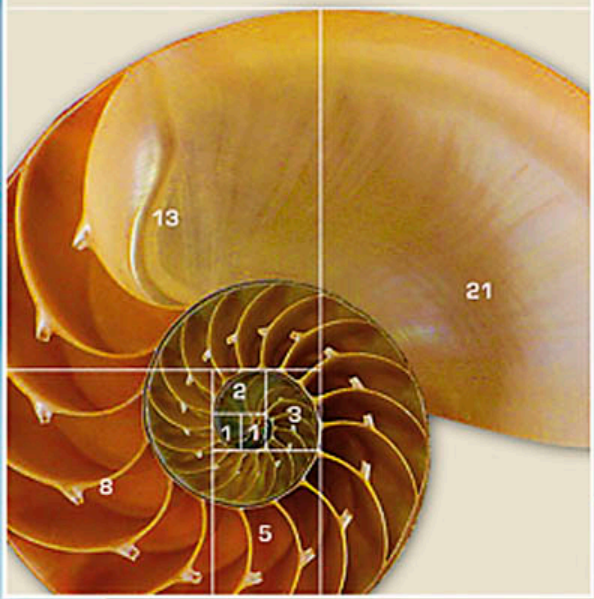
Νικόλαος Καραμπετάκης

Εισαγωγή στη Fortran 90/95/2003



2η έκδοση

ΕΚΔΟΣΕΙΣ ΖΗΤΗ

ISBN 978-960-456-280-0

Φωτοστοιχειοθεσία **Π. ΖΗΤΗ & Σια ΟΕ**
Εκτύπωση 18ο χλμ Θεσσαλονίκης - Περαίας
Βιβλιοδεσία Τ.Θ. 4171 • Περαία Θεσσαλονίκης • Τ.Κ. 570 19
Τηλ.: 2392.072.222 - Fax: 2392.072.229 • e-mail: info@ziti.gr

ΕΚΔΟΣΕΙΣ ΖΗΤΗ

www.ziti.gr

ΒΙΒΛΙΟΠΩΛΕΙΟ ΘΕΣΣΑΛΟΝΙΚΗΣ - ΚΕΝΤΡΙΚΗ ΔΙΑΘΕΣΗ:
Αρμενοπούλου 27 - 546 35 Θεσσαλονίκη • Τηλ.: 2310-203.720 • Fax 2310-211.305
e-mail: sales@ziti.gr

ΒΙΒΛΙΟΠΩΛΕΙΟ ΑΘΗΝΩΝ - ΕΝΩΣΗ ΕΚΔΟΤΩΝ ΒΙΒΛΙΟΥ ΘΕΣΣΑΛΟΝΙΚΗΣ:
Στοά του Βιβλίου (Πεσμαζόγλου 5) - 105 64 ΑΘΗΝΑ • Τηλ.-Fax: 210-3211.097

ΑΠΟΘΗΚΗ ΑΘΗΝΩΝ - ΠΩΛΗΣΗ ΧΟΝΔΡΙΚΗ:
Ασκληπιού 60 - Εξάρχεια 114 71, Αθήνα • Τηλ.-Fax: 210-3816.650 • e-mail: athina@ziti.gr

ΗΛΕΚΤΡΟΝΙΚΟ ΒΙΒΛΙΟΠΩΛΕΙΟ: www.ziti.gr

Πρόλογος

Το βιβλίο αυτό γράφτηκε για τις ανάγκες του μαθήματος «*Εισαγωγή στον Προγραμματισμό Η/Υ*»* που διδάσκεται στο Α′ εξάμηνο του Τμήματος Μαθηματικών του Α.Π.Θ.. Κύριος στόχος του βιβλίου αυτού είναι να εξοικιώσει τον αρχάριο προγραμματιστή με τις βασικές αρχές του προγραμματισμού των Η/Υ χρησιμοποιώντας την γλώσσα προγραμματισμού FORTRAN 90 καθώς και στοιχεία της FORTRAN 95 και FORTRAN 2003.

Το σημαντικότερο στο βιβλίο αυτό είναι ότι σχεδόν κάθε πρόγραμμα συνοδεύεται από τον αντίστοιχο αλγόριθμο, στον οποίο περιγράφονται τα βήματα επίλυσης του προβλήματος, δίνοντας έτσι την δυνατότητα στον αναγνώστη να μεταφέρει εύκολα αυτά που έμαθε και σε άλλες γλώσσες προγραμματισμού.

Για την καλύτερη κατανόηση της ύλης οι αναγνώστες παροτρύνονται

α) να εκτελέσουν τα έτοιμα προγράμματα που παρουσιάζονται στο βιβλίο έτσι ώστε να αποκτήσουν εξοικίωση με τον προγραμματισμό σε Fortran,

β) να προσπαθούν να επιλύουν τις δραστηριότητες αλλά και τις ασκήσεις επανάληψης που προτείνονται σε κάθε ενότητα που έχουν ως στόχο την ενεργό συμμετοχή τους και την εμπέδωση της ύλης. Σε περίπτωση αδυναμίας επίλυσης των δραστηριοτήτων θα πρότεινα μια καλή επανάληψη της ύλης που σχετίζεται με την επίλυση της δραστηριότητας.

Οι λύσεις των δραστηριοτήτων παρουσιάζονται στο τέλος κάθε ενότητας και χρησιμεύουν ως οδηγός αυτοαξιολόγησης για την πορεία εκμάθησης του αναγνώστη.

Το βιβλίο αυτό αποτελεί βελτιωμένη έκδοση του βιβλίου «*Εισαγωγή στην Fortran 90/95*» από τον ίδιο συγγραφέα. Στην έκδοση αυτή έχουν προστεθεί τα παρακάτω:

- Στο πρώτο κεφάλαιο (*Εισαγωγή στους Η/Υ και στο Περιβάλλον της Intel Visual Fortran*) προστέθηκε η ενότητα 1.6 η οποία εισάγει τους αναγνώστες στην έννοια της πολυπλοκότητας των αλγορίθμων, και συνεπώς στην μέτρηση της απόδοσης των αλγορίθμων που κατασκευάζουν, ενώ σε παραδείγματα που ακολουθούν στο υπόλοιπο βιβλίο αρκετές φορές αναλύεται και η πολυπλοκότητα των αλγορίθμων που δίνονται. Στο κεφάλαιο αυτό επίσης δίνονται οδηγίες για την υλοποίηση ενός

---

* Διαφάνειες και χρήσιμο υλικό από το μάθημα αυτό μπορεί να βρει ο αναγνώστης στις ιστοσελίδες

http://anemos.web.auth.gr/fortran90/default.htm και
http://eclass.auth.gr/courses/MATH104

προγράμματος στην *Intel Visual Fortran* στο περιβάλλον του *Microsoft Visual Studio 2008*.

