1. Ανάπτυξη λογισμικού

1.1 Η μηχανική λογισμικού.

Με τον όρο "ανάπτυξη λογισμικού" εννοούμε την θεωρία και τις πρακτικές τεχνικές και συγκεκριμένες μεθόδους που μπορούν να χρησιμοποιηθούν σε διάφορες εφαρμογές ανάπτυξης και συντήρησης έργων λογισμικού (software), είτε αυτές είναι εμπορικές εφαρμογές, είτε επιστημονικές, είτε εφαρμογές ελέγχου συστημάτων (system software). Είναι όμως η ανάπτυξη λογισμικού συνώνυμο του "κατασκευάζω προγράμματα ηλεκτρονικών υπολογιστών";

Στην πράξη οι περισσότεροι προγραμματιστές, αν και υποστηρίζουν ότι δουλεύουν βάσει κάποιας επιστημονικής μεθόδου στην ανάπτυξη των προγραμμάτων τους, ακολουθούν την πρακτική "κάνω κάτι και βλέπω τι θα γίνει στην συνέχεια". Ακολουθούν οι "κλασικοί" μηχανικοί (οι πολιτικοί ή οι ηλεκτρολόγοι) αυτήν την μέθοδο; Ασφαλώς όχι! Κανένας πολιτικός μηχανικός, για παράδειγμα, δεν αρχίζει να κτίζει ένα οίκημα, αν δεν έχει κάνει πρώτα τις απαραίτητες μελέτες που αφορούν το συγκεκριμένο έργο. Μελέτες που αφορούν το έδαφος στο οποίο θα στηριχθεί το οίκημα, την χωροταξία, τα θεμέλια, τον σκελετό, την τελική όψη του οικήματος και την διαμόρφωση του περιβάλλοντος γύρω από αυτό. Εφαρμόζει δοκιμασμένες τεχνικές και μεθόδους, χρησιμοποιεί τις απαιτούμενες γνώσεις του από την Φυσική, την Χημεία και τα Μαθηματικά και προτείνει κατά κανόνα συγκεκριμένα υλικά και τρόπους κατασκευής. Πριν ξεκινήσει η κατασκευή του οικήματος οι εργασίες που πρόκειται να γίνουν και ο τρόπος με τον οποίο θα εκτελεστούν περιγράφονται αναλυτικά σε μελέτες και σχέδια που έχουν εκπονηθεί. Ποιος προγραμματιστής εργάζεται με τον ίδιο τρόπο όταν πρόκειται να αναπτύξει ένα έργο λογισμικού;

Ακολουθώντας το παράδειγμα του μηχανικού στην ανάπτυξη λογισμικού, προσπαθούμε να χρησιμοποιήσουμε γνωστές τεχνικές και μεθόδους για την ολοκλήρωση ενός Συστήματος Πληροφορικής. Δυστυχώς όμως η τυποποίηση των τεχνικών και των μεθόδων στην πληροφορική δεν είναι ακόμα στο ίδιο επίπεδο με αυτό των άλλων μηχανικών. Είναι απαραίτητο οι επιστήμονες πληροφορικής να εργάζονται με το ίδιο τρόπο που εργάζονται οι υπόλοιποι μηχανικοί, όταν πρόκειται να αναπτύξουν κάποιο έργο λογισμικού με σκοπό να κατασκευάσουν ένα σύστημα πληροφορικής; Ένα σύστημα πληροφορικής δεν είναι τώρα πια ένα απλό εργαλείο κάποιου χρήστη ή μιας επιχείρησης, όπως είναι ίσως μια ταμειακή μηχανή ή ένα κοινό φωτοτυπικό μηχάνημα. Είναι το σύστημα μέσω του οποίου αποθηκεύονται, επεξεργάζονται και διακινούνται όλες οι πληροφορίες μιας επιχείρησης ή ενός οργανισμού. Τυχόν λάθη σε αυτό το σύστημα έχουν άμεσο αντίκτυπο στην μελλοντική πορεία του οργανισμού ή της επιχείρησης.

Η δημιουργία του τομέα της πληροφορικής ο οποίος ασχολείται με το αντικείμενο που περιγράψαμε προηγουμένως, ήταν επιβεβλημένη από την στιγμή που παρουσιάστηκε η ανάγκη κατασκευής μεγάλων έργων λογισμικού. Οι τεχνικές και οι μέθοδοι που χρησιμοποιούνται στην ανάπτυξη μικρών έργων λογισμικού (π.χ. παρακολούθηση πελατών ιατρείου) δεν βρίσκουν πάντα εφαρμογή σε μεγάλα έργα λογισμικού (π.χ. στην κατασκευή σχεδιαστικού πακέτου ή στην ανάπτυξη ενός λειτουργικού συστήματος). Ο αριθμός των μεγάλων έργων (projects) λογισμικού τα οποία δεν αποδίδουν τ' αναμενόμενα αποτελέσματα, που είναι αναξιόπιστα, δύσκολο να συντηρηθούν και με υψηλό κόστος κατασκευής, αυξάνει συνεχώς παρά την μεγάλη εξέλιξη της Πληροφορικής, και την παρουσία νέων εργαλείων ανάπτυξης λογισμικού. Ο τομέας της Πληροφορικής που ασχολείται με την τυποποίηση της ανάπτυξης του λογισμικού ονομάζεται Μηχανική Λογισμικού.

Είναι γενικά παραδεκτό ότι η τεχνολογία ανάπτυξης λογισμικού δεν ακολουθεί τους ρυθμούς εξέλιξης του υλικού (hardware). Το κόστος του υπολογιστικών συστημάτων και των περιφερειακών μειώνεται δραματικά, προσφέροντας πολύ ισχυρότερα συστήματα. Από την άλλη μεριά το κόστος υλοποίησης λογισμικού (software) ιδιαίτερα σε νέες εφαρμογές αυξάνει σημαντικά, χωρίς να προσφέρεται αντίστοιχη ποιότητα. Σκοπός λοιπόν της Μηχανικής

Λογισμικού είναι η τυποποίηση των διαφόρων τεχνικών και μεθόδων που έχουν κατά καιρούς παρουσιαστεί στην ανάπτυξη του λογισμικού και να υποδείξει την εφαρμογή αυτών σε διάφορα έργα λογισμικού με τελικούς στόχους:

(α) το χαμηλό κόστος παραγωγής λογισμικού

(β) την υψηλή ποιότητα και αποτελεσματικότητα του τελικού προϊόντος

(γ) την αξιοπιστία του συστήματος

(δ) την ευκολία συντήρησης του συστήματος

(ε) τη δυνατότητα μεταφοράς σε άλλο υπολογιστικό σύστημα

και τέλος - πολύ βασικό -

(στ) το προϊόν να είναι έτοιμο μέσα στα προκαθορισμένα χρονικά όρια.

Κάθε ένας από τους παραπάνω στόχους είναι παράλληλα και ένα πρόβλημα, καθώς η Μηχανική Λογισμικού δεν έχει καταφέρει μέχρι σήμερα να συστηματοποιήσει αποτελεσματικά την κατασκευή μεγάλων συστημάτων πληροφορικής και να προτείνει συγκεκριμένες μεθοδολογίες για πετύχει αυτούς τους στόχους.

1.2. Ο κύκλος ζωής λογισμικού.

Όλα τα μεγάλα συστήματα λογισμικού εκτός από τον χρόνο που χρειάζεται για να σχεδιαστούν και να κατασκευαστούν, για ένα μεγάλο χρονικό διάστημα θα χρησιμοποιηθούν (αυτός εξ' άλλου είναι και ο σκοπός των). Οι περισσότεροι ειδήμονες συμφωνούν λίγο-πολύ ότι τα στάδια της ζωής ενός έργου λογισμικού είναι:

- Η ανάλυση και ο καθορισμός προδιαγραφών του συστήματος
- Ο σχεδιασμός του συστήματος
- Η ανάπτυξη και η κατασκευή (υλοποίηση) των προγραμμάτων
- Ο έλεγχος της σωστής λειτουργίας
- Η χρήση και η συντήρηση του συστήματος

Η αρχική ιδέα του κύκλου ζωής ενός έργου λογισμικού δεν είναι καινούργια (Royce 1970) και υπάρχουν διάφορες παραλλαγές αυτής οι οποίες όμως δεν ξεφεύγουν του αρχικού σχεδιασμού. Για παράδειγμα κάποιοι προτείνουν κάθε στάδιο να χωρίζεται σε διάφορα υπο-στάδια.

Καθώς η ανάπτυξη ενός έργου λογισμικού είναι μια συνεχής διαδικασία, τα αποτελέσματα κάθε φάσης χρησιμοποιούνται σαν δεδομένα (feedback) για την επόμενη φάση (στάδιο). Αυτό έχει σαν αποτέλεσμα (και είναι σύνηθες το φαινόμενο) σε κάθε στάδιο να εντοπίζονται λάθη που έχουν γίνει στα προηγούμενα στάδια. Γι' αυτό ορισμένα μοντέλα (μεθοδολογίες) ανάπτυξης λογισμικού, προβλέπουν την επιστροφή σε προηγούμενο στάδιο του κύκλου ζωής ενός λογισμικού με σκοπό την διόρθωση των λαθών (Waterfall model) και επανεξέταση προηγουμένων φάσεων.

Άλλοι πάλι συνιστούν η διαδικασία ελέγχου να γίνεται σε κάθε στάδιο ανάπτυξης του λογισμικού και προτείνουν συγκεκριμένους τρόπους ή πρακτικές για επιτευχθεί αυτό. Η μέθοδος πρωτοτύπων (prototyping) για τον έγκαιρο εντοπισμό των λαθών προτείνει, σε ορισμένα στάδια τουλάχιστον, την κατασκευή ενός πρωτότυπου προϊόντος (με την βοήθεια για παράδειγμα μιας γλώσσας 4ης γενεάς - 4GL), το οποίο θα τίθεται στην διάθεση των μελλοντικών χρηστών του λογισμικού. Τα σχόλια των χρηστών, που θα προκύψουν από αυτή την δοκιμαστική λειτουργία, βοηθούν στην διάγνωση τυχόν λαθών. Γενικά η εμφάνιση λαθών, ιδιαίτερα σε στάδια όπου έχει τελειώσει η ανάπτυξη, σημαίνει αναξιοπιστία, παραπάνω κόστος και εγκυμονούν κινδύνους μη έγκαιρης παράδοσης του έργου.

Με την ευρεία της έννοια η Μηχανική Λογισμικού ασχολείται με όλες τις φάσεις της ζωής ενός λογισμικού, αλλά τα στάδια που κυρίως την ενδιαφέρουν είναι αυτά του σχεδιασμού, της κατασκευής και του ελέγχου του προϊόντος.

Η ανάλυση και ο καθορισμός των προδιαγραφών.

Στο στάδιο αυτό καταγράφονται αναλυτικά οι προδιαγραφές του συστήματος. Υπάρχουν:

- οι **λειτουργικέ**ς προδιαγραφές, οι οποίες δηλώνουν το τι πρέπει να επιτυγχάνει το σύστημα και
- οι **μη λειτουργικές** προδιαγραφές οι οποίες οριοθετούν τους διάφορους περιορισμούς που διέπουν το σύστημα.

Εδώ γίνεται η ανάλυση των στοιχείων που δίδονται στο σύστημα (δεδομένα), με κάθε λεπτομέρεια και περιγράφονται συνοπτικά οι ενέργειες που θα λάβουν χώρα, για να ικανοποιηθούν οι απαιτήσεις των χρηστών, καθώς και οι αλληλεπιδράσεις που θα προκύψουν μεταξύ αυτών. Επίσης περιγράφονται αναλυτικά τα αποτελέσματα που θα μας παρέχει η λειτουργία του συστήματος. Τέλος σε αυτό το στάδιο, λαμβάνονται αποφάσεις σχετικά τα μεγέθη και το είδος του hardware που πρόκειται να χρησιμοποιηθεί, καθώς και για το περιβάλλον εργασίας στο οποίο θα αναπτυχθεί το έργο (software) και ο τρόπος λειτουργίας. Όλα αυτά γίνονται με την στενή συνεργασία των μελλοντικών χρηστών του συστήματος οι οποίοι περιγράφουν τις ανάγκες και τις απαιτήσεις τους.

Ο σχεδιασμός του συστήματος.

Τα αποτελέσματα της ανάλυσης χρησιμοποιούνται για να γίνει ο σχεδιασμός του συστήματος. Καταγράφονται και περιγράφονται αναλυτικά ο τρόπος που θα γίνει η κάθε λειτουργία και οι τυχόν αλληλεπιδράσεις αυτών. Η περιγραφή αυτή πρέπει να είναι αρκετά αναλυτική και ίσως χρειαστεί να φτάσει μέχρι τον τρόπο κατασκευής ορισμένων προγραμμάτων (προδιαγραφές κώδικα). Τέλος ο τρόπος επικοινωνίας χρήστη / συστήματος επίσης καθορίζεται σε αυτό το στάδιο. Η Μηχανική λογισμικού είναι σε θέση σήμερα να προτείνει αρκετές μεθοδολογίες, οι οποίες μπορούν να εφαρμοστούν για να περιγραφεί κάποιος τυπικός σχεδιασμός (άρα ο έλεγχος του είναι δυνατός) ενός συστήματος.

Η υλοποίηση.

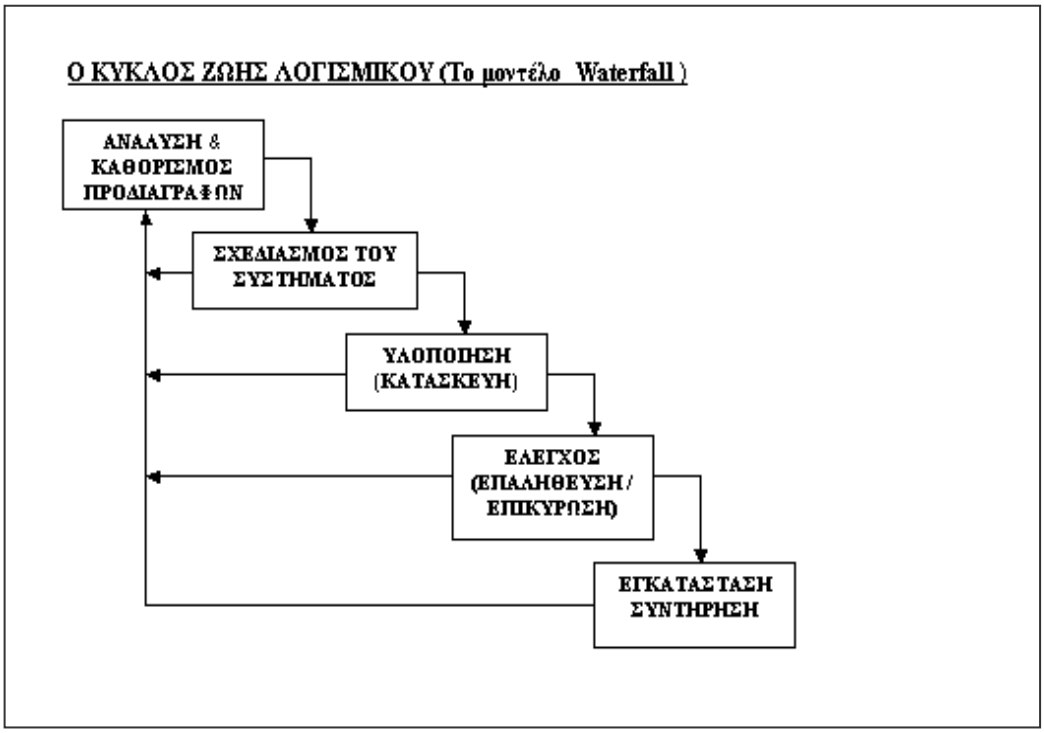
Γίνεται η κωδικοποίηση του σχεδιασμού του συστήματος σε κάποια γλώσσα προγραμματισμού και ενδεχομένως με την βοήθεια διαφόρων εργαλείων (έτοιμα προγράμματα, βιβλιοθήκες υποπρογραμμάτων, συστήματα διαχείρισης βάσεων δεδομένων, εργαλεία λειτουργικών συστημάτων κτλ). Η ανάπτυξη των αλγόριθμων και η δομή των προγραμμάτων περιγράφεται σε αυτή την φάση, πάντα έχοντας σαν οδηγό τον σχεδιασμό του συστήματος.

