Προγραμματισμός

Βασικές έννοιες – Ιστορική αναδρομή

Η έννοια του προγράμματος

Η περιγραφή της λύσης ενός προβλήματος, ως γνωστόν, γίνεται με τη βοήθεια ενός αλγορίθμου. Έτσι οι εντολές ενός προγράμματος εκφράζουν στη πραγματικότητα τα βήματα ενός αλγορίθμου σε γλώσσα που μπορεί να είναι κατανοητή από τον υπολογιστή. Οι αλγόριθμοι όπως ήδη μάθαμε, δεν είναι κάτι άγνωστο από την καθημερινή μας ζωή. Η «εκτέλεση» μιας συνταγής μαγειρικής ή οι οδηγίες για τον χρονοπρογραμματισμό μιας συσκευής video ή τα βήματα που ακολουθούμε για να θέσουμε σε λειτουργία την μηχανή του αυτοκινήτου αποτελούν ουσιαστικά αλγορίθμους.

Τα δεδομένα ενός προγράμματος αντιστοιχούν στα υλικά μιας συνταγής μαγειρικής ή στα στοιχεία που θα δώσουμε στην συσκευή video κατά τον χρονοπρογραμματισμό της ή στα χειριστήρια του αυτοκινήτου που θα ενεργοποιήσουμε την ώρα που το βάζουμε μπροστά. Τα δεδομένα δηλ. αναπαριστούν τις πληροφορίες που πρέπει να πάρει ένα πρόγραμμα, ώστε να φέρει σε πέρας επιτυχώς την αποστολή του.

Σε κάθε υπολογιστικό σύστημα υπάρχει η *Κεντρική μονάδα Επεξεργασίας* (*ΚΜΕ*) και η *Κύρια Μνήμη*. Στην Κύρια Μνήμη τοποθετούνται τα δεδομένα τα οποία είναι απαραίτητα για την επίλυση ενός προβλήματος. Παράλληλα είναι τοποθετημένες και οι εντολές οι οποίες υποδεικνύουν το πως θα γίνει η επεξεργασία των δεδομένων. Η ΚΜΕ παραλαμβάνει μία προς μία τις εντολές μαζί με τα αντίστοιχα δεδομένα, αναγνωρίζει τη σημασία της κάθε εντολής και την εκτελεί. Τα αποτελέσματα από την επεξεργασία επιστρέφονται στην Κύρια Μνήμη και μπορούν να αποτελέσουν δεδομένα κάποιας νέας επεξεργασίας, να εξαχθούν στην οθόνη του χρήστη, να αποθηκευτούν σε κάποια βοηθητική μονάδα μνήμης (π.χ. σκληρό δίσκο) κ.λπ. Εντολές, δεδομένα και πληροφορίες είναι τοποθετημένα στη μνήμη και επεξεργάζονται από την Κ.Μ.Ε. σε *ψηφιακή μορφή* (κωδικοποιημένα δηλαδή σε ακολουθίες ψηφίων του δυαδικού συστήματος αρίθμησης - 0 και 1). Με βάση τις οδηγίες του προγράμματος, η ΚΜΕ μπορεί να ελέγχει, να συγκρίνει τα δεδομένα και να εκτελεί με μεγάλη ταχύτητα στοιχειώδεις πράξεις ώστε να παραχθεί το αποτέλεσμα της επεξεργασίας.

Ιστορική αναδρομή

Από την δεκαετία του 1940 που τέθηκαν οι βάσεις της επιστήμης της πληροφορικής από τον Ελβετό καθηγητή Φον Νόυμαν, μέχρι σήμερα, παρότι η πληροφορική και οι υπολογιστές γενικά έχουν αναπτυχθεί εντυπωσιακά, οι βασικές αρχές της παραμένουν αναλλοίωτες. Η ψηφιακή λογική (δηλ. η αναπαράσταση όλων των πληροφοριών μέσα στον υπολογιστή με τα ψηφία 0 και 1) δεν έχει ακόμη ανατραπεί, και η απλή αρχή που υποστήριξε ο Φον Νόυμαν, ότι δηλ. με το συνδυασμό οκτώ δυαδικών ψηφίων (1 byte) αν τα δούμε σαν μία οντότητα, μπορούμε να αναπαραστήσουμε οποιαδήποτε πληροφορία επιθυμούμε, αποδείχθηκε σοφή.