- Στο δεύτερο κεφάλαιο (*Βασικά στοιχεία ενός προγράμματος στη Fortran 90*) προστέθηκαν επιπλέον πληροφορίες για τον τρόπο αποθήκευσης ακεραίων και πραγματικών αριθμών στην μνήμη του Η/Υ, με σκοπό να γίνει πιο εύκολα αντιληπτή η κατηγοριοποίηση των μεταβλητών που δέχονται ακέραιους και πραγματικούς αριθμούς στην Fortran. Επίσης με τον τρόπο αυτό γίνεται αντιληπτό ότι:
  α) τα μεγέθη των αριθμών που χρησιμοποιούμε στην Fortran είναι συγκεκριμένα και
  β) υπάρχει τις περισσότερες φορές μια διαφορά μεταξύ της ακριβής τιμής του αριθμού που δίνουμε και της τιμής που ουσιαστικά δέχεται ο υπολογιστής στην μνήμη του ακόμα και αν ο αριθμός που θέλουμε να αποθηκεύσουμε έχει πεπερασμένα δεκαδικά ψηφία.

  Με τον τρόπο αυτό γίνεται μια μικρή εισαγωγή σε έννοιες που συναντούμε στον κοινό κλάδο των Μαθηματικών αλλά και της Πληροφορικής που ονομάζεται *Αριθμητική Ανάλυση*. Στοιχεία από την Αριθμητική Ανάλυση συναντούμε και σε άλλα κεφάλαια όπως το κεφάλαιο 6 όπου γίνεται μια αναφορά στη μέθοδο Newton-Raphson, στο κεφάλαιο 8 όπου γίνεται ο υπολογισμός του π με την μέθοδο Monte-Carlo, πέρα από τις εφαρμογές του κεφαλαίου 10 που υπήρχαν και στην πρώτη έκδοση.
- Στο έβδομο κεφάλαιο (*Πίνακες*) υπάρχουν πλέον τρεις τρόποι ταξινόμησης πινάκων:
  α) *ταξινόμηση με επιλογή* (selection sort),
  β) *ταξινόμηση με αντιμετάθεση* (ή αλλιώς ταξινόμηση φυσαλίδας (bubble sort)) και
  γ) *ταξινόμηση με παρεμβολή* (insertion sort).
- Στο όγδοο κεφάλαιο (*Διαδικασίες και Συναρτήσεις*) αναλύονται άλλοι δύο αλγόριθμοι ταξινόμησης:
  α) ο αλγόριθμος *ταξινόμησης μέσω συγχώνευσης* (merge-sort) και
  β) ο αλγόριθμος *ταξινόμησης με διαμερισμό* (ή διαφορετικά quicksort).

  Με την παρουσίαση διαφορετικών αλγορίθμων ταξινόμησης θέλουμε να τονίσουμε ότι για το ίδιο πρόβλημα πολλές φορές υπάρχουν διαφορετικοί τρόποι επίλυσης, τους οποίους και μπορούμε να συγκρίνουμε μελετώντας την πολυπλοκότητα του καθενός από αυτούς. Έτσι γίνεται μια εισαγωγή στον σημαντικό αυτό κλάδο της *Πληροφορικής* που ασχολείται με *Αλγορίθμους και Πολυπλοκότητα*.
- Έχει προστεθεί ένα επιπλέον κεφάλαιο (το 9^{ο} κεφάλαιο), στο οποίο γίνεται μια εισαγωγή στους δείκτες (pointers) καθώς και στη χρήση τους για την δημιουργία δυναμικών δομών δεδομένων όπως οι απλά γραμμικές λίστες (singly linear lists).
- Τέλος το βιβλίο εμπλουτίστηκε με επιπλέον λυμένες αλλά και άλυτες ασκήσεις.

Θα ήθελα να ευχαριστήσω τον Ομότιμο Καθηγητή κ. Κωνσταντίνο Λάζο του Τμήματος Πληροφορικής του Α.Π.Θ., για τις πολύτιμες συμβουλές του όλα αυτά τα χρόνια που συνεργαστήκαμε, καθώς και την κα. Μπλέρη Καββαδία, τον κ. Γιώργο Ραχώνη, την κα. Κατερίνα Χατζηφωτεινού και την κα. Γρηγοριάδου Αναστασία με τους οποίους συνεργάστηκα στενά για μεγάλο διάστημα στη διδασκαλία του μαθήματος. Θα ήθελα επίσης να ευχαριστήσω την εταιρεία Compaq και την υπεύθυνη του προγράμματος κα. C. Staudinger, οι οποίοι διέθεσαν δωρεάν την Compaq Visual Fortran στο Τμήμα Μαθηματικών για την κάλυψη των αναγκών του μαθήματος. Τα προγράμματα που παρουσιάζονται έχουν γραφεί και εκτελεστεί στην Compaq Visual Fortran και στην Intel Visual Fortran. Τέλος, και πιο πολύ από όλους, θα ήθελα να ευχαριστήσω την οικογένεια μου για την υπομονή αλλά και συμπαράσταση που μου δείξανε κατά τη συγγραφή του βιβλίου.

Περιεχόμενα

4 Ο τελεστής ανάθεσης και οι εντολές εισόδου - εξόδου

5 Εντολές συνθήκης και διακλάδωσης

6 Οι εντολές επανάληψης

7 Πίνακες

11 Εφαρμογές

1ο Κεφάλαιο

Εισαγωγή στους Η/Υ και στο Περιβάλλον της Intel Visual Fortran

Σκοπός

Σκοπός του κεφαλαίου αυτού είναι να παρουσιάσει
α) μια συνοπτική περιγραφή της δομής των Η/Υ,
β) μια διαδικασία επίλυσης προβλημάτων με την βοήθεια του Η/Υ,
γ) έναν τρόπο μέτρησης της απόδοσης των αλγορίθμων,
δ) το περιβάλλον της Intel Visual Fortran.