Ο έλεγχος.

Πρόκειται για χρονοβόρα αλλά απαραίτητη διαδικασία. Το προϊόν όμως πρέπει να ελεγχθεί, να γίνει επιβεβαίωση των όσων κάνει (είναι πραγματικά αυτά που χρειάζεται ο χρήστης) και αν τα κάνει σωστά (επαλήθευση). Αν δηλαδή το λογισμικό πραγματικά ικανοποιεί τις προδιαγραφές και τις απαιτήσεις του χρήστη. Παρόλα αυτά όμως, συχνό είναι το φαινόμενο, να παρουσιάζονται λάθη ακόμα και κατά την διάρκεια της επόμενης φάσης.

Η χρήση και η συντήρηση.

Ασφαλώς είναι η μεγαλύτερη σε διάρκεια φάση κάποιου έργου λογισμικού και η φυσική συνέπεια της κατασκευής αυτού (εκμετάλλευση). Μετά το πέρας της τελικής φάσης του ελέγχου του συστήματος, αυτό εγκαθίσταται στο κατάλληλο hardware (υπολογιστές, δίκτυα κτλ) και αρχίζει η λειτουργία και η χρήση του. Τυχόν λάθη διορθώνονται και τυχόν αλλαγές (πάντα θα υπάρχουν αλλαγές) θα γίνονται όταν το επιβάλουν οι περιστάσεις (συντήρηση). Η βελτίωση εξ' άλλου ενός συστήματος, το οποίο βρίσκεται σε λειτουργία, είναι μια συνεχής απαίτηση του πελάτη και θα αποτελεί πάντα μια πρόκληση για τον κατασκευαστή.



1.3 Ο καθορισμός απαιτήσεων/προδιαγραφών

Δεν υπάρχει λόγος να ξεκινήσουμε την διαδικασία ανάπτυξης και κατασκευής ενός έργου λογισμικού, αν δε γνωρίζουμε ακριβώς ποια θα είναι η χρησιμότητα του. Δηλαδή τι ακριβώς θα κάνει, ποιο πρόβλημα (διαδικαστικό, διοικητικό, οικονομικό, πρόβλημα υπαρκτό γενικά) πρόκειται, το μελλοντικό σύστημα λογισμικού να λύσει και ποιες απαιτήσεις θα ικανοποιήσει. Το πρόβλημα αυτό μπορεί να είναι από μια απλή παρακολούθηση του λογιστηρίου μιας επιχείρησης ή ενός οργανισμού, έως την διοίκηση ενός πύργου ελέγχου κάποιου αεροδρομίου ή ακόμα τον έλεγχο και την διαχείριση ενός διαστημικού ταξιδιού, όπου η αξιοπιστία του συστήματος παίζει σημαντικό ρόλο και τυχόν λάθος μεταφράζεται σε αρκετά υψηλό κόστος.

Η κατανόηση του "φυσικού" προβλήματος από τους κατασκευαστές του λογισμικού (software developers) μπορεί να είναι έργο πολύ δύσκολο. Η εξέταση και η καταγραφή των υπηρεσιών και των αποτελεσμάτων, που θα είναι σε θέση να προσφέρει το υπό κατασκευή σύστημα πληροφορικής και ο προσδιορισμός των περιορισμών κάτω από τους οποίους πρέπει να λειτουργήσει αυτό ονομάζεται ΑΝΑΛΥΣΗ ΤΩΝ ΑΠΑΙΤΗΣΕΩΝ (Requirement Analysis). Αυτή την ανάλυση θα χρησιμοποιήσουν οι κατασκευαστές του συστήματος για να καθορίσουν τις προδιαγραφές και τα χαρακτηριστικά του. Ο ΚΑΘΟΡΙΣΜΟΣ ΤΩΝ ΠΡΟΔΙΑΓΡΑΦΩΝ ενός συστήματος (System Specification) είναι, όπως έχουμε αναφέρει, η πρώτη δραστηριότητα (φάση-στάδιο) που λαμβάνει χώρα στο κύκλο ζωής ενός έργου λογισμικού.

Η δραστηριότητα αυτή, του κύκλου ανάπτυξης ενός έργου λογισμικού, είναι πολύ σημαντική και επηρεάζει απόλυτα την περαιτέρω κατασκευή και την τελική συμπεριφορά του λογισμικού. Ένας από τους πιο συνηθισμένους λόγους αποτυχίας ενός έργου λογισμικού είναι αυτός της λανθασμένης ανάλυσης των απαιτήσεων και κατά συνέπεια του κακού προσδιορισμού των υπηρεσιών που θα προσφέρει (προδιαγραφές του συστήματος).

Ο Boar (το 1984) για παράδειγμα παρατηρεί ότι, το 50-80% των λαθών στην κατασκευή ενός συστήματος λογισμικού, οφείλονται στον "φτωχό" προσδιορισμό των αναγκών του χρήστη (πελάτη). Ο James Martin (1982) σημειώνει ότι το 80% των σφαλμάτων που παρουσιάζονται μετά την κατασκευή ενός συστήματος λογισμικού έχουν αφετηρία από αυτήν την πρώτη φάση της ανάπτυξης του λογισμικού. Το μήνυμα είναι σαφές:

«αν γίνουν λάθη στο στάδιο του καθορισμού των απαιτήσεων και προδιαγραφών, τότε τα λάθη αυτά ακολουθούν το λογισμικό σε όλη την διάρκεια του κύκλου της ζωής του. Και στον σχεδιασμό και στην κατασκευή και στη συντήρηση του (κυρίως στην τελευταία)».

Και το κυριότερο: τα λάθη αυτά κοστίζουν ακριβά!

Αντιμετώπιση.

Ο προσδιορισμός των απαιτήσεων και των προδιαγραφών δεν είναι καθόλου εύκολη υπόθεση. Συνήθως συνέρχονται μικτές ομάδες εργασίας που αποτελούνται από χρήστες (οι οποίοι ξέρουν καλά τις διαδικασίες) και από κατασκευαστές λογισμικού (οι οποίοι φυσικά έχουν σπουδάσει πληροφορική ή έχουν καλή γνώση από συστήματα πληροφορικής). Αυτές οι ομάδες εργασίας αναλαμβάνουν να φέρουν εις πέρας την ανάλυση των απαιτήσεων και τον καθορισμό των προδιαγραφών του συστήματος. Πολύ γρήγορα βέβαια γίνεται αντιληπτό ότι αυτοί (δηλ. οι κατασκευαστές και οι χρήστες) "μιλούν" μεταξύ τους διαφορετικές γλώσσες (gap analysis).

Είναι πολύ δύσκολο για ένα επιστήμονα πληροφορικής να προσεγγίσει το "φυσικό" πρόβλημα με τον ίδιο τρόπο που το αντιλαμβάνονται οι χρήστες. Από την άλλη μεριά οι χρήστες δεν είναι σε θέση φυσικά να προετοιμάσουν τις προδιαγραφές για ένα μελλοντικό σχεδιασμό του συστήματος το οποίο θα οδηγήσει στην τελική κατασκευή του λογισμικού, χωρίς να έχουν τις απαιτούμενες γνώσεις. Μια άλλη πολύ μεγάλη δυσκολία που παρουσιάζεται συνήθως είναι αυτή της έλλειψης εμπιστοσύνης και της δυσπιστίας, που δείχνουν οι μεν στους δε. Εκφράσεις όπως, "δεν ξέρουν τι πραγματικά θέλουν", "είναι άσχετοι", "αυτοί", "εμείς", "αδύνατον να καταλάβουν" κτλ, είναι πολύ συνηθισμένες.

Οι ιδανικές συνθήκες για να γίνει ένας σωστός καθορισμός των απαιτήσεων και των προδιαγραφών, είναι ίσως δυνατόν να επιτευχθούν στις εξής δύο περιπτώσεις:

1. Ο (ή οι) κατασκευαστής(ές) του λογισμικού να εργαστεί(ούν) κάτω από πραγματικές συνθήκες και να διαπιστώσει(ουν) ο(οι) ίδιος(οι) τις ανάγκες που παρουσιάζονται, τις απαιτήσεις που πρέπει να ικανοποιηθούν και να καταγράψει(ουν) τις διαδικασίες οι οποίες λαμβάνουν χώρα. Αυτό όμως κοστίζει ακριβά, μια και χρειάζεται αρκετό χρονικό διάστημα και απαιτούνται πολλές φορές και ειδικές γνώσεις (δηλαδή ο κατασκευαστής θα πρέπει ανάλογα με το πρόβλημα να γίνεται λογιστής, διευθυντής, οικονομολόγος, τεχνικός κτλ) και φυσικά σπάνια (για να ακριβολογούμε μάλλον ποτέ) δεν εφαρμόζεται.
2. Κάποιος(οι) από τους χρήστες να έχει(ουν) σπουδάσει την επιστήμη της πληροφορικής ή να έχει(ουν) αποκτήσει, με κάποιο τρόπο τις απαιτούμενες γνώσεις. Ούτε αυτό όμως είναι πάντα εφικτό. Είναι και αυτό πολύ σπάνιο φαινόμενο, που μπορεί να παρουσιαστεί κυρίως στα μικρά έργα κατασκευής λογισμικού.

Έχουν αναπτυχθεί (είναι δραστηριότητα της Μηχανικής Λογισμικού αυτή) και προτείνονται διάφορες τεχνικές και στρατηγικές, οι οποίες έχουν σκοπό να βοηθήσουν σε αυτή την επικοινωνία των χρηστών και των αναλυτών του συστήματος. Για παράδειγμα, μια έξυπνη ιδέα, για να αποφευχθούν παρεξηγήσεις και να τεθούν οι βάσεις για ένα σωστό καθορισμό των απαιτήσεων και των προδιαγραφών του λογισμικού, είναι η κατασκευή και η χρήση ενός "πρότυπου συστήματος". Με αυτό τον τρόπο είναι δυνατόν να γίνει πολύ εύκολα η ΕΠΙΚΥΡΩΣΗ των προδιαγραφών (είναι πραγματικά σωστές αυτές οι προδιαγραφές και οι απαιτήσεις του συστήματος;) το οποίο είναι ένα από τα μεγάλα προβλήματα που πρέπει να λύσει η Μηχανική Λογισμικού.

Και γραπτή τεκμηρίωση.

Έχοντας άμεση σχέση με το πρόβλημα ο καθορισμός των προδιαγραφών μπορεί να εκφραστεί, είτε σαν ελεύθερο κείμενο γραμμένο σε απλή γλώσσα, είτε σαν τυπικές μαθηματικές εκφράσεις. Και στις δύο περιπτώσεις απαιτείται γραπτό κείμενο (document) το οποίο θα συνοδεύεται από επεξηγηματικά (και για τις δύο πλευρές) παραρτήματα και σχεδιαγράμματα.

Αυτό το έγγραφο (document) μπορεί να είναι από μερικές δεκάδες έως μερικές εκατοντάδες σελίδες (υπάρχουν και κείμενα που είναι μερικές χιλιάδες σελίδες!). Σε αυτό περιγράφονται αναλυτικά, αλλά και περιληπτικά (για τους χρήστες, για παράδειγμα που δεν είναι σε θέση να καταλάβουν τεχνικούς όρους) οι προδιαγραφές του συστήματος και οι απαιτήσεις που περιμένουμε να ικανοποιήσει αυτό. Επίσης, πρέπει να είναι η βάση από την οποία θα ξεκινήσει ο σχεδιασμός (και μερικές φορές τον υποδεικνύει) και η κατασκευή του λογισμικού, να περιέχει αναλυτικές πληροφορίες για το σύστημα, οι οποίες θα είναι κατανοητές από τον χρήστη και στοιχεία, τα οποία θα χρειάζονται για την μελλοντική συντήρηση και επέκταση του έργου. Τέλος θα πρέπει να είναι το βασικό έγγραφο για την δημιουργία συμβολαίου υλοποίησης μεταξύ της κατασκευάστριας εταιρίας και της επιχείρησης (οργανισμού) που θα αγοράσει και θα χρησιμοποιήσει το προϊόν.

Γενικά για να γίνει η ανάλυση των απαιτήσεων, χρειάζεται λεπτομερής καταγραφή των δεδομένων (inputs) και των αποτελεσμάτων (outputs), αυστηρή παρακολούθηση της ροής των πληροφοριών, εντοπισμός των σημείων όπου μετασχηματίζονται οι πληροφορίες και ποιες ενέργειες λαμβάνουν χώρα σε κάθε μετασχηματισμό.

Με την ανάλυση των απαιτήσεων είμαστε σε θέση να καθορίσουμε τι ακριβώς θα κάνει το λογισμικό (software) και τι υπηρεσίες θα προσφέρει, δίνοντας λεπτομέρειες που αφορούν την διακίνηση των πληροφοριών, τα αποτελέσματα που θα έχουμε, την περιγραφή των λειτουργιών και των ενεργειών, που θα λαμβάνουν χώρα και τον τρόπο επικοινωνίας χρήστη-συστήματος. Έτσι μπορούμε να προετοιμάζουμε και ίσως να προτείνουμε τον σχεδιασμό του συστήματος και να κάνουμε χρήσιμες παρατηρήσεις σχετικά με την κατασκευή αυτού.

