Κεφάλαιο 4: Προγραμματισμός Συστημάτων Βιομηχανικού Ελέγχου

Σύνοψη
Στο κεφάλαιο αυτό εξηγείται πώς μπορεί να οργανωθεί και διαμορφωθεί η αρχιτεκτονική του λογισμικού το οποίο υλοποιεί τον έλεγχο βιομηχανικών μονάδων σύμφωνα με το διεθνές πρότυπο IEC 61131-3 (IEC61131-3, 2013) και ανεξάρτητα από το υλικό στο οποίο θα εγκατασταθεί. Το πρότυπο αυτό προτείνει καταρχήν ένα μοντέλο δόμησης της αρχιτεκτονικής του λογισμικού αποτελούμενο από ένα αφηρημένο ορισμό των υπολογιστικών πόρων στους οποίους θα εκτελεστεί και από διεργασίες, προγράμματα, συναρτησιακές οντότητες και συναρτήσεις. Το πρότυπο υποθέτει ότι κάθε βιομηχανικό σύστημα ελέγχου δειγματοληπτεί πληθώρα ψηφιακών και αναλογικών αισθητηρίων, τα οποία μετρούν τιμές φυσικών ελεγχόμενων μεταβλητών του βιομηχανικού εξοπλισμού και παράγει εξόδους σε ενεργοποιητές και τελικά στοιχεία τα οποία ρυθμίζουν αντίστοιχες χειριζόμενες μεταβλητές. Οι τιμές όλων των παραπάνω μεταβλητών μπορούν να αντιστοιχηθούν σε διάφορους τύπους δεδομένων. Για τη συγγραφή του λογισμικού ορίζονται γλώσσες προγραμματισμού και βιβλιοθήκες έτοιμων προγραμμάτων, συναρτησιακών οντοτήτων και συναρτήσεων.

Προαπαιτούμενη γνώση
Για να μπορέσει ο αναγνώστης να κατανοήσει το περιεχόμενο του κεφαλαίου αυτού είναι απαραίτητο να έχει στοιχειώδεις γνώσεις αντικειμενοστραφούς προγραμματισμού υπολογιστών.

4.1 Εισαγωγή

Ιστορικά, οι διατάξεις που κυριάρχησαν και κυριαρχούν ακόμη στο βιομηχανικό έλεγχο για την υλοποίηση των αλγορίθμων και διαδικασιών ελέγχου ήταν και είναι οι προγραμματιζόμενοι λογικοί ελεγκτές. Οι ραγδαίες εξελίξεις της μικροηλεκτρονικής, όμως, επέτρεψαν την κατασκευή ενός μεγάλου αριθμού διαφορετικών τύπων προγραμματιζόμενων λογικών ελεγκτών, αλλά και ελεγκτών που στηρίζονται σε προσωπικούς υπολογιστές με υλικό και λογισμικό, τα οποία έχουν χαρακτηριστικά πραγματικού χρόνου. Με όλους αυτούς τους υπολογιστές αυτοματοποιούνται σήμερα εξαιρετικά πολύπλοκες βιομηχανικές διεργασίες και δομούνται μεγάλα κατανεμημένα υπολογιστικά συστήματα τα οποία αποτελούνται από σύνολα προγραμματιζόμενων λογικών ελεγκτών, προσωπικών υπολογιστών, εξειδικευμένων ελεγκτών συγκεκριμένου μηχανολογικού εξοπλισμού, όπως είναι οι εργαλειομηχανές, σταθμών εποπτείας/χειρισμών, εποπτικού ελέγχου και συλλογής δεδομένων. Αυτό το ευρύ φάσμα υλικού και λογισμικού χρειάζεται να υποστηριχτεί από κατάλληλες γλώσσες και μεθόδους προγραμματισμού για την ποιοτική αλλά και χαμηλού κόστους δημιουργία πολύπλοκου λογισμικού εφαρμογής. Οι κλασικές γλώσσες και μέθοδοι οι οποίες χρησιμοποιήθηκαν στο παρελθόν αλλά και συνεχίζουν να χρησιμοποιούνται σε μεγάλο βαθμό μέχρι σήμερα, όπως είναι οι γλώσσες των σχεδίων επαφών (Ladder Relay Logic), λίστας εντολών και συναρτησιακών οντοτήτων έχουν φθάσει στα όριά τους. Με την εξέλιξη της τεχνολογίας των υπολογιστών και ιδιαίτερα με την παρατηρούμενη αύξηση της χωρητικότητας της μνήμης άρχισε να γίνεται αντιληπτό ότι είναι δυνατή η δημιουργία πολύπλοκων αλγορίθμων ελέγχου που θα μπορούσαν να βελτιώσουν σημαντικά την επίδοση των βιομηχανικών διεργασιών. Η υλοποίηση, όμως, των αλγορίθμων αυτών με τις κλασικές γλώσσες και μεθόδους δεν είναι εφικτή. Οι χρήστες θέλουν ομοιόμορφες, ανεξάρτητες από τον κατασκευαστή, υψηλού επιπέδου γλώσσες προγραμματισμού και εργαλεία ανάπτυξης λογισμικού όμοια με αυτά που υπάρχουν σήμερα στον κόσμο των προσωπικών υπολογιστών. Όπως προαναφέρθηκε, οι εντολές του προγράμματος στους ελεγκτές αυτούς εκτελούνται με μια καθορισμένη σειρά αρχίζοντας από την πρώτη εντολή. Κατά τη διάρκεια σάρωσης των εντολών του προγράμματος αν και υπολογίζονται οι τιμές των εξόδων, αυτές εγγράφονται στη μνήμη και μόνο όταν ολοκληρωθεί η σάρωση του προγράμματος, οι τιμές αυτές μεταφέρονται στις φυσικές εξόδους του ελεγκτή. Αυτός ο απλός κυκλικός τρόπος εκτέλεσης του προγράμματος κάνει τη συχνότητα εκτέλεσής του να εξαρτάται από το μήκος και την πολυπλοκότητά του. Η προσθήκη νέων εντολών επιμηκύνει το χρόνο σάρωσης του προγράμματος, γεγονός που μπορεί να κάνει αδύνατη την εκτέλεση του νέου προγράμματος και την ενεργοποίηση των φυσικών εξόδων μέσα στο χρονικό όριο που απαιτεί η εφαρμογή. Η υλοποίηση, όμως, των αλγορίθμων αυτών θα μπορούσε να γίνει από υπολογιστικά μέσα που θα διατηρούσαν τα χαρακτηριστικά εκείνα των προγραμματιζόμενων λογικών ελεγκτών που έχουν σχέση με την εξασφάλιση του χρόνου εκτέλεσης ενός προγράμματος εντός προβλεπτών ορίων, αλλά θα μπορούσαν ταυτόχρονα να επιτρέψουν στον προγραμματιστή να ελέγξει τις συχνότητες

εκτέλεσης των διαφόρων ακολουθιών εντολών του προγράμματος και ενεργοποίησης των εξόδων. Ακόμη μέσω της χρήσης υπολογιστών με περισσότερους του ενός επεξεργαστές, ο προγραμματιστής θα μπορούσε να αξιοποιήσει τις δυνατότητες της παράλληλης εκτέλεσης ακολουθιών εντολών ενός προγράμματος και να προγραμματίσει την καταγραφή ασύγχρονων γεγονότων που χρήζουν άμεσης εξυπηρέτησης διακόπτοντας την κυκλική εκτέλεση του κύριου κορμού του προγράμματος. Τέτοιοι υπολογιστές έχουν αρχίσει να διατίθενται εμπορικά τόσο από τους ίδιους τους κατασκευαστές των προγραμματιζόμενων λογικών ελεγκτών όσο και από άλλους κατασκευαστές που συνηθίζουν να τους αποκαλούν **υπολογιστές πραγματικού χρόνου** ή **βιομηχανικούς υπολογιστές**. Η ανάπτυξη, όμως, του λογισμικού των πολύπλοκων αλγορίθμων σε υπολογιστές με περισσότερο πολύπλοκη αρχιτεκτονική υλικού και λειτουργικού συστήματος από αυτή των κλασικών προγραμματιζόμενων ελεγκτών, με τη χρήση της γλώσσας των σχεδίων επαφών παρουσιάζει σημαντικά προβλήματα.. Είναι συχνό το φαινόμενο να μην μπορεί κανείς να συντάξει καλά δομημένα προγράμματα διότι οι κλασικές γλώσσες δεν υποστηρίζουν υπορουτίνες, διαδικασίες ή η υποστήριξη περιορίζεται στη δυνατότητα δημιουργίας ενός πολύ περιορισμένου αριθμού τέτοιων λογισμικών οντοτήτων. Ως αποτέλεσμα αυτής της αδυναμίας καθίσταται πολύ δύσκολη η διάσπαση ενός μεγάλου προγράμματος σε μικρότερα τμήματα με σαφείς διεπαφές μεταξύ τους. Κατά συνέπεια είναι πολύ δύσκολο π.χ. για ένα τμήμα ενός προγράμματος σε γλώσσα σχεδίων επαφών να διαβάσει και να αλλάξει τις καταστάσεις επαφών και εξόδων που χρησιμοποιούνται σε άλλο τμήμα του προγράμματος, δηλαδή είναι πολύ δύσκολο να επιτύχουμε συνένωση μικρότερων προγραμμάτων σε μεγαλύτερες οντότητες προσβάσιμες από άλλες οντότητες λογισμικού μόνο στο επίπεδο των δεδομένων.

Ένα άλλο πρόβλημα που συναντά συχνά ο συντάκτης προγραμμάτων σε γλώσσα σχεδίων επαφών είναι η δημιουργία προγράμματος μεγάλου μήκους, η διαχείριση του οποίου είναι δύσκολη διότι για την υλοποίηση των ίδιων περίπου λογικών πράξεων χρειάζεται να γίνει επανάληψη συνόλων εντολών με μικρές διαφοροποιήσεις μεταξύ τους. Ένα τέτοιο πρόγραμμα θα ήταν αυτό που ανιχνεύει την εμφάνιση φωτιάς σε εκατοντάδες χώρους μιας βιομηχανικής εγκατάστασης. Γράφοντας μια φορά ένα πρόγραμμα υπό τη μορφή μιας ενιαίας οντότητας που να μπορεί να χρησιμοποιείται πολλές φορές χωρίς να προστίθεται ο κώδικάς του στον κύριο κορμό του προγράμματος κάθε φορά που χρειάζεται, θα μείωνε σημαντικά το μήκος του συνολικού προγράμματος εφαρμογής.

Άλλο ένα πρόβλημα που παρουσιάζει η γλώσσα των σχεδίων επαφών είναι η περιορισμένη ή ανύπαρκτη υποστήριξη δομών δεδομένων. Συνήθως τα δεδομένα αποθηκεύονται και διαβάζονται διευθυνσιοδοτώντας μεμονωμένα ψηφία καταχωρητών ή ολόκληρους καταχωρητές. Μολονότι αυτή η δυνατότητα είναι θετική διότι επιτρέπει το χειρισμό διακριτών εισόδων/εξόδων και σημαιών με ευκολότερο τρόπο από αυτόν των γνωστών δομημένων γλωσσών, ωστόσο με την επέκταση της χρήσης των προγραμματιζόμενων λογικών ελεγκτών σε πολύπλοκες εφαρμογές όπου δεν υπάρχει η δυνατότητα ομαδοποίησης δεδομένων σε δομές, καθίσταται η γλώσσα δύσχρηστη. Σαν παράδειγμα θεωρείστε ένα αισθητήριο πίεσης που συνδέεται σε διακριτή είσοδό του. Το σύστημα διαχείρισης των εισόδων του θα δεσμεύσει ένα απλό δυαδικό ψηφίο σε έναν καταχωρητή μνήμης, η εκάστοτε λογική κατάσταση του οποίου θα αντικατοπτρίζει την ενεργοποίηση ή μη του αισθητηρίου. Με ένα απλό ψηφίο, όμως, δεν είναι δυνατόν να αποτυπωθούν όλες οι άλλες καταστάσεις του αισθητηρίου. Η εφαρμογή μπορεί να απαιτεί την απομόνωση του αισθητηρίου, την τοποθέτησή του σε κατάσταση δοκιμής, την καταγραφή της χρονικής στιγμής που ενεργοποιείται και τη διέγερση μιας προειδοποιητικής ένδειξης, όταν το αισθητήριο παραμένει ενεργοποιημένο για κάποια χρονική περίοδο. Όλα τα επιπλέον δεδομένα που σχετίζονται με την παραπάνω λειτουργία του αισθητηρίου ιδανικά θα πρέπει να αποθηκεύονται υπό τη μορφή μιας ενιαίας δομής δεδομένων που θα μπορεί να διευθυνσιοδοτείται με αναφορά σε ένα μοναδικό όνομα. Στις κλασικές γλώσσες προγραμματισμού των λογικών ελεγκτών τα δεδομένα αυτά είναι τοποθετημένα σε θέσεις διάσπαρτες μέσα στη μνήμη έτσι ώστε η πιθανότητα να γίνει αναφορά σε δεδομένα που αφορούν άλλο αισθητήριο να είναι μεγάλη.

Σε πολλές βιομηχανικές εφαρμογές τα προγράμματα χρειάζεται να ακολουθήσουν μία ή περισσότερες ακολουθίες εντολών ανάλογα με τις καταστάσεις των ελεγχόμενων φυσικών διεργασιών. Σαν παράδειγμα μπορεί να θεωρήσει κανείς μια μεταφορική ταινία μιας γραμμής παραγωγής προϊόντων. Το πρόγραμμα μπορεί να χρειαστεί να ελέγξει στην αρχή την κατάσταση στην οποία βρίσκεται κάποιος βοηθητικός εξοπλισμός και στη συνέχεια να εκκινήσει τους κινητήρες της ταινίας τον ένα μετά τον άλλο. Μετά το πρόγραμμα μπορεί να χρειαστεί να ελέγξει το κατά πόσο η ταινία έχει αρχίσει να κινείται με την επιθυμητή ταχύτητα και να εκκινήσει άλλες μηχανές και διατάξεις στη γραμμή παραγωγής. Ο συμβατικός τρόπος σύνταξης ενός τέτοιου προγράμματος είναι να αντιστοιχηθεί σε κάθε κατάσταση της ακολουθίας εντολών και σε κάθε αιτία που προκαλεί την αλλαγή της μιας κατάστασης σε άλλη ένα δυαδικό ψηφίο. Όμως με μια τέτοια προσέγγιση, όταν ο αριθμός των ακολουθιών εντολών είναι μεγάλος, δημιουργείται πρόβλημα συντήρησης και αναγνωσιμότητας

του προγράμματος. Η δυσκολία της αναγνωσιμότητας προκύπτει από το γεγονός ότι η λογική ελέγχου της φυσικής διεργασίας εμπλέκεται με τη λογική ελέγχου των ακολουθιών των εντολών.

