadaniem. W Automation Studio przewidziano szereg narzędzi służących do usuwania usterek w językach programowania wysokiego poziomu:

- Tryb monitora
- Podgląd zmiennych - "Watch"
- Funkcja "Line coverage"
- Podpowiedzi (tooltips)
- Debugger
- Lista "Cross reference"



Diagnostics and service \ Diagnostics tool \ Debugger

Diagnostics and service \ Diagnostics tool \ Variable watch

Diagnostics and service \ Diagnostics tool \ Monitors \ Programming languages in monitor mode \ Line coverage

Diagnostics and Service \ Diagnostic tools \ Monitors \ Programming languages in monitor mode \ Powerflow

Project management \ The workspace \ Output window \ Cross reference

## 8 Ćwiczenia

### 8.1 Ćwiczenie - Podnośnik pudeł

#### Ćwiczenie: Podnośnik pudeł

Dwa przenośniki taśmowe (**doConvTop**, **doConvBottom**) przenoszą pudła do podnośnika.

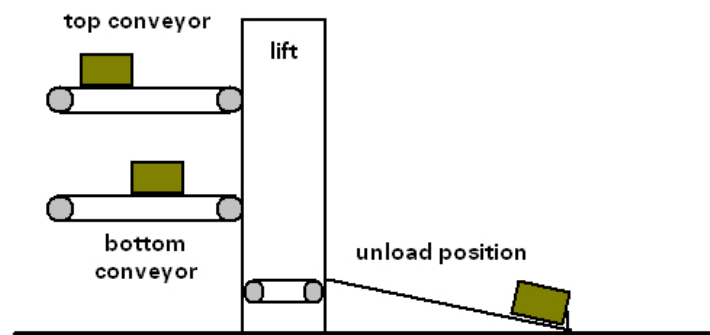
Jeżeli fotokomórka (**diConvTop** lub **diConvBottom**) zostaje aktywowana, odpowiadający jej przenośnik taśmowy zatrzyma się i zostanie zasygnalizowane żądanie opuszczenia podnośnika.

Jeżeli podnośnik nie jest pożądaný, wraca w odpowiednie położenie (**doLiftTop**, **doLiftBottom**).

Kiedy podnośnik znajduje się w zadanej pozycji (**diLiftTop**, **diLiftBottom**), przenośnik taśmowy podnośnika (**doConvLift**) zostaje włączony aż do chwili, gdy całe pudło znajdzie się na podnośniku (**diBoxLift**).

Podnośnik przechodzi następnie w położenie rozładunkowe (**doLiftUnload**). Kiedy je osiąga (**diLiftUnload**), pudło trafia na przenośnik taśmowy rozładunkowy.

Gdy tylko pudło opuści podnośnik, ten będzie gotowy przyjąć kolejne żądanie.