Προσδοκώμενα αποτελέσματα

Έχοντας διαβάσει το κεφάλαιο αυτό θα είσαι σε θέση:

- να δώσεις τον ορισμό του υπολογιστή,
- να περιγράψεις τα δύο βασικά μέρη από τα οποία αποτελείται ο Η/Υ,
- να δώσεις το βασικό ορισμό του υλικού (hardware) ενός Η/Υ, να περιγράψεις τη βασική δομή του καθώς και να δώσεις μια σύντομη περιγραφή των τμημάτων που το απαρτίζουν,
- να δώσεις τον ορισμό του λογισμικού, καθώς και να περιγράψεις δύο βασικές κατηγορίες λογισμικού,
- να ορίσεις τι είναι αλγόριθμος,
- να ορίσεις τι είναι γλώσσα προγραμματισμού και να περιγράψεις τις δύο μεγάλες κατηγορίες στις οποίες χωρίζονται οι γλώσσες προγραμματισμού,
- να διαχωρίσεις την λειτουργία του διερμηνέα από αυτή του μεταφραστή,
- να περιγράψεις την διαδικασία εκτέλεσης και μετάφρασης ενός προγράμματος,
- να περιγράψεις τα στάδια του προγραμματισμού,
- να περιγράψεις με ποιον τρόπο μετρούμε την απόδοση ενός αλγορίθμου,
- να εισάγεις ένα πρόγραμμα γραμμένο σε FORTRAN 90/95/2003 στο περιβάλλον της Intel Visual Fortran και να το εκτελείς.

Έννοιες κλειδιά

- Υλικό (hardware)
- Λογισμικό (software)
- Λειτουργικό σύστημα
- Κύρια Μνήμη (RAM)
- Κεντρική μονάδα επεξεργασίας (CPU)
- Λογισμικό συστήματος
- Αλγόριθμος
- Λογισμικό εφαρμογών
- Πηγαίος Κώδικας
- Αντικείμενο Πρόγραμμα
- Ασυμπτωτική πολυπλοκότητα
- $\Theta(f(n)), O(f(n)), \Omega(f(n)), o(f(n)), \omega(f(n))$

Εισαγωγικές Παρατηρήσεις

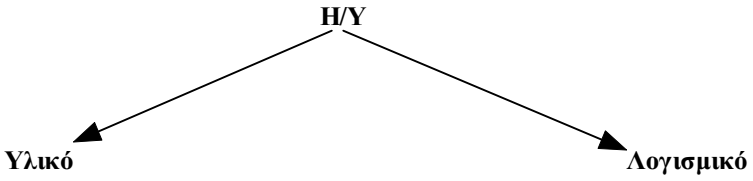
Στο κεφάλαιο αυτό θα εισάγουμε βασικές έννοιες που αφορούν τη δομή του Η/Υ και θα περιγράψουμε τα στάδια υλοποίησης ενός προγράμματος με τη βοήθεια της FORTRAN 90/95/2003.

Πιο συγκεκριμένα, στην ενότητα 1.1 θα αναφέρουμε ότι ένας υπολογιστής αποτελείται από δύο βασικά μέρη: το υλικό (hardware) και το λογισμικό (software) και θα προχωρήσουμε στη συνοπτική περιγραφή τους στις ενότητες 1.2 και 1.3. Στην ενότητα 1.4 θα αναφέρουμε τι είναι αλγόριθμος ενώ στην ενότητα 1.5 θα δούμε τι είναι γλώσσα προγραμματισμού και σε ποιες βασικές κατηγορίες μπορούμε να χωρίσουμε τις γλώσσες προγραμματισμού. Στην ενότητα 1.6 θα αναφέρουμε τα 7 βασικά στάδια του προγραμματισμού. Στην ενότητα 1.7 αναφέρουμε μεθόδους κατηγοριοποίησης της γενικής συμπεριφοράς των αλγορίθμων μέσω της ασυμπτωτικής πολυπλοκότητας του χρόνου εκτέλεσης. Τέλος, στην ενότητα 1.8 θα εξηγήσουμε πώς μπορεί να υλοποιηθεί στην Intel Visual Fortran (έκδοση 11) στο περιβάλλον του Microsoft Visual Studio 2008 ένα πρόγραμμα που έχουμε γράψει στη FORTRAN 90/95/2003.

1.1 Ποια είναι τα βασικά μέρη από τα οποία αποτελείται ένας Η/Υ;

Ο ηλεκτρονικός υπολογιστής (Η/Υ) είναι μια ηλεκτρονική συσκευή, η οποία επεξεργάζεται με μεγάλη ακρίβεια και ταχύτητα έναν τεράστιο όγκο πληροφοριών. Η βασική διαφορά του από άλλες ηλεκτρονικές συσκευές βρίσκεται στη δυνατότητα προγραμματισμού του. Είναι δυνατό ο ίδιος Η/Υ, με τη βοήθεια διαφορετικών *προγραμμάτων* να είναι το ίδιο χρήσιμος σε ένα Φυσικό, σε ένα Μαθηματικό, σε ένα Μηχανικό κ.τ.λ.

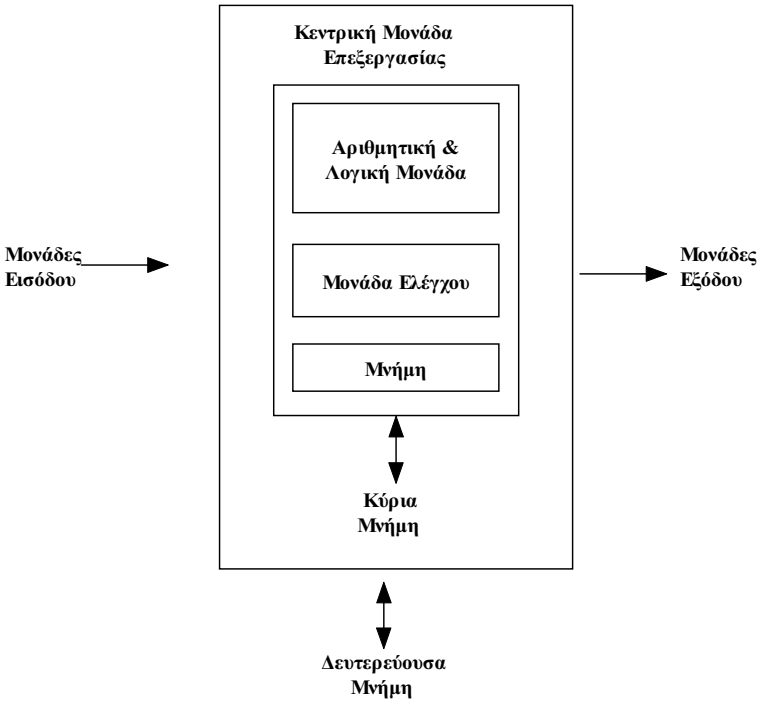
Ο Η/Υ αποτελείται από το *Hardware* ή *υλικό* και το *Software* ή *λογισμικό*. Με τον όρο *Hardware* αναφερόμαστε στις διάφορες μονάδες εισόδου, επεξεργασίας και εξόδου του Η/Υ, όπως το πληκτρολόγιο, η κεντρική μονάδα επεξεργασίας, η οθόνη κ.τ.λ., ενώ με τον όρο *Software* στα προγράμματα τα οποία κατευθύνουν τον Η/Υ. Τα προγράμματα αυτά αποτελούν ένα σύνολο από οδηγίες γραμμένες κατά τέτοιο τρόπο, ώστε να είναι άμεσα κατανοητές από τον Η/Υ.