Τελικά, η εκτέλεση αριθμητικών πράξεων είναι δύσκολη. Η μέθοδος που υιοθετήθηκε από τους περισσότερους κατασκευαστές είναι η χρήση του γραφικού συμβόλου του αριθμητικού κουτιού μέσα στο οποίο δηλώνεται η μαθηματική πράξη και οι καταχωρητές ή μεταβλητές στις οποίες είναι αποθηκευμένες οι τιμές των τελεστών της πράξης. Η χρονική στιγμή εκτέλεσης της πράξης προσδιορίζεται από την ενεργοποίηση λογικής που προτάσσεται του αριθμητικού συμβόλου.

Προκειμένου να αρθούν οι παραπάνω περιορισμοί αναπτύχθηκε το πρότυπο IEC 61131-3 (IEC61131-3, 2013). Το πρότυπο αυτό προτείνει ένα μοντέλο δόμησης της αρχιτεκτονικής κάθε εφαρμογής λογισμικού που αφορά τον έλεγχο βιομηχανικών διεργασιών και γλώσσες που μπορούν να υποστηρίξουν το μοντέλο αυτό. Τόσο το μοντέλο όσο και οι γλώσσες υποστηρίζουν:

- Μια καλά δομημένη από πάνω προς τα κάτω ή από κάτω προς τα επάνω οργάνωση του προγράμματος.
- Αυστηρή χρήση τύπων δεδομένων, έτσι ώστε να είναι δυνατή η ανίχνευση σε επίπεδο σημαντικής ανάλυσης του μεταγλωττιστή η εγγραφή από τον προγραμματιστή λανθασμένων δεδομένων σε μια μεταβλητή.
- Δυνατότητα εκτέλεσης τμημάτων του προγράμματος σε διαφορετικές χρονικές στιγμές, με διαφορετικούς ρυθμούς και παράλληλα σε περισσότερους του ενός επεξεργαστές,
- Περιγραφή με γραφικό τρόπο πολλών και διαφορετικών ακολουθιών εντολών.
- Τη δημιουργία δομών δεδομένων.
- Επιλογή διαφορετικών γλωσσών προγραμματισμού για τη σύνταξη τμημάτων ενός ενιαίου προγράμματος εφαρμογής, επιτρέποντας έτσι τη χρήση της πλέον κατάλληλης γλώσσας για καθένα από τα επιμέρους προβλήματα βιομηχανικού ελέγχου που αντιμετωπίζονται από το συνολικό λογισμικό της εφαρμογής.
- Ενθυλάκωση στοιχείων λογισμικού σε ενιαίες οντότητες (software encapsulation), και περιορισμό της πρόσβασης μιας οντότητας στα δεδομένα και διαδικασίες μιας άλλης οντότητας (information hiding).

Γενικά η συμμόρφωση με τις υποδείξεις του προτύπου θα μπορούσε να οδηγήσει στην ανάπτυξη λογισμικών οντοτήτων οι οποίες υλοποιούν τυπικές λειτουργίες και αλγορίθμους βιομηχανικού ελέγχου με δυνατότητα μεταφοράς και εκτέλεσής τους σε συστήματα διαφορετικών κατασκευαστών. Στην αγορά έχουν, ήδη, εμφανιστεί προϊόντα μηχανικής λογισμικού τα οποία στηρίζονται στις αρχές του προτύπου αυτού και μπορούν να παράγουν εκτελέσιμα προγράμματα στις περισσότερες από τις προτεινόμενες γλώσσες για περισσότερα του ενός υπολογιστικά μέσα είτε αυτά είναι προγραμματιζόμενοι ελεγκτές ή υπολογιστές πραγματικού χρόνου. Επίσης αυτά τα εργαλεία μηχανικής λογισμικού μπορούν να παράγουν εκτελέσιμα προγράμματα για προσωπικούς υπολογιστές παρόλο που τα λειτουργικά συστήματα των Windows και Linux δεν ενδείκνυνται για τέτοιου είδους εφαρμογές.

Στη συνέχεια το βιβλίο αυτό επικεντρώνεται στην παρουσίαση αυτού του μοντέλου, των γλωσσών καθώς επίσης και των εργαλείων μηχανικής λογισμικού με τα οποία μπορεί κάποιος να αναπτύξει, να αποσφαλματώσει και να εκτελέσει σε προγραμματιζόμενους λογικούς ελεγκτές, σε υπολογιστές πραγματικού χρόνου και σε προσωπικούς υπολογιστές λογισμικό γραμμένο σύμφωνα με τις υποδείξεις του προτύπου. Για περισσότερη εμβάθυνση στη μεθοδολογία προγραμματισμού και στον προγραμματισμό συστημάτων βιομηχανικού ελέγχου, ο αναγνώστης μπορεί να αναφερθεί στη βιβλιογραφία που παρατίθεται στο τέλος του κεφαλαίου αυτού (Bonfati, Monari, & Sampieri, 1997), (Kart-Heinz & Tiegelkamp, 2010), (Lewis, 1995).

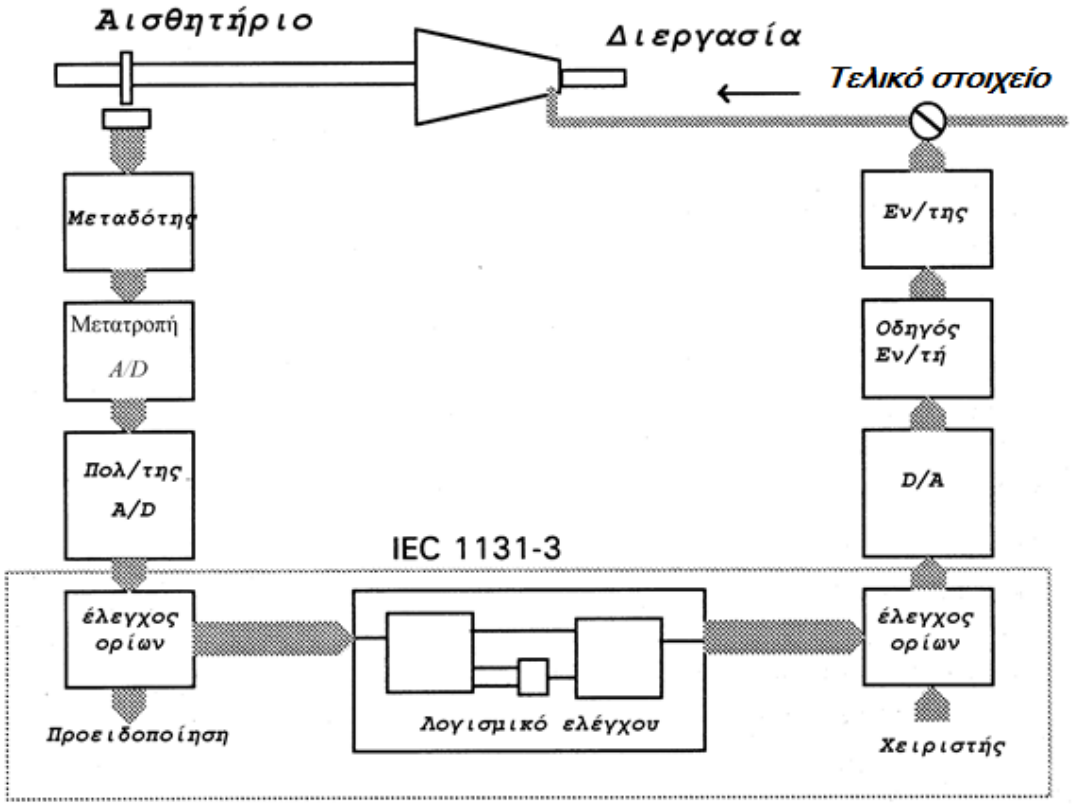
4.2 Το μοντέλο λογισμικού του προτύπου IEC 61131-3

Ένα πρόγραμμα ή λογισμικό που αφορά το χώρο εφαρμογών του βιομηχανικού ελέγχου διαφέρει από ένα συμβατικό πρόγραμμα, π.χ. επεξεργασίας κειμένου ή επίλυσης μιας διαφορικής εξίσωσης, ως προς το ότι πρέπει να αλληλοεπιδρά με το φυσικό περιβάλλον. Προφανώς δεν είναι δυνατόν να ορίσει κάποιος τη δομή ή την αρχιτεκτονική ενός τέτοιου προγράμματος χωρίς να λάβει υπόψη του, να κατανοήσει και να προσδιορίσει τον τρόπο επικοινωνίας και τις αλληλεπιδράσεις του προγράμματος τόσο με το φυσικό περιβάλλον όσο και με το λογισμικό συστήματος από το οποίο είτε δανείζεται υπηρεσίες ή μέσω του οποίου ρυθμίζεται ο

τρόποεκτέλεσής του. Σύμφωνα με τα όσα εκτέθηκαν και στα προηγούμενα κεφάλαια του βιβλίου είτε το λογισμικό συστήματος είτε το λογισμικό εφαρμογής πρέπει να διαθέτουν:

- Διεπαφές Εισόδων/Εξόδων οι οποίες πραγματοποιούν :
  - (α) την ανάγνωση τιμών από τις αναλογικές, ψηφιακές και διακριτές εισόδους του υπολογιστή στις οποίες συνδέονται τα αισθητήρια διαφόρων τύπων, όπως είναι οι μικρό-διακόπτες, οι μετατροπείς πίεσης σε ηλεκτρικά σήματα, τα θερμοζεύγη, κ.ά.,
  - (β) την ανανέωση των τιμών των εξόδων του υπολογιστή που συνδέονται με τους ενεργοποιητές των τελικών στοιχείων των χειριζόμενων μεταβλητών, όπως είναι οι οδηγοί βαλβίδων, σωληνοειδείς βαλβίδες, σερβομηχανισμοί, θερμαντήρες, κ.ά.
- Διεπαφές επικοινωνίας για την ανταλλαγή δεδομένων με τους άλλους υπολογιστές με τους οποίους έχει σφικτή ή χαλαρή σύνδεση ο υπολογιστής ο οποίος φιλοξενεί το λογισμικό εφαρμογής, καθώς επίσης και με σταθμούς ή μιμικά διαγράμματα απεικόνισης και χειρισμού της κατάστασης της ελεγχόμενης διεργασίας.
- Διεπαφές συστήματος για την αρχικοποίηση, την εκτέλεση και τη διακοπή της εκτέλεσης του λογισμικού εφαρμογής.

Το πρότυπο υποθέτει ότι οι τιμές των εξωτερικών αισθητηρίων είναι διαθέσιμες σε θέσεις μνήμης του συγκεκριμένου υπολογιστή. Επίσης, υποθέτει ότι οι τιμές εξόδου που υπολογίζονται για την οδήγηση ενεργοποιητών και ενδείξεων, ανανεώνουν τα περιεχόμενα συγκεκριμένων θέσεων της μνήμης του υπολογιστή. Η αντιστοίχιση θέσεων μνήμης σε φυσικές διατάξεις που είναι συνδεδεμένες με τις εισόδους και εξόδους του υλικού του υπολογιστή δεν καθορίζεται στο πρότυπο και είναι διαφορετική για υπολογιστές διαφορετικών κατασκευαστών. Στην Εικόνα 4.1 παρουσιάζεται μια ολοκληρωμένη εικόνα των οργάνων και εξαρτημάτων τα οποία απαιτούνται για την υλοποίηση ενός κλειστού βρόχου ελέγχου με υπολογιστή που έχει προγραμματισθεί σύμφωνα με το πρότυπο IEC-61131. Τα στοιχεία του βρόχου που καλύπτει η περιγραφή του προτύπου είναι αυτά που περιέχονται μέσα στο πλαίσιο με διακεκομμένη γραμμή.



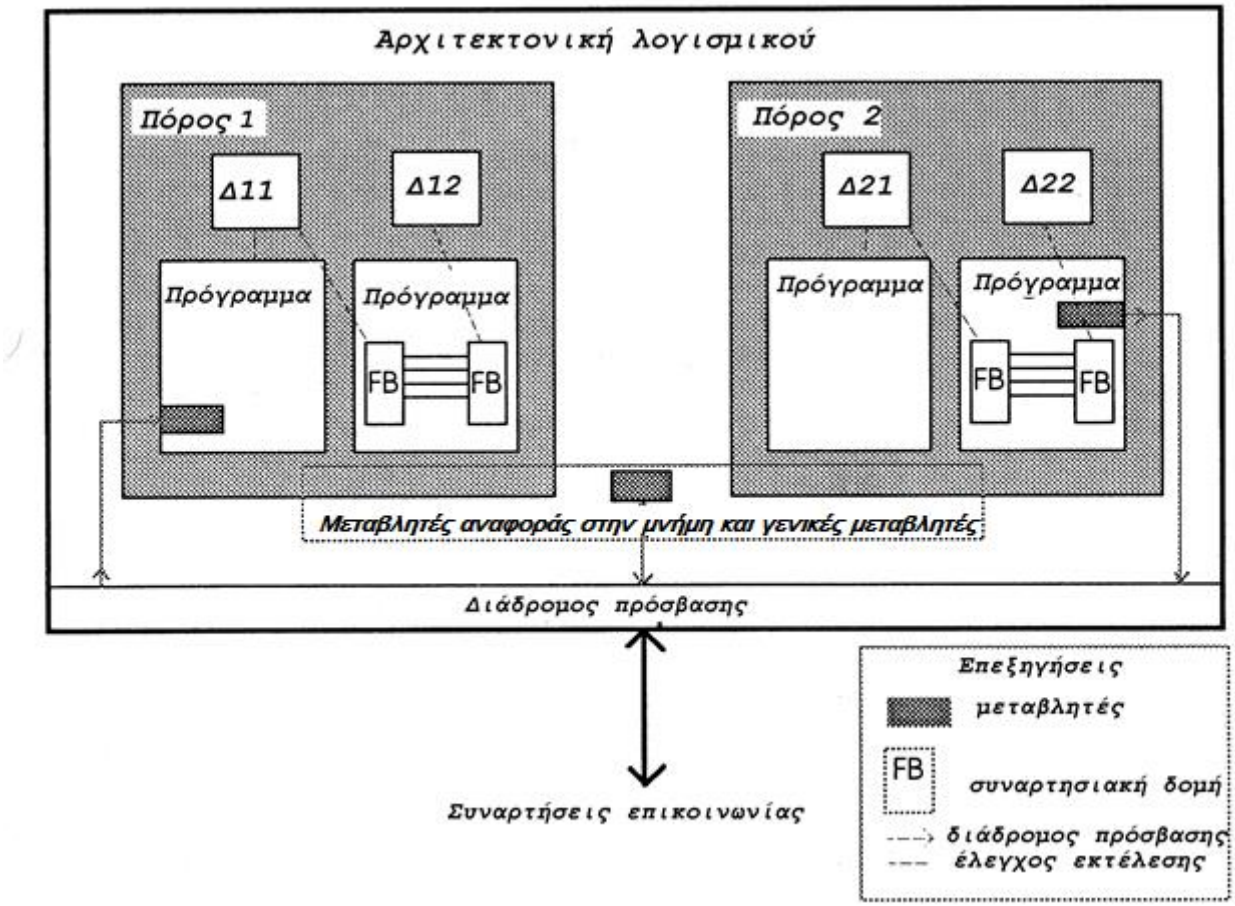
Εικόνα 4.1 *Βρόχος ελέγχου με υπολογιστή ανοικτής αρχιτεκτονικής*

Το μοντέλο έχει ιεραρχική δομή. Κάθε επίπεδο της ιεραρχίας αποκρύπτει πολλά από τα χαρακτηριστικά των επιπέδων που βρίσκονται κάτω από αυτό.