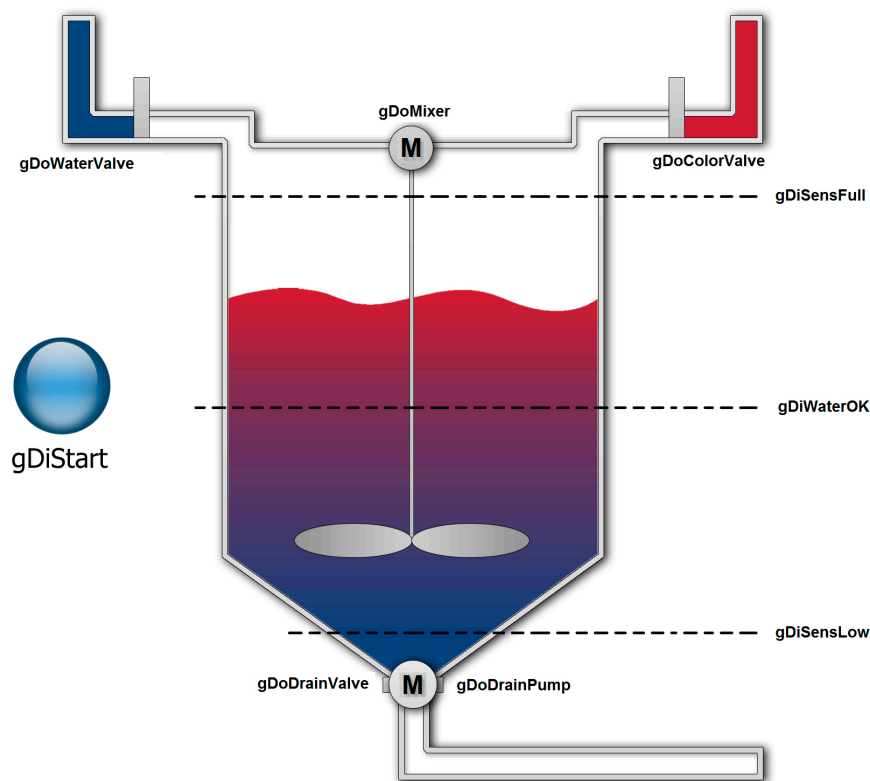


Numer 18: Podnośnik pudeł



## 8.2 Ćwiczenie - Mieszadło do farb dyspersyjnych

W tym ćwiczeniu będziemy programować system mieszania, który miesza wodę i farbę, w wyniku czego powstaje dyspersja.



Numer 19: Schemat systemu mieszania farb

**Program mieszania będzie działać zgodnie z następującą procedurą:**

- Program mieszania czeka, aż zostanie naciśnięty przycisk start (**gDiStart**).
- Woda (**gDoWaterValve**) jest dodawana do zbiornika aż do załączenia czujnika "**gDiWaterOK**".
- Urządzenie mieszające (**gDoMixer**) uruchamia się, a farba (**gDoColorValve**) jest dodawana aż do załączenia czujnika "**gDiSensFull**".
- Czas mieszania powinien wynosić 30 sekund.
- Zawór (**gDoDrainValve**) i pompa (**gDoDrainPump**) są włączane, aby opróżnić zbiornik.
- Proces opróżniania jest zakończony, gdy zostanie wyzwolony sygnał "**gDiSensLow**".
- Stan początkowy jest przywracany.

### Ćwiczenie: Realizacja mieszadła do farb dyspersyjnych

Podstawowym zadaniem jest analiza procesu mieszania farb dyspersyjnych, a następnie zaprogramowanie go krok po kroku w tekście strukturalnym. Aby to zrobić, należy wykonać następujące czynności:

- 1) Naskicuj proces.

- Zdefiniuj kroki.
  - Zdefiniuj przejścia.  
(warunki przejścia z jednego kroku do następnego)
  - Zdefiniuj akcje.  
(fragmenty programu, które są wywoływane bezpośrednio w kroku)
- 2) [opcjonalnie] Określ, które kroki mogą zostać przeniesione do akcji zgodnie z normą IEC.
- 3) Realizacja założeń w programie w tekście strukturalnym



Blok czasowy z biblioteki "STANDARD" powinien zostać wykorzystany do realizacji czasu mieszania. Upewnij się, że blok czasowy jest resetowany po upływie zadanego czasu.

## 9 Podsumowanie

Teks strukturalny (ST) jest językiem programowania wysokiego poziomu, który odznacza się bardzo dużymi możliwościami funkcjonalnymi. Zawiera on wszystko, co niezbędne, do tworzenia aplikacji dedykowanych dla określonych zadań. Znasz już ogólne informacje o konstruktach i możliwościach dostępnych w ST.



Dokumentacja pomocy Automation Studio opisuje szczegółowo wszystkie te konstrukty. Należy pamiętać, że omawiany język programowania jest najbardziej przydatny, jeżeli korzystasz w nim z funkcji arytmetycznych i formułujesz obliczenia matematyczne.

## 10 Załącznik

### 10.1 Tablice

W odróżnieniu od zmiennych o podstawowych typach danych, tablice składają się z kilku zmiennych o tym samym typie danych. Do każdego pojedynczego elementu można się dostać poprzez nazwę tablicy i wartość indeksu.

Indeks używany, aby dostać się do wartości w tablicy, nie może wykraczać poza jej rzeczywisty rozmiar. Rozmiar tablicy jest definiowany, kiedy zmienna jest deklarowana.