Σημειώστε ότι σε αυτή την φάση δεν ασχολούμεθα καθόλου με το «πώς» θα γίνει η ανάπτυξη και η κατασκευή του συστήματος. Απλά καταγράφεται το «τι» ακριβώς θα κάνει αυτό (ΚΑΘΟΡΙΣΜΟΣ ΠΡΟΔΙΑΓΡΑΦΩΝ). Ο σχεδιασμός και η κατασκευή του λογισμικού γίνεται σε άλλες φάσεις αργότερα (δείτε τον κύκλο ζωής λογισμικού). Η ανάλυση των απαιτήσεων πρέπει να δίνει απάντηση σε μια σειρά από ερωτήματα:

- Ποιες είναι οι ανάγκες του χρήστη που ζητά το σύστημα;
- τι περιμένει ο χρήστης από το σύστημα;
- ποια είναι τα διαδικαστικά και λειτουργικά προβλήματα;
- έχουν γίνει αντιληπτές οι επιδράσεις του συστήματος στη επιχείρηση ή τον οργανισμό που θα το εγκαταστήσει;
- πώς θα γίνει η χρήση του συστήματος;
- πώς θα γίνει η μεταβίβαση από το χειρωνακτικό (ή το παλιό μηχανογραφικό) σύστημα στο νέο;
- υπάρχουν άτομα να βοηθήσουν στην εφαρμογή του συστήματος;
- ποια θα είναι τα τεχνικά χαρακτηριστικά του λογισμικού;
- σε ποια υποδομή (υλικό, λειτουργικά συστήματα, δίκτυα) θα στηριχθεί το υπό ανάπτυξη σύστημα;
- ποιο θα είναι το περιβάλλον ανάπτυξης (το προγραμματιστικό περιβάλλον, οι γλώσσες προγραμματισμού, τα εργαλεία που θα χρησιμοποιηθούν);
- σε ποιο ηλεκτρονικό περιβάλλον εργασίας θα εργάζεται ο χρήστης;
- με τι είδους πληροφορίες έχουμε να κάνουμε;
- τι περιορισμοί υπάρχουν;
- ποια θα πρέπει να είναι η απόδοση του συστήματος; τι κριτήρια πρέπει να μπουν ώστε να γίνει η αξιολόγηση της απόδοσης (η γρήγορη ανταπόκριση είναι αρκετή;)

- ποια θα είναι η επεκτασιμότητα του συστήματος

  κλπ κλπ ...

φυσικά ο κατάλογος των ερωτημάτων δεν εξαντλείται εδώ.

Το αποτέλεσμα τις ανάλυσης των προδιαγραφών θα είναι όπως προαναφέραμε ένα γραπτό κείμενο με ακριβείς πληροφορίες για το σύστημα το οποίο θα είναι σημείο αναφοράς για τους κατασκευαστές αλλά και για τους χρήστες. Θα επαναλάβουμε εδώ ότι αυτό το κείμενο θα περιέχει μια απλή καταγραφή των λειτουργιών, χωρίς ν' αναφέρει πώς θα γίνουν αυτές και θα ακολουθεί το λογισμικό καθ' όλη την διάρκεια της ζωής του.

Ο Heniger (1980) ισχυρίζεται ότι ένα κείμενο όπου γίνεται ο προσδιορισμός των χαρακτηριστικών ενός συστήματος θα πρέπει:

- να ασχολείται μόνο με την εξωτερική συμπεριφορά του συστήματος (τι εμφανίζεται στο χρήστη, και με ποιο τρόπο θα εργάζεται αυτός)
- να επισημαίνει τους περιορισμούς που θα υπάρχουν κατά την διάρκεια της ανάπτυξης του έργου,
- να δίνει την δυνατότητα αλλαγών,
- να αποτελεί σημείο αναφοράς για τους συντηρητές του έργου,
- να προβλέπει και να είναι φτιαγμένο έχοντας υπόψη τις επόμενες φάσεις-στάδια του έργου και
- να είναι σε θέση να δίνει αποδεκτές απαντήσεις σε απρόβλεπτες καταστάσεις.

Επίσης οι πληροφορίες που θα υπάρχουν στο προαναφερθέν κείμενο πρέπει να είναι ακριβείς και να εντοπίζονται εύκολα. Συνίσταται να υπάρχει πίνακας περιεχομένων και να εξηγείται η σχετική ορολογία (και από την πλευρά της πληροφορικής αλλά και από την μεριά της εφαρμογής). Αν είναι τέλος δυνατόν να είναι κατανοητό από τους χρήστες αλλά ταυτόχρονα πολύ συγκεκριμένο για τους κατασκευαστές και αναλυτικό για τους συντηρητές. Το τελευταίο είναι πολύ δύσκολο να επιτευχθεί γι' αυτό και συχνά υπάρχουν δύο έγγραφα. Ένα για τους χρήστες και ένα για τους κατασκευαστές και συντηρητές.

Ο Sommerville στο βιβλίο "Software Engineering" (1985) συνιστά ότι το κείμενο όπου θα καθορίζονται οι προδιαγραφές ενός συστήματος πρέπει να περιέχει:

1. Μια εισαγωγή, όπου θα τεκμηριώνεται η ανάγκη ανάπτυξης του συστήματος και ο ρόλος του μέσα στη επιχείρηση, εταιρεία ή οργανισμό. Επίσης θα πρέπει να γίνεται μια περιληπτική περιγραφή των εξυπηρετήσεων που θα προσφέρονται στους χρήστες.
2. Τις βασικές αρχές του συστήματος. Μια γενική άποψη του όλου συστήματος με μια σύντομη περιγραφή των υπηρεσιών που θα προσφέρει αυτό και τις επιδράσεις του στο περιβάλλον. Όπου είναι δυνατόν να γίνεται χρήση σχεδίων και γραφημάτων.
3. Τις απαιτήσεις του συστήματος σε hardware όπου περιγράφονται τα υπολογιστικά συστήματα, οι επικοινωνίες, τυχόν εξειδικευμένα συστήματα (π.χ barcode readers) και οι ιδανικές και ελάχιστες συνθέσεις.
4. Τις λειτουργικές απαιτήσεις. Πλήρης περιγραφή των υπηρεσιών οι οποίες θα παρέχονται στους χρήστες.
5. Ο σχεδιασμός της βάσης δεδομένων. Η δομή των πληροφοριών και οι σχέσεις που υπάρχουν μεταξύ τους.
6. Οι μη λειτουργικές απαιτήσεις. Αυτές είναι οι περιορισμοί κάτω από τους οποίους καλείται να λειτουργήσει το λογισμικό.

7. Διάφορες πληροφορίες για την συντήρηση και την επέκταση του συστήματος. Εδώ αναφέρονται οι κύριες προϋποθέσεις πάνω στις οποίες στηρίζεται το σύστημα και καταγράφονται οι αλλαγές που έχουν γίνει για διάφορους λόγους.
8. Ορολογία όπου εξηγούνται οι τεχνικοί και ειδικοί όροι.
9. Περιεχόμενα και ευρετήριο.

Κατασκευή πρότυπου (υποδείγματος ;)

Όπως έχουμε ήδη αναφέρει, ένα πρότυπο προϊόν είναι δυνατόν να χρησιμοποιηθεί για να γίνει πιο εύκολο το έργο το έργο της ομάδας που έχει αναλάβει τον καθορισμό των απαιτήσεων και των προδιαγραφών (prototyping method).

Η ανάπτυξη και η εφαρμογή της μεθόδου ολοκληρώνεται σε 4 στάδια (Geoff Simons, Introducing Software Engineering, 1987):

1. Προσδιορισμών των αρχικών απαιτήσεων
2. Σχεδιασμός και κατασκευή προτύπου
3. Παρουσίαση και εγκατάσταση προτύπου
4. Αξιολόγηση προτύπου

Όπως παρατηρείτε είναι δυνατόν να έχουμε ένα κύκλο ζωής λογισμικού μέσα σε ένα κύκλο ζωής ενός άλλου λογισμικού.

Ίσως θεωρηθεί παράδοξο το γεγονός ότι για να κάνουμε τον καθορισμό των προδιαγραφών και των απαιτήσεων ενός συστήματος πρέπει να επαναλάβουμε την διαδικασία, αλλά λάβετε υπόψη σας ότι για τον καθορισμό των απαιτήσεων του προτύπου συστήματος δεν χρειάζεται να κάνουμε εκτεταμένη έρευνα. Κάποιος ερευνητής παρατηρεί ότι από μερικές ημέρες έως μερικές εβδομάδες είναι αρκετό καλό χρονικό περιθώριο για τον καθορισμό των προδιαγραφών του προτύπου προϊόντος. Για το κύριο προϊόν το χρονικό διάστημα θα είναι αρκετά μεγαλύτερο.

Για την κατασκευή του προτύπου προϊόντος υπάρχουν αρκετά εργαλεία (Γλώσσες 4ης γενεάς, συστήματα διοίκησης βάσεων δεδομένων κλπ), τα οποία μπορούν να χρησιμοποιηθούν ώστε να γίνει αυτή πολύ σύντομα.

Μετά τη παρουσίαση και την εγκατάσταση το πρότυπο προϊόν τίθεται σε λειτουργία και παραδίδεται στους χρήστες. Αυτοί το χρησιμοποιούν και διατυπώνονται τα σχετικά σχόλια. Από τα σχόλια αυτά γίνονται αντιληπτές πολλές παρεξηγήσεις και εντοπίζονται τυχόν σφάλματα.

Σε ένα πρότυπο σύστημα μας ενδιαφέρει κυρίως η ικανοποίηση των λειτουργικών προδιαγραφών και όχι των μη λειτουργικών (ασφάλεια, ταχύτητα, αξιοπιστία κτλ). Και αυτό διότι επιθυμούμε την γρήγορη και φτηνή κατασκευή του προτύπου.

Τα πλεονεκτήματα που έχουμε με την χρήση αυτής της μεθόδου είναι:

- προσδιορίζονται εύκολα τυχόν παρανοήσεις, σφάλματα και παραλείψεις
- γίνεται αντιληπτός ο τρόπος παροχής και διακίνησης των πληροφοριών.
- τεκμηριώνεται η αναγκαιότητα του συστήματος, έστω και αν το πρότυπο δεν είναι πλήρες
- το πρότυπο είναι δυνατόν να χρησιμοποιηθεί σαν βάση στην περαιτέρω ανάπτυξη και κατασκευή του λογισμικού (ιδιαίτερα αν χρησιμοποιηθούν τα ίδια εργαλεία) και
- οι απαιτήσεις και οι προδιαγραφές που θα προκύψουν είναι επικυρωμένες (το πιο σημαντικό)

Το μεγαλύτερο μειονέκτημα που έχουμε όταν κατασκευάζουμε ένα πρότυπο προϊόν, είναι το υψηλό κόστος που χρειάζεται.

3.4 Ο σχεδιασμός του συστήματος

Όταν συμπληρωθεί η φάση του καθορισμού των απαιτήσεων και των προδιαγραφών του συστήματος, προχωρούμε στο στάδιο του σχεδιασμού αυτού. Για να επιτευχθεί ο σχεδιασμός ενός συστήματος θα ληφθούν υπόψη, τα συμπεράσματα που έχουν προέλθει από την προηγούμενη φάση της ανάλυσης, αλλά θα πρέπει να γίνει με τέτοιο ώστε να είναι δυνατή κάποια μελλοντική αλλαγή, καθώς είναι πιθανόν νέα δεδομένα να προκύψουν κατά την διάρκεια της δημιουργίας του σχεδιασμού.

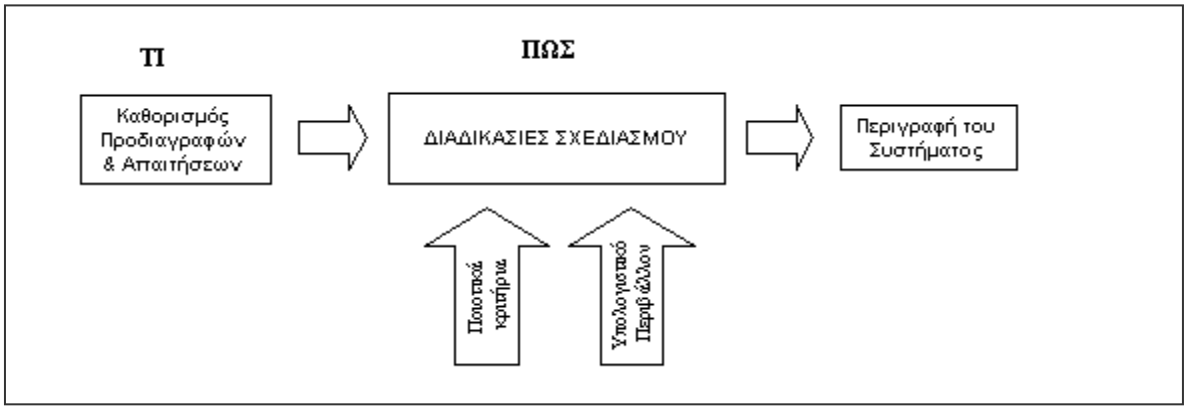
Σε αυτό το στάδιο θα πρέπει επίσης να ληφθούν σοβαρά υπόψη το περιβάλλον εργασίας και ο ανθρώπινος παράγοντας (ο τρόπος με τον οποίον ο χρήστης, σκέπτεται, θυμάται, λαμβάνει αποφάσεις κλπ.). Ένα πραγματικά "φιλικό" σύστημα, πρέπει να είναι σε θέση να προσφέρει βοήθεια στον χρήστη από την πρώτη στιγμή που θα τεθεί σε λειτουργία.

Υπάρχουν πολλοί τρόποι να ορίσει κάποιος την έννοια του σχεδιασμού:

" η διαδικασία, όπου εφαρμόζοντας διάφορες τεχνικές και μεθόδους, κατορθώνει κάποιος να ορίσει ένα σύστημα αρκετά αναλυτικά, ώστε να είναι δυνατή η πραγματοποίηση αυτού" (Taylor, 1959)

" το τελικό προϊόν ενός σχεδιασμού είναι μια πλήρης και λεπτομερής περιγραφή, μέσω της οποίας είναι δυνατόν να βεβαιωθεί η κατασκευή ενός συστήματος" (Teague & Pidgeon, 1985)

Πολλοί εξ' άλλου συγγραφείς και ερευνητές θεωρούν τον σχεδιασμό σαν *"μια διαδικασία η οποία μας οδηγεί από το ΤΙ στο ΠΩΣ"*. Πιο συγκεκριμένα ο σχεδιασμός είναι η διαδικασία της δημιουργίας μιας αναλυτικής περιγραφής, με σκοπό να ικανοποιήσει συγκεκριμένες προδιαγραφές κάτω από ορισμένους περιορισμούς που υπάρχουν ή έχουν τεθεί. Με αυτήν την έννοια ο σχεδιασμός ενός συστήματος λογισμικού είναι ακριβώς αντίστοιχος όπως σε κάθε άλλο τομέα (εκτός της πληροφορικής).



Κριτήρια αξιολόγησης σχεδιασμού είναι πραγματικά πολύ δύσκολο να τεθούν. Ο Pressman (1982) προσπαθεί να θέσει μερικά κριτήρια αξιολόγησης του σχεδιασμού ενός συστήματος:

- Πρέπει να είναι ιεραρχικά δομημένος
- Να είναι χωρισμένος σε τμήματα
- Κάθε τμήμα θα πρέπει να έχει ανεξάρτητα λειτουργικά γνωρίσματα
- Να ακολουθείται κάποια συστηματική μέθοδος επαναληπτικά, έχοντας πάντα υπόψη τις προδιαγραφές του συστήματος.