4.3 Αρχιτεκτονική λογισμικού

Μέχρι τώρα ο όρος λογισμικό και πρόγραμμα χρησιμοποιήθηκαν σαν ταυτόσημες έννοιες. Στο πρότυπο IEC 61131-3 (IEC61131-3, 2013) οι έννοιες αυτές διαφοροποιούνται. Το **πρόγραμμα** ορίζεται σαν μια οντότητα που μπορεί να εμπεριέχεται στο λογισμικό εφαρμογής και η οποία από μόνη της δεν μπορεί να εκτελεστεί αλλά πρέπει να κληθεί από μια άλλη οντότητα, τη **διεργασία.** Αυτή μπορεί να ελέγχει την περιοδική ή τη μοναδική εκτέλεση ενός συνόλου προγραμμάτων και/η συναρτησιακών οντοτήτων. Η έννοια της **συναρτησιακής οντότητας** έχει εξηγηθεί στο κεφάλαιο 2. Το λογισμικό είναι ένα ευρύτερο σύνολο διεργασιών, προγραμμάτων, συναρτησιακών οντοτήτων και του περιβάλλοντος εκτέλεσης του συνόλου των οντοτήτων αυτών. Το περιβάλλον εκτέλεσης είναι μια απλοποιημένη σύλληψη της δομής του υλικού του υπολογιστή, όπως αυτή παρουσιάζεται στο χρήστη από το λειτουργικό σύστημα και συνιστά έναν **πόρο (resource) του υπολογιστή**. Ως πόρος μπορεί να θεωρηθεί κάθε διεπαφή του λειτουργικού συστήματος μεταξύ χρήστη και υλικού μέσω της οποίας μπορεί να εκτελεστεί ένα πρόγραμμα και να επικοινωνήσει με τις φυσικές εισόδους και εξόδους (I/O) του υπολογιστή. Ένα πρόγραμμα ή μια διεργασία δεν μπορούν να εκτελεστούν χωρίς να έχει ανατεθεί η εκτέλεσή τους σε κάποιον πόρο του υπολογιστή. Συνήθως ένας πόρος θα προϋπάρχει μέσα στον υπολογιστή, αλλά δεν αποκλείεται και η δυνατότητα να προσφέρεται ένας πόρος από ένα άλλο σύστημα. Παραδείγματος χάριν, ένας πόρος μπορεί να προσομοιωθεί από έναν προσωπικό υπολογιστή για να υποστηρίξει τη δοκιμαστική εκτέλεση ενός προγράμματος. Το ενδιαφέρον χαρακτηριστικό της έννοιας του πόρου είναι ότι, ενώ είναι μια οντότητα του λογισμικού συστήματος, αντανακλά τη φυσική δομή του υπολογιστή όπου είναι εγκατεστημένο. Σαν παράδειγμα αναφέρεται το γεγονός ότι το λογισμικό συστήματος ενός υπολογιστή με μια φυσική κάρτα επεξεργαστή μπορεί να έχει σχεδιαστεί έτσι ώστε να παρουσιάζει στο χρήστη δυο πόρους για εκτέλεση προγραμμάτων εφαρμογής. Το αντίθετο μπορεί να συμβαίνει στην περίπτωση υπολογιστή με περισσότερους του ενός επεξεργαστές όπου το λογισμικό συστήματος μπορεί να παρουσιάζει στο χρήστη περιβάλλον εκτέλεσης του προγράμματος με έναν πόρο.

Η αρχιτεκτονική του λογισμικού εφαρμογής είναι μια περιγραφή των οντοτήτων από τις οποίες αποτελείται, του τρόπου με τον οποίο επικοινωνούν οι οντότητες αυτές τόσο μεταξύ τους όσο και με οντότητες του λογισμικού του συστήματος του υπολογιστή στον οποίο φιλοξενείται το λογισμικό εφαρμογής και καθορίζει τον πόρο στον οποίο θα εκτελεστεί. Η αρχιτεκτονική αυτή μπορεί να επικοινωνεί με άλλες αρχιτεκτονικές λογισμικού άλλων υπολογιστών με καθορισμένες διεπαφές που προδιαγράφονται με τη χρήση, επίσης, συγκεκριμένων εντολών γλώσσας προγραμματισμού. Στην Εικόνα 4.2 παρουσιάζεται γραφικά η αρχιτεκτονική λογισμικού εφαρμογής με δύο πόρους στο καθέναν από τους οποίους έχει ανατεθεί ο έλεγχος της εκτέλεσης δύο προγραμμάτων από δυο ανεξάρτητες διεργασίες.



Εικόνα 4.2*: Μοντέλο λογισμικού σύμφωνα με το πρότυπο IEC 61131-3.*

Ένα πρόγραμμα μπορεί να δομηθεί από οντότητες οι οποίες μπορεί να έχουν γραφεί σε διαφορετικές γλώσσες η καθεμία. Τυπικά ένα πρόγραμμα αποτελείται από έναν αριθμό διασυνδεόμενων συναρτησιακών οντοτήτων (function blocks) οι οποίες μπορούν να ανταλλάσσουν δεδομένα μέσω λογικών συνδέσεων. Ένα πρόγραμμα μπορεί να διαβάσει και να γράψει σε μεταβλητές I/O καθώς και να επικοινωνήσει με άλλα προγράμματα του ίδιου λογισμικού. Η εκτέλεση των διαφορετικών τμημάτων ενός προγράμματος, π.χ. επιλεγμένες συναρτησιακές οντότητες, μπορεί να ελέγχεται από τις διεργασίες (tasks).

Στο μοντέλο που έχει περιγραφεί δεν υπάρχουν κρυμμένοι ή υπονοούμενοι μηχανισμοί για την εκτέλεση των προγραμμάτων. Με άλλα λόγια, ένα πρόγραμμα ή συναρτησιακή οντότητα θα παραμένει ανεκτέλεστη εκτός και αν (α) ανατεθεί η εκτέλεση σε μια διεργασία, (β) η διεργασία έχει προγραμματιστεί να εκτελείται περιοδικά ή η εκτέλεσή της έχει ενεργοποιηθεί από την αλλαγή της κατάστασης μιας μεταβλητής. Θα πρέπει, όμως, να σημειωθεί ότι στην περίπτωση των συναρτησιακών οντοτήτων, όταν αυτές δε σχετίζονται με μια συγκεκριμένη διεργασία, θα εκτελούνται πάντοτε από τη διεργασία που εκτελεί το πρόγραμμα στο οποίο περιέχονται.

Η χρήση της συναρτησιακής οντότητας είναι η βάση του μοντέλου του προτύπου IEC 61131-3 (IEC61131-3, 2013) Χρησιμοποιώντας συναρτησιακές οντότητες είναι δυνατόν να δομηθούν πολύ καλά δομημένα και αληθινά ιεραρχημένα προγράμματα. Μια συναρτησιακή οντότητα έχει δύο χαρακτηριστικά: (1) τις παραμέτρους εισόδου/εξόδου οι οποίες μπορούν να χρησιμοποιηθούν ως συνδέσεις με άλλες συναρτησιακές οντότητες και μεταβλητές, (2) τον αλγόριθμο, που είναι κώδικας προγράμματος ο οποίος εκτελείται κάθε φορά που καλείται η σχετική συναρτησιακή οντότητα

Ο αλγόριθμος επεξεργάζεται τις τιμές που έχουν οι παράμετροι εισόδου και μεταβλητές της συναρτησιακής οντότητας, για να παράξει νέες τιμές για τις παραμέτρους εξόδου. Μερικές φορές ο αλγόριθμος

ενδέχεται να δημιουργήσει νέες τιμές για εσωτερικές μεταβλητές οι οποίες μπορεί να είναι τοπικές στη συγκεκριμένη συναρτησιακή οντότητα ή γενικές αν έχουν δηλωθεί σε πρόγραμμα, πόρο ή στο επίπεδο της αρχιτεκτονικής του λογισμικού, όπως θα εξηγηθεί σε επόμενη ενότητα. Επειδή κάθε συναρτησιακή οντότητα μπορεί να αποθηκεύει τιμές, χαρακτηρίζεται εξαιτίας αυτής της ιδιότητας ως οντότητα που μπορεί να βρίσκεται σε μια κατάσταση. Το πρότυπο καθορίζει ένα ρεπερτόριο τυπικών συναρτησιακών οντοτήτων, οι οποίες πρέπει να περιλαμβάνονται σε κάθε περιβάλλον ανάπτυξης λογισμικού εφαρμογής. Τέτοιες οντότητες θα μπορούσαν να είναι τα προγράμματα που υλοποιούν τις λειτουργίες συνδυαστικών και ακολουθιακών λογικών κυκλωμάτων, όπως είναι τα δισταθή στοιχεία R-S, μετρητές, ρολόγια πραγματικού χρόνου και άλλα σχετικά κυκλώματα. Το σημαντικότερο, όμως, είναι ότι το πρότυπο επιτρέπει στο σχεδιαστή λογισμικού να δημιουργήσει νέες διαφορετικές συναρτησιακές οντότητες από αυτές που διατίθενται σε βιβλιοθήκη. Το πρότυπο καθορίζει ακόμα και τις συναρτήσεις που πρέπει να περιλαμβάνονται σε περιβάλλοντα ανάπτυξης λογισμικού. Η **συνάρτηση** σε αντιδιαστολή με τη συναρτησιακή οντότητα είναι μια άλλη οντότητα λογισμικού η οποία εκτελείται με ένα συγκεκριμένο σύνολο τιμών εισόδων, παράγει ωστόσο ένα και μοναδικό αποτέλεσμα. Τα πιο αντιπροσωπευτικά παραδείγματα συναρτήσεων είναι οι μαθηματικές συναρτήσεις υπολογισμού ημίτονου, SIN( ), και συνημίτονου, COS( ). Η διαφορά μεταξύ της έννοιας της συνάρτησης και της συναρτησιακής δομής είναι ότι η συνάρτηση δεν αποθηκεύει τιμές εσωτερικών μεταβλητών, δε χαρακτηρίζεται, δηλαδή, από μια κατάσταση. Επομένως, για συγκεκριμένες τιμές εισόδου παράγει πάντοτε το ίδιο αποτέλεσμα. Αντίθετα η συναρτησιακή οντότητα χαρακτηρίζεται από την κατάσταση στην οποία βρίσκεται όταν επεξεργάζεται δεδομένα εισόδου. Σε διαδοχικές εκτελέσεις του κώδικα της με τις ίδιες τιμές εισόδων παράγει διαφορετικές τιμές εξόδου. Τυπικό παράδειγμα συναρτησιακής οντότητας που παράγει διαφορετικές τιμές εξόδου σε διαδοχικές χρονικές στιγμές όταν οι τιμές εισόδου παραμένουν οι ίδιες, είναι η συναρτησιακή οντότητα της γεννήτριας γραμμικά μεταβαλλόμενης τάσης (ramp). Αν και οι τιμές εισόδου που προσδιορίζουν την κλίση της γραμμής και τη τελική τιμή είναι αμετάβλητες, κάθε φορά που εκτελείται η συναρτησιακή οντότητα η έξοδός της αυξάνει κατά μια μικρή ποσότητα έως ότου φθάσει στην τελική τιμή.

Για τον ορισμό και την περιγραφή της αρχιτεκτονικής του λογισμικού εφαρμογής, το πρότυπο προτείνει εντολές ανεξάρτητες από τις εντολές της κάθε γλώσσας που υποστηρίζει, με τις οποίες μπορούν να ορίζονται οι οντότητες της αρχιτεκτονικής, η ανάθεση της εκτέλεσής τους σε πόρους του υπολογιστή και ο καθορισμός της ακολουθίας εκτέλεσής τους από τις διεργασίες. Γενικά το πρότυπο ονομάζει τις οντότητες του πόρου, διεργασίας, προγράμματος, συναρτησιακής οντότητας και συνάρτησης **"μονάδες οργάνωσης λογισμικού (program organization units ή POU)"** και θεωρεί ότι σε κάθε επίπεδο του λογισμικού, είτε αυτό είναι το επίπεδο της αρχιτεκτονικής είτε είναι το επίπεδο μιας μονάδας οργάνωσης λογισμικού, υποστηρίζονται οι ίδιοι τύποι δεδομένων και μεταβλητών. Όλοι αυτοί οι τύποι δεδομένων μπορούν να ορίζονται με τα ίδια σύνολα χαρακτήρων, τις ίδιες δηλωτικές λέξεις, τον ίδιο τρόπο παράθεσης επεξηγηματικών σχολίων με τις ίδιες εντολές τόσο σε επίπεδο αρχιτεκτονικής όσο και σε επίπεδο μονάδας οργάνωσης λογισμικού. Τέλος, με τις γλώσσες προγραμματισμού που προτείνονται στο πρότυπο, μπορούν να συντάσσονται μόνον οι μονάδες οργάνωσης λογισμικού των προγραμμάτων, συναρτησιακών οντοτήτων και συναρτήσεων.

Στο κείμενο που ακολουθεί περιγράφονται οι τύποι δεδομένων και μεταβλητών που υποστηρίζονται από το πρότυπο και η σημασιολογία και οι συντακτικοί κανόνες των εντολών ορισμού της αρχιτεκτονικής του λογισμικού, ορισμού τύπου δεδομένων και μεταβλητών και καθορισμού των διεργασιών.

4.4 Σύνολο χαρακτήρων

Για να εξασφαλιστεί η δυνατότητα μεταφοράς ενός προγράμματος από ένα σύστημα σε άλλο, κάθε πληροφορία υπό μορφή κειμένου θα πρέπει να χρησιμοποιεί ένα συγκεκριμένο και περιορισμένο σύνολο γραμμάτων της αλφαβήτου, αριθμητικών ψηφίων και άλλων συμβόλων. Το σύνολο αυτό καθορίζεται από το πρότυπο ISO 646 "Basic code table" (ISO/IEC 546, 1991) _και περιέχει όλα τα γράμματα, σύμβολα, αριθμούς και χαρακτήρες που περιλαμβάνονται στο πρότυπο ASCII και χρησιμοποιούνται στους προσωπικούς υπολογιστές. Το πρότυπο, όμως, προσφέρει εναλλακτικές λύσεις στις περιπτώσεις των χαρακτήρων # και €, ενώ όταν σε ένα εθνικό σύνολο γραμματοσειρών ο χαρακτήρας της κάθετης γραμμής (|) δεν προβλέπεται, αυτός μπορεί να αντικατασταθεί με το σύμβολο του θαυμαστικού (!). Η χρήση, όμως, ειδικών εθνικών χαρακτήρων δεν επιτρέπεται αλλά μπορούν να περιληφθούν σαν επέκταση του προτύπου αυτού στο συγκεκριμένο εθνικό σύνολο χαρακτήρων. Η χρήση κεφαλαίων και μικρών γραμμάτων δε διαφοροποιεί το όνομα οποιασδήποτε μεταβλητής ή μονάδας οργάνωσης προγράμματος παρά μόνον όταν χρησιμοποιούνται για να ορίσουν ακολουθία χαρακτήρων (string) ή σχόλια που θα εκτυπωθούν.