W programie indeks może być stałą wartością liczbową, zmienną, stałą lub literałem wyliczeniowym.

Name	Typ	Wart
aPressure	INT[0..9]	
aPressure[0]	INT	123
aPressure[1]	INT	555
aPressure[2]	INT	0
aPressure[3]	INT	552
aPressure[4]	INT	32767
aPressure[5]	INT	9700
aPressure[6]	INT	0
aPressure[7]	INT	9
aPressure[8]	INT	0
aPressure[9]	INT	13

Numer 20: Tablica zmiennych typu INT z zakresem od 0 do 9 odpowiada 10 różnym elementom tablicy.

#### Deklarowanie i używanie tablic

Gdy tablica jest deklarowana, musi zostać podany typ danych i rozmiar. Powszechną praktyką jest stosowanie wartości 0 jako najmniejszego indeksu tablicy. Należy pamiętać, że wtedy maksymalny indeks tablicy 10-cio elementowej wynosi 9.



#### Deklaracja

```
VAR
    aPressure : ARRAY[0..9] OF INT := [10(0)];
END_VAR
```

#### Kod programu

```
(* Przypisanie wartości 123 do elementu z indeksem 0 *)
aPressure[0] := 123;
```

Table 29: Deklarowanie tablicy 10-cio elementowej, indeks początkowy = 0

Podczas próby uzyskania dostępu do elementu tablicy o indeksie 10, kompilator wyświetla następujący komunikat o błędzie:

#### Kod programu

```
aPressure[10] := 75;
```

#### Komunikat o błędzie

```
Error 1230: The constant value '10' is not in range '0..9'.
```

Table 30: Próba dostępu do elementu tablicy o indeksie spoza dopuszczalnego zakresu



Aby zadeklarować tablicę 10-cio elementową, można w edytorze deklaracji wpisać "USINT [0..9]" lub "USINT [10]". W obu tych przypadkach zostanie utworzona tablica o indeksie początkowym 0 i indeksie maksymalnym 9.

**Zadanie: Tworzenie tablicy "aPressure"**

- 1) Dodaj nowy program "int\_array" w podglądzie logicznym "Logical View"
- 2) Otwórz okno deklaracji zmiennych.
- 3) Zadeklaruj tablicę "aPressure".

Tablica powinna zawierać 10 elementów. Najmniejszy indeks tablicy to 0. Typ danych musi być "INT".

- 4) Wykorzystaj tablicę w kodzie programu.

Użyj indeks aby uzyskać dostęp do tablicy w kodzie programu. W tym celu skorzystaj z określonych wartości liczbowych, stałych i zmiennych.

- 5) Spróbuj, w nieprawidłowy sposób, uzyskać dostęp do tablicy w kodzie programu.

Wywołaj dostęp do elementu tablicy o indeksie 10, a następnie przeanalizuj komunikat w oknie danych wyjściowych.



Podczas próby przypisania `aPressure[10] := 123;`, kompilator zgłasza następujący komunikat o błędzie.

**Error 1230: The constant value '10' is not in range '0..9'.**

Kompilator nie jest w stanie sprawdzić, czy dostęp do tablicy jest prawidłowy, jeśli przypisanie odbywa się za pomocą zmiennej.

```
index := 10;
aPressure[index] := 123;
```

**Deklarowanie tablicy za pomocą stałych**

Użycie określonych wartości liczbowych w deklaracjach i kodzie programu prowadzi zwykle do oprogramowania, które jest trudne do modyfikacji i analizy; w związku z tym o wiele lepszym pomysłem jest użycie stałych.

Indeksy (maksymalny i minimalny) tablicy mogą być określone za pomocą stałych. Stałe te mogą być następnie wykorzystywane w kodzie programu, aby ograniczyć indeks stosowany do uzyskania dostępu do tablicy.