Φυσικά είναι αδύνατον για τους σχεδιαστές λογισμικού να φτάσουν σε ένα τελικό αποτέλεσμα με την πρώτη προσπάθεια. Η διαδικασία σχεδιασμού είναι μια μάλλον επαναλαμβανόμενη πράξη. Αρχικά γίνεται ένας πρόχειρος σχεδιασμός οποίος προοδευτικά, μετά από συνεχείς επαναλήψεις και προσθέτοντας νέες λεπτομερείς (αναλύοντας τον) μεταβάλλεται σε κάποιο συστηματικό σχεδιασμό.

Οι περισσότεροι συγγραφείς και ερευνητές συνιστούν να γίνεται προσέγγιση του σχεδιασμού με μια "δομημένη μεθοδολογία" η οποία θα είναι βασικά ένας οδηγός για την δημιουργία του λογισμικού. Ο ΔΟΜΗΜΕΝΟΣ ΣΧΕΔΙΑΣΜΟΣ υιοθετεί την ιδέα της διαίρεσης ενός μεγάλου προβλήματος σε διάφορα μικρότερα. Η ίδια διαδικασία μπορεί κατόπιν να εφαρμοστεί και στα μικρότερα προβλήματα που έχουν προκύψει και ούτω καθ' εξής. Τελικά το μεγάλο αρχικό πρόβλημα μετατρέπεται σε ένα σύνολο μικρών προβλημάτων τα οποία μπορούν να σχεδιαστούν και να κατασκευαστούν ευκολότερα. Σε έναν λοιπόν δομημένο σχεδιασμό η κατασκευή του συστήματος εκφράζεται με τα διάφορα συστατικά μέρη αυτού και με τις σχέσεις που υπάρχουν ανάμεσα σε αυτά. Μετά είμαστε σε θέση να προχωρήσουμε σε μια πιο λεπτομερή περιγραφή του συστήματος.

Δεν είναι δυνατόν να υποθέσουμε ότι πάντα έχουμε έναν μόνο (μοναδικό) σχεδιασμό ο οποίος θα ικανοποιεί κάποια(ες) συγκεκριμένη(ες) προδιαγραφή(ές). Συνήθως υπάρχουν διάφοροι τρόποι σχεδιασμού οι οποίοι μπορούν να μας οδηγήσουν στην ικανοποίηση αυτής(ών) της(ων) προδιαγραφής(ών). Αυτό είναι κυρίως μέλημα των ατόμων (σχεδιαστών) που προβαίνουν στον σχεδιασμό και εξαρτάται κατά πολύ από την εμπειρία και την τεχνογνωσία που διαθέτουν. Εξ' άλλου πολύ σπάνια κάποιος σχεδιαστής ξεκινά από το μηδέν. Σχεδόν πάντα υπάρχουν προηγούμενοι σχεδιασμοί στους οποίους μπορεί να αναφερθεί και να κάνει χρήση της μεθοδολογίας κατασκευής αυτών.

Έχουνε εφευρεθεί και προτείνονται από την Μηχανική Λογισμικού διάφοροι συστηματικοί τρόποι, μέθοδοι και τεχνικές με σκοπό να βοηθήσουν τους σχεδιαστές συστημάτων στο έργο τους. Ο στόχος όλων αυτών των τεχνικών είναι πάντα ο ίδιος. Αρχίζοντας από τις προδιαγραφές του συστήματος να δώσουν τρόπους, ώστε να επιτευχθεί μια συστηματική περιγραφή αυτού από την οποία θα είναι δυνατή η κατασκευή του συστήματος.

Ο σχεδιασμός όμως είναι μια δημιουργική διαδικασία. Αν και όπως αναφέραμε προηγουμένως υπάρχουν τεχνικές και προτείνονται διάφορες μεθοδολογίες η τελική κρίση και απόφαση ανήκει αποκλειστικά στον σχεδιαστή του συστήματος, του οποίου η εμπειρία

και η διαίσθηση είναι οι βασικοί παράγοντες που θα συμβάλουν σε σημαντικό βαθμό στον τελικό σχεδιασμό.

3.5 Η κατασκευή λογισμικού

Το λογισμικό ουσιαστικά είναι τα προγράμματα που κατασκευάζουμε στους ηλεκτρονικούς υπολογιστές για την επίλυση κάποιου προβλήματος. Τα προγράμματα είναι σύνολα εντολών, τα οποία πληροφορούν το υπολογιστή πώς να επικοινωνήσουν με τον χρήστη και πώς να γίνει η διαχείριση των μονάδων του για υλοποιηθεί κάποια επεξεργασία δεδομένων. Οι πρώτοι υπολογιστές απαιτούσαν προγραμματισμό σε «γλώσσα μηχανής» (machine language) η οποία ήταν η –μοναδική- γλώσσα προγραμματισμού που καταλάβαινε η Κεντρική Μονάδα Επεξεργασίας (επεξεργαστής / processor) κάθε υπολογιστικού συστήματος.

Οι γλώσσες μηχανής (καθώς και η αντίστοιχες συμβολικές αυτών -assembly) αναφέρονται σαν γλώσσες προγραμματισμού «χαμηλού επιπέδου». Το να γράφει όμως κανείς προγράμματα σε γλώσσα μηχανής (ή έστω και στην συμβολική γλώσσα αυτής) ήταν εξαιρετικά δύσκολο και χρονοβόρο. Για αυτό τον λόγο σταδιακά δημιουργήθηκαν οι διάφορες γλώσσες «υψηλού επιπέδου» όπως η C, η FORTRAN, η PASCAL και άλλες, που όμως για να εκτελέσει κανείς προγράμματα γραμμένα σε αυτές, είναι απαραίτητο να γίνει προηγουμένως κάποιου είδους επεξεργασία αυτών από τους αντίστοιχους compilers (μεταγλωττιστές) και interpreters (μεταφραστές;). Οι compilers / interpreters παίρνουν ένα πρόγραμμα γραμμένο σε κάποια γλώσσα υψηλού επιπέδου (πηγαίο) και μετά από κάποια επεξεργασία (compilation) παράγουν ένα κώδικα ο οποίος είναι αναγνωρίσιμος και μπορεί να εκτελεστεί από την Κεντρική Μονάδα Επεξεργασίας (CPU – processor) ενός συγκεκριμένου υπολογιστικού συστήματος (εκτελέσιμο πρόγραμμα -executable). Βέβαια αυτή η μετατροπή (μεταγλώττιση) του πηγαίου κώδικα και το εκτελέσιμο πρόγραμμα που παράγεται έχουν άμεση σχέση με όχι μόνο με την Κεντρική Μονάδα Επεξεργασίας, αλλά και με την αρχιτεκτονική του υπολογιστή και το Λειτουργικό Σύστημα (για παράδειγμα τα Windows, το Unix κλπ) που τον ελέγχει.

Υπάρχουν δύο κύριες μεθοδολογίες προγραμματισμού για να αναπτύξει κάποιος προγράμματα σε υπολογιστικά συστήματα:

- Ο δομημένος προγραμματισμός (structured programming) και
- Ο αντικειμενοστραφής προγραμματισμός (object oriented programming)

Στον δομημένο προγραμματισμό οι ενότητες του προγράμματος (εντολή ή κατάλληλα διατεταγμένες εντολές) εκτελούνται η μια μετά την άλλη. Με αντίστοιχες εντολές ελέγχου που τοποθετούνται στο πρόγραμμα από τον προγραμματιστή, αλλάζει αντίστοιχα η σειρά εκτέλεσης των ενοτήτων του προγράμματος.

Στον αντικειμενοστραφή προγραμματισμό η πληροφορία διαμορφώνεται σε αντικείμενα (objects). Η επεξεργασία της πληροφορίας γίνεται με αντίστοιχες ενότητες κώδικα (εντολές της γλώσσας προγραμματισμού) που βρίσκεται σε κάθε αντικείμενο. Τα αντικείμενα επικοινωνούν μεταξύ τους ανταλλάσσοντας δεδομένα και δεν υπάρχει συγκεκριμένη σειρά εκτέλεσης των ενοτήτων.

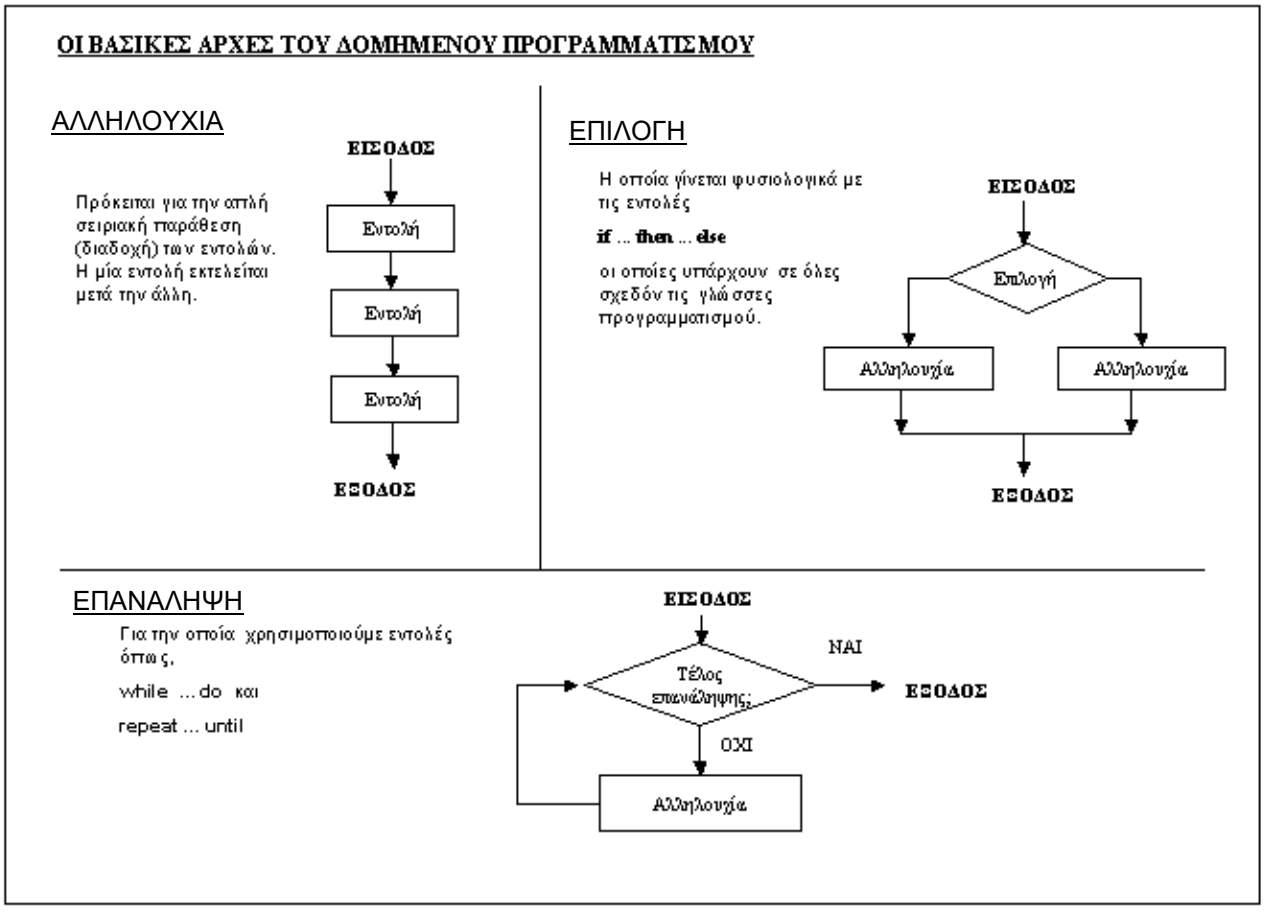
2. Ο δομημένος προγραμματισμός

Το 1965 ο E.W. Dijkstra σε συνέδριο της IFIP πρότεινε τον περιορισμό της χρήσης της εντολής GOTO στα προγράμματα ηλεκτρονικών υπολογιστών. Οι Bohm & Jacopini το 1966 διατύπωσαν και απέδειξαν το παρακάτω θεώρημα:

> *Ανεξάρτητα από την πολυπλοκότητα του ελέγχου της ροής των εντολών ενός προγράμματος, αυτό «μπορεί» να αντικατασταθεί με ένα άλλο πρόγραμμα το οποίο κάνει χρήση τριών βασικών δομών:*
>
> 1. *την **αλληλουχία** των εντολών, που πρόκειται ουσιαστικά για μια απλή σειριακή παράθεση (διαδοχή) των εντολών, δηλαδή η μια εντολή εκτελείται μετά την άλλη.*
> 2. *την **επιλογή**, που υλοποιείται φυσιολογικά με τις εντολές **if ... Then ... else,** οι οποίες υπάρχουν σε όλες σχεδόν τις γλώσσες προγραμματισμού.την **επανάληψη**, για την οποία χρησιμοποιούμε εντολές όπως **while ... do** και **repeat ... until.***

Με άλλα λόγια, η αυθαίρετη μεταφορά του ελέγχου της ροής ενός προγράμματος, από ένα σημείο σε κάποιο άλλο (το οποίο φυσικά γίνεται με την γνωστή εντολή GOTO) θεωρείται πλεονασμός και δεν χρειάζεται να εφαρμοστεί στην κατασκευή ενός προγράμματος!

Παρατηρήστε ότι το θεώρημα αναφέρει ότι "μπορεί" να αντικατασταθεί ένα πρόγραμμα με ένα άλλο, αλλά δεν λέει πώς!



Στο προηγούμενο σχήμα κάθε τετράγωνο (κόμβος) μπορεί να αποτελείται από μια ή και περισσότερες εντολές οι οποίες όμως διέπονται από τους ίδιους κανόνες.

Αν δούμε προσεκτικά πώς είναι σχεδιασμένες οι δομές θα δούμε παρατηρήσουμε ότι:

(α) έχουν μια μόνο είσοδο και μια έξοδο

(β) καμία από τις δομές δεν αποτελείται από περισσότερα από τρία τετράγωνα (κόμβους) και

(γ) η είσοδος είναι στη αρχή και η έξοδος στο τέλος.

Όταν εξετάζουμε ένα πρόγραμμα το οποίο είναι γραμμένο ακολουθώντας αυτές τις τρεις βασικές δομές, τότε έχουμε την δυνατότητα να το προσεγγίσουμε από διαφορετικά επίπεδα "ανάλυσης" και "αφαίρεσης" και σε κάθε επίπεδο το πρόγραμμα είναι κατανοητό χωρίς να χρειάζεται κάποιος να αναφερθεί σε άλλα επίπεδα.

Αν και στην πράξη (κυρίως σε μεγάλα projects, όπου ο πειραματισμός και ο έλεγχος είναι ακριβός) δεν έχουν αποδειχθεί πλήρως τα πλεονεκτήματα που έχουμε από τον περιορισμό της χρήσης της εντολής GOTO, μελέτες που έχουν γίνει σε μικρά projects δείχνουν ότι, ο έλεγχος, η διόρθωση και η επέκταση προγραμμάτων που έχουν κατασκευαστεί και αναπτυχθεί χρησιμοποιώντας τις μεθόδους του Δομημένου Προγραμματισμού είναι σαφώς ευκολότερα.