4.5 Αναγνωριστικά

Προκειμένου να καθοριστεί το όνομα διαφόρων στοιχείων στις γλώσσες προγραμματισμού, χρησιμοποιούνται αναγνωριστικές ακολουθίες χαρακτήρων για να δοθούν ονόματα σε μεταβλητές, τύπους δεδομένων, συναρτησιακές οντότητες και προγράμματα. Ένα αναγνωριστικό μπορεί να είναι μια ακολουθία χαρακτήρων, ψηφίων και υπογραμμίσεις, με τους περιορισμούς ότι:

(α) Ο πρώτος χαρακτήρας δεν μπορεί να είναι αριθμητικό ψηφίο

(β) Δεν υπάρχουν δύο διαδοχικοί υπογραμμισμένοι χαρακτήρες.

Αναγνωριστικά μπορούν να γραφούν και με μικρά γράμματα καθώς επίσης και με υπογραμμισμένο τον πρώτο χαρακτήρα, αλλά δεν επιτρέπεται η εισαγωγή κενών (spaces). Μερικά παραδείγματα αναγνωριστικών είναι τα ακόλουθα:

- W123_PV W12_3PV aTemp1 _PROG1

Τα παρακάτω αναγνωριστικά δεν επιτρέπονται

- W123__PV W12 3PV 1Tempa Q%TY12

Επίσης, για να είναι δυνατή η διαπίστωση της μοναδικότητας ενός αναγνωριστικού, πρέπει αυτό να διαφέρει από ένα άλλο σε κάποιον από τους πρώτους έξι χαρακτήρες του. Παραδείγματος χάριν τα παρακάτω δύο αναγνωριστικά θα θεωρηθούν ταυτόσημα σε έναν υπολογιστή:

- A12345_XY A12345_GG

Συνεπώς, για να είναι δυνατή η μεταφορά λογισμικού από έναν υπολογιστή σε άλλο, θα πρέπει να αποφεύγεται η χρήση αμφιλεγόμενων αναγνωριστικών. Αυτό εξασφαλίζεται με τη διαφοροποίηση των έξι πρώτων χαρακτήρων ενός αναγνωριστικού από ένα άλλο αναγνωριστικό.

4.6 Λέξεις-Κλειδιά

Οι λέξεις κλειδιά είναι ορισμένες συγκεκριμένες λέξεις ή ακολουθίες λέξεων που χρησιμοποιούνται τόσο για τον ορισμό του τύπου των δεδομένων των μονάδων οργάνωσης λογισμικού όσο και για τον ορισμό τους. Παραδείγματος χάριν για τον ορισμό του ονόματος μιας συνάρτησης χρησιμοποιούνται οι λέξεις FUNCTION και END_FUNCTION ανάμεσα από τις οποίες πρέπει να δηλωθεί το όνομα της συνάρτησης.

Για να αποφευχθεί η σύγχυση που μπορεί να προκληθεί από τη χρήση αναγνωριστικών που συμπίπτουν με λέξεις-κλειδιά, πρέπει τα χρησιμοποιούμενα αναγνωριστικά να διαφοροποιούνται από τις λέξεις αυτές. Έτσι έναν γενικό κανόνα που πρέπει να ακολουθεί ο κάθε προγραμματιστής είναι να αποφεύγει να χρησιμοποιεί ως αναγνωριστικά τις λέξεις:

- TYPE, TRUE PROGRAM TASK RETURN STEP FUNCTION

Τα αναγνωριστικά που χρησιμοποιούνται για τον ορισμό συναρτήσεων και συναρτησιακών οντοτήτων πρέπει να θεωρούνται ως δεσμευμένες λέξεις-κλειδιά και δεν πρέπει να χρησιμοποιούνται . Τέτοιες λέξεις είναι οι:

- TON RS SIN COS

4.7 Σχόλια

Σε κάθε γλώσσα προγραμματισμού η οποία υποστηρίζεται από το πρότυπο μπορούν να εισαχθούν σχόλια που να αποτελούνται από μία μέχρι πολλές γραμμές, με εξαίρεση τη γλώσσα της Λίστας Εντολών (IL). Στη γλώσσα αυτή τα σχόλια μπορούν να τοποθετηθούν εκεί όπου μπορούν να εισαχθούν ένα ή περισσότερα κενά. Τα σχόλια δηλώνονται και τοποθετούνται μεταξύ των χαρακτήρων (* ……*).

Ένα παράδειγμα δήλωσης σχολίου είναι το ακόλουθο:

```
 (* Activate Pump *)
(************************ *)
(* Main turbine interlock logic *)
(************************)
```

4.8 Τύποι δεδομένων

Το πρότυπο υποστηρίζει μια σειρά τύπων δεδομένων κατάλληλων να χειρίζονται τις τιμές όλων των μεταβλητών που εμφανίζονται στο βιομηχανικό έλεγχο. Η σειρά περιλαμβάνει ακεραίους δεκαδικούς και δυαδικούς αριθμούς καθώς επίσης και αριθμούς κινητής υποδιαστολής, χρόνους και ημερομηνίες, ακολουθίες χαρακτήρων και bits, bytes και words για λειτουργίες σε επίπεδο συμβολικής γλώσσας και γλώσσας μηχανής. Στη συνέχεια περιγράφονται η μορφή και ο τρόπος δήλωσης μεταβλητών που αντιστοιχούν σε κάθε συγκεκριμένο τύπο δεδομένων.

Για τον καθορισμό του τύπου των δεδομένων μιας μεταβλητής χρησιμοποιούνται ειδικά δηλωτικά προθέματα πριν από τη μεταβλητή ή την τιμή που έχει η μεταβλητή αυτή, π.χ. με INT#12, LINT#342., δηλώνεται ότι ο αριθμός 12 είναι ένας ακέραιος αριθμός που μπορεί να εκφραστεί με ένα δυαδικό κώδικα 8 ψηφίων, ενώ ο αριθμός 342 είναι ένας ακέραιος αριθμός που μπορεί να εκφραστεί με ένα δυαδικό κώδικα μήκους 16 ψηφίων. Στους πίνακες που ακολουθούν δίνονται τα δηλωτικά προθέματα για τους διάφορους τύπους δεδομένων, που υποστηρίζονται από το πρότυπο καθώς και παραδείγματα δήλωσης αντίστοιχων δεδομένων.

4.8.1 Ακέραιοι αριθμοί

Όπως προκύπτει από τον πίνακα 4.1, υπάρχουν διάφοροι τύποι ακεραίων δεδομένων. Η επιλογή του κατάλληλου τύπου εξαρτάται από το μέγεθος των αριθμών που θα αποθηκευτούν.

Τύπος Δεδομένων	Περιγραφή	Μήκος δυαδικής παράστασης	Περιοχή τιμών
SINT	Σύντομος ακέραιος	8	-128 μέχρι +128
INT	Ακέραιος	16	-32768 μέχρι 32767
DINT	Διπλός Ακέραιος	32	-2^{31} μέχρι $2^{31}-1$
LINT	Μακρύς ακέραιος	64	-2^{63} μέχρι $2^{63}-1$
USINT	Μη προσημασμένος σύντομος ακέραιος	8	0 μέχρι 255
UINT	Μη προσημασμένος ακέραιος	16	0 μέχρι $2^{16}-1$
UDINT	Μη προσημασμένος διπλός ακέραιος	32	0 μέχρι $2^{32}-1$
ULINT	Μη προσημασμένος μακρύς ακέραιος	64	0 μέχρι $2^{64}-1$

Πίνακας 4.1 *Ακέραιοι τύποι δεδομένων.*

Έτσι αν πρόκειται να χρησιμοποιηθεί μια μεταβλητή για την καταγραφή του αριθμού των αντικειμένων που μεταφέρονται από μια μεταφορική ταινία και είναι γνωστό ότι ο αριθμός δε θα είναι μεγαλύτερος από 100, τότε ο τύπος δεδομένων SINT θα ήταν ο ενδεδειγμένος τύπος δεδομένων που θα έπρεπε να οριστεί για τη μεταβλητή. Αν, όμως, η μεταβλητή επρόκειτο να χρησιμοποιηθεί για την καταμέτρηση των παλμών που στέλνει ένας αποκωδικοποιητής γωνιακής ταχύτητας άξονα, ο οποίος μπορεί να πάρει μεγάλες θετικές και αρνητικές τιμές, τότε ο ενδεδειγμένος ορισμός δεδομένων είναι ο "LINT". Όσον αφορά τον τρόπο γραφής των δεδομένων των διαφόρων ακεραίων τύπων, σημειώνεται ότι οι δεκαδικοί ακέραιοι γράφονται με τον κλασικό τρόπο της παράθεσης των ψηφίων κάθε δεκαδικής τάξης, το ένα δίπλα στο άλλο, με το ψηφίο της μεγαλύτερης δεκαδικής τάξης πρώτο στα αριστερά , π.χ. –123, 0, +463. Όσον αφορά αριθμούς άλλων αριθμητικών συστημάτων ακολουθούνται οι εξής συμβάσεις. Για τη γραφή αριθμών στο δυαδικό σύστημα προτάσσεται η ακολουθία

χαρακτήρων 2# πριν από το δυαδικό αριθμό, π.χ. 2#1111_1111 ή 2#0000_1000. Για τη γραφή αριθμών στο οκταδικό σύστημα, προτάσσεται η ακολουθία 8# πριν από τον οκταδικό αριθμό, π.χ. 8#377, 8#020. Όμοια, για τη γραφή αριθμών του δεκαεξαδικού συστήματος προτάσσεται η ακολουθία 16#, π.χ. 16#FF, 16#A0.

4.8.2 Αριθμοί κινητής υποδιαστολής

Οι τύποι δεδομένων κινητής υποδιαστολής που υποστηρίζονται από το πρότυπο καταγράφονται στον πίνακα 4.2.

Τύπος δεδομένων	Περιγραφή	Αρ. ψηφίων δυαδικής παράστασης	Περιοχή τιμών
REAL	Πραγματικοί αριθμοί	32	$\pm 10^{\pm 38}$ με ακρίβεια $1/2^{23}$
LREAL	Μακρείς πραγματικοί αριθμοί	64	$\pm 10^{\pm 308}$ με ακρίβεια $1/2^{53}$

Πίνακας 4.2 *Τύποι δεδομένων κινητής υποδιαστολής.*

Οι τύποι δεδομένων 'REAL' και 'LREAL' χρησιμοποιούνται κυρίως για να αποθηκευτούν πολύ μεγάλοι, θετικοί ή αρνητικοί αριθμοί όπως είναι ο αριθμός 13459500.0 ή πολύ μικροί κλασματικοί αριθμοί όπως είναι ο αριθμός –0.000123. Οι τυπικές χρήσεις των αριθμών κινητής υποδιαστολής είναι:

- Στην αποθήκευση στον υπολογιστή των αναλογικών μετρήσεων που λαμβάνονται από αισθητήρια πίεσης, θερμοστοιχεία, ταχύμετρα κτλ. (Lewis, 1995), (Kart-Heinz & Tiegelkamp, 2010).
- Στους υπολογισμούς αλγορίθμων που χρησιμοποιούνται σε κλειστούς βρόχους ελέγχου, όπως είναι ο αλγόριθμος PID.
- Στον υπολογισμό του μεγέθους των δράσεων επί των τελικών στοιχείων των βρόχων ελέγχου.

Η γραφή των αριθμών κινητής υποδιαστολής γίνεται με τη χρήση των ψηφίων του δεκαδικού συστήματος και το διαχωρισμό του ακέραιου μέρους του αριθμού από το κλασματικό με υποδιαστολή. Για τη γραφή πολύ μικρών και πολύ μεγάλων αριθμών χρησιμοποιείται η εκθετική παράσταση που δίνει τη δύναμη του 10, με την οποία ο αριθμός πολλαπλασιάζεται. Το κεφαλαίο γράμμα E ή το μικρό e χρησιμοποιούνται για να δηλωθεί η δύναμη του 10. Οι αριθμοί 123.21, -0.001298, 1.6E-10, 0.9E20, 0.23276e+14, είναι μερικά τυπικά παραδείγματα γραφής αριθμών κινητής υποδιαστολής τύπου REAL και LREAL αντίστοιχα.

4.8.3 Χρονική διάρκεια

Σε πολλές εφαρμογές βιομηχανικού ελέγχου χρειάζεται να καταγράφεται η χρονική διάρκεια η οποία πέρασε από τη στιγμή της εμφάνισης κάποιου γεγονότος. Η δήλωση της διάρκειας αυτής μπορεί να γίνει με δύο μορφές γραφής, τη σύντομη και την πλήρη. Στη σύντομη γραφή η δήλωση γίνεται προτάσσοντας τους χαρακτήρες T# και μετά τη χρονική διάρκεια εκφρασμένη στη μορφή:

D για ημέρες, h για ώρες, m για λεπτά, s για δευτερόλεπτα, ms για milliseconds

Έτσι η δήλωση της χρονικής διάρκειας 12 ημερών, 3 ωρών και 3 δευτερολέπτων, στη σύντομη μορφή θα γίνει με την παράσταση T#12d3h3s.

Στην πλήρη γραφή η δήλωση γίνεται προτάσσοντας τη λέξη TIME και το χαρακτήρα # και μετά τη χρονική διάρκεια εκφρασμένη στη μορφή:

D για ημέρες_ h για ώρες_ m για λεπτά_ s για δευτερόλεπτα_ ms για milliseconds

Η δήλωση της διάρκειας των 16 ημερών, 5 ωρών, 3 λεπτών και 4 δευτερολέπτων στην πλήρη μορφή θα γίνει με την παράσταση TIME#16d_5h_3m_4s. Το τελευταίο πεδίο της δήλωσης μπορεί να γίνει και σε κλασματική μορφή, π.χ. T#12d3.5h, T#10.12s.

4.8.4 Ημερομηνία και ώρα

Για τη δήλωση ημερομηνίας και ώρας χρησιμοποιούνται οι λέξεις που σημειώνονται στον πίνακα 4.3.

Τύπος δεδομένων	Περιγραφή	Χρήση
DATE	Ημερομηνία	Αποθήκευση ημερομηνίας
TIME_OF_DAY ή TOD	Time of day	Αποθήκευση ώρας της ημέρας ή πραγματικού ρολογιού
DATE_AND_TIME ή DT	Ημερομηνία και ώρα ημέρας	Αποθήκευση ημ/νίας και ώρας

Πίνακας 4.3 *Τύποι δεδομένων ημερομηνίας και ώρας.*

Αυτοί οι τύποι δεδομένων έχουν μεγάλη εφαρμογή στον έλεγχο διεργασιών διαλείποντος έργου (batch processes). Τυπικές χρήσεις της ημερομηνίας και ώρας είναι:

- Στην καταγραφή της ημερομηνίας και χρονικής στιγμής που συμβαίνει κάποιο γεγονός και προειδοποίηση σφάλματος,
- Στην επιβεβαίωση της πραγματοποίησης ενός γεγονότος κατά τη διάρκεια της ημέρας, εβδομάδας ή ακόμη και μήνα. Για παράδειγμα ένας αντιδραστήρας πρέπει να προθερμαίνεται κάθε Δευτέρα πρωί αυτόματα για να είναι έτοιμος για την παραγωγή της εβδομάδας.
- Στην καταγραφή του γεγονότος της διακοπής της ηλεκτρικής τροφοδοσίας και επαναφοράς της για τον υπολογισμό του χρόνου που ετέθη η διεργασία εκτός λειτουργίας.