**Deklaracja**

```
VAR CONSTANT
    MAX_INDEX : USINT := 9;
END_VAR
VAR
    aPressure : ARRAY[0..MAX_INDEX] OF INT ;
    index : USINT := 0;
END_VAR
```

**Kod programu**

```
IF index > MAX_INDEX THEN
    index := MAX_INDEX;
END_IF
aPressure[index] := 75;
```

Table 31: Deklarowanie tablicy za pomocą stałej



Kod programu został zaktualizowany tak, że indeks stosowany do uzyskania dostępu do tablicy jest ograniczony do maksymalnej wartości indeksu tablicy. Zaletą jest to, że tablice mogą być zmniejszane lub powiększane bez konieczności dokonywania zmian w kodzie programu.



Programming \ Variables and data types \ Data types \ Derived data types \ Arrays

## Zadanie: Obliczenie sumy i średniej wartości

Należy obliczyć średnią wartość elementów tablicy "aPressure". Program musi być skonstruowany tak, aby jak najmniejsza ilość zmian w programie była konieczna do zmodyfikowania rozmiaru tablicy.

- 1) Oblicz sumę za pomocą pętli.

Stałe liczbowe nie mogą być stosowane w kodzie programu.

- 2) Obliczenie wartości średniej

Typ danych dla wartości średniej musi być taki sam, jak typ danych dla tablicy ("INT").



Stała, która jest wykorzystywana do określenia rozmiaru w deklaracji tablicy, może być również używana jako wartość końcowa dla pętli. Podczas obliczania wartości średniej ta sama stała może być używana również w operacji dzielenia. Podczas dodawania poszczególnych elementów tablicy musi zostać użyty większy typ danych niż typ danych zdefiniowany dla zmiennej wyjściowej (na przykład "DINT"). Obliczona wartość może być z powrotem przekształcona na typ danych "INT" za pomocą jawnej konwersji typu danych.

## Tablice wielowymiarowe

Tablice mogą również składać się z kilku wymiarów. Deklaracja i zastosowanie w tym przypadku mogą wyglądać następująco:

<b>Deklaracja</b>	<pre>VAR     Array2Dim : ARRAY [0..6,0..6] OF INT; END_VAR</pre>	
<b>Kod programu</b>	<pre>(*Zählweise mit 0 beginnend*) Array2Dim[2,2] := 11;</pre>	<p>Numer 21: Dostęp do wartości w kolumnie 3, wiersz 3</p>

Table 32: Deklaracja i wykorzystanie tablicy dwuwymiarowej o wymiarach 7x7



Nieprawidłowa próba uzyskania dostępu do tablicy w kodzie programu przy użyciu stałej liczbowej, stałej lub literału wyliczeniowego zostanie wykryta i zatrzymana przez kompilator.

Nieprawidłowa próba uzyskania dostępu do tablicy w kodzie programu za pomocą zmiennej nie jest wykrywana przez kompilator i może prowadzić do błędu pamięci podczas pracy sterownika. Błędów pamięci można uniknąć poprzez ograniczenie indeksu tablicy do prawidłowego zakresu.

Biblioteka IEC Check może zostać zaimportowana do projektu w Automation Studio i pomóc zlokalizować błędy występujące podczas pracy sterownika.



Programming \ Libraries \ IEC Check library

## 10.2 Rozwiązania do ćwiczeń

### Ćwiczenie: Sterowanie oświetleniem

<b>Deklaracja:</b>	<pre> VAR     btnLightOn: BOOL;     btnLightOff: BOOL;     doLight: BOOL; END_VAR </pre>
<b>Kod programu:</b>	<pre> doLight := (btnLightOn OR doLight) AND NOT bntLightOff; </pre>

Table 33: Przełącznik dwupozycyjny (on/off), przekaźnik z zatraskiem

### Ćwiczenie: Akwarium

<b>Deklaracja:</b>	<pre> VAR     aiTemp1 : INT;     aiTemp2 : INT;     aoAvgTemp : INT; END_VAR </pre>
<b>Kod programu:</b>	<pre> aoAvgTemp := DINT_TO_INT((INT_TO_DINT(aiTemp1)     + aiTemp2) / 2); </pre>

Table 34: Jawną konwersja typu danych przed dodawaniem i po dzieleniu

### Ćwiczenie: Stacja meteorologiczna – Część 1

<b>Deklaracja:</b>	<pre> VAR     aiOutside : INT;     sShowText : STRING[80]; END_VAR </pre>
--------------------	---