Γλώσσες μηχανής

Η μοναδική μορφή προγράμματος που είναι κατανοητή από την ΚΜΕ ενός υπολογιστή είναι ο κώδικας ή γλώσσα μηχανής. Η ανάπτυξη προγράμματος από τους προγραμματιστές σε γλώσσα μηχανής ήταν και παραμένει μια ιδιαίτερα επίπονη και χρονοβόρα εργασία αφού απαιτεί πολύ καλή γνώση του υλικού και ανάπτυξη πολύ αναλυτικών αλγορίθμων.

Συμβολικές γλώσσες ή γλώσσες χαμηλού επιπέδου

Ένα πρώτο βήμα για την υπερπήδηση των προηγούμενων εμποδίων ήταν η ανάπτυξη γλωσσών προγραμματισμού που ήταν πιο κοντά στην ανθρώπινη γλώσσα. Οι πρώτες γλώσσες που αναπτύχθηκαν ήταν οι ***συμβολικές γλώσσες (assembly languages)*** ή ***γλώσσες χαμηλού επιπέδου (low level languages)***.

Οι συμβολικές γλώσσες διευκόλυναν σε ένα βαθμό τους ανθρώπους στην ανάπτυξη προγραμμάτων. Όμως είχαν και σοβαρά μειονεκτήματα τα σημαντικότερα των οποίων ήταν: α) Η ανάπτυξη πολύ αναλυτικών αλγορίθμων και β) Η πλήρης εξάρτηση τους από την ΚΜΕ για την οποία είχαν δημιουργηθεί, με αποτέλεσμα να μην μπορεί ένα πρόγραμμα να τρέξει σε διαφορετικό υπολογιστή.

Γλώσσες υψηλού επιπέδου

Το επόμενο βήμα (στα τέλη της δεκαετίας του 1950) ήταν η ανάπτυξη γλωσσών ***υψηλού επιπέδου*** όπως ονομάζονται. Κάθε εντολή ή πρόταση σε μια γλώσσα υψηλού επιπέδου μπορεί να αντιστοιχεί και σε ένα πακέτο εντολών σε γλώσσα μηχανής. Το πρόβλημα με αυτές τις γλώσσες όμως είναι ότι ενώ είναι ευκολότερα κατανοητές από τον άνθρωπο, δεν είναι καθόλου κατανοητές από τον υπολογιστή. Το πρόβλημα αυτό το λύνουν ειδικά προγράμματα που ονομάζονται ***μεταφραστές*** (***compilers***) ή ***διερμηνείς*** (***interpreters***) που αναλαμβάνουν να μετατρέψουν ένα πρόγραμμα από γλώσσα υψηλού επιπέδου, σε κώδικα γλώσσας μηχανής (εκτελέσιμο πρόγραμμα). Η διαδικασία της μετάφρασης κοστίζει βέβαια σε χρόνο, αλλά η ανάπτυξη ταχύτατων υπολογιστών τα επόμενα χρόνια, έθεσε αυτό το πρόβλημα ουσιαστικά στο περιθώριο.

Ένα από τα σπουδαιότερα χαρακτηριστικά των γλωσσών προγραμματισμού υψηλού επιπέδου που βοήθησε σημαντικά στην εξάπλωσή τους, είναι ***η ανεξαρτησία τους από τον υπολογιστή*** που θα εκτελεστεί το πρόγραμμα. Αυτό σημαίνει ότι ένα πρόγραμμα γραμμένο σε μια γλώσσα υψηλού επιπέδου μπορεί να εκτελεστεί σε οποιαδήποτε μηχανή αρκεί φυσικά να υπάρχει το ειδικό μεταφραστικό πρόγραμμα - εργαλείο για την αντίστοιχη μηχανή. Ο προγραμματιστής απαλλάσσεται από την ανάγκη γνώσης των ιδιαιτεροτήτων του υλικού κάθε μηχανής και μένει προσηλωμένος στην επίτευξη του στόχου του που δεν είναι άλλος από την κατασκευή ενός προγράμματος που να πληροί τις προδιαγραφές που έχουν τεθεί.