Οι τρόποι με τον οποίους δηλώνονται ημερομηνία και ώρα είναι και στην περίπτωση αυτή δύο, ο σύντομος και ο πλήρης. Τα χαρακτηριστικά προθέματα που ορίζουν αυτόν τον τύπο των δεδομένων με τον πλήρη τρόπο είναι αυτά που είναι γραμμένα στον πίνακα 4.3, ενώ τα προθέματα του σύντομου τρόπου είναι: D# για την ημερομηνία, TOD# για την ώρα της ημέρας, DT# για την ημερομηνία και ώρα της ημέρας. Παρακάτω δίνονται παραδείγματα δήλωσης ημερομηνίας, ώρας και ταυτόχρονα ημερομηνίας και ώρας.

Δήλωση ημερομηνίας με σύντομη μορφή D#1994-06-10 (10 Ιουνίου 1994) ή
d#1995-01-13 (13 Ιανουαρίου 1995)
Δήλωση ημερομηνίας με πλήρη μορφή DATE#1993-10-15 (15 Οκτωβρίου 1993)

Δήλωση ώρας με σύντομη μορφή TOD#10:10:30 (ώρα 10 και 10 λεπτά και 30 δευτερόλεπτα)
Δήλωση ώρας με σύντομη μορφή TOD#23:59:59 (1 δευτερόλεπτο πριν από τα μεσάνυκτα)
Δήλωση ώρας με πλήρη μορφή TIME_OF_DAY#05:00:00:56 (0.56 δευτερόλεπτα μετά τις 5).
Δήλωση ημερομηνίας και ώρας με σύντομο τρόπο DT#1993-06-12-15:36:55.40 (12 Ιουνίου 1993, ώρα 15 και 36 λεπτά και 55.4 δευτερόλεπτα)
Δήλωση ημερομηνίας και ώρας με πλήρη μορφή DATE_AND_TIME#1995-02-01-12:00:00 (Μεσημέρι της 1ης Φεβρουαρίου 1995).

4.8.5 Ακολουθία χαρακτήρων

Κατά τη σύνταξη λογισμικού βιομηχανικού ελέγχου χρειάζεται να γραφούν ακολουθίες χαρακτήρων για:

- ταυτοποίηση παρτίδων παραγωγής π.χ. 'JOB_X32A3',
- αποστολή μηνυμάτων σε σταθμούς εποπτείας και χειρισμών, π.χ. 'Starting Vessel Purge',
- αποστολή μηνυμάτων μέσω καναλιών επικοινωνίας σε άλλες διατάξεις

Για τη δήλωση τέτοιων ακολουθιών χρησιμοποιείται η λέξη STRING ακολουθούμενη από την επιθυμητή ακολουθία χαρακτήρων μέσα σε ομοιωματικά (‘’).

Η δήλωση μιας ακολουθίας χαρακτήρων μπορεί να περιλαμβάνει εκτυπώσιμους και μη εκτυπώσιμους χαρακτήρες. Οι μη εκτυπώσιμοι χαρακτήρες τοποθετούνται στην ακολουθία προτάσσοντας το σύμβολο '$' στη δεκαεξαδική αναπαράσταση του χαρακτήρα. Υπάρχει, επίσης, ένας αριθμός από δεσμευμένα γράμματα που, όταν τοποθετηθούν μετά το σύμβολο $, δηλώνουν, συνήθως, χρησιμοποιούμενους χαρακτήρες ελέγχου. Τεχνικές για να εισάγουμε χαρακτήρες ελέγχου παρατίθενται στον πίνακα 4.4.

Κώδικας	Εξήγηση
$$	εισαγωγή του συμβόλου $
$´	εισαγωγή του χαρακτήρα ´
$L ή $1	εισαγωγή χαρακτήρα ελέγχου αλλαγής γραμμής
$N ή $n	εισαγωγή χαρακτήρα ελέγχου νέας γραμμής
$P ή $p	εισαγωγή χαρακτήρα ελέγχου νέας σελίδας
$P ή $r	εισαγωγή χαρακτήρα ελέγχου 'ENTER' (carriage return)
$T ή $t	εισαγωγή χαρακτήρα tab

Πίνακας 4.4 *Χαρακτήρες ελέγχου.*

Παρακάτω δίνονται μερικά παραδείγματα από δηλώσεις ακολουθιών χαρακτήρων.

'Batch number AX45_65' - μήνυμα χωρίς χαρακτήρες ελέγχου
'End of report $N' -μήνυμα που περιλαμβάνει το χαρακτήρα ελέγχου για την αλλαγή της σελίδας
'$01$02$10' - τρεις χαρακτήρες εκφρασμένοι σε δεκαεξαδικό κώδικα που αντιστοιχούν στους με δεκαδικούς αριθμούς 1, 2 και 16.
‘’ - μια κενή ακολουθία

4.8.6 Δυαδική ακολουθία

Για την παράσταση της κατάστασης διακριτών εισόδων, εξόδων και σημαιών που πρέπει να αποσταλεί ή να παραληφθεί από διατάξεις και όργανα τα οποία βρίσκονται μακριά από τον υπολογιστή ή για το χειρισμό χαμηλού επιπέδου επικοινωνίας δεδομένων μεταξύ μονάδων του υλικού του υπολογιστή, χρειάζεται να χρησιμοποιηθούν ακολουθίες δυαδικών ψηφίων. Ο ορισμός τέτοιων ακολουθιών μπορεί να γίνει με τα προθέματα που δίνονται στον πίνακα 4.5.

Τύπος δεδομένων	Περιγραφή	αρ. ψηφίων	Χρήση
BOOL	Ακολουθία 1 δυαδικού ψηφίου	1	Δήλωση λογικής κατάστασης
BYTE	Ακολουθία 8 δυαδικών ψηφίων	8	Δεδομένα σε δυαδικό κώδικα
WORD	Ακολουθία 16 δυαδικών ψηφίων	16	Δεδομένα σε δυαδικό κώδικα
DWORD	Ακολουθία 32 δυαδικών ψηφίων	32	Δεδομένα σε δυαδικό κώδικα
LWORD	Ακολουθία 64 δυαδικών ψηφίων	64	Δεδομένα σε δυαδικό κώδικα

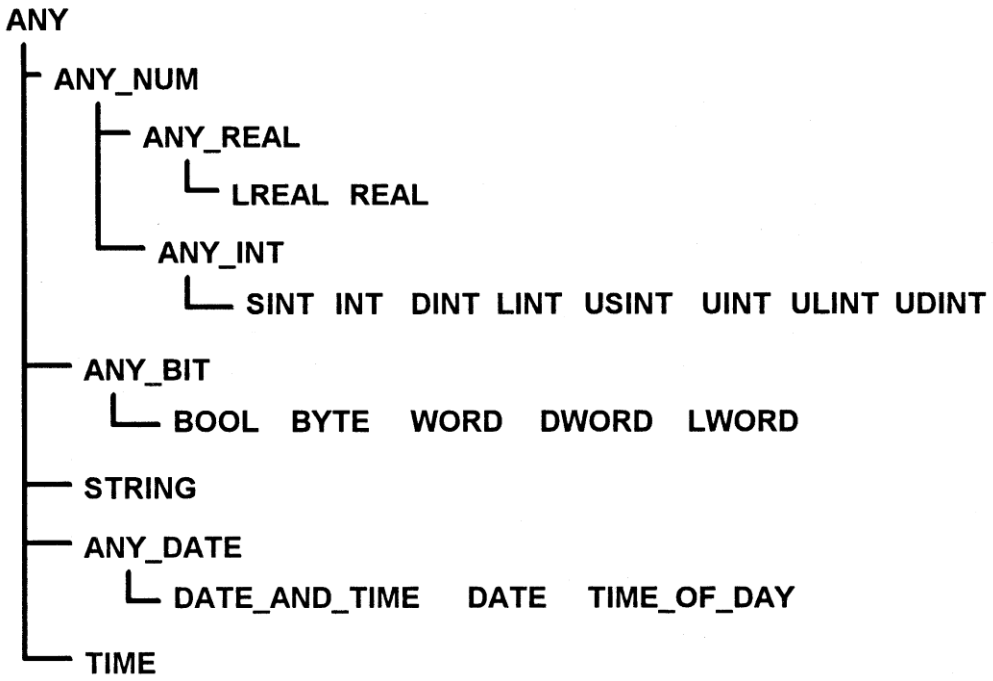
Πίνακας 4.5 *Τύποι δεδομένων δυαδικής ακολουθίας.*

Ανάλογα με το μήκος της ακολουθίας των δυαδικών ψηφίων χρησιμοποιείται και το αντίστοιχο πρόθεμα. Έτσι, για τον ορισμό των τιμών λογικών μεταβλητών που σχετίζονται με ψηφιακές εισόδους και εξόδους, που είναι ένα μοναδικό δυαδικό ψηφίου, χρησιμοποιείται η λέξη BOOL και ακολουθεί το μνημονικό σύμβολο της μεταβλητής.

Οι μεταβλητές τύπου BOOL χρησιμοποιούνται σε λογικές συναρτήσεις μανταλώσεων και συνδυαστικής λογικής, για τη καταγραφή της κατάστασης που βρίσκεται μια συσκευή, π.χ. αν μια αντλία είναι σε λειτουργία (ON) ή εκτός λειτουργίας (OFF). Παρόμοια είναι και η χρήση των άλλων προθεμάτων του πίνακα 4.5 για τον ορισμό μεταβλητών που οι τιμές τους μπορεί να δίνονται σε 8-bits, 16-bits, 32-bits και 64-bits δυαδικούς αριθμούς.

4.9 Γενικοί τύποι δεδομένων

Οι τύποι δεδομένων που ορίσθηκαν στην προηγούμενη ενότητα μπορούν να συσχετισθούν μεταξύ τους με τον ιεραρχικό τρόπο που δείχνεται στην Εικόνα 4.3.



Εικόνα 4.3 *Ιεραρχία των βασικών τύπων δεδομένων.*

Η έννοια αυτής της ιεράρχησης των δεδομένων είναι ότι ορίζοντας έναν ανώτερο ιεραρχικά τύπο δεδομένων για τις μεταβλητές μιας συνάρτησης, η συνάρτηση αυτή μπορεί να χρησιμοποιηθεί με δεδομένα κάθε άλλου τύπου που ιεραρχικά είναι κατώτερος του τύπου που αρχικά ορίστηκε. Αν π.χ. οριστεί ως ο τύπος δεδομένων για τις μεταβλητές Y και X της συνάρτησης Y= AVE(X) να είναι ANY_NUM τότε η συνάρτηση

αυτή θα μπορούσε να υπολογιστεί και με δεδομένα τύπου ANY_REAL, LREAL και ANY_INT. Με άλλα λόγια η συνάρτηση AVE θα μπορούσε να χρησιμοποιηθεί με δεδομένα που είναι είτε κινητής υποδιαστολής είτε ακέραιοι αριθμοί. Γενικά όλες οι μεταβλητές πρέπει να έχουν μια αρχική τιμή που αν δεν οριστεί συνεπάγεται ότι είναι 0 για τις μεταβλητές τύπου δεδομένων ANY_NUM ή κενή ακολουθία χαρακτήρων (‘’) για τις μεταβλητές τύπου STRING και D#0001-01-01 για μεταβλητές τύπου ANY_DATE.

4.10 Παραγόμενοι τύποι δεδομένων

Για να βοηθηθεί η αναγνωσιμότητα ενός προγράμματος και για να εξασφαλιστεί η συνεπής χρησιμοποίηση των δεδομένων, νέοι τύποι δεδομένων μπορούν να οριστούν από τους βασικούς τύπους που αναπτύχθηκαν παραπάνω. Ένας τέτοιος νέος τύπος δηλώνεται παρεμβάλλοντας τον ορισμό του μεταξύ των λέξεων TYPE και END_TYPE. Έτσι, ο νέος τύπος δεδομένων PRESSURE μπορεί να ορισθεί ως εξής:

```
TYPE
 PRESSURE: REAL;
END_TYPE
```

Επίσης, μπορούν να δημιουργηθούν σύνθετοι τύποι δεδομένων ορίζοντας μια δομή που να αποτελείται από δεδομένα των βασικών τύπων. Η δήλωση μιας τέτοιας δομής γίνεται περιλαμβάνοντας μεταξύ των λέξεων STRUCT και END_STRUCT τον ορισμό της νέας αυτής δομής δεδομένων. Στο κάθε μέλος της δομής μπορεί να δοθεί ένα όνομα με συγκεκριμένο νόημα . Π.χ. τα δεδομένα από ένα αισθητήριο πίεσης μπορεί να αποτελούνται από:

(α) την τρέχουσα τιμή μιας αναλογικής μέτρησης,
(β) την κατάσταση του αισθητηρίου, δηλαδή, αν είναι εντός ή εκτός λειτουργίας,
(γ) την ημερομηνία της ρύθμισής του,
(δ) τη μέγιστη τιμή της ασφαλούς λειτουργίας του,
(ε) τον αριθμό των προειδοποιήσεων που ελήφθησαν μέσα σε ορισμένη χρονική περίοδο λόγω υπέρβασης ορίων πίεσης.

Όλα αυτά τα δεδομένα μπορούν να οριστούν ως ένας νέος τύπος δεδομένων με το όνομα PRESSURE_SENSOR ως εξής:

```
TYPE PRESSURE_SENSOR
STRUCT
        INP: PRESSURE;
                STATUS: BOOL
                        CALIBRATION: DATE;
                                HIGH LIMIT: REAL;
                                        ALARM COUNT: INT;
END_TYPE
```

Ακόμη μπορεί να οριστεί ένας τύπος δεδομένων που θα επιτρέπει την απόδοση διαφορετικών μνημονικών ονομάτων στις διάφορες καταστάσεις που μπορεί να παίρνει η τιμή μιας μεταβλητής. Αν π.χ. όλες οι διατάξεις μιας βιομηχανικής μονάδας έχουν έναν αριθμό διαφορετικών τρόπων λειτουργίας τους, τότε ένας τύπος δεδομένων κατάλληλος για μεταβλητές που προσδιορίζουν την κατάσταση στην οποία βρίσκεται κάποια διάταξη θα μπορούσε να οριστεί ως εξής:

```
TYPE DEVICE MODE
(INITIALIZING, RUNNING, STANDBY, FAULTY);
END_TYPE
```

Έτσι κάθε μεταβλητή του τύπου DEVICE_MODE μπορεί να παίρνει μία από τις καταστάσεις που απαριθμούνται στη δήλωση του συγκεκριμένου τύπου δεδομένων.