Ο Δομημένος Προγραμματισμός δεν είναι απλά μια προσπάθεια περιορισμού της εντολής GOTO. Το γεγονός ότι δεν χρειάζεται να χρησιμοποιήσουμε την προαναφερθείσα εντολή, απορρέει από την εφαρμογή των τριών μεθόδων του Δομημένου Προγραμματισμού.

Κύριος στόχος του Δομημένου Προγραμματισμού είναι η γρήγορη κατασκευή αξιόπιστων προγραμμάτων τα οποία θα είναι εύκολα κατανοητά. Αυτό θα έχει σαν αποτέλεσμα την αύξηση της παραγωγικότητας και την εύκολη μελλοντική συντήρηση και επέκταση ενός συστήματος πληροφορικής.

Τις περασμένες δεκαετίες ο Δομημένος Προγραμματισμός καθιερώθηκε σαν την κατ' εξοχήν μεθοδολογία ανάπτυξης και κατασκευής προγραμμάτων ηλεκτρονικών υπολογιστών.

Παράδειγμα ΔΟΜΗΜΕΝΟΥ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ:

```
Αρχή προγράμματος
        εντολή
        .....
        While (....) do
                εντολή
                ......
                εντολή
                If (...) then
                        εντολή
                        ...
                Else
                        εντολή
                        ...
                End if
                εντολή
                …
                Repeat
                        εντολή...
                        ...
                Until (...)
        End While
        εντολή
        …..
Τέλος προγράμματος.
```

3. Τεχνικές Προγραμματισμού

3.1 Τα προγράμματα

Η ανάπτυξη ενός προγράμματος σε κάποια γλώσσα προγραμματισμού ουσιαστικά περιλαμβάνει τις εξής βασικές ενότητες:

- εντολές για την έναρξη και τον τερματισμό του προγράμματος
- τις δηλώσεις (ορισμούς –declarations) των μεταβλητών, για την δέσμευση μνήμης όπου θα φιλοξενηθούν τα δεδομένα που πρόκειται να επεξεργαστούμε.
- εντολές, οι οποίες αναλαμβάνουν την επικοινωνία με τον χρήστη, την είσοδο των δεδομένων, την επεξεργασία των δεδομένων και τον έλεγχο της ροής του προγράμματος και
- εντολές, ο οποίες θα εμφανίσουν τα αποτελέσματα της επεξεργασίας.

Οι εντολές έναρξης και τερματισμού του προγράμματος είναι ουσιαστικά κάποιες λέξεις-κλειδιά που διαφέρουν από γλώσσα σε γλώσσα προγραμματισμού (main, program κλπ). Το ίδιο συμβαίνει και με τις εντολές επεξεργασίας, ελέγχου της ροής του προγράμματος και τις εντολές επικοινωνίας (εισόδου / εξόδου) με τον χρήστη. Οι εντολές ενός προγράμματος εκτελούνται με την σειρά η μια μετά την άλλη. Η σειρά εκτέλεσης είναι δυνατόν να αλλάξει με την χρήση των εντολών ελέγχου της ροής του προγράμματος (if-then-else) ή οπότε κρίνεται αναγκαίο να επαναληφθεί η εκτέλεση κάποιας ομάδας εντολών με την αντίστοιχη χρήση εντολών επανάληψης (for, while, repeat).

Τα δεδομένα (πληροφορίες - data) είναι απαραίτητα στοιχεία ενός προγράμματος, καθώς οι βασικές λειτουργίες ενός προγράμματος είναι η επεξεργασία αυτών των δεδομένων και η εξαγωγή αποτελεσμάτων (δηλαδή άλλα νέα δεδομένα). Απαιτείται λοιπόν η δέσμευση κάποιων χώρων μνήμης για να αποθηκευτούν αυτά τα δεδομένα, κατά την διάρκεια της εκτέλεσης του προγράμματος. Αυτοί οι χώροι μνήμης που τους χρησιμοποιούμε για την φύλαξη δεδομένων, όταν εκτελείται ένα πρόγραμμα, ονομάζονται ανάλογα με την χρήση τους σταθερές ή μεταβλητές. Σε κάθε γλώσσα προγραμματισμού μπορούμε με διάφορους τρόπους να δηλώσουμε (καθορίσουμε) κάποιους τύπους μεταβλητών / σταθερών, τις οποίες θα χρησιμοποιήσουμε για την αποθήκευση δεδομένων, όταν εκτελείται ένα πρόγραμμα. Οι μεταβλητές λοιπόν είναι κομμάτια μνήμης του υπολογιστή, όπου αποθηκεύονται τα δεδομένα που επεξεργαζόμαστε κατά την διάρκεια της εκτέλεσης του προγράμματος. Στις μεταβλητές αυτές δίνουμε ονόματα, ώστε να μπορούμε εύκολα να αναφερόμαστε σε αυτές όταν τις χρησιμοποιούμε στο πρόγραμμα.

Οι γλώσσες προγραμματισμού μας δίνουν την δυνατότητα, αφού καθορίσουμε και δώσουμε ονόματα σε μεταβλητές, στην συνέχεια να καταχωρούμε σε αυτές όποια τιμή επιθυμούμε, που έχει βέβαια σχέση με το πρόγραμμα. Η δήλωση των μεταβλητών γίνεται συνήθως στην αρχή του προγράμματος. Σε γενικές γραμμές αυτοί οι τύποι των μεταβλητών που μπορούμε να ορίσουμε σε κάποια γλώσσα προγραμματισμού είναι:

1. Ακέραιος: για την αποθήκευση ακεραίων αριθμών.
2. Πραγματικός: για την αποθήκευση πραγματικών αριθμών.
3. Χαρακτήρας: για την αποθήκευση χαρακτήρων και γραμματοσειρών.

Φυσικά ένας τύπος μεταβλητής σε κάποια γλώσσα προγραμματισμού καθορίζει τον τρόπο χειρισμού αυτής. Συγκεκριμένα προσδιορίζει το είδος των τιμών (δεδομένων) που μπορεί να αποθηκεύσει, και καθορίζει ένα σύνολο λειτουργιών που είναι δυνατόν να εφαρμοσθούν πάνω στις μεταβλητές αυτού του τύπου. Βέβαια ανάλογα με την γλώσσα προγραμματισμού υπάρχουν διάφοροι τύποι μεταβλητών:

	C	Pascal	Java	Fortran 77
Ακέραιος	int	integer	int	INTEGER
Πραγματικός	Float	Real	double	REAL
Χαρακτήρας	char	char	string	CHARACTER

Επιπρόσθετα, οι γλώσσες προγραμματισμού μας δίνουν την δυνατότητα να κατασκευάζουμε παράλληλα με το κυρίως πρόγραμμα μικρότερα υποπρογράμματα τα οποία συνήθως εκτελούν κάποιες συγκεκριμένες εργασίες. Συνήθως αυτά τα υποπρογράμματα δέχονται παραμέτρους και καλούνται από το κυρίως πρόγραμμα. Η ιδέα πίσω από αυτά τα υποπρογράμματα είναι να μειωθεί το μέγεθος του κώδικα και να διευκολυνθεί η δομημένη ανάπτυξη προγραμμάτων. Σε διάφορες γλώσσες προγραμματισμού αυτά τα υποπρογράμματα ονομάζονται «συναρτήσεις» (στην Pascal και στην C) «διαδικασίες» (procedures – PASCAL) «ρουτίνες» / «υπο-ρουτίνες» (FORTRAN) ή και «παράγραφοι» (paragraphs –COBOL).

3.2 Η άλγεβρα του BOOL

Στην άλγεβρα του Bool (παρουσιάστηκε τον 19ο αιώνα) οι μαθηματικές εκφράσεις καταλήγουν πάντα σε δύο τιμές: το Αληθές (True) και το Ψευδές (False) εισάγοντας την συγκεκριμένη μαθηματική λογική (Boolean logic). Οι δύο αυτές καταστάσεις «βόλεψαν» τα μέγιστα την Πληροφορική, η οποία ως γνωστόν στηρίζεται σε ψηφιακά σύστημα, όπου κυριαρχεί η έκφραση του περνά / δεν περνά (1/0) ρεύμα. Η δημιουργία και η αποτίμηση λογικών εκφράσεων της άλγεβρας του Bool, αποτελεί θεμελιώδη γνώση για τον προγραμματισμό (γλώσσες προγραμματισμού) αλλά για την χρήση (εφαρμογές, πχ. επεξεργασία φύλλων εργασίας, εκμετάλλευση βάσεων δεδομένων κλπ) συστημάτων πληροφορικής.

Στην κλασική αριθμητική οι όροι μιας πράξης είναι οι αριθμοί και οι τελεστές τα σύμβολα της πρόσθεσης, της αφαίρεσης, του πολλαπλασιασμού και της διαίρεσης (πχ 5+3, 7- 4, 5x2, 20/4). Στην άλγεβρα του Bool οι όροι είναι προτάσεις, οποίες είναι δυνατόν να αποδειχθούν Αληθείς ή Ψευδείς και οι τελεστές είναι τα λογικά **ΚΑΙ** (AND), **ΕΙΤΕ** (OR) και **ΟΧΙ** (NOT).

Για παράδειγμα η πρόταση:

Το 4 είναι μεγαλύτερο από το 3 (4>3 στα μαθηματικά)

είναι σαφώς είναι Αληθής.

Ας θεωρήσουμε μια άλλη έκφραση:

Είμαι πάνω από 40** ΚΑΙ **διδάσκω στο ΕΛΜΕΠΑ

Η έκφραση είναι αληθής αν και οι δύο προτάσεις (είμαι πάνω από 40, διδάσκω στο ΕΛΜΕΠΑ) είναι αληθείς. Αν δεν είμαι πάνω από 40 ή αν δεν διδάσκω στο ΕΛΜΕΠΑ, τότε η συγκεκριμένη έκφραση είναι Ψευδής.

Η έκφραση:

*Είμαι πάνω από 60 **ΕΙΤΕ** έχω μισθό κάτω από 180.000 δραχμές*

είναι Αληθής αν είναι Αληθής μόνο μια από τις πιο πάνω προτάσεις.

Οι τελεστές **AND**, **OR** και **NOT** στην άλγεβρα του Bool αντίστοιχα στα μαθηματικά συμβολίζονται με τα σύμβολα **^**, **v** και **¬**. Η αποτίμηση τους συνήθως γίνεται με την βοήθεια πινάκων αληθείας:

AND	T	F
T	T	F
F	F	F

OR	T	F
T	T	T
F	T	F

NOT	T	F
	F	T

Στην Λογική του Bool, όταν συνθέτουμε προτάσεις και εκφράσεις, συχνά χρησιμοποιούνται σχεσιακοί τελεστές (ή τελεστές σύγκρισης) η αποτίμηση των οποίων μας δείχνει κατά πόσο αυτές οι προτάσεις και εκφράσεις μας είναι αληθείς ή ψευδείς και κατά συνέπεια το συμπέρασμα μας είναι αληθές ή ψευδές. Αυτοί οι σχεσιακοί τελεστές (σύγκρισης) είναι:

= (ίσον),
< (μικρότερο),
> (μεγαλύτερο),
≥ (μεγαλύτερο ή ίσο),
≤ (μικρότερο ή ίσο) και
≠ (διάφορο).

Έτσι λοιπόν οι τελεστές της άλγεβρας Bool και οι σχεσιακοί τελεστές μπορούν να συνδυαστούν για να συνθέσουν λογικές εκφράσεις:

- *Μισθός < 100.000* ***ΕΙΤΕ*** *ηλικία < 60 ετών*
- *Μέτρηση > 56* ***ΚΑΙ*** *μέτρηση < 20*
- *Θερμοκρασία > 40^o* ***ΚΑΙ*** *θερμοκρασία < 75^o*
- *Υγρασία < 45% και υγρασία > 20%*
- *[(θερμοκρασία > 40^o)* ***ΕΙΤΕ*** *(υγρασία < 10%)]* ***ΚΑΙ*** *(ποσότητα νερού > 8000lt)*

Οι γλώσσες προγραμματισμοί προσφέρουν την δυνατότητα σύνθεσης τέτοιων λογικών εκφράσεων με την χρήση κατάλληλων λογικών και σχεσιακών τελεστών, οι οποίοι φυσικά διαφέρουν από γλώσσα σε γλώσσα προγραμματισμού.

	C	Pascal	Java	FORTRAN
AND	&&	AND	&	.AND.
OR	| |	OR	| |	.OR.
NOT	!	NOT	!	.NOT.
=	==	=	==	.EQ.
>	>	>	>	.GT.
<	<	<	<	.LT.
≥	>=	>=	>=	.GE.
≤	<=	<=	<=	.LE.
≠	!=	<>	!=	.NE.

Οι λογικές προτάσεις και εκφράσεις χρησιμοποιούνται στην κατασκευή προγραμμάτων υπολογιστών, κυρίως στον σχηματισμό των εντολών ελέγχου και επανάληψης. Τότε όταν θέλουμε να δρομολογήσουμε κάποια ροή εκτέλεσης των εντολών (δηλαδή, ποιες πρέπει να εκτελεστούν και ποιες όχι) ή για καθοριστεί αντίστοιχα μια επαναληπτική διαδικασία (πόσες φορές να εκτελεστεί ένα σύνολο εντολών) θα πρέπει να ικανοποιηθούν (ή όχι) κάποιες συνθήκες, οι οποίες διατυπώνονται με λογικές προτάσεις ή εκφράσεις:

	C	Pascal

Εντολές **Ελέγχου**	if (temp > 30 && temp <60)
	{
	mask = 5;
	printf (“mask is 8 now\n);
	}
	else
	mask = 7;
		if (temp > 30 AND temp < 60) then
		begin
		mask := 5;
		writeln (‘mask is 8 now’);
		else
		mask:=7;

Εντολές **Επανάληψης**	While (συνθήκη) {
	Εντολή (ες)
	}
		While (συνθήκη) do
		begin
		Εντολή (ες)
		end;

3.3 Οι αλγόριθμοι

Όταν κατασκευάζουμε κάποιο πρόγραμμα εκτός από την γνώση της γλώσσας προγραμματισμού θα πρέπει απαραίτητα να έχουμε κατανοήσει πλήρως το πρόβλημα που αντιμετωπίζουμε. Συνήθως τρία είναι τα βασικά βήματα για να δημιουργηθεί ένα πρόγραμμα το οποίο θα κάνει κάποια επεξεργασία δεδομένων και θα επιλύει ένα συγκεκριμένο πρόβλημα με την βοήθεια ενός ηλεκτρονικού υπολογιστή:

1. Ο καθορισμός του προβλήματος και ο προσδιορισμός της συμπεριφοράς που επιθυμούμε να έχει αυτό το πρόγραμμα για να λύσει το συγκεκριμένο πρόβλημα (τι δεδομένα θα δώσουμε, τι αποτελέσματα θέλουμε να έχουμε και με ποιόν τρόπο).
2. Η ανάπτυξη ενός αλγορίθμου (τρόπου σκέψης) για σχεδιάσουμε πως θα κάνουμε την επίλυση του προβλήματος, δηλαδή με ποιο τρόπο θα γίνει η επιθυμητή επεξεργασία των δεδομένων.
3. Ο μετασχηματισμός αυτού του αλγορίθμου σε κάποια γλώσσα προγραμματισμού, ώστε να είναι δυνατή η εκτέλεση του, μετά από την κατάλληλη μεταγλώττιση (compilation) από τον ηλεκτρονικό υπολογιστή (κωδικοποίηση του αλγορίθμου σε κάποια γλώσσα προγραμματισμού π.χ. C, PASCAL, COBOL, FORTRAN κλπ)

Εμείς εδώ θα ασχοληθούμε με θέματα από το δεύτερο κυρίως βήμα, δηλαδή με τον τρόπο που σκεπτόμαστε για την ανάπτυξη κάποιου αλγορίθμου, ο οποίος θα μας οδηγήσει στην επίλυση του προβλήματος. Πιο συγκεκριμένα θα προσπαθήσουμε να δούμε πως σχεδιάζεται ένας αλγόριθμος (τεχνικές), θα εξετάσουμε μερικούς από αυτούς δίνοντας παραδείγματα.