Table 35: Instrukcja "IF"

Kod programu:

```

IF aiOutside < 18 THEN
    sShowText := 'Cold';
ELSIF (aiOutside >= 18) AND (aiOutside <= 25) THEN
    sShowText := 'Opt';
ELSE
    sShowText := 'Hot';
END_IF;

```

Table 35: Instrukcja "IF"

## Ćwiczenie: Stacja meteorologiczna – Część 2

Deklaracja:

```

VAR
    aiOutside : INT;
    aiHumidity: INT;
    sShowText : STRING[80];
END_VAR

```

Kod programu:

```

IF aiOutside < 18 THEN
    sShowText := 'Cold';
ELSIF (aiOutside >= 18) AND (aiOutside <= 25) THEN
    IF (aiHumid >= 40) AND (aiHumid <= 75) THEN
        sShowText := 'Opt';
    ELSE
        sShowText := 'Temp. Ok';
    END_IF
ELSE
    sShowText := 'Hot';
END_IF;

```

Table 36: Zagnieżdżona instrukcja "IF"

## Ćwiczenie: Kontrolowanie poziomu

Deklaracja:

```

VAR
    aiLevel : INT;
    PercentLevel : UINT;
    doLow : BOOL;
    doOk : BOOL;
    doHigh : BOOL;
    doAlarm : BOOL;
END_VAR

```

Table 37: Instrukcja "CASE" do odpytywania wartości i zakresów wartości



**Kod programu:**

```

(*scaling the analog input to percent*)
PercentLevel := INT_TO_UINT(aiLevel / 327);
(*reset all outputs*)
doAlarm := FALSE;
doLow := FALSE;
doOk := FALSE;
doHigh := FALSE;

CASE PercentLevel OF
  0:      (*-- level alarm*)
    doAlarm := TRUE;
  1..24: (*-- level is low*)
    doLow := TRUE;
  25..90: (*-- level is ok*)
    doOk := TRUE;
  ELSE   (*-- level is high*)
    doHigh := TRUE;
END_CASE

```

Table 37: Instrukcja "CASE" do odpytywania wartości i zakresów wartości

**Ćwiczenie: Wyszukiwanie z zakończeniem****Deklaracja:**

```

VAR CONSTANT
  MAXNUMBERS : UINT := 99;
END_VAR
VAR
  aNumbers : ARRAY[0..MAXNUMBERS] OF INT;
  nCnt : INT;
END_VAR

```

**Kod programu:**

```

nCnt := 0;
REPEAT
  IF aNumbers[nCnt] = 10 THEN
    (*found the number 10*)
    EXIT;
  END_IF
  nCnt := nCnt + 1;
  UNTIL nCnt > MAXNUMBERS
END_REPEAT

```

Table 38: Instrukcja "REPEAT", przerywanie przeszukiwania, ograniczanie cykli

Ćwiczenie: Całkowite obciążenie dźwigu

Deklaracja:	<pre>VAR CONSTANT     MAX_INDEX: UINT := 4; END_VAR VAR     aWeights : ARRAY[0..MAX_INDEX] OF INT;     wCnt : INT;     sumWeight : DINT;     avgWeight : INT; END_VAR</pre>
Kod programu:	<pre>sumWeight := 0; FOR wCnt := 0 TO MAX_INDEX DO     sumWeight := sumWeight + aWeights[wCnt]; END_FOR avgWeight := DINT_TO_INT (sumWeight / (MAX_INDEX + 1));</pre>

Table 39: Instrukcja "FOR", dodawanie ciężaru

Ćwiczenie: Bloki funkcyjne

Deklaracja:	<pre>VAR     TON_Test : TON;     CTU_Test : CTU;     diSwitch : BOOL;     diCountImpuls : BOOL;     diReset : BOOL; END_VAR</pre>
Kod programu:	<pre>TON_Test(IN := diSwitch, PT := T#5s); CTU_Test(CU := diCountImpuls, RESET := diReset);</pre>

Table 40: Wywołanie "TON" i "CTU"