Οι πρώτες ιστορικά γλώσσες υψηλού επιπέδου καλούνται ***διαδικασιακές ή αλγοριθμικές γλώσσες*** αφού βασικό δομικό στοιχείο τους είναι η περιγραφή των διαδικασιών ή αλγορίθμων που συνθέτουν ένα πρόγραμμα. Χαρακτηριστικές γλώσσες αυτής της κατηγορίας είναι οι γλώσσες COBOL που αναπτύχθηκε για εμπορικές εφαρμογές, η FORTRAN που αναπτύχθηκε για επιστημονικές εφαρμογές και η BASIC που αναπτύχθηκε σαν γλώσσα γενικού σκοπού για εκπαίδευση αρχαρίων.

Η επόμενη εξέλιξη (στα τέλη της δεκαετίας του 1960) των γλωσσών προγραμματισμού υψηλού ήταν οι ***δομημένες γλώσσες*** (***structured languages***) με κύριο χαρακτηριστικό τη διευκόλυνση στη συγγραφή και

τη διόρθωση προγραμμάτων. Χαρακτηριστικές δομημένες γλώσσες υψηλού επιπέδου είναι οι γλώσσες ALGOL, PASCAL και C.

Μια ακόμη προσπάθεια να έλθει ο προγραμματισμός ποιο κοντά στην ανθρώπινη λογική, έφερε σαν αποτέλεσμα (στα τέλη της δεκαετίας του 1970 και ιδιαίτερα τη δεκαετία του 1980) την καθιέρωση των ***αντικειμενοστραφών γλωσσών προγραμματισμού*** (***Object Oriented Languages***). Βασικά δομικά στοιχεία σε μια τέτοια γλώσσα είναι τα αντικείμενα. Τα αντικείμενα αλληλεπιδρούν μεταξύ τους κατά τρόπον ώστε να εξασφαλίζεται η λειτουργία του προγράμματος. Χαρακτηριστικές αντικειμενοστραφείς γλώσσες υψηλού επιπέδου είναι οι γλώσσες JAVA και C++.

Τέλος η εμφάνιση υπολογιστών (στα τέλη της δεκαετίας του 1980 και τη δεκαετία του 1990) που υποστηρίζονταν από γραφικά περιβάλλοντα εργασίας άλλαξε προς το καλύτερο τον τρόπο επικοινωνίας ανθρώπου και μηχανής, με αποτέλεσμα πολλές γλώσσες προγραμματισμού να εξελιχθούν σε ***γλώσσες οπτικού προγραμματισμού*** (***visual programming***) ή ***οδηγούμενου από γεγονότα προγραμματισμού*** (***object driven programming***). Βασικά δομικά στοιχεία σε μια τέτοια γλώσσα είναι η δυνατότητα σχεδίασης προγραμμάτων με γραφικό (οπτικό) τρόπο και ενεργοποίησης λειτουργιών μετά την αναγνώριση από το πρόγραμμα της εκτέλεσης ενός γεγονότος (Π.χ. το πάτημα του ποντικιού). Χαρακτηριστικές οπτικές γλώσσες υψηλού επιπέδου είναι οι γλώσσες visual basic, visual C ++ και delphi.

Ενδιάμεσα έχουν παρουσιασθεί και οι γλώσσες ανάπτυξης εφαρμογών τεχνητής νοημοσύνης. Οι γλώσσες αυτές διέπονται από τις αρχές του συναρτησιακού και λογικού προγραμματισμού που θα παρουσιάσουμε παρακάτω. Οι κυριότερες γλώσσες της κατηγορίας είναι η LISP και η PROLOG. Βασικό χαρακτηριστικό τους είναι η δυνατότητα αναπαράστασης γεγονότων και κανόνων που καθορίζουν τις σχέσεις ανάμεσα στα γεγονότα, σε μια ενιαία μορφή. Οι γλώσσες αυτές δεν είναι ευρέως διαδεδομένες ενώ η χρήση τους περιορίζεται στην ανάπτυξη συγκεκριμένων εφαρμογών. Παρ' όλα αυτά είναι οι πλέον κατάλληλες στον τομέα της τεχνητής νοημοσύνης.