Αλγόριθμος (algorithm) είναι μια διαδικασία, δηλαδή μια διατεταγμένη σειρά από ενέργειες / εντολές, οι οποίες επεξεργάζονται ορισμένα δεδομένα, για να μας οδηγήσουν σε κάποιο συγκεκριμένο επιθυμητό αποτέλεσμα (λύση του προβλήματος). Δηλαδή ο αλγόριθμος είναι ένα σχεδιαστικό εργαλείο για την ανάπτυξη προγραμμάτων, το οποίο χρησιμοποιείται πριν την τελική σύνταξη κάποιου προγράμματος.

Στην περιγραφή ενός αλγορίθμου πρέπει να υπάρχει μια μόνο αρχή και ένα τέλος και οι ενέργειες / εντολές που περιέχονται σε αυτόν θα πρέπει να είναι διατυπωμένες με σαφήνεια (χωρίς παρερμηνείες). Αυτές οι ενέργειες / εντολές είναι δυνατόν αν το επιθυμούμε να τις οργανώσουμε σε ενότητες και πρέπει να εκφράζονται με ένα πεπερασμένο αριθμό βημάτων (δηλαδή όχι άπειρο πλήθος εντολών). Ο αλγόριθμος όταν εφαρμόζεται στον προγραμματισμό θα πρέπει να μπορεί να μετασχηματίζεται (κωδικοποιείται) εύκολα σε κάποια γλώσσα προγραμματισμού.

Στην συνέχεια θα δούμε πως μπορούμε να διατυπώσουμε έναν αλγόριθμο, χρησιμοποιώντας κάποιες τυποποιημένες τεχνικές. Για παράδειγμα, έστω ότι έχω ένα σύνολο ακεραίων και θέλω να βρω τον μεγαλύτερο από αυτούς (το πρόβλημα).

Πως σκέφτεται κανείς για να βρει τον μεγαλύτερο ακέραιο; Προφανώς, τους συγκρίνει μεταξύ τους και διαλέγει τον μεγαλύτερο. Αν προσπαθήσουμε να αναλύσουμε αυτήν ακριβώς την διαδικασία (τις διάφορες συγκρίσεις δηλαδή) που ακολουθούμε, θα είχαμε την εξής περιγραφή:

1. Παίρνω τον πρώτο αριθμό και τον θεωρώ σαν τον μεγαλύτερο (υποτίθεται ότι δεν έχω δει τους άλλους ακεραίους ακόμη).
2. Μετά συγκρίνω αυτόν που θεωρώ σαν μεγαλύτερο με τον επόμενο.
3. Αν ο επόμενος αριθμός είναι μεγαλύτερος από αυτόν που θεωρώ σαν μεγαλύτερο, τότε θεωρώ τον νέο αριθμό σαν μεγαλύτερο και τον συγκρίνω με τον αμέσως επόμενο.
4. Αν όμως, ο αμέσως επόμενος δεν είναι μεγαλύτερος, εξακολουθώ να θεωρώ σαν μεγαλύτερο αυτόν που έχω εντοπίσει τελευταίο και τον συγκρίνω με τον αμέσως επόμενο κλπ.

Πιο συγκεκριμένα:

Έστω ότι έχω τους αριθμούς 7, 5, 11, 12, 3 και 9 η πορεία της σκέψης μου είναι:

- βλέπω το 7 πρώτα. Το θεωρώ μεγαλύτερο.
- μετά βλέπω το 5. Το 7 είναι μεγαλύτερο από το 5 άρα θεωρώ ακόμη το 7 μεγαλύτερο.
- μετά βλέπω το 11. Το 11 > 7 άρα θεωρώ το 11 μεγαλύτερο.
- μετά βλέπω το 12. Το 12 > 11 άρα θεωρώ το 12 μεγαλύτερο.
- μετά βλέπω το 3. Εξακολουθεί το 12 να είναι μεγαλύτερο και
- τέλος το 9, το τελευταίο στοιχείο άρα το 12 είναι το μεγαλύτερο.

Η περιγραφή ενός αλγορίθμου αφορά μια οργανωμένη και καλά δομημένη παρουσίαση του διαδικασίας που ακολουθούμε για να έχουμε την επίλυση του προβλήματος μας. Αυτή η παρουσίαση θα πρέπει να γίνει με τέτοιο τρόπο ώστε να είναι δυνατή η κωδικοποίηση του αλγορίθμου σε κάποια γλώσσα προγραμματισμού. Μια πολύ απλή περιγραφή ενός αλγορίθμου είναι προσπαθήσουμε να εκφράσουμε τον τρόπο σκέψης μας με βήματα. Για παράδειγμα μια περιγραφή με βήματα του αλγορίθμου για να βρούμε τον μεγαλύτερο αριθμό θα μπορούσε να είναι η ακόλουθη:

Β0. Θεωρώ μια μεταβλητή Α στην οποία θα βάζω κάθε αριθμό που διαβάζω και μια μεταβλητή ΜΑΧ στην οποία θα αποθηκεύω τον εκάστοτε μεγαλύτερο.

Β1. Διαβάζω τον πρώτο αριθμό Α.

Β2. Τον θεωρώ μεγαλύτερο και δίνω στην μεταβλητή ΜΑΧ την τιμή του πρώτου αριθμού.

Β3. Διαβάζω τον επόμενο αριθμό Α.

Β4. Μήπως τελείωσα; αν ναι, το περιεχόμενο του ΜΑΧ είναι ο μεγαλύτερος. ΤΕΛΟΣ.

Β5. Ισχύει η συνθήκη Α > ΜΑΧ; Αν ναι, τότε το ΜΑΧ ← Α, αλλιώς πάω στον στο Β3 να διαβάσω τον επόμενο αριθμό (ΕΠΑΝΑΛΗΨΗ)

Αλλά ας δούμε και κάποιο άλλο πρόβλημα:

Να βρεθεί ο μέγιστος κοινός διαιρέτης (Μ.Κ.Δ.) δυο αριθμών Μ και Ν, ακολουθώντας το σκεπτικό του Ευκλείδη. Το σκεπτικό του Ευκλείδη βασίζεται στο γεγονός ότι όταν διαιρούμε ένα αριθμό Μ με έναν μικρότερο του Ν, τότε ο ΜΚΔ του υπολοίπου Υ και του μικρότερου Ν είναι ο ίδιος με τον ΜΚΔ των Μ και Ν. Επίσης είναι γνωστό ότι ο ΜΚΔ ενός αριθμού με το 0 είναι ο ίδιος ο αριθμός.

Έχουμε λοιπόν τον αλγόριθμο:

Β0. Διαβάζω τους αριθμούς Μ και Ν.

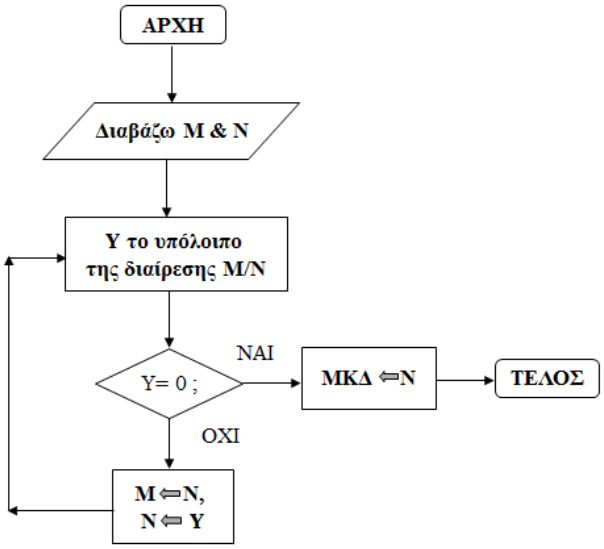
Β1. Διαιρώ τον Μ με τον Ν. Έστω Υ το υπόλοιπο.

Β2. Αν έχουμε ότι Υ = 0 τότε ο Ν είναι ΜΚΔ και ΤΕΛΟΣ.

Β3. Μ ← Ν, Ν ← Υ. Πήγαινε στο Β1.

Μέχρι τώρα για να περιγράψουμε ένα αλγόριθμο χρησιμοποιούμε απλές προτάσεις που περιγράφουν ενέργειες και οργανωμένες σε αριθμημένα βήματα για να διαφυλαχθεί η ροή των διαδικασιών. Ένας άλλος τρόπος έκφρασης ενός αλγορίθμου είναι αυτός με λογικά διαγράμματα ροής, όπου σχήματα περιγράφουν ενέργειες και η ροή των διαδικασιών υλοποιείται ακολουθώντας βέλη.

Για παράδειγμα ο αλγόριθμος του Ευκλείδη μπορεί να εκφραστεί με κάποιο λογικό διάγραμμα ροής ως εξής:



Ο αλγόριθμος του Ευκλείδη (Λογικό Διάγραμμα Ροής)

Η ίδια η κατασκευή ενός προγράμματος για την αντιμετώπιση ενός προβλήματος με την βοήθεια του υπολογιστή μπορεί να περιγραφεί "αλγοριθμικά" με βήματα:

Β1. Προσδιορισμός του προβλήματος.

Β2. Σχεδίαση και δημιουργία αλγορίθμου.

Β3. Κωδικοποίηση του αλγορίθμου,

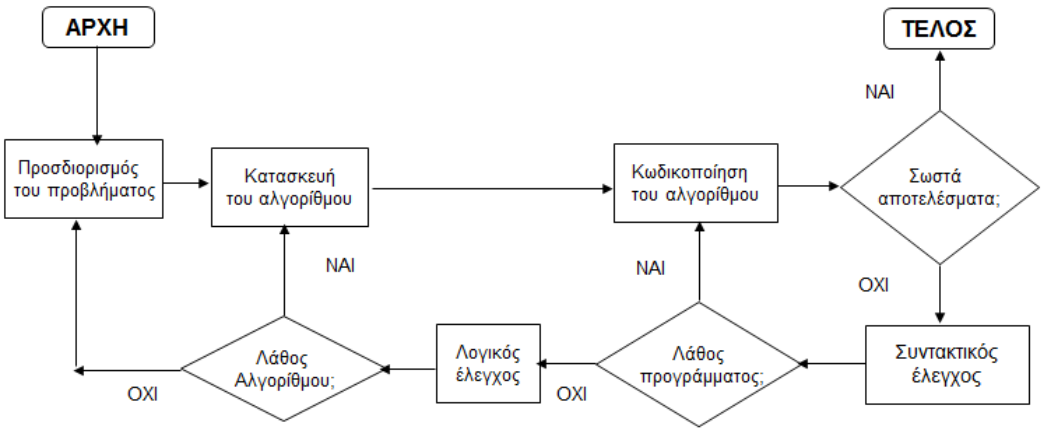
Β3.α Το πρόγραμμα δίνει σωστά αποτελέσματα; αν ναι ΤΕΛΟΣ

Β3.β Συντακτικός έλεγχος: υπάρχει λάθος στο πρόγραμμα; αν ναι πήγαινε στο Β3.

Β3.γ Λογικός έλεγχος: υπάρχει λάθος στον αλγόριθμο; αν ναι πήγαινε στο Β2.

Β3.δ Αν εξακολουθούν τα λάθη πήγαινε στο Β1.

ή με λογικό διάγραμμα:



Ο αλγόριθμος για την κατασκευή ενός προγράμματος

Όπως είδαμε αν και η διατύπωση ενός αλγορίθμου μπορεί να γίνει στην φυσική γλώσσα, υπάρχουν διάφοροι δομημένοι τρόποι για να εκφράσουμε έναν αλγόριθμο. Είτε με σχήματα (flowcharts), είτε με λόγια (λογικές εκφράσεις), είτε ακόμη και με μια γλώσσα προγραμματισμού. Στην συνέχεια για την περιγραφή των αλγορίθμων θα χρησιμοποιήσουμε τους δυο τυποποιημένους τρόπους έκφρασης:

- αυτόν με τα διαγράμματα, εισάγοντας ορισμένα αντιπροσωπευτικά για κάποιες χαρακτηριστικές διαδικασίες σχήματα και
- αυτόν με προτάσεις εισάγοντας κάποιες λέξεις κλειδιά όπως ακριβώς γίνεται στις γλώσσες προγραμματισμού (ψευδοκώδικας).

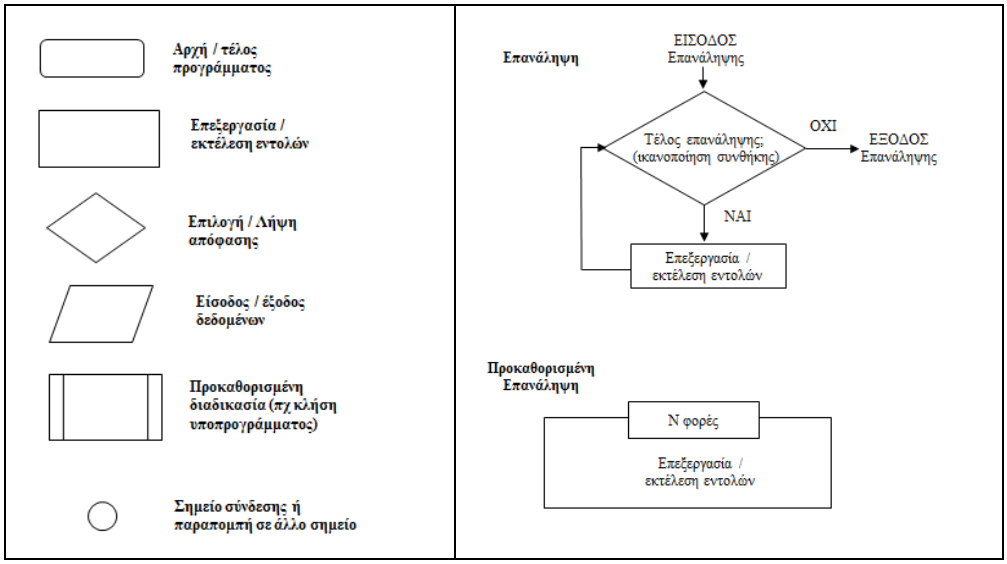
3.3.1 Τα λογικά διαγράμματα ροής

Τα λογικά διαγράμματα ροής (flowcharts) είναι ουσιαστικά μια γραφική αναπαράσταση των βημάτων που πρέπει να ακολουθήσουμε για να φτάσουμε στην λύση ενός προβλήματος. Ένα διάγραμμα ροής αποτελείται από διάφορα συγκεκριμένα σχήματα που ενώνονται μεταξύ τους με γραμμές ροής οι οποίες έχουν ένδειξη φοράς (προσανατολισμένες με βέλη / τόξα), για τον καθορισμό της ροής του διαγράμματος.