## Ćwiczenie: Podnośnik pudeł

## Deklaracja:

```

VAR CONSTANT
    WAIT : UINT := 0;
    TOP_POSITION : UINT := 1;
    BOTTOM_POSITION : UINT := 2;
    GETBOX : UINT := 3;
    UNLOAD_POSITION : UINT := 4;
    UNLOAD_BOX : UINT := 5;
END_VAR

VAR
    (*-- digital outputs*)
    doConvTop: BOOL;
    doConvBottom: BOOL;
    doConvLift: BOOL;
    doLiftTop: BOOL;
    doLiftBottom: BOOL;
    doLiftUnload: BOOL;
    (*-- digital inputs*)
    diConvTop: BOOL;
    diConvBottom: BOOL;
    diLiftTop: BOOL;
    diLiftBottom: BOOL;
    diLiftUnload: BOOL;
    diBoxLift: BOOL;
    (*-- status variables*)
    selectLift: UINT;
    ConvTopOn: BOOL;
    ConvBottomOn: BOOL;
END_VAR

```

Table 41: Możliwe rozwiązanie ćwiczenia z podnośnikiem pudeł

## Kod programu:

```
doConvTop := NOT diConvTop OR ConvTopOn;
doConvBottom := NOT diConvBottom OR ConvBottomOn;
CASE selectLift OF
  (*-- wait for request*)
  WAIT:
    IF (diConvTop = TRUE) THEN
      selectLift := TOP_POSITION;
    ELSIF (diConvBottom = TRUE) THEN
      selectLift := BOTTOM_POSITION;
    END_IF

  (*-- move lift to top position*)
  TOP_POSITION:
    doLiftTop := TRUE;
    IF (diLiftTop = TRUE) THEN
      doLiftTop := FALSE;
      ConvTopOn := TRUE;
      selectLift := GETBOX;
    END_IF
```

Table 41: Możliwe rozwiązanie ćwiczenia z podnośnikiem pudeł

```

(*-- move lift to bottom position*)
BOTTOM_POSITION:
    doLiftBottom := TRUE;
    IF (diLiftBottom = TRUE) THEN
        doLiftBottom := FALSE;
        ConvBottomOn := TRUE;
        selectLift := GETBOX;
    END_IF

(*-- move box to lift*)
GETBOX:
    doConvLift := TRUE;
    IF (diBoxLift = TRUE) THEN
        doConvLift := FALSE;
        ConvTopOn := FALSE;
        ConvBottomOn := FALSE;
        selectLift := UNLOAD_POSITION;
    END_IF

(*-- move lift to unload position*)
UNLOAD_POSITION:
    doLiftUnload := TRUE;
    IF (diLiftUnload = TRUE) THEN
        doLiftUnload := FALSE;
        selectLift := UNLOAD_BOX;
    END_IF

(*-- unload the box*)
UNLOAD_BOX:
    doConvLift := TRUE;
    IF (diBoxLift = FALSE) THEN
        doConvLift := FALSE;
        selectLift := WAIT;
    END_IF
END_CASE

```

Table 41: Możliwe rozwiązanie ćwiczenia z podnośnikiem pudeł

## Ćwiczenie: Realizacja mieszadła do farb dyspersyjnych

### Deklaracje zmiennych

```

(*****
* COPYRIGHT -- Bernecker + Rainer
*****
* Program: mixer
* File: mixer.var
* Author: Academy
* Created: February 04, 2015
*****
* Local variables of program mixer
*****)

```

```
(*state machine variables*)
VAR
    sMixerState : enMixerStates := enWAIT_FOR_START;
END_VAR
(*Digital input signals*)
VAR
    gDiStart : BOOL := FALSE;
    gDiWaterOK : BOOL := FALSE;
    gDiSensFull : BOOL := FALSE;
    gDiSensLow : BOOL := FALSE;
END_VAR
(*Digital output signals*)
VAR
    gDoWaterValve : BOOL := FALSE;
    gDoColorValve : BOOL := FALSE;
    gDoMixer : BOOL := FALSE;
    gDoDrainPump : BOOL := FALSE;
    gDoDrainValve : BOOL := FALSE;
END_VAR
(*Function block instance variables*)
VAR
    TON_Mixer : TON;
END_VAR
```