Γλώσσες τέταρτης γενιάς

Αν οι προηγούμενες γενιές γλωσσών προγραμματισμού απευθύνονται κυρίως σε επαγγελματίες της πληροφορικής οι γλώσσες **τέταρτης γενιάς** (ή γλώσσες ερωταποκρίσεων) μπορούν να χρησιμοποιηθούν και από ερασιτέχνες και από επαγγελματίες χρήστες. Είναι σχεδιασμένες για την διαχείριση πληροφοριών που υπάρχουν σε μια βάση δεδομένων. Ο χρήστης μπορεί να υποβάλλει ερωτήσεις ή να ενημερώσει την βάση δεδομένων εκφράζοντας τις επιθυμίες του με ένα τρόπο που πλησιάζει ακόμη περισσότερο την φυσική γλώσσα. Για παράδειγμα "*Βρες όλους τους μαθητές που έχουν γράψει πάνω από 10 στο μάθημα Φυσική Γενικής παιδείας Γ λυκείου και αποφοίτησαν από σχολείο της Αθήνας*».χαρακτηριστικό παράδειγμα τέτοιας γλώσσας είναι η SQL.

Ταξινόμηση Γλωσσών Προγραμματισμού	
Με βάση τα χαρακτηριστικά τους	**Με βάση τη περιοχή χρήσης**
Διαδιακασιακές ή αλγοριθμικές (COBOL,	Γλώσσες επιστημονικού σκοπού (FORTRAN)

FORTRAN, BASIC)	
Δομημένες (Algol, Pascal, C)	Γλώσσες εμπορικής χρήσης (COBOL)
Αντικειμενοστραφείς (C++, Java)	Γλώσσες τεχνητής νοημοσύνης (PROLOG, LISP)
Οπτικές (Visual BASIC, Delphi)	Γλώσσες γενικής χρήσης (PASCAL, BASIC)
Μη Διαδιακασιακές (PROLOG)	Γλώσσες ειδικής χρήσης Π.χ. Γλώσσες προσανατολισμένες στην εκπαίδευση (LOGO)
Συναρτησιακές (LISP)	Γλώσσες προγραμματισμού συστημάτων (C, C++)
Γλώσσες ερωταποκρίσεων (SQL)	Γλώσσες δικτυακών εφαρμογών (Java)

Φυσικές και τεχνητές γλώσσες και τεχνικές σχεδίασης προγραμμάτων

Φυσικές και τεχνητές γλώσσες

Και οι φυσικές και οι τεχνητές γλώσσες έχουν κοινά χαρακτηριστικά που προσδιορίζονται από γλωσσολογικούς κανόνες που αφορούν στο αλφάβητο, το λεξιλόγιο, τη γραμματική και τη σημασιολογία κάθε γλώσσας. Υπάρχουν όμως και μεγάλες διαφορές μεταξύ φυσικών και τεχνητών γλωσσών όπως:

- *Η αυστηρότητα στην έκφραση*, αφού στις τεχνητές γλώσσες δεν επιτρέπεται καμία παρέκκλιση από τους συντακτικούς κανόνες της ενώ στις φυσικές γλώσσες η σύνταξη είναι σαφώς πιο χαλαρή.
- *Η εξέλιξη* των φυσικών γλωσσών είναι καθημερινό φαινόμενο αφού αποτελούν ένα ζωντανό σύστημα επικοινωνίας που αλλάζει ανάλογα με τις εποχές και τις συνθήκες της ζωής που εξελίσσεται. Αντίθετα οι τεχνητές γλώσσες εξελίσσονται και προσαρμόζονται μόνον ύστερα από συνειδητή ανθρώπινη επέμβαση.

Μια γλώσσα χαρακτηρίζεται από:

- Το ***αλφάβητό*** της, που είναι το σύνολο των χαρακτήρων (συμβόλων) που χρησιμοποιεί (πεζά και κεφαλαία γράμματα, σημεία στίξης, ψηφία και ειδικούς χαρακτήρες)
- Το ***λεξιλόγιό*** της, που είναι το σύνολο των αποδεκτών σημασιολογικά από τη γλώσσα ακολουθιών χαρακτήρων που ονομάζονται λέξεις
- Τη ***Γραμματική*** της, που αποτελείται από το τυπικό της (χρόνοι και μορφές κάθε λέξης) και από το συντακτικό της που αφορά ένα σύνολο κανόνων για τον τρόπο σύνθεσης των λέξεων σε προτάσεις.
- Τη ***σημασιολογία*** της, που αφορά στο σύνολο των κανόνων ερμηνείας των λέξεων και προτάσεων της γλώσσας.