Σχήμα 1.1 Τα μέρη του Η/Υ.

1.2 Ποια είναι η δομή του υλικού ενός Η/Υ;

Στο παρακάτω σχήμα φαίνονται τα βασικά μέρη από τα οποία αποτελείται το υλικό του Η/Υ:



Σχήμα 1.2 Δομή του υλικού του Η/Υ

όπου:

Μονάδες Εισόδου: Είναι το σύνολο των μονάδων με τις οποίες επιτυγχάνεται η είσοδος των πληροφοριών στον Η/Υ, π.χ. πληκτρολόγιο, ποντίκι, κ.τ.λ.

Κύρια Μνήμη (RAM): Η μονάδα αυτή χρησιμοποιείται για την αποθήκευση του προγράμματος, των δεδομένων, καθώς επίσης και των ενδιάμεσων και τελικών αποτελεσμάτων του προγράμματος, πριν αυτά εμφανιστούν στην οθόνη ή τυπωθούν στον εκτυπωτή ή γραφτούν σε κάποια μονάδα δευτερεύουσας μνήμης, όπως ο μαγνητικός δίσκος, το CD-ROM, το DVD-ROM, κ.λπ.

Κεντρική Μονάδα Επεξεργασίας (CPU): Η μονάδα στην οποία γίνονται οι κάθε είδους επεξεργασίες των πληροφοριών, καθώς και ο συντονισμός και έλεγχος των λειτουργιών του Η/Υ. Αποτελείται από την αριθμητική και λογική μονάδα, τη μονάδα ελέγχου και μονάδες μνήμης.

Μονάδες Εξόδου: Είναι το σύνολο των μονάδων με τις οποίες επιτυγχάνεται η έξοδος των αποτελεσμάτων από τον Η/Υ π.χ. οθόνη, εκτυπωτής, κ.τ.λ.

Μερικές μονάδες του Η/Υ, όπως ο σκληρός δίσκος, και το modem, μπορούν να θεωρηθούν ως μονάδες εισόδου-εξόδου γιατί δίνουν πληροφορίες στον Η/Υ και παίρνουν από αυτόν.

1.3 Ποιες είναι οι βασικές κατηγορίες λογισμικού;

Ο Η/Υ αποτελεί μια ηλεκτρονική συσκευή, η οποία λειτουργεί μόνο κάτω από ένα σύνολο προγραμμάτων, τα οποία όλα μαζί αποτελούν το λογισμικό ή software του Η/Υ. Το λογισμικό του Η/Υ χωρίζεται σε δύο κατηγορίες:

α) στο λογισμικό συστήματος και

β) στο λογισμικό των εφαρμογών.

Σχήμα 1.3 Κατηγορίες λογισμικού

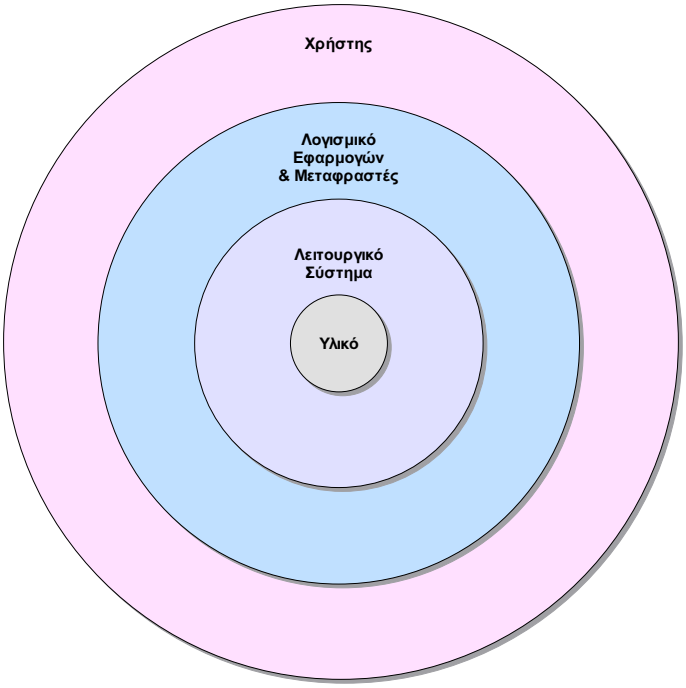
Λογισμικό Συστήματος: Αποτελείται από όλα εκείνα τα προγράμματα που εξασφαλίζουν τη σύνδεση όλων των μονάδων μεταξύ τους και τον έλεγχο των λειτουργιών

τους. Το λογισμικό συστήματος περιλαμβάνει κυρίως το λειτουργικό σύστημα και τα μεταφραστικά προγράμματα.

Το *λειτουργικό σύστημα* (Λ.Σ.) είναι ο συνδετικός κρίκος μεταξύ του υλικού από τη μια πλευρά και των προγραμμάτων εφαρμογής από την άλλη. Πρωταρχικός ρόλος του Λ.Σ. είναι η εύκολη και αποδοτική χρήση του υλικού από το χρήστη. Σε γενικές γραμμές το Λ.Σ. καθορίζει στον υπολογιστή πώς να καλεί κάποιο πρόγραμμα από μια μονάδα αποθήκευσης, πώς να αποθηκεύει δεδομένα σ' αυτές, πώς να χειρίζεται την οθόνη και τον εκτυπωτή και σε γενικές γραμμές αναλαμβάνει την διαχείριση και συντονισμό του υλικού του υπολογιστή που απαιτείται για την εκτέλεση μιας ολοκληρωμένης εργασίας. Το Λ.Σ. και η αρχιτεκτονική των Η/Υ έχουν επηρεάσει το ένα το άλλο σε μεγάλο βαθμό. Η ραγδαία εξέλιξη, λοιπόν, στο χώρο της αρχιτεκτονικής των Η/Υ οδήγησε και στην αντίστοιχη εξέλιξη του χώρου των λειτουργικών συστημάτων (αν και πολλές φορές συνέβη και το αντίθετο).

Σήμερα υπάρχουν δύο κατηγορίες λειτουργικών συστημάτων:

α) Τα λειτουργικά συστήματα που μπορούν να εξυπηρετούν ένα χρήστη κάθε φορά και χρησιμοποιούνται στους μικροϋπολογιστές. Τέτοια λειτουργικά συστήματα είναι το WINDOWS XP, τα WINDOWS NT Workstation, Windows 2000, Windows 7, Windows Vista κ.τ.λ.



