Odstępstwa od normy IEC 61131-3 w kompilatorze CPDev.STComp05 – dotyczy wersji: 0.4.11.0 (i następnych)

1. W deklaracjach zmiennych lokalnych `VAR` nie można użyć modyfikatora `AT` pozycjonującego zmienną.
2. Deklaracji zmiennej globalnej nie można zacząć od `AT`.
3. `VAR_EXTERNAL` występuje tylko w programach (nie ma ich w blokach funkcjonalnych).
4. Nie obsługiwane są ścieżki dostępu do zmiennych `VAR_ACCESS`.
5. Identyfikatory są przechowywane w przestrzeniach nazw. W każdej przestrzeni nazwa identyfikatora musi być unikalna. Każdy projekt, czy importowana biblioteka jest odrębną przestrzenią nazw. Nazwy krótkie identyfikatorów w różnych przestrzeniach nazw mogą się powtarzać. Reguły wyszukiwania identyfikatorów o krótkiej nazwie są wykonywane następująco:
   a) w bieżącej przestrzeni w której jest wywoływany,
   b) w przestrzeniach importowanych, ale tylko na ich głównym poziomie i w zależności od typu żądania (żądanie z określoną kategorią typu, żądanie z dowolną kategorią typu) sprawdzane jest również występowanie dopasowania w pozostałych przestrzeniach nazw – w przypadku znalezienia takiego dopasowania sygnalizowany jest błąd o niejednoznaczności identyfikatora (Błędy: „Ambigous ...", „Multiple name found ..." itp.). W takich przypadkach trzeba używać pełnej nazwy w stylu: `PRZESTRZEN.IDENTYFIKATOR`; Systemowe przestrzenie nazw zawierają niedozwolony znak dla identyfikatora dolar ($) np (`$DEFAULT`, `$VMSYS`). Aby dostać się do takich przestrzeni w przypadku niejednoznaczności należy użyć dyrektywy `NS` (NameSpace) np (`(*$NS $DEFAULT*)ADD(3, 4)`).
6. Inne niedziesiętne systemy liczbowe uzyskamy poprzez wpisanie: podstawy systemu (dziesiętnie), symbolu #, ciągu alfanumerycznego. Np. (`5#24` – to dziesiętnie 14); wspierane są również nie informatyczne systemy liczbowe np trójkowy, czwórkowy itp.
7. Wymuszenie typu literału numerycznego: np. `BOOL#1`, `WORD#16#55AA`. Typ literału jest (obecnie) zawsze `INT` gdy liczba nie zawiera znaku kropki (`.`) i zawsze `REAL` gdy literał numeryczny zawiera kropkę. W niektórych przypadkach może występować implikacja typu stałej.
8. Opcjonalne wejścia bloków funkcjonalnych definiujemy wtedy gdy jawnie podana jest wartość początkowa danego wejścia (i/lub dyrektywa `PAROPT` po nazwie wejścia).
9. Nie obsługiwane są elementy konfiguracji (książka Kasprzyka str. 228) czyli słowa kluczowe i ich znaczenie takie jak: `CONFIGURATION`, `RESOURCE ... ON ...` , `TASK ... (INTERVAL := ..., PRIORITY := ...)` , `PROGRAM ... WITH ... : ...` itp. Zadania deklarowane są za pomocą dyrektywy `TASKS`.
10. Interpretacje typów własnych deklarowanych za pomocą słowa kluczowego `TYPE ... END_TYPE`, zostały oprogramowane dla struktur i tablic. Pozostałe typy takie jak aliasy (np: USINT, UINT, UDINT), wyliczenia (enum) i typy okrojone nie zostały jeszcze zaimplementowane i nie zalecane jest ich używanie. Zamiast nich należy używać typów podstawowych.
11. Brakuje typu `STRING` w kompilatorze. Próba jego użycia zakończy się komunikatem o nieznanym typie.
12. Instrukcja `CASE` wymaga w definicji punktu wejścia (ciąg przed dwukropkiem) podania stałej lub stałych bezpośrednich, nie mogą być to identyfikatory zadeklarowane jako stałe lub wyrażenia dające w wyniku wartość stałą. Na etapie kompilacji nie są sprawdzane czy zakresy punktów wejścia nie pokrywają się, oraz czy definicje punktów wejścia mają porównywalne typy ze zmienną sterującą.
13. Typ pętli `FOR` określany za pomocą wyrażenia stałego następującego po słowie kluczowym `BY`, jest to jedyne miejsce w którym wymagana jest wartość stała na etapie kompilacji. Gdy

ma wartość ujemną to sprawdzane jest czy zmienna sterująca pętlą jest większa bądź równa wartości końcowej; gdy ma wartość dodatnią to sprawdzane jest czy zmienna sterująca ma wartość mniejszą bądź równą wartości końcowej. Wartość inkrementacji nie może być 0.

14. Podczas wywoływania dowolnej funkcji (użytkownika, wbudowanej, czy *in-line*) nie używamy podstawień do nazw parametrów (w przeciwieństwie do bloków funkcjonalnych). Przykładowy zapis wywołania funkcji: `X := SEL(BWY, P, 1);` bloku funkcjonalnego: `BSWI1(S:=BWY, IN1 := Pb, IN2 := TRUE);` Kolejność parametrów bloku funkcjonalnego jest dowolna. Kolejność parametrów funkcji jest ustalona przez programistę.
15. Bloki komunikacyjne `COM_SM`*x* są szczególnym przypadkiem bloków funkcjonalnych fizycznie nie wywoływanych w trakcie działania programu. Rezerwują one tylko obszar pamięci dla danych podsystemu komunikacyjnego w sterowniku SMC. Próba przypisania wyjść tego bloku bezpośrednio w wywołaniu bloku funkcjonalnego kończy się niepowodzeniem. Wejścia oznaczone jako stałe muszą mieć identyczne wartości w wywołaniu każdej instancji.