Τεχνικές σχεδίασης προγραμμάτων

Με το πέρασμα του χρόνου τα προγράμματα μεγαλώνουν σε όγκο, δυνατότητες και πολυπλοκότητα, προκειμένου να καλύψουν συνθετότερες ανάγκες και να επιλύσουν δυσκολότερα προβλήματα.

Στόχος των αναλυτών και των προγραμματιστών είναι πάντα η δημιουργία λειτουργικών, αξιόπιστων και απαλλαγμένων από λάθη προγραμμάτων στο μικρότερο δυνατό χρόνο αλλά και με δυνατότητες εύκολης

τροποποίησης και επέκτασης. Για να επιτευχθεί αυτός ο στόχος υιοθετήθηκαν διάφορες μεθοδολογίες και τεχνικές που χρησιμοποιούνται ευρέως ακόμη και σήμερα.

Μια μεθοδολογία ανάλυσης, σχεδίασης και συγγραφής λογισμικού (προγραμμάτων) που στηρίζεται στην τεχνική της «*από επάνω προς τα κάτω*» επίλυσης προβλημάτων (***ιεραρχική σχεδίαση***) και ονομάζεται ***δομημένος προγραμματισμός*** αναπτύχθηκε και διαδόθηκε ευρύτατα στο χώρο της πληροφορικής.

ΧΑΡΑΚΤΗΡΙΣΤΙΚΑ ΔΟΜΗΜΕΝΟΥ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

- Στηρίζεται στην ιεραρχική σχεδίαση και προγραμματισμό
- Εμπεριέχει τις αρχές του τμηματικού προγραμματισμού
- Χρησιμοποιεί τις δομές ακολουθίας, επιλογής και επανάληψης και αποφεύγει τη χρήση της GOTO.

Η βασική ιδέα στο δομημένο προγραμματισμό είναι η ανάλυση σε μικρότερα υποπροβλήματα (τμήματα): Ένα σύνθετο πρόβλημα μπορεί να διασπαστεί σε λιγότερο ή περισσότερο ανεξάρτητα **υποπροβλήματα (τμήματα).** Σε πρώτο επίπεδο λοιπόν η λύση του προβλήματος δίνεται σαν ένας γενικός αλγόριθμος που θεωρεί ότι υπάρχει η λύση για κάθε υποπρόβλημα. Κάθε υποπρόβλημα με τη σειρά του ίσως να μπορεί να διασπαστεί σε ακόμα μικρότερα υποπροβλήματα. Σε δεύτερο επίπεδο σχεδιάζονται αλγοριθμικά οι λύσεις του κάθε υποπροβλήματος. Οι σχετικοί αλγόριθμοι και πάλι θεωρούν ότι υπάρχουν οι λύσεις για κάθε μικρότερο υποπρόβλημα. Όταν φτάσουμε σε κάποιο επίπεδο όπου η επίλυση του κάθε υποπροβλήματος ξεχωριστά είναι εύκολη τότε σταματάμε και σχεδιάζουμε τους αντίστοιχους λεπτομερειακούς πια αλγορίθμους.

Ο κάθε επιμέρους αλγόριθμος υλοποιείται και σε μια ενότητα προγράμματος η οποία διέπεται από τις αρχές του τμηματικού προγραμματισμού. Στον ***τμηματικό προγραμματισμό*** ένα πρόγραμμα αποτελείται από **ενότητες** οι οποίες πρέπει να χαρακτηρίζονται από όσο το δυνατόν μεγαλύτερο *βαθμό ανεξαρτησίας* από τις υπόλοιπες ενότητες και από *υψηλό βαθμό συνεκτικότητας* (να υλοποιούν μια συγκεκριμένη διαδικασία ή λειτουργία μόνο). Η σύνθεση όλων των παραπάνω ενοτήτων σε ένα ενιαίο πρόγραμμα μας δίνει την τελική λύση του προβλήματος.