Σχήμα 1.4 Σχέση Υλικού – Λογισμικού - Χρήστη.

β) Τα λειτουργικά συστήματα που έχουν την δυνατότητα να εξυπηρετούν συγχρόνως περισσότερους από ένα χρήστες (σε δίκτυα υπολογιστών). Τέτοια λειτουργικά συστήματα είναι το Solaris, Unix/Linux και διάφορες εκδόσεις των Microsoft Windows όπως Windows Server 2008, Novel Netware, Cisco IOS, Junos κ.τ.λ..

Σήμερα πλέον τα λειτουργικά συστήματα που χρησιμοποιούνται για ατομικούς υπολογιστές διαθέτουν δυνατότητες για την εξυπηρέτηση ενός δικτύου υπολογιστών όπως το MAC OS X και όλες οι εκδόσεις των Microsoft Windows.

Λογισμικό Εφαρμογών: Εδώ ανήκουν όλα τα προγράμματα ή σύνολα προγραμμάτων που γράφονται για να διεκπεραιώνουμε με τη βοήθεια του Η/Υ διάφορες εργασίες, όπως προγράμματα επεξεργασίας κειμένου, προγράμματα σχεδίασης, λογιστικά φύλλα κ.λ.π.

1.4 Τι είναι αλγόριθμος;

Αλγόριθμος είναι η ακριβής περιγραφή μιας αυστηρά καθορισμένης σειράς ενεργειών που πρέπει να ακολουθήσουμε ώστε να φέρουμε σε πέρας μια διαδικασία ή να λύσουμε ένα πρόβλημα, με την προϋπόθεση ότι η διαδικασία αυτή θα τερματίσει σε πεπερασμένο χρόνο. Στην περίπτωση της Πληροφορικής χρησιμοποιούμε για την περιγραφή αυτή μια γλώσσα που έχει ίδια δομή μ'αυτή που χρησιμοποιεί ο υπολογιστής και η οποία περιγράφει σειρές βημάτων με τρόπο αυστηρό, μαθηματικά ξεκάθαρο, χωρίς ασάφειες και διφορούμενα. Ένας από τους πιο γνωστούς ιστορικούς αλγόριθμους είναι ο αλγόριθμος του Ευκλείδη για την εύρεση του μέγιστου κοινού διαιρέτη δύο ακεραίων αριθμών ο οποίος περιέχεται στο 7^{ο} βιβλίο των **Στοιχείων**.

7^{ο} βιβλίο των Στοιχείων – Πρόταση 2

Δύο ἀριθμῶν δοθέντων μὴ πρώτων πρὸς ἀλλήλους τὸ μέγιστον αὐτῶν κοινὸν μέτρον εὑρεῖν. Ἔστωσαν οἱ δοθέντες δύο ἀριθμοὶ μὴ πρῶτοι πρὸς ἀλλήλους οἱ ΑΒ, ΓΔ. δεῖ δὴ τῶν ΑΒ, ΓΔ τὸ μέγιστον κοινὸν μέτρον εὑρεῖν. Εἰ μὲν οὖν ὁ ΓΔ τὸν ΑΒ μετρεῖ, μετρεῖ δὲ καὶ ἑαυτόν, ὁ ΓΔ ἄρα τῶν ΓΔ, ΑΒ κοινὸν μέτρον ἐστίν. καὶ φανερόν, ὅτι καὶ μέγιστον: οὐδεὶς γὰρ μείζων τοῦ ΓΔ τὸν ΓΔ μετρήσει. Εἰ δὲ οὐ μετρεῖ ὁ ΓΔ τὸν ΑΒ, τῶν ΑΒ, ΓΔ ἀνθυφαιρουμένου ἀεὶ τοῦ ἐλάσσονος ἀπὸ τοῦ μείζονος λειφθήσεταί τις ἀριθμός, ὃς μετρήσει τὸν πρὸ ἑαυτοῦ. μονὰς μὲν γὰρ οὐ λειφθήσεται: εἰ δὲ μή, ἔσονται οἱ ΑΒ, ΓΔ πρῶτοι πρὸς ἀλλήλους: ὅπερ οὐχ ὑπόκειται. λειφθήσεταί τις ἄρα ἀριθμός, ὃς μετρήσει τὸν πρὸ ἑαυτοῦ. καὶ ὁ μὲν ΓΔ τὸν ΒΕ μετρῶν λειπέτω ἑαυτοῦ ἐλάσσονα τὸν ΕΑ, ὁ δὲ ΕΑ τὸν ΔΖ μετρῶν λειπέτω ἑαυτοῦ ἐλάσσονα τὸν ΖΓ, ὁ δὲ ΓΖ τὸν ΑΕ μετρείτω. ἐπεὶ οὖν ὁ ΓΖ τὸν ΑΕ μετρεῖ, ὁ δὲ ΑΕ τὸν ΔΖ μετρεῖ, καὶ ὁ ΓΖ ἄρα τὸν ΔΖ μετρήσει: μετρεῖ δὲ καὶ ἑαυτόν: καὶ ὅλον ἄρα τὸν ΓΔ μετρήσει. ὁ δὲ ΓΔ

τὸν ΒΕ μετρεῖ: καὶ ὁ ΓΖ ἄρα τὸν ΒΕ μετρεῖ: μετρεῖ δὲ καὶ τὸν ΕΑ: καὶ ὅλον ἄρα τὸν ΒΑ μετρήσει: μετρεῖ δὲ καὶ τὸν ΓΔ: ὁ ΓΖ ἄρα τοὺς ΑΒ, ΓΔ μετρεῖ. ὁ ΓΖ ἄρα τῶν ΑΒ, ΓΔ κοινὸν μέτρον ἐστίν. λέγω δή, ὅτι καὶ μέγιστον. εἰ γὰρ μή ἐστιν ὁ ΓΖ τῶν ΑΒ, ΓΔ μέγιστον κοινὸν μέτρον, μετρήσει τις τοὺς ΑΒ, ΓΔ ἀριθμοὺς ἀριθμὸς μείζων ὢν τοῦ ΓΖ. μετρείτω, καὶ ἔστω ὁ Η. καὶ ἐπεὶ ὁ Η τὸν ΓΔ μετρεῖ, ὁ δὲ ΓΔ τὸν ΒΕ μετρεῖ, καὶ ὁ Η ἄρα τὸν ΒΕ μετρεῖ: μετρεῖ δὲ καὶ ὅλον τὸν ΒΑ: καὶ λοιπὸν ἄρα τὸν ΑΕ μετρήσει. ὁ δὲ ΑΕ τὸν ΔΖ μετρεῖ: καὶ ὁ Η ἄρα τὸν ΔΖ μετρήσει: μετρεῖ δὲ καὶ ὅλον τὸν ΔΓ: καὶ λοιπὸν ἄρα τὸν ΓΖ μετρήσει ὁ μείζων τὸν ἐλάσσονα: ὅπερ ἐστὶν ἀδύνατον: οὐκ ἄρα τοὺς ΑΒ, ΓΔ ἀριθμοὺς ἀριθμός τις μετρήσει μείζων ὢν τοῦ ΓΖ: ὁ ΓΖ ἄρα τῶν ΑΒ, ΓΔ μέγιστόν ἐστι κοινὸν μέτρον: [ὅπερ ἔδει δεῖξαι]. Πόρισμα Ἐκ δὴ τούτου φανερόν, ὅτι ἐὰν ἀριθμὸς δύο ἀριθμοὺς μετρῇ, καὶ τὸ μέγιστον αὐτῶν κοινὸν μέτρον μετρήσει: ὅπερ ἔδει δεῖξαι.