Σε πολλές εφαρμογές των προγραμματιζόμενων ελεγκτών χρειάζεται να χρησιμοποιηθούν μεταβλητές που να μπορούν να αποθηκεύσουν πίνακες τιμών. Ένας πίνακας μπορεί να περιλαμβάνει δεδομένα πολλών και διαφορετικών βασικών και παραγόμενων τύπων. Έτσι, αν θεωρήσουμε ότι σε μια μεταβλητή θέλουμε να αντιστοιχήσουμε το σύνολο των πιέσεων που αναπτύσσονται σε διάφορα σημεία ενός δοχείου, τότε ένας κατάλληλος τρόπος θα ήταν η χρήση του τύπου των δεδομένων VESSEL PRESS_DATA που ορίζεται ως εξής:

```
TYPE VESSEL PRESS_DATA:
ARRAY[1…20] OF PRESSURE;
END_TYPE

TYPE VESSEL_MATRIX:
ARRAY[1…3, 1…4] OF VESSEL_PRESS_DATA;
END_TYPE
```

Το πώς θα οριστούν οι διαστάσεις του πίνακα και ο αριθμός των υποπινάκων από τους οποίους μπορεί να αποτελείται δεν προσδιορίζεται στο πρότυπο. Ο πίνακας του πρώτου παραδείγματος είναι ένα διάνυσμα 20 στοιχείων, ενώ ο πίνακας του δεύτερου παραδείγματος είναι δύο διαστάσεων 3X4 στοιχείων.

Σε όλους τους παραγόμενους τύπους δεδομένων μπορούν να δοθούν αρχικές τιμές που θα αντικαταστήσουν τις τιμές των βασικών τύπων δεδομένων από τους οποίους συνίστανται. Οι νέες τιμές μπορούν να δοθούν κατά τον ορισμό του τύπου των δεδομένων αυτών. Για τον ορισμό του τύπου δεδομένων “PRESSURE” ορίζεται και η αρχική τιμή κάθε μεταβλητής του τύπου αυτού να είναι ίση με 1.0 ως εξής.

```
TYPE PRESSURE: REAL:=1.0;
END_TYPE
```

Στα παραδείγματα που ακολουθούν, δείχνεται πώς αρχικοποιούνται δύο άλλοι τύποι δεδομένων, ο τύπος “PRESSURE SENSOR” και ο τύπος “DEVICE MODE”.

```
TYPE PRESSURE_SENSOR:
STRUCT
INPUT: PRESSURE:=2.00;
STATUS: BOOL:=0;
CALIBRATION:
DATE:=DT#1994-01-10;
HIGH LIMIT: REAL:=30.0;
ALARM COUNT:INT:=0;
END_STRUCT;
END_TYPE

TYPE DEVICE MODE:
(INITIALIZING, RUNNING, STANDBY, FAULTY):=STANDBY;
END_TYPE
```

Για την αρχικοποίηση πινάκων μπορεί να δοθούν τιμές σε κάθε στοιχείο του πίνακα. Στο παράδειγμα που ακολουθεί σε καθένα από τα πρώτα 10 στοιχεία δίνεται η τιμή 1.1, σε καθένα από τα επόμενα πέντε δίνεται η τιμή 1.3 και στα επόμενα πέντε η τιμή 1.7.

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

Είναι, επίσης, δυνατόν να δίνονται τιμές αρχικοποίησης των τύπων δεδομένων που χρησιμοποιούνται σε δομές δεδομένων που παράγονται από τους τύπους αυτούς. Ως παράδειγμα δίνεται η αρχικοποίηση του

τύπου δεδομένων PRESSURE_SENSOR που χρησιμοποιείται για τον ορισμό του νέου παραγόμενου τύπου δεδομένων GAS_PRESSURE_SENSOR.

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

Στην περίπτωση αυτή δίνονται διαφορετικές τιμές στις μεταβλητές PRESSURE και HIGH_LIMIT οι οποίες συνιστούν τη δομή PRESSURE_SENSOR και η οποία χρησιμοποιείται για τον ορισμό της νέας δομής δεδομένων GAS_PRESSURE_SENSOR.

4.11 Μεταβλητές

Μπορούν να οριστούν μεταβλητές μέσα σε κάθε μονάδα οργάνωσης λογισμικού, όπως είναι τα προγράμματα, οι συναρτησιακές οντότητες, οι πόροι, οι διεργασίες και ακόμη στο επίπεδο περιγραφής της αρχιτεκτονικής του λογισμικού. Οι μεταβλητές μπορούν να οριστούν ως τοπικές (local) μεταβλητές, όταν πρόσβαση στις μεταβλητές αυτές μπορεί να επιτευχθεί μόνο μέσα από τη μονάδα οργάνωσης λογισμικού στην οποία ορίζονται. Μπορούν εναλλακτικά να οριστούν ως γενικές (global), όταν η πρόσβαση σε αυτές μπορεί να γίνει από όλες τις οντότητες λογισμικού που περιλαμβάνονται στη μονάδα οργάνωσης λογισμικού στην οποία ορίζονται. Π.χ. μια γενική μεταβλητή μπορεί να οριστεί σε έναν πόρο, οπότε θα είναι προσβάσιμη από τις διεργασίες, προγράμματα και συναρτησιακές οντότητες τις οποίες περιλαμβάνει ο συγκεκριμένος πόρος.

Προκειμένου να μεταφερθούν δεδομένα μεταξύ μονάδων οργάνωσης λογισμικού που ανήκουν σε διαφορετικές αρχιτεκτονικές, το πρότυπο περιλαμβάνει τις υπηρεσίες του ορισμού της **διαδρομής πρόσβασης (access paths).** Κάθε αρχιτεκτονική μπορεί να έχει έναν αριθμό από καθορισμένες μεταβλητές στις οποίες μπορούν να έχουν πρόσβαση σε άλλες αρχιτεκτονικές. Το πρότυπο υποθέτει ότι υπάρχει κάποια μορφή επικοινωνίας μεταξύ διαφορετικών αρχιτεκτονικών χωρίς να προσδιορίζει επακριβώς τις λεπτομέρειες αυτής της επικοινωνίας, π.χ. τα πρωτόκολλα και τις υπηρεσίες των επιπέδων OSI, τα χαρακτηριστικά του δικτύου, π.χ. δίκτυο τύπου Ethernet ή τύπου Fieldbus. Οι διαδρομές πρόσβασης ορίζονται με ειδικού τύπου μεταβλητές που δηλώνονται με VAR_ACCESS. Ανάγνωση ή εγγραφή των μεταβλητών αυτών μπορεί να γίνει από άλλες απομακρυσμένες αρχιτεκτονικές. Με άλλα λόγια ο σχεδιαστής του λογισμικού μπορεί να προσδιορίσει ένα σύνολο μεταβλητών που τυπικά συνιστά τη διεπαφή μιας αρχιτεκτονικής με τις άλλες.

Γενικά, η αρχιτεκτονική και οι πόροι μέσω των διεργασιών είναι υπεύθυνοι για την εκτέλεση των προγραμμάτων που εμπεριέχουν. Τα προγράμματα δεν έχουν πάντοτε τον έλεγχο των συναρτησιακών τους οντοτήτων διότι μερικές τέτοιες οντότητες ελέγχονται απευθείας από διεργασίες που εκτελούν τις οντότητες αυτές είτε περιοδικά είτε μετά από την εμφάνιση κάποιου σποραδικού γεγονότος.

Όσον αφορά υπηρεσίες εκκίνησης και διακοπής της λειτουργίας λογισμικού που αντιστοιχεί σε συγκεκριμένη αρχιτεκτονική, αυτές δεν περιγράφονται στο πρότυπο, αλλά θεωρούνται εξωτερικές υπηρεσίες που πρέπει να προσφέρονται από τους σταθμούς εποπτείας και χειρισμών κάθε συστήματος βιομηχανικού ελέγχου. Το πρότυπο, όμως, περιγράφει τη συμπεριφορά του λογισμικού κατά την εκκίνηση και τη διακοπή λειτουργίας του. Έτσι σύμφωνα με το πρότυπο κατά την εκκίνηση του λογισμικού που περιλαμβάνεται σε μια αρχιτεκτονική πρέπει:

- όλες οι γενικές μεταβλητές να αρχικοποιούνται και όλοι οι πόροι να εκκινούν,
- κάθε μεταβλητή μέσα στον πόρο αυτό να αρχικοποιείται και όλες οι διεργασίες να
- ενεργοποιούνται όταν ένας πόρος εκκινεί,
- όλα τα προγράμματα και οι συναρτησιακές οντότητες που ελέγχονται από τη διεργασία να αρχίζουν την εκτέλεσή τους όταν ενεργοποιείται μια διεργασία.

Κατά τη διακοπή της λειτουργίας του εν-λόγω λογισμικού πρέπει:

- όλοι οι πόροι που εμπεριέχει να σταματούν να λειτουργούν,
- όλες οι διεργασίες να απενεργοποιούνται με αποτέλεσμα να διακόπτεται η εκτέλεση όλων των ελεγχόμενων από τη διεργασία προγραμμάτων και συναρτησιακών οντοτήτων, όταν η λειτουργία ενός πόρου διακοπεί.

Εκτός από το χαρακτηρισμό των μεταβλητών σε τοπικές και γενικές, το πρότυπο προβλέπει την περαιτέρω ένταξή τους σε κατηγορίες μέσα στο επίπεδο προγράμματος, συναρτησιακής οντότητας και συναρτήσεως. Οι κατηγορίες αυτές των μεταβλητών είναι:

- (α) των εξωτερικών μεταβλητών
- (β) των μεταβλητών εισόδου
- (γ) των μεταβλητών εξόδου
- (δ) των μεταβλητών εισόδου/εξόδου
- (ε) των μεταβλητών της άμεσης απεικόνισης θέσεων μνήμης.

Όπως αναφέρθηκε στα προηγούμενα, οι γενικές μεταβλητές μπορούν να οριστούν είτε στο επίπεδο της περιγραφής της αρχιτεκτονικής είτε των πόρων, είτε του προγράμματος του λογισμικού. Για την πρόσβαση, όμως, μιας γενικής μεταβλητής από μια μονάδα οργάνωσης προγράμματος που βρίσκεται σε κατώτερο επίπεδο από αυτό στο οποίο ορίσθηκε ως γενική, χρειάζεται να ορισθεί η ίδια μεταβλητή ως εξωτερική μέσα στη μονάδα από την οποία επιδιώκεται η πρόσβαση. Παραδείγματος χάριν, ας θεωρήσουμε μια αρχιτεκτονική με όνομα LINE_1 που περιλαμβάνει ένα πρόγραμμα που ονομάζεται PRODUCT_1. Μέσα στο πρόγραμμα υπάρχει μια συναρτησιακή οντότητα που ονομάζεται Ramp_Speed. Η γενική μεταβλητή Line_Speed που ορίζεται ως γενική στο επίπεδο της αρχιτεκτονικής μπορεί να είναι προσβάσιμη από το επίπεδο της συναρτησιακής οντότητας Ramp_Speed δηλώνοντας μέσα σε αυτή τη συναρτησιακή οντότητα μια εξωτερική μεταβλητή με το ίδιο όνομα. Ο ορισμός μιας γενικής μεταβλητής γίνεται παρεμβάλλοντας τους ορισμούς των τύπων των μεταβλητών ανάμεσα στις λέξεις VAR_GLOBAL …. END_VAR, ενώ για τον ορισμό μιας εξωτερικής μεταβλητής χρησιμοποιούνται οι λέξεις VAR_EXTERNAL … END_VAR. Έτσι η εφαρμογή των παραπάνω στο παράδειγμα που μόλις αναφέρθηκε, έχει ως αποτέλεσμα τους εξής ορισμούς:

```
VAR_GLOBAL
Line_Speed : LREAL;
Job_Number: INT;
END_VAR

VAR_EXTERNAL
Line_Speed : LREAL;
Job_Number: INT;
END_VAR
```

Οι μεταβλητές εισόδου δρουν ως μηχανισμοί εισόδου δεδομένων στις μονάδες οργάνωσης προγράμματος από εξωτερικές πηγές. Δηλώνονται με τις λέξεις VAR_INPUT … END_VAR στα προγράμματα, συναρτήσεις και συναρτησιακές οντότητες. Στη συνέχεια δίνεται ένα παράδειγμα ορισμού μεταβλητών αυτής της κατηγορίας.

```
VAR_INPUT
Setpoint : REAL;
Max_Count : USINT;
END_VAR
```

Όμοια, οι μεταβλητές εξόδου δρουν ως μηχανισμοί εγγραφής δεδομένων από τις μονάδες οργάνωσης προγράμματος και εγγραφής των δεδομένων αυτών σε εξωτερικές μεταβλητές. Η δήλωση μεταβλητών αυτής της κατηγορίας γίνεται με τις λέξεις VAR_OUTPUT … END_VAR, όπως φαίνεται στο παράδειγμα που ακολουθεί

```
VAR_OUTPUT
Message :STRING(10);
Status : BOOL;
```

END_VAR

Είναι δυνατόν να οριστούν μεταβλητές που θα δρουν ταυτόχρονα και ως μεταβλητές εισόδου και ως μεταβλητές εξόδου μιας μονάδας οργάνωσης προγράμματος. Δηλώνονται με τις λέξεις VAR_IN_OUT … END_VAR και μπορούν να λάβουν τιμές από εξωτερικές μεταβλητές. Η τυπική χρήση των μεταβλητών αυτών είναι ο έλεγχος του τρόπου λειτουργίας μιας συναρτησιακής οντότητας. Ας υποθέσουμε ότι μια συναρτησιακή οντότητα έχει μια μεταβλητή εισόδου/εξόδου που ονομάζεται AUTO και η οποία συνδέεται με την εξωτερική μεταβλητή MAIN_MODE. Η αρχική τιμή της AUTO που μπορεί να είναι η INIT, μπορεί να γραφεί στην MAIN_MODE με λογισμικό ανεξάρτητο της συναρτησιακής οντότητας. Όταν η συναρτησιακή οντότητα αρχίζει την εκτέλεσή της, διαβάζει την είσοδο AUTO. Κατά την εκτέλεσή της η συναρτησιακή οντότητα μπορεί να αλλάξει τον τρόπο λειτουργίας της εγγράφοντας στην AUTO την τιμή READY. Ο νέος τρόπος λειτουργίας της οντότητας μπορεί να διαβαστεί είτε στο επίπεδο της οντότητας από τη μεταβλητή AUTO είτε στο επίπεδο άλλης μονάδας οργάνωσης προγράμματος από τη μεταβλητή MAIN_MODE. Η χρησιμοποίηση μεταβλητών αυτής της κατηγορίας ενδείκνυται όταν χρειάζεται να περαστούν τα δεδομένα μεταβλητών με πολλά στοιχεία σε μια μονάδα οργάνωσης προγράμματος. Τότε, αντί να χρησιμοποιηθούν μεμονωμένες μεταβλητές των κατηγοριών VAR_INPUT και VAR_OUTPUT, είναι προτιμότερο να χρησιμοποιηθεί μεταβλητή της κατηγορίας VAR_IN_OUT, διότι στην περίπτωση αυτή εγγράφεται στη μονάδα οργάνωσης προγράμματος η διεύθυνση της μεταβλητής και όχι αυτή καθαυτή η μεταβλητή. Το παράδειγμα που ακολουθεί δείχνει πώς μπορεί να ορισθεί μια τέτοια μεταβλητή.

```
TYPE
MODE_LIST : ( INIT, READY, RUNNING, STOPPED );
 END_TYPE
 VAR_IN_OUT
 AUTO : MODE_LIST;
 END_VAR
```

Οι μεταβλητές απεικόνισης θέσεων μνήμης επιτρέπουν την αναφορά στα περιεχόμενα των θέσεων μνήμης του υπολογιστή. Κάθε μεταβλητή απεικόνισης θέσης μνήμης δηλώνεται με το χαρακτήρα '%'. Ο χαρακτήρας αυτός ακολουθείται από κώδικα ενός έως δύο γραμμάτων που προσδιορίζει αν η θέση της μνήμης σχετίζεται με είσοδο, έξοδο ή καταχωρητή της κύριας μνήμης του υπολογιστή καθώς και τον τύπο της οργάνωσης των δεδομένων στη συγκεκριμένη θέση, δηλαδή αν το περιεχόμενο της μνήμης είναι εκφρασμένο σε κώδικα μεμονωμένων bits ή bytes ή words. Τα γράμματα που χρησιμοποιούνται για να δηλώσουν τη σχέση της θέσης της μνήμης με είσοδο, έξοδο, εσωτερικό καταχωρητή και την οργάνωση των δεδομένων, δίνονται αντίστοιχα στους Πίνακες 4.6 και 4.7. Βασικά, η ανάγκη χρήσης τέτοιων μεταβλητών παρουσιάζεται στους προγραμματιζόμενους λογικούς ελεγκτές και σε υπολογιστές που προσομοιώνουν τις αρχές λειτουργίας των προγραμματιζόμενων λογικών ελεγκτών. Ως γνωστό ο χώρος των διευθύνσεων της μνήμης_κάθε προγραμματιζόμενου ελεγκτή είναι οργανωμένος σε τρεις περιοχές (α) την περιοχή των θέσεων μνήμης στις οποίες αποθηκεύονται τιμές διακριτών, ψηφιακών και αναλογικών εισόδων, (β) την περιοχή των θέσεων μνήμης που αποθηκεύονται οι τιμές των διακριτών, ψηφιακών και αναλογικών εξόδων και (γ) την περιοχή που αποθηκεύονται προσωρινά δεδομένα.