## Cykliczna część programu

```
(*****
* COPYRIGHT -- Bernecker + Rainer
*****
* Program: mixer
* File: mixerCyclic.st
* Author: Academy
* Created: February 04, 2015
*****
* Implementation of program mixer
*****)
```

## PROGRAM \_CYCLIC

```
(*reset all outputs - will be set in the individual states*)
gDoWaterValve := FALSE;
gDoColorValve := FALSE;
gDoMixer := FALSE;
gDoDrainValve := FALSE;
gDoDrainPump := FALSE;
TON_Mixer.IN := FALSE;

(* implementation of dispersion mixer state machine *)
CASE sMixerState OF
    (*wait until operator starts process by start button*)
    enWAIT_FOR_START:
        IF gDiStart = TRUE THEN
            sMixerState := enFILL_WATER;
```

```

        END_IF

        (*fill water into the reservoir until limit is reached*)
    enFILL_WATER:
        IF gDiWaterOK = TRUE THEN
            sMixerState := enFILL_COLOR;
        END_IF

        gDoWaterValve := TRUE;

        (*add color and mix dispersion until reservoir is full*)
    enFILL_COLOR:
        IF gDiSensFull = TRUE THEN
            sMixerState := enMIX_TIME;
        END_IF

        gDoColorValve := TRUE;
        gDoMixer := TRUE;

        (*mix color and water until time is elapsed*)
    enMIX_TIME:

        TON_Mixer.IN := TRUE;
        TON_Mixer.PT := T#30s;

        IF TON_Mixer.Q = TRUE THEN
            sMixerState := enBOTTLE_DISPERSION;
        END_IF

        gDoMixer := TRUE;

        (*continue mixing and bottle dispersion until reservoir is empty*)
    enBOTTLE_DISPERSION:
        IF gDiSensLow = TRUE THEN
            sMixerState := enWAIT_FOR_START;
        END_IF

        gDoDrainValve := TRUE;
        gDoDrainPump := TRUE;
        gDoMixer := TRUE;

    END_CASE

    (*call timer function block*)
    TON_Mixer();

END_PROGRAM

```

## Seminaria i Moduły Szkoleniowe

Automation Academy organizuje szkolenia w wybranych zagadnieniach, zarówno dla klientów B&R, jak i własnych pracowników.

**Automation Academy to najszybszy sposób na rozwinięcie własnych umiejętności!**

Nasze szkolenia pomogą Państwu wzbogacić swoją wiedzę na temat inżynierii automatyzacji.

Po ukończeniu szkolenia będą mogli Państwo wdrażać wydajnie i skutecznie rozwiązania z dziedziny automatyki, korzystając z technologii oferowanych przez B&R. Dzięki temu zyskają Państwo przewagę konkurencyjną, a Państwa firma będzie mogła szybciej reagować na stale zmieniające się wymagania rynku.



## Automation Studio szkolenia i moduły treningowe

Programowanie i konfiguracja	Diagnostyka i serwis
SEM210 - Podstawy SEM246 - IEC 61131-3 język programowa Tekst Strukturalny ST* SEM250 - Zarządzanie pamięcią i danymi	SEM920 - Diagnostyka i serwis dla użytkowników końcowych SEM920 - Diagnostyka i serwis z Automation Studio SEM950 - POWERLINK konfiguracja i diagnostyka*
SEM410 - Zintegrowany system napędowy* SEM441 - System napędowy: Elektroniczna synchronizacja i krzywki cam** SEM480 - Hydraulika** SEN1110 - Grupy osi i kontrolowanie ścieżki ruchu**	Jeżeli nie znajdziecie Państwo szkolenia odpowiedniego dla Państwa potrzeb, B&R oferuje także szkolenia niestandardowe, które możecie Państwo ustalić w porozumieniu z przedstawicielem sprzedaży. SEM099 - Szkolenie indywidualne
SEM510 - Zintegrowany system bezpieczeństwa* SEM450 - Bezpieczny system napędowy***	Po więcej informacji zachęcamy do odwiedzenia naszej strony internetowej*****: <a href="http://www.br-automation.com/academy">www.br-automation.com/academy</a>
SEM610 - Zintegrowana wizualizacja*	