Πριν την εμφάνιση του δομημένου προγραμματισμού χρησιμοποιούνταν ευρέως η εντολή **GOTO** (Πήγαινε), η οποία μπορούσε να μεταφέρει τον έλεγχο του προγράμματος όπου επιθυμούσε ο προγραμματιστής (μπροστά ή πίσω), με αποτέλεσμα ένα πρόγραμμα να είναι σχεδόν αδύνατο να κατανοηθεί και να διορθωθεί επειδή δεν υπήρχε μια αυστηρή δομή σχεδίασης. Ο δομημένος σχεδιασμός (παρότι για λόγους συμβατότητας με παλαιότερα προγράμματα υποστηρίζει την εντολή GOTO) δεν συνιστά τη χρήση αυτής της εντολής. Οι δομημένες γλώσσες εξυπηρετώντας τους στόχους του προγραμματιστή που αναφέρθηκαν προηγουμένως και στηρίζονται στην αποδεδειγμένη αρχή ***ότι για να δομηθεί ένα οποιοδήποτε πρόγραμμα χρειάζονται μόνο οι δομές της ακολουθίας, της επιλογής και της επανάληψης*** και συνδυασμός αυτών των δομών. Επίσης ***κάθε ενότητα προγράμματος πρέπει να έχει μόνο μια είσοδο και έξοδο***.

Οι περισσότερες γλώσσες υψηλού επιπέδου υποστηρίζουν την ανάπτυξη δομημένων προγραμμάτων γι αυτό ονομάζονται και δομημένες γλώσσες. Ο κάθε αλγόριθμος ειδικότερος ή γενικότερος παίρνει την μορφή ενός υποπρογράμματος. Το τελικό πρόγραμμα κτίζεται από τα υποπρογράμματα που κατασκευάζονται το καθένα ξεχωριστά. Το κάθε υποπρόγραμμα έχει συγκεκριμένο όνομα και δικά του δεδομένα. Ένα υποπρόγραμμα μπορεί να κληθεί από υποπρογράμματα υψηλότερου ή του ίδιου επιπέδου. Κατά την κλήση ενός υποπρογράμματος το καλούν υποπρόγραμμα μπορεί να μεταβιβάσει στο υποπρόγραμμα που καλείται διάφορα δεδομένα. Όταν το υποπρόγραμμα ολοκληρώσει την εργασία του επιστρέφει τον έλεγχο στο υποπρόγραμμα που το κάλεσε. Σε πολλές περιπτώσεις υποπρογραμμάτων η επιστροφή του ελέγχου στο καλούν υποπρόγραμμα συνοδεύεται και από την επιστροφή κάποιας τιμής. Αυτή μπορεί να είναι το αποτέλεσμα από την εκτέλεση υπολογισμών στο εσωτερικό του υποπρογράμματος που κλήθηκε, κ.λπ.

Παράδειγμα:

Έστω ότι πρέπει να κατασκευαστεί πρόγραμμα που να διαβάζει την βαθμολογία των μαθητών ενός τμήματος, να υπολογίζει και να εμφανίζει στην οθόνη τα εξής: α) Τον αριθμό των μαθητών που περνάνε τη τάξη. β) Τον μέσο όρο των μαθητών που περνάνε τη τάξη. γ) Τον μέσο όρο των μαθητών όλου του τμήματος. Μια λύση πρώτου επιπέδου θα μπορούσε να είναι.

ΑΡΧΗ

ΔΙΑΒΑΣΕ τον βαθμό του μαθητή

ΟΣΟ υπάρχουν και άλλοι βαθμοί μαθητών **ΕΠΑΝΕΛΑΒΕ**

Μέτρησε μαθητές πάνω από τη βάση

Υπολόγισε μέσο όρο πάνω από τη βάση

Υπολόγισε μέσο όρο όλων των μαθητών

ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ

ΕΜΦΑΝΙΣΕ αποτελέσματα

ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ

Το κάθε βήμα του παραπάνω γενικού αλγορίθμου αποτελεί και ένα υποπρόβλημα που εύκολα μπορεί να επιλυθεί αλγοριθμικά. Αφού κατασκευαστούν λοιπόν οι αντίστοιχοι αλγόριθμοι τότε εύκολα μπορούν να γραφτούν ως υποπρογράμματα σε μια γλώσσα προγραμματισμού. Εύκολα επίσης μπορεί να γραφτεί ως υποπρόγραμμα και ο παραπάνω γενικός αλγόριθμος.