Μέσα σε αυτά τα σχήματα περιγράφονται περιληπτικά σε φυσική γλώσσα οι διάφορες ενέργειες που υλοποιούνται σε κάθε σχήμα. Συνήθως ανάλογα με το περίγραμμα ενός σχήματος, υπονοείται και το είδος της δραστηριότητας που λαμβάνει χώρα σε αυτό. Υπάρχουν ειδικά σχήματα για την έναρξη και το τέλος του αλγορίθμου. Μερικές παρατηρήσεις για τον σχεδιασμό λογικών διαγραμμάτων:

1. Για κάθε σχήμα (εκτός αυτών της έναρξης και του τέλους) υπάρχει μια τουλάχιστον είσοδος και μια τουλάχιστον έξοδος (προσανατολισμένες γραμμές ροής).
2. Κάθε λογικό διάγραμμα ροής ξεκινάει στην αρχή της σελίδα και η ροή του έχει διεύθυνση προς το τέλος της σελίδας, χωρίς όμως αυτό να είναι δεσμευτικό.
3. Υπάρχουν σύμβολα σύνδεσης όταν απαιτείται η σύνδεση μακρινών σημείων ή όταν συνεχίζεται το διάγραμμα σε άλλη σελίδα απεικόνισης.
4. Αν η περιγραφή του αλγορίθμου είναι μεγάλη ή περίπλοκη, η ανάπτυξη του διαγράμματος μπορεί να περιλαμβάνει και άλλες σελίδες. Σε αυτή την περίπτωση τα λογικά διαγράμματα σε διαφορετικές σελίδες συνδέονται μεταξύ τους με τα αντίστοιχα σύμβολα σύνδεσης.
5. Τέλος, θα πρέπει να προσέχουμε ώστε να μην διασταυρώνονται οι γραμμές ροής του σχεδιαγράμματος.

Ένα βασικό μειονέκτημα των λογικών διαγραμμάτων ροής είναι η έκταση που μπορεί αυτό να πάρει αν ο αλγόριθμος που θέλουμε να περιγράψουμε είναι μεγάλος και πολύπλοκος. Στον πίνακα που ακολουθεί παρουσιάζονται τα πιο συνηθισμένα σχήματα που χρησιμοποιούνται για την κατασκευή λογικών διαγραμμάτων ροής.



Βασικές δομές λογικών διαγραμμάτων

3.3.2 Ο ψευδοκώδικας

Ο ψευδοκώδικας είναι ουσιαστικά μια διατύπωση ενός αλγορίθμου, η οποία πολύ εύκολα μπορεί να μετασχηματιστεί σε πρόγραμμα κάποιας γλώσσας προγραμματισμού. Σε μια τέτοια διατύπωση συνήθως υπάρχουν κάποιοι (όχι αυστηροί) κανόνες και ορισμένες λέξεις-κλειδιά αντίστοιχες με αυτές που συναντάμε στις γλώσσες προγραμματισμού. Με αυτό τον τρόπο ο προγραμματιστής επικεντρώνεται στον τρόπο επίλυσης του προβλήματος (δηλαδή τον αλγόριθμο) χωρίς να ασχολείται με προγραμματιστικές λεπτομέρειες (κυρίως την σύνταξη) και τις ιδιαιτερότητες της γλώσσας προγραμματισμού που πρόκειται να χρησιμοποιηθεί. Στην πραγματικότητα κάποιος μπορεί να διατυπώσει ένα αλγόριθμο με την βοήθεια ψευδοκώδικα, χωρίς να γνωρίζει καμία γλώσσα προγραμματισμού. Με τον ψευδοκώδικα μπορούμε να διατυπώσουμε μεγάλους και πολύπλοκους αλγορίθμους ευκολότερα από ότι με τα λογικά διαγράμματα ροής.

Μια τυποποίηση της διατύπωσης του ψευδοκώδικα θα πρέπει να περιλαμβάνει κάποιες συγκεκριμένες ομάδες ψευδοεντολών και ψευδοδηλώσεων όπως ακριβώς στις γλώσσες προγραμματισμού:

Μεταβλητές

N: αριθμός στοιχείων πίνακα
Μ: Μέτρηση θερμοκρασίας
Ε: Εμβαδόν τριγώνου
Μισθός_υ: ποσό μισθοδοσίας

Εντολές εισόδου / εξόδου

Διάβασε N
Τύπωσε Ε
Τύπωσε «κάποιο μήνυμα»

Εντολές εκχώρησης

N ← 5
Μ ← 30

Εντολές ελέγχου

αν *(συνθήκη)* ***τότε***
Ενέργειες / εντολές (αν η συνθήκη ικανοποιείται)
τέλος (αν)

αν *(συνθήκη)* ***τότε***
Ενέργειες / εντολές (αν η συνθήκη ικανοποιείται)
διαφορετικά
Ενέργειες / εντολές (αν η συνθήκη ΔΕΝ ικανοποιείται)
τέλος (αν)

Εντολές επανάληψης (βρόγχου)

επανάληψη *N* ***φορές / συνεχώς***
Ενέργειες / εντολές επανάληψης
τέλος (επανάληψης)

όσο *(συνθήκη)*
Ενέργειες / εντολές επανάληψης όσο ικανοποιείται η συνθήκη
τέλος (όσο)

επανάληψη
Ενέργειες / εντολές επανάληψης όσο ικανοποιείται η συνθήκη
όσο *(συνθήκη)*

Υποπρόγραμμα / κλήση υποπρογραμμάτων

υποπρόγραμμα *ονομα_υποπρογράμματος (παράμετροι)*
Ενέργειες / εντολές υποπρογράμματος
Επιστροφή (παράμετροι)
τέλος (υποπρόγραμμα)

κλήση υποποπρογράμματος *ονομα_υποπρογράμματος (παράμετροι)*

Παρατηρήσεις στην διατύπωση ψευδοκώδικα*:*

- *Για τον σχηματισμό των συνθηκών χρησιμοποιούμε λογικές εκφράσεις και προτάσεις με την βοήθεια των λογικών τελεστών και των τελεστών συσχέτισης (από την λογική Bool).*
- *Για τις πράξεις χρησιμοποιούμε τα σύμβολα των πράξεων και παρενθέσεις.*
- *Οι εντολές / ενέργειες είναι δυνατόν να χωρίζονται μεταξύ τους με κάποιο προσυμφωνημένο σύμβολο (κόμμα ή ερωτηματικό).*
- *Αν το επιθυμείτε μπορείτε να χρησιμοποιήσετε τα βήματα (Β1, Β2, κλπ) για να χωρίσετε το αλγόριθμο σε ενότητες. Σε αυτή την περίπτωση μπορείτε να χρησιμοποιήσετε και την ψευδοεντολή ΠΗΓΑΙΝΕ στο Βx, αν και ο δομημένος προγραμματισμός δεν το συνιστά!*

3.3.3 Παραδείγματα αλγορίθμων με χρήση πινάκων (ψευδοκώδικας)

Ο πίνακας (array) είναι μια αρκετά διαδεδομένη δομή αναπαράστασης στοιχείων (data, δεδομένα) στην μνήμη του υπολογιστή, όταν δημιουργούμε κάποιο πρόγραμμα. Η δομή του πίνακα καθορίζεται σε κάθε σχεδόν γλώσσα προγραμματισμού.

Η αναφορά σε συγκεκριμένο στοιχείο του πίνακα, γίνεται χρησιμοποιώντας το όνομα του πίνακα ακολουθούμενο από ένα δείκτη (ή τους αντίστοιχους δείκτες αν ο πίνακας μας είναι πολυδιάστατος) που δηλώνει την θέση του στοιχείου στον πίνακα, π.χ.

Αν έχω τον μονοδιάστατο πίνακα Α = (1, 5, 7, 10) τότε A_1=1 (πρώτο στοιχείο του πίνακα), A_3=7 (τρίτο στοιχείο του πίνακα) κλπ

Η κύρια χρησιμότητα της δομής του πίνακα στις γλώσσες προγραμματισμού, είναι ότι με ένα όνομα μεταβλητής μπορούμε να αναφερθούμε σε πολλές μεταβλητές στο πρόγραμμα μας. Η ιδέα αυτής της μαζικής αποθήκευσης στοιχείων προέρχεται φυσικά από τα μαθηματικά.

Ας δούμε το πρόβλημα του <u>αθροίσματος δυο μονοδιάστατων πινάκων</u> με n στοιχεία ο κάθε ένας.

- αν έχω τους πίνακες Α και Β όπου:
  $A = A_1, A_2,..., A_n$ και
  $B = B_1, B_2,..., B_n$

τότε η πρόσθεση αυτών των πινάκων είναι ένας νέος πίνακας C:
$C= A_1+B_1, A_2+B_2,...,A_n+B_n$

Ο αλγόριθμος για αυτήν την πρόσθεση των πινάκων είναι αρκετά εύκολος:

<u>Αλγόριθμος ΠΡΟΣΘΕΣΗ_ΠΙΝΑΚΩΝ.</u>

Β0. ΑΡΧΗ
Β1. Α, Β, C: πίνακες ακεραίων
N: ο αριθμός στοιχείων κάθε πίνακα
i: ακέραιος (μετρητής)
Β2. ***διάβασε*** πίνακες Α και Β, i ← 1
Β2. ***όσο*** (i <= N)
$C_i \leftarrow A_i + B_i$
i ← i + 1
τέλος (όσο)
Β3. ***τύπωσε*** πίνακα C
Β4. **ΤΕΛΟΣ**

Αναζήτηση στοιχείου σ' ένα πίνακα

Ένα άλλο πολύ σημαντικό πρόβλημα είναι πως μπορεί να διαπιστωθεί αν υπάρχει κάποιο συγκεκριμένο στοιχείο ανάμεσα στα περιεχόμενα ενός πίνακα (εύρεση στοιχείου, searching). Για παράδειγμα:

> *Όταν μας δίνεται ένας πίνακας 10 ακέραιων, υπάρχει ο αριθμός Μ ανάμεσα στους αριθμούς του πίνακα;*

Δεν έχουμε παρά να πάρουμε όλα τα στοιχεία του πίνακα και να τα συγκρίνουμε ένα-ένα με τον αριθμό Μ. Αν βρεθεί ότι κάπου υπάρχει ισότητα τότε τελειώσαμε. Εύκολα μπορώ να περιγράψουμε τον αλγόριθμο σε ψευδοκώδικα με βήματα:

Αλγόριθμος ΕΥΡΕΣΗ-1.

Β0. ΑΡΧΗ
Β1. Α: πίνακας ακεραίων, Ν: ο αριθμός στοιχείων του πίνακα, i: ακέραιος (μετρητής)
Β3. ***διάβασε*** πίνακα Α, Μ, i ← 1
Β4. ***όσο*** (i <= N)
αν (A_i = M) **τότε**
τύπωσε «ο αριθμός υπάρχει στον πίνακα»
τέλος (αν)
i ← i + 1
τέλος (όσο)
Β5. ΤΕΛΟΣ

Παρατηρήστε ότι ο αλγόριθμος είναι ατελής. Ναι μεν βρίσκει αν υπάρχει ο ζητούμενος αριθμός, αλλά δεν δίνει κάποιο μήνυμα αν δεν υπάρχει αυτός ανάμεσα στους αριθμούς του πίνακα. Επιπλέον, όσες φορές τον συναντήσει μέσα στον πίνακα, τόσα μηνύματα τυπώνει. Θα ήταν καλύτερα αν απλά μου έβρισκε ότι ο συγκεκριμένος αριθμός υπάρχει ή όχι ανάμεσα στους αριθμούς του πίνακα. Για να το πετύχω αυτό χρησιμοποιώ μια παραπάνω μεταβλητή met, η οποία θα έχει αρχικά τιμή 0 και θα αυξάνεται (αθροιστής) κάθε φορά που ο αλγόριθμος βρίσκει κάποια ισότητα:

Αλγόριθμος ΕΥΡΕΣΗ-2.

Β0. ΑΡΧΗ
Β1. Α: πίνακας ακεραίων, Ν: ο αριθμός στοιχείων του πίνακα, i, met: ακέραιοι (μετρητές)
Β3. ***διάβασε*** πίνακα Α, αριθμό Μ
i <- 1, met=0
Β4. ***όσο*** (i <= N)
αν (A_i = M) ***τότε***
met ← met+1
τέλος (αν)
i ← i + 1
τέλος (όσο)
Β5. ***αν*** (met > 0) ***τότε***
τύπωσε «ο αριθμός υπάρχει στον πίνακα met φορές»
διαφορετικά
ΤΥΠΩΣΕ «ο αριθμός ΔΕΝ υπάρχει στον πίνακα»
Β6. ΤΕΛΟΣ

Επιπρόσθετα, ο αθροιστής met μας δείχνει πόσες φορές υπάρχει ο αριθμός μας στον πίνακα.

Ταξινόμηση στοιχείων πίνακα

Ένα άλλο αρκετά σημαντικό πρόβλημα είναι η ταξινόμηση των στοιχείων ενός πίνακα κατά φθίνουσα ή αύξουσα διάταξη. Για παράδειγμα επιθυμούμε τα στοιχεία ενός πίνακα Α ακεραίων αριθμών να τοποθετηθούν κατ' αύξουσα σειρά (*παρατήρηση: για λόγους απλότητας στο εξής όταν μιλάμε για ταξινόμηση θα εννοούμε ταξινόμηση ακέραιων αριθμών*

κατ' αύξουσα σειρά. Έτσι θα είναι πολύ πιο εύκολο να εξετάσουμε τους διάφορους αλγόριθμους ταξινόμησης.)

Το πρόβλημα λοιπόν για τους παρακάτω αλγόριθμους είναι, να τοποθετήσουμε τα στοιχεία ενός μονοδιάστατου πίνακα ακέραιων κατά αύξουσα σειρά.