### Przegląd materiałów szkoleniowych

TM210 – Praca z Automation Studio TM213 – Automation Runtime TM223 – Diagnostyka Automation Studio TM230 – Podstawy Programowania TM240 – Logika drabinkowa (LD) TM241 – Diagram Bloków Funkcyjnych (FBD) TM242 – Język Sekwencji Działania (SFC) TM246 – Tekst strukturalny (ST) TM250 – Zarządzanie Pamięcią i Danymi	TM600 – Wprowadzenie do Wizualizacji TM610 – Praca ze Zintegrowaną Wizualizacją TM630 – Wytyczne do Tworzenia Wizualizacji TM640 – Alarmy, Trendy i Diagnostyka TM670 – Zaawansowane Obiekty Wizualizacji
TM440 – Sterowanie Napędami: Podstawy TM410 – Zintegrowany System Napędów TM440 - System napędowy Podstawowe funkcje TM441 - System napędowy: Elektroniczne krzywki i profile Cam TM1110 - Zintegrowane sterowanie napędami (Grupy osi) TM1111 - Zintegrowane sterowanie napędami (Sterowanie trajektorią ruchu) TM450 - Koncepcja sterowania napędami i konfiguracja TM460 – Uruchamianie Silników	TM920 - Diagnostyka i serwis TM923 - Diagnostyka z automation studio TM950 - POWERLINK konfiguracja i diagnostyka
TM500 – System Bezpieczeństwa - Wprowadzenie TM510 – Praca z Edytorem SafeDESIGNER TM540 – Zintegrowane Napędy Bezpieczne	TM280 - Condition Monitoring for Vibration Measurement TM480 – Podstawy Hydrauliki TM481 – Valve-based Hydraulic Drives TM482 – Hydraulic Servo Pump Drives TM490 - Printing Machine Technology
	Materiały szkoleniowe oprócz wersji drukowanej są także dostępne na naszej stronie internetowej w wersji elektronicznej (wymagane logowanie):  Po więcej informacji odwiedź naszą stronę internetową <a href="http://www.br-automation.com/academy">www.br-automation.com/academy</a>

## Sterowanie procesami szkolenia i materiały szkoleniowe

Sterowanie procesami szkolenia standardowe	Sterowanie procesami materiały szkoleniowe
SEM841 - Sterowanie procesami: Basic 1 SEM842 - Sterowanie procesami: Basic 2 SEM890 - Zaawansowane rozwiązania sterowania procesami	TM800 – APROL: Koncepcja Systemu TM810 – APROL: Ustawienia, Konfiguracja i Odzyskiwanie TM811 – APROL: Runtime System TM812 – APROL: Zarządzanie Operatorami TM813 - APROL: portal web TM820 - rozwiązania APROL TM830 – APROL: Tworzenie Projektu TM835 - APROL: Konfiguracja ST-SFC TM840 - APROL – Zarządzanie Parametrami i Zasadami TM850 - APROL – Kontrola Sterownika i INA TM860 - APROL – Inżynieria Bibliotek TM865 - APROL – Przewodnik po Bibliotekach TM870 - APROL Programowanie w języku Python TM880 - APROL: Raportowanie

\* SEM 210 - Podstawy jest kursem wymagającym do przystąpienia do tego szkolenia.

\*\* SEM410 - Zintegrowany system napędowy jest wymagany do szkolenia

\*\*\* SEM410 - zintegrowany system napędowy i SEM510 - Zintegrowany system bezpieczeństwa

\*\*\*\*Wszystkie szkolenia wyszczególnione są w zakładce Akademia/Szkolenia.

\*\*\*\*\*Nazwy szkoleń mogą się różnić w każdym z państwNie wszystkie szkolenia dostępne są w każdym z państw



Sterowanie procesami szkolenia standardowe

Sterowanie procesami materiały szkoleniowe

TM890 – Podstawy Systemu LINUX







TM246TRE.00-POL

V1.0.8.2 ©2017/03/10 by B&R. Wszelkie prawa zastrzeżone.  
Wszystkie zarejestrowane znaki towarowe są własnością ich właścicieli.  
Zastrzegamy sobie prawo do zmian technicznych.