Σχήμα 1.5 7^{ο} βιβλίο των Στοιχείων του Ευκλείδη
όπου διατυπώνεται ο ευκλείδιος αλγόριθμος από την βιβλιοθήκη
του Clay Mathematics Institute http://claymath.org/

Η λέξη ***αλγόριθμος*** προέρχεται από το όνομα του άραβα μαθηματικού Abu Ja'far Mohammed ibn Musa al Khowarizmi (περίπου 780 μ.Χ. – 850 μ.Χ.), το οποίο σημαίνει ο Μωχάμετ, ο υιός του Μωϋσή από το Χαρίζμ που είναι η σημερινή πόλη Khiva του Ουζμπεκιστάν. Το βιβλίο του al Khowarizmi για το ινδικό σύστημα υπολογισμού «Κανόνες σύνθεσης και αναγωγές» διασώθηκε μόνο στα λατινικά όπου ξεκινάει με τον τίτλο "Dixit algorismi .." (όπως είπε ο al Khowarizmi) και είχε μεγάλη απήχηση στον

$12^{ο}$ αιώνα σε σημείο που ο υπολογισμός ονομάστηκε algorism και από εκεί προέκυψε ο όρος *algorithm* ή *αλγόριθμος*. Ένα άλλο γνωστό βιβλίο του ίδιου συγγραφέα είναι το "Al-kitab al muhtasar fi hisab al-jabr wa-l-muqabala" (το συνοπτικό βιβλίο των υπολογισμών μέσω της αποκατάστασης και της αναγωγής) όπου το "al-jabr" που σημαίνει αποκατάσταση "restoration", είναι η διαδικασία μεταφοράς μιας ποσότητας από την μια πλευρά της εξίσωσης στην άλλη με ταυτόχρονη αλλαγή πρόσημου π.χ. $x^2 = 2x - 3x^2 \Leftrightarrow x^2 + 3x^2 = 2x$ ενώ το "al-muqabala" που σημαίνει αναγωγή "reduction", είναι η διαδικασία της αναγωγής ομοίων όρων π.χ $4x^2 = 2x$. Η εργασία αυτή ονομάστηκε εν συντομία al-jabr και από αυτή προήλθε η ονομασία algebra ή ***άλγεβρα***.

Σχήμα 1.6 Σελίδα από την μετάφραση στα λατινικά του βιβλίου του al Khowarizmi που αρχίζει με τις λέξεις "Dixit algorismi .." (όπως είπε ο al Khowarizmi)

1.5 Τι είναι γλώσσα προγραμματισμού;

Η ανάγκη επικοινωνίας του ανθρώπου με τον Η/Υ σε μια κοινή γλώσσα ανέπτυξε τις διάφορες γλώσσες προγραμματισμού. *Γλώσσα προγραμματισμού* είναι ένα σύνολο γραμμάτων, αριθμών, λέξεων και συντομογραφικών μνημονικών σημείων που διέπονται από ειδικό συντακτικό και χρησιμοποιούνται στην ανάπτυξη αλγορίθμων στον υπολογιστή.

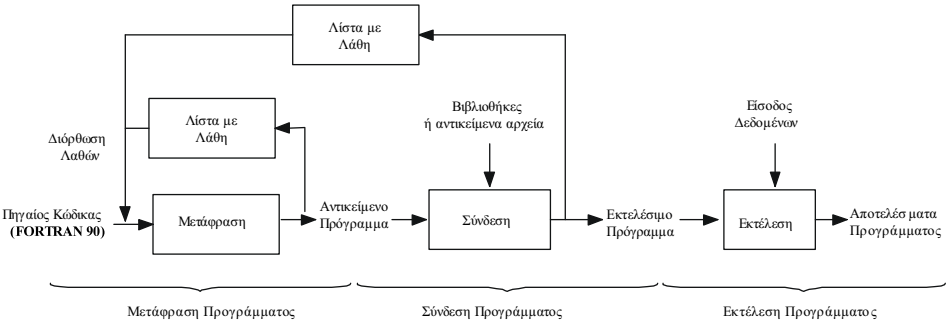
Η ποικιλία των γλωσσών προγραμματισμού οφείλεται στο ότι κάποιες γλώσσες υπερτερούν έναντι άλλων σε συγκεκριμένες εφαρμογές, ή προσφέρουν καλύτερη υποστήριξη, ή είναι ευκολότερες στην εκμάθηση τους. Γενικά, τις γλώσσες προγραμματισμού μπορούμε να τις χωρίσουμε σε δύο κατηγορίες:

Οι *γλώσσες χαμηλού επιπέδου* που βρίσκονται κοντά στις στοιχειώδεις εντολές μηχανής ενός συγκεκριμένου Η/Υ και γι' αυτό διαφέρουν από Η/Υ σε Η/Υ. Η γλώσσα που είναι πιο κοντά στο επίπεδο της μηχανής είναι η γλώσσα μηχανής. Η ανάγκη επίτευξης μια γλώσσας χαμηλού επιπέδου που να μοιάζει περισσότερο στη γλώσσα του ανθρώπου, χωρίς να ξεφεύγει από το επίπεδο της μηχανής, δημιούργησε τη γλώσσα Assembly. Υπάρχει μάλιστα ειδικό πρόγραμμα που ονομάζεται Assembler, το οποίο αναλαμβάνει να μεταφράσει τα προγράμματα που είναι γραμμένα στη γλώσσα αυτή σε γλώσσα μηχανής.