(α) Απλή ταξινόμηση

Θα προσπαθήσουμε από το πίνακα Α που μας δίνεται να δημιουργήσουμε ένα νέο πίνακα Β ο οποίος θα αποτελείται από τα στοιχεία του Α σε αύξουσα σειρά όμως. Για αυτό το λόγο θα χρησιμοποιήσουμε στην περιγραφή του αλγορίθμου μας, εκτός από τον πίνακα Α και ένα άλλον πίνακα Β ο οποίος θα έχει τον ίδιο αριθμό στοιχείων. Όμως πως θα δημιουργήσουμε τον πίνακα Β με τα ταξινομημένα στοιχεία; Αρχικά θα βρούμε το μικρότερο στοιχείο του Α και θα το τοποθετήσουμε στην πρώτη θέση του Β. Στην συνέχεια πρέπει να αγνοήσουμε αυτό το μικρότερο στοιχείο στον πίνακα Α και να βρούμε το μικρότερο από τα υπόλοιπα στοιχεία για να το βάλουμε στην δεύτερη θέση του πίνακα Β. Πώς όμως θα γίνει αυτό; Μια πολύ απλή λύση θα ήταν να βάζαμε στη θέση του μικρότερου στοιχείου στον πίνακα Α, ένα άλλο στοιχείο με πολύ μεγάλη τιμή (πχ. 99999). Έτσι όταν ψάχνουμε για το δεύτερο μικρότερο στοιχείο αυτό δεν πρόκειται να είναι το πρώτο. Ας το δούμε με ένα παράδειγμα:

Έστω ο πίνακας Α=(8, 256, 37, 1, 45). Ο πίνακας Β είναι άδειος αρχικά Β=(, , , ,)

Βρίσκουμε τον μικρότερο στοιχείο του Α. Είναι το 1. Τοποθετούμε το 1 στην πρώτη θέση του Β. Στην θέση του 1 στον Α βάζουμε τον αριθμό 999. Έτσι οι δύο μας πίνακες γίνονται:

Α= (8, 256, 37, 999, 45) και Β=(1, , , ,)

Δουλεύοντας με τον ίδιο τρόπο και βρίσκουμε το δεύτερο μικρότερο στοιχείο του Α. Είναι το 8. Οι πίνακες τώρα είναι:

Α= (999, 256, 37, 999, 45) και Β=(1, 8, , ,)

Συνεχίζοντας με τον ίδιο τρόπο θα καταλήξουμε να έχουμε τους πίνακες:

Α= (999, 999, 999, 999, 999) και Β=(1, 8, 37, 45, 256)

Δηλαδή ο πίνακας Β περιέχει ταξινομημένα τα στοιχεία του Α.

Ας διατυπώσουμε τώρα τον αλγόριθμο μας:

Αλγόριθμος ΤΑΞΙΝΟΜΗΣΗ-ΑΠΛΗ

Β0. ΑΡΧΗ

Β1. Α: πίνακας ακεραίων, Ν: ακέραιος (αριθμός στοιχείων Α)
j, k, i: ακέραιοι (j, i μετρητές)

Β2. ***διάβασε*** Α, i ← 0

Β3. ***επανάληψη*** Ν ***φορές***
i ← i +1
min ← A_1 (θεωρούμε το 1ο στοιχείο σαν μικρότερο)
j ← 2
όσο (j <= N)
αν (min > A_j) ***τότε***
min ← A_j
k ← j
τέλος (αν)
j = j+1
τέλος (όσο)
B_i ← A_k
A_k ← πολύ μεγάλη τιμή
τέλος (επανάληψης)

Β4. ***τύπωσε*** το Β

Β5. ΤΕΛΟΣ

(β) ταξινόμηση με ανταλλαγή:

Μπορούμε όμως να σκεφτούμε διαφορετικά και να διατυπώσουμε κάποιο άλλο αλγόριθμο χωρίς την χρήση του βοηθητικού πίνακα Β. Κάθε φορά που βρίσκουμε ένα μικρότερο αριθμό να του αλλάζουμε την θέση του στον πίνακα και τον βάζουμε απευθείας στην σωστή σειρά, ανταλλάσσοντας θέση με το στοιχείο που βρίσκεται στην σωστή σειρά. Αλλά ας δούμε καλύτερα με ένα παράδειγμα πως σκεφτόμαστε:

Έστω ότι έχω τους αριθμούς

8, 256, 37, 1, 45

από αυτούς βρίσκω τον μικρότερο. Είναι το 1. Τοποθετώ τώρα το 1 στη θέση του 8 που είναι πρώτο στη σειρά και στην θέση του 1 βάζω το 8. Έτσι τώρα ο πίνακας μας διαμορφώνεται ως εξής:

1, 256, 37, 8, 45

Στην συνέχεια αγνοούμε το πρώτο στοιχείο του πίνακα και επικεντρωνόμαστε στα υπόλοιπα στοιχεία, δηλαδή στους αριθμούς 256, 37, 8, 45.

Βρίσκουμε τον μικρότερο από αυτούς (είναι το 8) και το τοποθετούμε στη θέση του πρώτου στη νέα σειρά (είναι το 256). Ο πίνακας μας γίνεται τώρα:

1, 8, 37, 256, 45

Μετά παίρνουμε την νέα σειρά 37, 256, 45 και συνεχίζουμε την ίδια διαδικασία.

Δουλεύοντας με βάση τα παραπάνω μπορούμε να διατυπώσουμε τώρα έναν αλγόριθμο για την ταξινόμηση των στοιχείων ενός πίνακα με ανταλλαγή. Αρχικά αυτό που χρειαζόμαστε είναι έναν αλγόριθμο, ο οποίος θα μας βρίσκει το μικρότερο στοιχείο ενός πίνακα. Θα περιγράψουμε αυτόν τον αλγόριθμο σαν υποπρόγραμμα, το οποίο θα το καλούμε από κυρίως αλγόριθμο της ανταλλαγής. Σε αυτό το υποπρόγραμμα ορίζουμε σαν k τον δείκτη του μικρότερου στοιχείου που θα βρεθεί μέσα στον πίνακα μεταξύ των m και n δεικτών του πίνακα (δηλαδή τα m και n ορίζουν μεταξύ ποιών δεικτών θα πρέπει να βρεθεί το μικρότερο στοιχείο).

ΥΠΟΠΡΟΓΡΑΜΜΑ MIN (A, m, n, k)
Β1. A: Πίνακας ακεραίων, k: ακέραιος (ο δείκτης του μικρότερου στοιχείου που θα βρεθεί)
m: ακέραιος (αρχικός δείκτης), n: ακέραιος (τελικός δείκτης), j: ακέραιος (μετρητής)
Β2. min ← A_m, k ← m, j ← m+1
Β3. ***όσο*** (j <= n)
αν (min > A_j) ***τότε***
min ← A_j
k← j
τέλος (αν)
j ← j+1
τέλος (όσο)
Β4. ***τέλος (υποπρογράμματος)***

Με την χρήση του υποπρογράμματος **MIN** που περιγράψαμε ο αλγόριθμος της ταξινόμησης των στοιχείων ενός πίνακα διατυπώνεται ως εξής:

Αλγόριθμος ΤΑΞΙΝΟΜΗΣΗ_ΑΝΤΑΛΛΑΓΗ

Β0. ΑΡΧΗ

Β1. Α: πίνακας ακεραίων, Ν, Κ, i, temp: ακέραιοι (Ν ο αριθμός στοιχείων Α)

Β2. ***διάβασε*** πίνακα Α, i← 1

Β3. ***όσο*** (i < N)

κλήση υποπρογράμματος MIN(A, i, N, K)

temp ← A_i

A_i ← A_K

A_K ← temp

i ← i+1

τέλος (όσο)

Β4. ***τύπωσε*** το Α

Β5. ΤΕΛΟΣ

Τι διάφορα έχει αυτός ο αλγόριθμος από τον πρώτο; Στηρίζονται στο ίδιο τρόπο σκέψης αλλά η ανάπτυξη τους είναι διαφορετική.

Γιατί ο δεύτερος αλγόριθμος είναι καλύτερος;

(α) διότι ο πρώτος χρησιμοποιεί έναν πίνακα παραπάνω (άρα περισσότεροι μνήμη).

(β) διότι το Β3 του πρώτου παίρνει περισσότερο χρόνο (γιατί;)

Ενδιαφέρον όμως παρουσιάζει και ο τρόπος που γίνεται η ανταλλαγή των δυο στοιχείων του πίνακα στον δεύτερο αλγόριθμο. Όπως βλέπουμε για να μην χάσουμε το περιεχόμενο μιας θέση του πίνακα όταν κάνουμε ανταλλαγή, χρησιμοποιούμε μια επιπλέον μεταβλητή την temp.

(γ) Bubble sort:

Ας δούμε ένα ακόμα τρόπο σκέψης για ταξινόμηση που ονομάζεται bubble sort (ταξινόμηση φυσαλίδας;). Υποθέτουμε ότι έχουμε να ταξινομήσουμε τους αριθμούς:

7, 5, 2, 11, 8

Το bubble sort λειτουργεί ως εξής:

Οι δυο πρώτοι αριθμοί συγκρίνονται και ο μικρότερος τοποθετείται στην πρώτη θέση του πίνακα. Μετά συγκρίνω τον δεύτερο με τον τρίτο και ο μικρότερος πάει δεύτερος. Μετά τον τρίτο με τον τέταρτο και ο μικρότερος πάει πρώτος σε αυτό το ζευγάρι και συνεχίζω με τον ίδιο τρόπο μέχρι να φτάσω στο τέλος του πίνακα. Επαναλαμβάνω την διαδικασία ξανά και συνεχίζω να την επαναλαμβάνω έως ότου να μην λάβει χώρα καμία ανταλλαγή μεταξύ των στοιχείων του πίνακα.

Ας το κάνουμε καλύτερα με τους παραπάνω αριθμούς.

- Συγκρίνω το 7 με το 5. Το 5 πάει πρώτο. Τώρα η σειρά είναι:

  5, 7, 2, 11, 8

- Συγκρίνω το 7 με το 2 . Το 2 πάει πρώτο και έχω:

  5, 2, 7, 11, 8

- Συγκρίνω το 7 με το 11. Καμία αλλαγή στη σειρά.

- Συγκρίνω το 11 με το 8 . Η σειρά γίνεται:

  5, 2, 7, 8, 11

Αν "περάσω" ξανά τον πίνακα με τον ίδιο τρόπο έχω :

2, 5, 7, 8, 11

αφού θα αλλάξει το 2 με το 5 και αν επαναλάβω την διαδικασία δεν θα υπάρξουν ανταλλαγές. Αυτό σημαίνει ότι οι αριθμοί μου μέσα στον πίνακα είναι ταξινομημένοι και εδώ τελειώνει ο αλγόριθμος μου. Η διατύπωση του αλγορίθμου σε ψευδοκώδικα:

Αλγόριθμος BUBBLE_SORT
Β0. ΑΡΧΗ
Β1. Α: πίνακας Ν ακεραίων, i, k, temp, flag: ακέραιοι (i, k μετρητές)
Β2. ***διάβασε*** Α, i ←1
Β3. ***όσο*** (i <= N)
k ← 1, flag ← 0
επανάληψη N-1 ***φορές***
αν (A_{k+1} < A_k) ***τότε***
temp ← A_k , A_k ← A_{k+1} , A_{k+1} ← temp (ανταλλαγή)
flag ← 1 (ένδειξη ότι έγινε τουλάχιστον 1 ανταλλαγή)
τέλος (αν)
k ← k +1
τέλος (επανάληψη)
αν (flag = 0) ***τότε*** i ← N+1 (εκβιάζω την έξοδο από το ΟΣΟ)
διαφορετικά i ← i+1
τέλος (αν)
τέλος (όσο)
Β4. ***τύπωσε*** το Α
Β5. ΤΕΛΟΣ

Και μια εφαρμογή: binary search.

Αν κάποιος πίνακας έχει ταξινομηθεί προηγουμένως, η αναζήτηση ενός συγκεκριμένου στοιχείου σε αυτόν είναι πολύ πιο εύκολη υπόθεση, διότι είναι δυνατόν να αποφύγουμε όλες τις συγκρίσεις του ζητούμενου στοιχείου με τα στοιχεία του πίνακα. Μια πολύ γνωστή μέθοδος για να το πετύχουμε αυτό είναι το binary search. Αλλά ας δούμε πως δουλεύει αυτή.

Πρώτα χωρίζουμε τον πίνακα σε δυο ίσα (αν είναι δυνατόν) μέρη. Συγκρίνοντας το τελευταίο στοιχείο του πρώτου μέρους (ή το πρώτο του δεύτερου μέρους) με το ζητούμενο στοιχείο, μπορούμε εύκολα (μη ξεχνάμε ότι ο πίνακας είναι ταξινομημένος) να διαπιστώσω σε ποιο μέρος από τα δύο θα βρίσκεται (αν υπάρχει) το ζητούμενο στοιχείο. Στην συνέχεια εφαρμόζω την ίδια διαδικασία για αυτό το μέρος (δηλαδή το χωρίζω ξανά σε δυο) και προχωράω με αυτόν τον τρόπο μέχρι να βρω (αν υπάρχει) το ζητούμενο στοιχείο. Ας το δούμε με ένα παράδειγμα.

Έστω ότι έχουμε τον ταξινομημένο πίνακα Α

A = (1, 2, 4, 7, 11, 16, 22, 29, 37) - 9 στοιχεία και

ότι θέλουμε να διαπιστώσουμε ότι ο αριθμός 29 βρίσκεται ανάμεσα στα στοιχεία του πίνακα.

Χωρίζω τον πίνακα σε δυο κομμάτια

A1 = (1, 2, 4, 7) και
A2 = (11, 22, 29, 37)

Το 29 συγκρινόμενο με το 7 ή το 11 είναι μεγαλύτερο άρα θα βρίσκεται (αν υπάρχει) στον Α2.

Χωρίζω τώρα τον Α2 σε δυο υπο-πίνακες:

A21 = (11, 22) και
A22 = (29, 37)

Το 29 συγκρινόμενο με το 22 είναι μεγαλύτερο άρα θα βρίσκεται στο Α22 (αν σύγκρινα με το πρώτο στοιχείο του δεύτερου μέρους θα είχα τελειώσει).

Χωρίζω το Α22 σε δυο μέρη:

Α221 = (29) και
Α222 = (37)

Το μοναδικό στοιχείο του Α221 είναι το ζητούμενο άρα τέλος.

Φυσικά στην ανάπτυξη του αλγόριθμου δεν χρησιμοποιούμε τόσους πολλούς πίνακες, αλλά δουλεύουμε με δείκτες μέσα στον πίνακα. Έτσι θα πρέπει να ορίζουμε δύο δείκτες Α και Δ (αριστερός και δεξιός δείκτης) οι οποίοι μας επιτρέπουν να παρακολουθούμε σε ποιο κομμάτι του πίνακα αναφερόμαστε κάθε φορά. Για να δούμε όμως τον αλγόριθμο:

Αλγόριθμος binary_search
Β0. ΑΡΧΗ
Β1. Χ: Πίνακας Ν ακεραίων, Μ: ακέραιος (ζητούμενο στοιχείο)
Α, Δ, i, Κ, flag: ακέραιοι
Β2. ***διάβασε*** τον Α και το Μ
Β3. Α ← 1, Δ← Ν
Κ ← ακέραιο μέρος της διαίρεσης (Α+Δ)/2
Β4. ***όσο*** (flag = 0 ***είτε*** Α > Δ)
αν (X_K = Μ) ***τότε*** flag=1, ***τέλος (αν)***
αν (X_K < Μ) ***τότε***
Α ← Κ+1
διαφορετικά
Δ ← Κ-1
τέλος (αν)
τέλος (όσο)
Β5. ***αν*** (flag = 1) ***τότε***
τύπωσε «Υπάρχει το στοιχείο στον πίνακα»
διαφορετικά
τύπωσε «Δεν υπάρχει το στοιχείο στον πίνακα»
τέλος (αν)
Β6. ΤΕΛΟΣ