Πρώτο γράμμα	Ερμηνεία
I	θέση μνήμης που αποθηκεύει δεδομένα εισόδου
Q	θέση μνήμης που αποθηκεύει δεδομένα εξόδου
M	εσωτερικός καταχωρητής

Πίνακας 4.6 *Δηλωτικοί χαρακτήρες της σχέσης θέσης μνήμης με είσοδο, έξοδο και εσωτερικό καταχωρητή υπολογιστή.*

Δεύτερο γράμμα	Ερμηνεία
X	Bit
B	Byte (8 bits)
W	Word (16 bits)
D	Double word (32 bits)
L	Long word (64 bits)

Πίνακας 4.7 *Δηλωτικοί χαρακτήρες του μήκους δεδομένων*.

Το τρίτο μέρος της μεταβλητής αποτελείται από ένα ή περισσότερα αριθμητικά πεδία που χωρίζονται μεταξύ τους με τελεία (.). Τα πεδία αυτά προσδιορίζουν τη διεύθυνση της μνήμης και διαφέρουν από ελεγκτή σε ελεγκτή. Γενικά, όμως, τα πεδία αυτά μπορούν να θεωρηθούν ότι σχηματίζουν μια ιεραρχική διεύθυνση που καθορίζει υποδιαιρέσεις, μονάδες, κανάλια, κτλ. Παρακάτω δίνονται μερικά παραδείγματα ορισμού μεταβλητών αναφοράς σε θέσεις μνήμης.

```
%I100 (* Το ψηφίο 100 της μνήμης εισόδων *)
%IX100 (* Το ίδιο ψηφίο με %I100 *)
%IW122 (* Η λέξη 122 της μνήμης εισόδων *)
%IW10.1.21 (* Κανάλι 21 στη μονάδα 1 που είναι - *)
(* τοποθετημένη στη σχισμή 10 *)
%QL100 (* Η λέξη των 64 bits με διεύθυνση)
(* μνήμης εξόδων 100 *)
%MW132 (* Η λέξη 132 της μνήμης των *)
(* εσωτερικών καταχωρητών *)
```

Κάθε μεταβλητή μπορεί να συνοδεύεται από τα χαρακτηριστικά (attributes) RETAIN, CONSTANT και AT. Όταν μια σειρά μεταβλητών έχει χαρακτηριστεί ως RETAIN, τότε σε περίπτωση διακοπής της ηλεκτρικής τροφοδοσίας του υπολογιστή οι τιμές που είχαν οι μεταβλητές τη στιγμή της διακοπής διατηρούνται και επαναφέρονται όταν ο υπολογιστής τροφοδοτηθεί ξανά με ισχύ. Στο παράδειγμα που ακολουθεί δείχνεται ο τρόπος ορισμού των μεταβλητών Speed_Profile και Max_Speed με χαρακτηριστικό RETAIN.

```
VAR_OUT RETAIN
Speed_Profile: ARRAY[1…4] OF REAL;
Max_speed : REAL;
END_VAR
```

Με το χαρακτηριστικό CONSTANT οι τιμές που έλαβαν οι μεταβλητές μιας λίστας κατά την αρχικοποίησή τους και οι οποίες δεν ανήκουν στη κατηγορία των εξωτερικών μεταβλητών, δεν μπορούν να τροποποιηθούν καθ' όλη τη διάρκεια εκτέλεσης του λογισμικού. Το παράδειγμα που ακολουθεί δείχνει τον τρόπο ορισμού μεταβλητών με χαρακτηρισμό CONSTANT.

```
VAR CONSTANT
Startup_Speed: REAL :=12.3;
Gear Ratio: SINT :=12;
END_VAR
```

Με το χαρακτηριστικό AT μια μεταβλητή κλειδώνεται με συγκεκριμένη διεύθυνση θέσης μνήμης της επιλογής του προγραμματιστή και δεν αφήνεται στο μεταγλωττιστή να αντιστοιχήσει τη μεταβλητή σε θέση μνήμης. Ως παράδειγμα αναφέρεται ότι η μεταβλητή INPUT_1 μπορεί να αντιστοιχηθεί στη μνήμη με διεύθυνση 100, γράφοντας την εντολή:

```
VAR
```

```
SCAN_DATA AT %IW100 : ARRAY[1…8] OF SINT;
DIG_OUTPUTS AT %QX120 : ARRAY[0…15] OF BOOL;
END_VAR
```

Όσον αφορά τις μεταβλητές πρόσβασης σε άλλες προγραμματιζόμενες διατάξεις με τις οποίες μπορεί να επικοινωνεί ο υπολογιστής μέσω δικτύου δεδομένων, αυτές μπορούν να συσχετίζονται μόνο σε μεταβλητές των κατηγοριών

- εισόδου ή εξόδου ενός προγράμματος,
- γενικές μεταβλητές,
- μεταβλητές αναφοράς σε θέσεις μνήμης,
- πίνακες δεδομένων.

Επίσης, μπορούν να συνοδεύονται από τα χαρακτηριστικά READ_ONLY και READ_WRITE, τα οποία καθορίζουν αν η προγραμματιζόμενη διάταξη που επικοινωνεί με τον υπολογιστή μπορεί μόνο να διαβάσει τη μεταβλητή πρόσβασης ή και να τροποποιήσει την τιμή της αντίστοιχα. Στο παράδειγμα που ακολουθεί, μέσω των μεταβλητών πρόσβασης σε άλλες προγραμματιζόμενες διατάξεις επιτυγχάνεται ανάγνωση ή εγγραφή τιμών από ή στις γενικές μεταβλητές SPEED και LENGTH του λογισμικού εφαρμογής ενός προγραμματιζόμενου ελεγκτή και στην παράμετρο εισόδου START_UP του προγράμματος LINE1. Προγραμματιζόμενες διατάξεις που επικοινωνούν με τον ελεγκτή μπορούν να χρησιμοποιήσουν τις μεταβλητές LINE_START_UP, LINE_SPEED και GOOD_CABLE για να διαβάσουν και να γράψουν δεδομένα από και στις μεταβλητές START_UP και SPEED, αλλά και να διαβάσουν μόνον τη μεταβλητή LENGTH.

```
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
```

Τέλος, θα πρέπει να επισημανθεί ότι κάθε μεταβλητή μπορεί να αρχικοποιηθεί με τιμή διαφορετική από αυτήν που της δίνει ο μεταγλωττιστής αν περιληφθεί στο λογισμικό η σχετική εντολή, η οποία έχει τη μορφή που δείχνεται στα παρακάτω παραδείγματα.

Παράδειγμα αρχικοποίησης δύο εσωτερικών μεταβλητών

```
VAR
 Process_Runs : INT :=10;
 Max Temp : Real := 350.0;
END_VAR
```

Παράδειγμα αρχικοποίησης επιλεγμένων μεταβλητών μιας δομής δεδομένων

```
VAR GLOBAL
 SENSOR1 : PRESSURE_SENSOR ( PRESSURE := 4.0, HIGH_LIMIT := 50.0);
END_VAR
```

4.12 Ορισμός συναρτησιακών οντοτήτων

Νέοι τύποι συναρτήσεων και συναρτησιακών οντοτήτων μπορούν να οριστούν σχεδόν με τον ίδιο τρόπο που ορίζονται νέοι τύποι μεταβλητών και δομών δεδομένων. Για τον ορισμό ενός νέου τύπου συναρτησιακής οντότητας χρησιμοποιούνται οι λέξεις FUNCTION_BLOCK και END_FUNCTION_BLOCK, μεταξύ των οποίων πρέπει να παρεμβληθεί ο κατάλογος του ορισμού των παραμέτρων εισόδου, εξόδου και εσωτερικών μεταβλητών και να ακολουθήσει το πρόγραμμα που υλοποιεί τη συναρτησιακή οντότητα. Το πρόγραμμα αυτό πρέπει να είναι γραμμένο σε μια από τις γλώσσες ST, FBD, LD, SFC και IL, που υποστηρίζονται από το

πρότυπο και αναπτύσσονται στο_επόμενο κεφάλαιο. Το πρόγραμμα της συναρτησιακής οντότητας μπορεί να περιέχει αναφορές σε στιγμιότυπα_άλλων συναρτησιακών οντοτήτων είτε της βιβλιοθήκης του περιβάλλοντος στο οποίο αναπτύσσεται το λογισμικό ή σε οντότητες που έχουν ήδη οριστεί. Επίσης, μπορεί να περιέχει αναφορές σε εξωτερικές μεταβλητές, όπως είναι οι γενικές μεταβλητές που ορίζονται σε επίπεδο προγράμματος, στο οποίο εντάσσεται η νέα συναρτησιακή οντότητα, σε επίπεδο πόρου και σε επίπεδο αρχιτεκτονικής. Αντίστοιχα μπορούν να οριστούν στιγμιότυπα του νέου τύπου συναρτησιακής οντότητας με τη χρήση των εντολών ορισμού μεταβλητών ή δομών δεδομένων. Στιγμιότυπα νέου τύπου συναρτησιακής οντότητας μπορούν να οριστούν μόνο σε επίπεδο προγράμματος ή άλλης συναρτησιακής οντότητας. Στη συνέχεια παρατίθεται ένα παράδειγμα ορισμού ενός νέου τύπου συναρτησιακής οντότητας και ενός στιγμιότυπού του σε επίπεδο άλλης συναρτησιακής οντότητας.

4.12.1 Παράδειγμα ορισμού νέας συναρτησιακής οντότητας

Ας θεωρήσουμε ότι επιθυμούμε να ορίσουμε μια συναρτησιακή οντότητα η οποία ονομάζεται COUNTER. Αυτή η οντότητα θα πρέπει να έχει μια είσοδο με όνομα MODE και μια έξοδο με όνομα OUT. Η είσοδος MODE μπορεί να παίρνει μόνον τις εξής τρεις τιμές:

- RESET
- COUNT
- HOLD

Όταν η συναρτησιακή οντότητα τίθεται σε κατάσταση RESET, η έξοδός της είναι πάντοτε 0. Όταν τίθεται σε κατάσταση COUNT, η έξοδος OUT αυξάνεται κατά 1 μετά από κάθε εκτέλεση του προγράμματος της οντότητας. Όταν η δομή τίθεται σε κατάσταση HOLD, στην έξοδό της διατηρείται η τιμή που έλαβε η μεταβλητή OUT την τελευταία χρονική στιγμή πριν από την αλλαγή της κατάστασης. Ο ορισμός του τύπου της συναρτησιακής οντότητας COUNTER γίνεται ως εξής. Πρώτα ορίζεται σε επίπεδο αρχιτεκτονικής ένας τύπος δεδομένων με όνομα ModeType που σχετίζεται με την είσοδο MODE. Στη συνέχεια ορίζεται η συναρτησιακή οντότητα και εντός της οντότητας ορίζονται πρώτα οι τύποι δεδομένων, αρχικοποιούνται οι παράμετροι εισόδου και εξόδου και γράφεται ο κώδικας του προγράμματος της οντότητας. Ο κώδικας του προγράμματος στο παράδειγμα είναι γραμμένος στη γλώσσα ST που αναπτύσσεται στο επόμενο κεφάλαιο. Οι συγκεκριμένες εντολές IF που χρησιμοποιούνται στο παράδειγμα είναι ίδιες με αυτές της γλώσσας Pascal και C και επομένως η κατανόηση του προγράμματος είναι εφικτή.

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

FUNCTION_BLOCK COUNTER
 VAR_INPUT
 MODE : ModeType :=RESET;
 END_VAR
 VAR_OUTPUT
 OUT : INT :=0;
 END_VAR
 IF MODE=RESET THEN
 OUT :=0;
 ELSEIF MODE =COUNT THEN
 OUT :=OUT + 1;
END_IF;
END_FUNCTION_BLOCK
```

Ο ορισμός ενός στιγμιότυπου της παραπάνω συναρτησιακής οντότητας γίνεται μέσα σε άλλη μονάδα οργάνωσης προγράμματος. Στο πρόγραμμα COUNT1 το οποίο ακολουθεί δίνεται ένα παράδειγμα ορισμού του στιγμιότυπου C1 της παραπάνω συναρτησιακής οντότητας COUNTER.