Οι *γλώσσες υψηλού επιπέδου* που προσεγγίζουν την ανθρώπινη γλώσσα περισσότερο από ότι οι γλώσσες χαμηλού επιπέδου και είναι ειδικά σχεδιασμένες, ώστε προγράμματα τα οποία είναι γραμμένα σ' αυτές να εκτελούνται από οποιονδήποτε υπολογιστή. Τέτοιες γλώσσες είναι οι BASIC, FORTRAN, PASCAL, PROLOG, C++, Java, κ.τ.λ. Το πρόγραμμα που γράφουμε σε μια γλώσσα υψηλού επιπέδου μεταφράζεται πάντοτε μέσω ειδικών προγραμμάτων στη γλώσσα που καταλαβαίνει ο Η/Υ, δηλαδή τη γλώσσα μηχανής. Τα προγράμματα που κάνουν τη μετάφραση αυτή είναι ο Interpreter ή ο Compiler.

Interpreter (Διερμηνέας)	Μεταφράζει μια-μια τις εντολές συγχρόνως με την εκτέλεση τους.
Compiler (Μεταφραστής)	Μεταφράζει μια φορά ολόκληρο το πρόγραμμα. Στη συνέχεια το μεταφρασμένο πρόγραμμα εκτελείται.

Η Fortran 90/95/2003, που θα αναλύσουμε παρακάτω, διαθέτει μεταφραστή για τη μετατροπή του προγράμματος σε γλώσσα μηχανής.



Σχήμα 1.7 Φάση υλοποίησης ενός προγράμματος.

Ένα πρόγραμμα γραμμένο σε μια γλώσσα υψηλού επιπέδου ονομάζεται συνήθως *πηγαίος κώδικας* (source code). Μετά τη μετατροπή του σε γλώσσα μηχανής από το μεταφραστή (compiler) δημιουργείται το *αντικείμενο πρόγραμμα* (object program). Το αντικείμενο πρόγραμμα αποθηκεύεται σε αρχείο που έχει το όνομα του αρχείου του πηγαίου κώδικα και επέκταση .obj. Το αντικείμενο πρόγραμμα στη συνέχεια συνδέεται με βιβλιοθήκες της γλώσσας προγραμματισμού και άλλα αντικείμενα προγράμματα προκειμένου να δημιουργηθεί ο εκτελέσιμος κώδικας ή αλλιώς το εκτελέσιμο πρόγραμμα (executable program). Το εκτελέσιμο πρόγραμμα αποθηκεύεται σε αρχείο που έχει το όνομα του αρχείου του πηγαίου κώδικα και επέκταση .exe. Το παραπάνω διάγραμμα δείχνει την διαδικασία υλοποίησης ενός προγράμματος.

Δραστηριότητα 1.1 Να περιγράψετε τη διαδικασία μετάφρασης και εκτέλεσης ενός προγράμματος. Ισχύει η ίδια διαδικασία για προγράμματα που είναι γραμμένα σε γλώσσα μηχανής;

1.6 Τι είναι προγραμματισμός;

Προγραμματισμός είναι μια διαδικασία που αποτελείται από καθορισμένα στάδια, σε καθένα από τα οποία γίνονται διάφορες ενέργειες με σκοπό το μετασχηματισμό του προβλήματος και της μεθόδου επίλυσης του σε μια μορφή που να είναι κατανοητή και αποδεκτή από τον Η/Υ.

Τα *στάδια του προγραμματισμού* είναι τα ακόλουθα:

Φάση Ανάλυσης (Analysis)

- Η αναγνώριση, ο ορισμός και ο καθορισμός των προδιαγραφών του προβλήματος.
- Εντοπισμός των χρήσιμων εννοιών και απλοποίηση ορισμένων στοιχείων.

Φάση Σχεδιασμού (Design)

- Σκιαγράφηση της λύσης και διάκριση ανεξαρτήτων διαδικασιών.
- Επιλογή και περιγραφή ενός *αλγόριθμου* που να είναι:
  α) *περατός*, να τερματίζει μετά από την εκτέλεση ενός πεπερασμένου αριθμού βημάτων,
  β) *σαφής*, κάθε βήμα να καθορίζεται χωρίς καμιά αμφιβολία για τον τρόπο εκτέλεσης του,
  γ) *γενικός*, να μπορεί δηλαδή να επιλύει ένα πρόβλημα στην πιο γενική περίπτωση. Σίγουρα ένας αλγόριθμος που υπολογίζει το εμβαδό ενός οποιουδήποτε τριγώνου γνωρίζοντας τα μήκη των πλευρών του είναι πιο καλός από

έναν παρόμοιο που λύνει το πρόβλημα μόνο για ορθογώνια τρίγωνα.

δ) *αποτελεσματικός*, να μπορεί δηλαδή κάθε βήμα του να εκτελεσθεί και από ένα άτομο με την χρήση χαρτιού και μολυβιού σε πεπερασμένο χρόνο,

ε) *αποδοτικός*, η έννοια της απόδοσης ενός αλγορίθμου σχετίζεται με το πλήθος των βημάτων που απαιτούνται για την ολοκλήρωση του (χρονική πολυπλοκότητα) καθώς και με τον όγκο των δεδομένων που απαιτεί για την εκτέλεση του (χωρική πολυπλοκότητα). Ας πάρουμε για παράδειγμα αλγορίθμους που έχουν ως στόχο την επίλυση συστημάτων διαφορικών εξισώσεων που οδηγούν σε μοντέλα πρόβλεψης καιρού. Ένας αλγόριθμος που θα υπολόγιζε την πρόγνωση του *αυριανού* καιρού σε παραπάνω από μια μέρα δεν θα ήταν καθόλου χρήσιμος.

στ) *ευσταθής*, αν δηλαδή υπάρχει ένα μικρό σφάλμα κατά την εισαγωγή των δεδομένων, να προσπαθεί να διατηρήσει το αντίστοιχο σφάλμα που θα προκύψει στα αποτελέσματα όσο το δυνατό πιο μικρό.

Φάση Υλοποίησης (Implementation)

- Επιλογή της γλώσσας προγραμματισμού βάσει της φύσης του προβλήματος, των γνώσεών μας σε γλώσσες προγραμματισμού και τις δυνατότητες ή περιορισμούς που μας θέτει ο Η/Υ. Κωδικοποίηση του αλγορίθμου σε πρόγραμμα.