```
PROGRAM COUNT1
 VAR_INPUT
InputMode : ModeType; (* ορισμός μεταβλητής InputMode που μπορεί *)
 (*να επηρεάζεται από άλλη μονάδα *)
 (*οργάνωσης προγράμματος *)
END_VAR
 VAR_OUTPUT
 Max_Count : INT;
 END_VAR
 C1 : COUNTER; (* Κλήση του στιγμιότυπου C1 θέτοντας την *)
 END_VAR (* παράμετρο MODE στην εκάστοτε τιμή της *)
 C1 (MODE :=InputMode);
 Max_Count := C1.OUT; (* Αποθήκευσε την τιμή της εξόδου του*
 (*στιγμιότυπου σε μια μεταβλητή *)
END_PROGRAM
```

4.13 Ορισμός προγράμματος

Το πρόγραμμα είναι η μεγαλύτερη μονάδα οργάνωσης λογισμικού και μπορεί να οριστεί μόνο στο επίπεδο του πόρου. Πρακτικά ως ιδέα το πρόγραμμα είναι όμοιο με συναρτησιακή οντότητα και περιλαμβάνει εισόδους, εξόδους και κώδικα που περιγράφει τη συμπεριφορά του, γραμμένο σε μία ή περισσότερες γλώσσες. Γενικά τα προγράμματα θεωρούνται σε σχέση με τις συναρτησιακές οντότητες ως οντότητες λογισμικού με μεγάλου μήκους κώδικα και εκτεταμένη χρήση των βιβλιοθηκών των συναρτήσεων και συναρτησιακών οντοτήτων.

Για τον ορισμό ενός προγράμματος χρησιμοποιούνται οι λέξεις PROGRAM και END_PROGRAM μεταξύ των οποίων πρέπει να περιληφθούν οι ορισμοί των εσωτερικών μεταβλητών και των μεταβλητών εισόδου/εξόδου, ακολουθούμενοι από τον κώδικα που καθορίζει τη συμπεριφορά του προγράμματος. Ο ορισμός ενός προγράμματος περιλαμβάνει συνήθως και στιγμιότυπα συναρτησιακών οντοτήτων και εξωτερικών μεταβλητών, δηλαδή γενικών μεταβλητών οι οποίες ορίζονται στα επίπεδα του πόρου και της αρχιτεκτονικής του λογισμικού. Στο παράδειγμα που ακολουθεί ορίζεται ένα πρόγραμμα και ένα στιγμιότυπο του προγράμματος.

Ορισμός προγράμματος

```
PROGRAM fermenter
 VAR_INPUT
 Reagent_Code : INT;
 Sterilize : BOOL;
 Ferment_Period : TIME;
 END_VAR
 VAR_OUTPUT
 Yield : REAL;
 Status := WORD;
 END_VAR
 VAR
 Ph_Loop, Temp_Loop : PID;
 Phase : INT :=1;
 END_VAR
(* κώδικας προγράμματος*)
END_PROGRAM
```

Ορισμός στιγμιότυπου

```
PROGRAM Line1 : Fermenter;
PROGRAM CT1, CT2 : COUNT1;
```

Line1 είναι ένα στιγμιότυπο του προγράμματος Fermenter , ενώ CT1 και CT2 είναι στιγμιότυπα της συναρτησιακής οντότητας COUNT1. Ο ορισμός ενός προγράμματος μπορεί να περιλαμβάνει και συνδέσεις των εισόδων και εξόδων του με μεταβλητές που ορίζονται εκτός προγράμματος, όπως αυτό φαίνεται στο ακόλουθο παράδειγμα.

```
PROGRAM Line1 Fermenter ( Reagent_Code : = A1,
(Bonfati, Monari, & Sampieri, 1997) Yield => AJ_43, Status=> KX56 );
```

Πολλές φορές χρειάζεται να ανατεθεί ο έλεγχος της εκτέλεσης ενός στιγμιότυπου προγράμματος σε διεργασία. Για να γίνει η ανάθεση αυτή χρησιμοποιείται η λέξη WITH, όπως φαίνεται και στο παράδειγμα που ακολουθεί

```
PROGRAM line2 WITH Slow_Task : Packaging_Line,
Speed := %IW21_10,
Product _rate => %qw33;
Sampler1 WITH Fast_task;
Sealer1 WITH Fast_task ;
```

Σύμφωνα με τις εντολές της ανάθεσης, line2 είναι ένα στιγμιότυπο του προγράμματος Packaging_Line που τρέχει κάτω από τον έλεγχο της διεργασίας Slow_task. Η είσοδος του προγράμματος Speed λαμβάνει μια τιμή που προέρχεται κατευθείαν από μεταβλητή αναφοράς σε μνήμη, η οποία μνήμη σχετίζεται με συγκεκριμένη είσοδο του υπολογιστή. Η έξοδος του προγράμματος Product_Rate γράφεται σε μια θέση μνήμης του υπολογιστή που σχετίζεται με συγκεκριμένη έξοδό του. Τα στιγμιότυπα των συναρτησιακών οντοτήτων Sampler1 και Sealer1 δηλώνονται μέσα στο πρόγραμμα να τρέχουν κάτω από τον έλεγχο της διεργασίας Fast_Task.

4.14 Ορισμός πόρου

Ο πόρος αντιστοιχεί σε μονάδα επεξεργασίας που μπορεί να εκτελεί προγράμματα. Ο ορισμός ενός πόρου γίνεται στο επίπεδο περιγραφής της αρχιτεκτονικής του λογισμικού με τη λέξη RESOURCE η οποία ακολουθείται από ένα χαρακτηριστικό σύμβολο αναγνώρισης της μονάδας επεξεργασίας και του τύπου του επεξεργαστή που χρησιμοποιεί ο πόρος. Επιπλέον περιλαμβάνονται στον ορισμό του πόρου ορισμοί μεταβλητών, διεργασιών και προγραμμάτων. Ο ορισμός περατώνεται με τη γραφή της λέξης END_RESOURCE. Παρακάτω δίνεται ένα παράδειγμα ορισμού του πόρου Res2, που εκτελεί το πρόγραμμα diagnose1 συνεχώς και το πρόγραμμα Logger1 κάτω από τον έλεγχο της διεργασίας LOG_TASK, στον επεξεργαστή 8044.

```
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.
```

4.15 Ορισμός διεργασίας

Στο βιομηχανικό έλεγχο ο σχεδιαστής του λογισμικού επιθυμεί να έχει έναν ευέλικτο τρόπο καθορισμού των ρυθμών εκτέλεσης διαφόρων προγραμμάτων και συναρτησιακών οντοτήτων του λογισμικού. Ως παράδειγμα θα μπορούσε να αναφέρει κανείς την περίπτωση ελέγχου από ένα προγραμματιζόμενο λογικό ελεγκτή της θερμοκρασίας του παραγόμενου ατμού από έναν ατμολέβητα ταυτόχρονα με την υλοποίηση της λογικής οδήγησης σε πλήρως ανοικτή θέση των βαλβίδων εκτόνωσης της πίεσης στο εσωτερικό του ατμολέβητα, σε περίπτωση που είτε από λανθασμένο χειρισμό ή από απόφραξη των σωληνώσεων απομάστευσης του ατμού αυξηθεί σε επικίνδυνη τιμή η πίεση στον ατμολέβητα. Για τον έλεγχο της θερμοκρασίας η σχετική συναρτησιακή οντότητα χρειάζεται να εκτελείται μια φορά κάθε 5 min, ενώ για την παρακολούθηση της πίεσης του ατμολέβητα και την ενεργοποίηση της εντολής ανοίγματος των βαλβίδων εκτόνωσης το σχετικό πρόγραμμα χρειάζεται να εκτελείται κάθε 5 ms. Με την ανάθεση της εκτέλεσης προγραμμάτων και συναρτησιακών οντοτήτων σε διαφορετικές διεργασίες, είναι δυνατόν να ρυθμιστεί η εκτέλεση των προγραμμάτων αυτών έτσι ώστε να γίνεται είτε κατά περιόδους είτε σε περίπτωση που αλλάζει η κατάσταση μιας λογικής μεταβλητής.

Για τον ορισμό μιας διεργασίας χρησιμοποιείται η λέξη TASK ακολουθούμενη από το συμβολικό όνομα της διεργασίας και τις τιμές των τριών προαιρετικά χρησιμοποιούμενων παραμέτρων, που αναφέρονται στον Πίνακα 4.8. Παρακάτω δίνονται τρία τυπικά παραδείγματα ορισμού διεργασιών.

```
TASK FAST_INTERLOCKS (INTERVAL := t#30ms, PRIORITY := 0);
TASK LOG_TASK (SINGLE ;=LogFlag, PRIORITY := 3);
TASK CONTROL_TASK (INTERVAL := t#500ms, PRIORITY ;= 1);
```

‘Όπως προαναφέρθηκε η εκτέλεση στιγμιότυπων συναρτησιακών οντοτήτων και προγραμμάτων μπορεί να ανατεθεί σε διαφορετικές διεργασίες. Αυτό προϋποθέτει ότι, όταν τμήματα ενός προγράμματος εκτελούνται με διαφορετικούς ρυθμούς, θα πρέπει ορισμένοι έξοδοι να ενημερώνονται μέσα σε καθορισμένα χρονικά όρια σε περίπτωση που διαπιστωθεί αλλαγή στην κατάσταση των εισόδων. Η ανάθεση της εκτέλεσης ενός στιγμιότυπου προγράμματος ή συναρτησιακής οντότητας γίνεται με τη λέξη WITH, όπως αυτό παρουσιάστηκε στο παράδειγμα της ενότητας 4.1.

Παράμετρος	τύπος δεδομένων	Περιγραφή
SINGLE	BOOL	Ορισμός λογικής μεταβλητής που κατά τη μετάβαση της από 0 σε 1 προκαλεί την εκτέλεση της διεργασίας μια φορά
INTERVAL	TIME	Ορισμός της περιόδου που πρέπει να μεσολαβήσει μεταξύ διαδοχικών εκτελέσεων της διεργασίας
PRIORITY	UINT	Το επίπεδο προτεραιότητας της διεργασίας. Το 0 θεωρείται η υψηλότερη προτεραιότητα και αυξάνοντας τον αριθμό σε 1, 2, 3, 4 μειώνεται η προτεραιότητα.

Πίνακας 4.8 *Παράμετροι ορισμού διεργασίας.*

Είναι, όμως, δυνατόν η εκτέλεση στιγμιότυπων συναρτησιακών οντοτήτων να μην ανατίθεται σε άλλες διεργασίες, αλλά να γίνεται από την ίδια διεργασία στην οποία ανήκει το μητρικό πρόγραμμα. Επίσης, όταν ή εκτέλεση ενός στιγμιότυπου προγράμματος δεν έχει ανατεθεί σε συγκεκριμένη διεργασία, τότε το στιγμιότυπο αυτό εκτελείται με τη χαμηλότερη προτεραιότητα.

4.16 Ορισμός αρχιτεκτονικής λογισμικού

Η αρχιτεκτονική του λογισμικού περιγράφει τη δομή του λογισμικού το οποίο γράφεται για κάποια εφαρμογή και κατά κανόνα θα περιλαμβάνει τουλάχιστον έναν πόρο. Η αρχιτεκτονική αναφέρεται σε ένα συγκεκριμένο τύπο υπολογιστή και συγκεκριμένη διαμόρφωση του υλικού του. Γενικά η αρχιτεκτονική του λογισμικού θα αντιστοιχεί σε υπολογιστή που θα περιλαμβάνει:

- συγκεκριμένους επεξεργαστές, π.χ. Intel pendium, ARM , κτλ.,
- συγκεκριμένες διευθύνσεις για κάθε είσοδο και έξοδο,
- συγκεκριμένες δυνατότητες λειτουργικού συστήματος, όπως είναι μέγιστος αριθμός διεργασιών και μέγιστος ρυθμός εκτέλεσης προγράμματος.

Η δήλωση της αρχιτεκτονικής του λογισμικού γίνεται με τη λέξη CONFIGURATION και περατώνεται με τη λέξη END_CONFIGURATION. Η δήλωση αυτή μπορεί να γίνεται είτε με κείμενο ή με γραφικό τρόπο. Ένα παράδειγμα περιγραφής της αρχιτεκτονικής λογισμικού παρατίθεται στη συνέχεια.

```
CONFIGURATION unit_1_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.Start: 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
```

Στο παράδειγμα αυτό υποτίθεται ότι ο υπολογιστής έχει δύο επεξεργαστές τους : PROC_386 και PROC_8044 και κατά συνέπεια ορίζονται δύο πόροι με τα συμβολικά ονόματα Res1 και Res2. Ο πρώτος πόρος Res1 περιέχει το στιγμιότυπο turbine1 του προγράμματος turbine, που έχει συνδέσεις των παραμέτρων των εισόδων και των εξόδων του με γενικές μεταβλητές και μεταβλητές αναφοράς στη μνήμη. Η εκτέλεση των στιγμιότυπων των συναρτησιακών οντοτήτων loop1, ramp1 και io_scanner1, οι οποίες ανήκουν στο στιγμιότυπο του προγράμματος turbine1 έχει ανατεθεί σε διαφορετικές διεργασίες που εκτελούνται περιοδικά και διαφορετικά από αυτήν του στιγμιότυπου του προγράμματος. Οι γενικές μεταβλητές, όπως είναι η G_speed_setpoint, οι οποίες δηλώνονται στο επίπεδο της αρχιτεκτονικής του λογισμικού, μπορούν να διαβάζονται και να εγγράφονται από όλα τα προγράμματα και τις συναρτησιακές οντότητες οι οποίες ανήκουν στη συγκεκριμένη αρχιτεκτονική. Ο δεύτερος πόρος περιέχει τα δύο στιγμιότυπα προγραμμάτων logger1 και diagnose1. Το στιγμιότυπο logger1 τρέχει κάτω από τον έλεγχο της διεργασίας LOG_TASK, η ενεργοποίηση της οποίας γίνεται όταν αλλάζει η κατάσταση της γενικής μεταβλητής G_log_event από 0 σε 1. Το στιγμιότυπο αυτό εγγράφει δεδομένα στον πίνακα G_Log_Data. Η εκτέλεση του στιγμιότυπου του προγράμματος diagnose1 περιλαμβάνεται στην ουρά των υπό εκτέλεση προγραμμάτων κάθε φορά που ο δρομολογητής του λειτουργικού συστήματος διαμορφώνει την ουρά αυτή. Η χρήση των μεταβλητών UNIT1_START και UNIT1_ALARM δίνει τη δυνατότητα σε ένα χειριστή να αρχίσει μια ακολουθία εκκίνησης μέσα από το πρόγραμμα turbine1 γράφοντας δεδομένα από το σταθμό χειρισμών και εποπτείας στη μεταβλητή UNIT1_START και να ελέγξει για την ύπαρξη προειδοποιήσεων υπέρβασης ορίων κανονικής λειτουργίας διαβάζοντας τη μεταβλητή UNIT1_ALARM. Τέλος, εκτελώντας το στιγμιότυπο του προγράμματος logger1 εγγράφονται δεδομένα στη μεταβλητή UNIT1_LOG. Τα δεδομένα αυτά γίνονται προσβάσιμα από το σταθμό χειρισμών και εποπτείας μέσω της μεταβλητής πρόσβασης UNIT1_DATA.

Βιβλιογραφία/Αναφορές

Bonfati, F., Monari, P.-D., & Sampieri, U. (1997). *IEC 1131-3 Programming Methodology.* Seyssins France: CJ International.

IEC61131-3. (2013). *Programmable controllers-Part 3 Programming languagres.* International Electrotechnical Commission.

ISO/IEC 646. (1991). *Information technology -- ISO 7-bit coded character set for information interchange.* International Standards Organization.

Kart-Heinz, J., & Tiegelkamp, M. (2010). *Programming Industrial Automation Systems.* Berlin Heindelberg: Springer-Verlag.

Lewis, R. W. (1995). *Programming industrial control systems using IEC 1131-3.* London: The Institution of Electrical Engineers.