Εκσφαλμάτωση (Debugging)

- Έλεγχος προγράμματος για ανίχνευση λαθών. Τα λάθη χωρίζονται σε τρεις κατηγορίες:

  α) *συντακτικά λάθη* (syntax errors), λάθη δηλαδή ως προς τον τρόπο σύνταξης του προγράμματος στην συγκεκριμένη γλώσσα προγραμματισμού π.χ. REED αντί για READ,

  β) *λογικά λάθη* (logical errors), λάθη που δεν είναι άμεσα αντιληπτά και συνήθως οφείλονται σε λανθασμένη απόδοση του αλγορίθμου ή σε περιορισμούς της γλώσσας ή του συστήματος και οδηγούν σε λάθη εκτέλεσης (run time errors) π.χ. X=–A/B χωρίς να ελέγξουμε αν $B \neq 0$ και

  γ) *λάθη εκτέλεσης*, λάθη που προκύπτουν κατά την διαδικασία ανάγνωσης/εγγραφής, χρήσης μη επιτρεπτών δεδομένων, χρήσης λανθασμένων ορισμάτων σε συναρτήσεις, αριθμητικών λαθών (π.χ. πολύ μεγάλοι αριθμοί) ή γενικά σε λάθη του συστήματος.

- Εφαρμογή του προγράμματος σε ποικίλα δεδομένα για να διαπιστωθούν τυχόν λάθη.

Τεκμηρίωση

- Το πρόγραμμα είναι φρόνιμο να περιέχει πάντα σχόλια όπου θα επεξηγούνται τα επιμέρους στάδια του (*εσωτερική τεκμηρίωση*). Θα πρέπει επίσης να συντάσσεται ένας φάκελος που να περιέχει:

α) το πρόγραμμα με την λειτουργική δομή του προγράμματος,
β) τον αλγόριθμο που επιλέχτηκε,
γ) τις τεχνικές που χρησιμοποιήθηκαν για την υλοποίηση του αλγορίθμου,
δ) τα πλεονεκτήματα και μειονεκτήματα του συγκεκριμένου αλγόριθμου,
ε) αποτελέσματα δοκιμών,
στ) τον κώδικα,
ζ) αναλυτική επεξήγηση του κάθε τμήματος του αλγόριθμου κ.α. (*εξωτερική τεκμηρίωση*).

Οι οδηγίες αυτές θα βοηθήσουν τον προγραμματιστή (ή και τρίτους) όταν το ξαναδιαβάσει μετά από κάποιο ορισμένο χρονικό διάστημα προκειμένου να το διορθώσει όπως και να δανειστεί τεχνικές που χρησιμοποίησε στο συγκεκριμένο πρόγραμμα για την επίλυση παρόμοιων προβλημάτων.

Συντήρηση

- Το πρόγραμμα θα πρέπει να συντηρείται ανά τακτά χρονικά διαστήματα με την διόρθωση απρόβλεπτων λαθών που έχουν προκύψει αλλά και με την τροποποίηση-βελτίωση του λαμβάνοντας υπόψη νέα δεδομένα και νέες εξελίξεις στον τομέα που το αφορά.

Παρακάτω δίνουμε ένα παράδειγμα όπου εφαρμόζουμε τα τρία πρώτα στάδια του προγραμματισμού.

Παράδειγμα. Να υπολογιστεί το εμβαδόν ενός τριγώνου ΑΒΓ.

Φάση Ανάλυσης

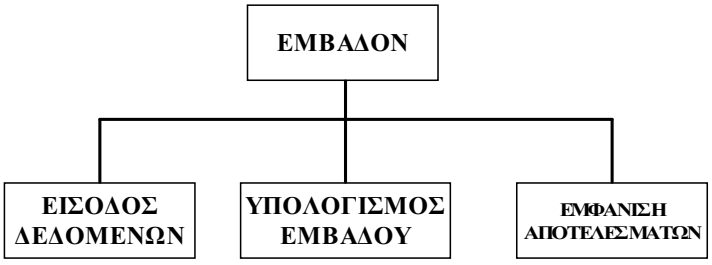
- Διαπιστώνουμε ότι το παραπάνω πρόβλημα δεν είναι καλά ορισμένο, διότι θα μπορούσαμε να υπολογίσουμε το εμβαδόν του τριγώνου μόνο αν γνωρίζαμε τις 3 πλευρές του, ή τις 2 πλευρές και μια γωνία του, κ.ο.κ.. Επαναδιατυπώνουμε λοιπόν το πρόβλημα με σαφήνεια.
  «Να υπολογιστεί το εμβαδόν ενός τριγώνου ΑΒΓ εάν είναι γνωστά τα μήκη των πλευρών του ΑΒ, ΒΓ, ΓΑ.»
- Καθορίζουμε την *είσοδο* στο πρόβλημά μας, που είναι τα μήκη των τριών πλευρών, έστω a,b,c και την *έξοδο* που θα είναι το εμβαδόν του τριγώνου, έστω Ε.

Φάση Σχεδιασμού

- Έχουμε 3 ανεξάρτητες διαδικασίες:

  $1^η$ διαδικασία. Είσοδος των δεδομένων.

2η διαδικασία. Υπολογισμός του εμβαδού μέσω του τύπου του Ήρωνα.
3η διαδικασία. Έξοδος των αποτελεσμάτων.



- Οι παραπάνω 3 διαδικασίες μπορούν να παρασταθούν μέσω του παρακάτω ψευδοκώδικα:

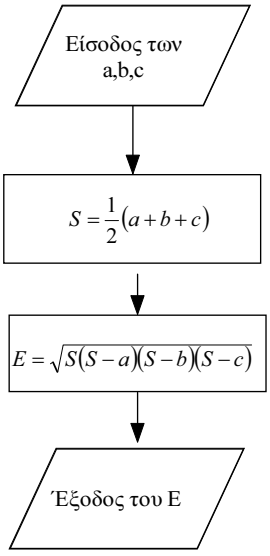
Ψευδοκώδικας[*]
Βήμα 1ο. Διάβασε a,b,c.

Βήμα 2ο. $S \leftarrow \frac{1}{2}(a+b+c)$

Βήμα 3ο. $E \leftarrow \sqrt{S(S-a)(S-b)(S-c)}$

Βήμα 4ο. Αποτελέσματα // E //.

ή του διπλανού λογικού διαγράμματος[†] ή διαγράμματος ροής (δες Παράρτημα Α, Λογικό Διάγραμμα, σελ. 365)



Φάση Υλοποίησης

- Το πρόγραμμα αυτό είναι πολύ απλό και μπορεί να υλοποιηθεί σε οποιαδήποτε γλώσσα προγραμματισμού, χωρίς κάποια ιδιαίτερα πλεονεκτήματα έναντι κάποιας άλλης γλώσσας. Παρακάτω το πρόγραμμα υλοποιείται στη FORTRAN 90.

---

[*] Ο ψευδοκώδικας είναι μια περιγραφή του τρόπου επίλυσης ενός προβλήματος με σύντομες και περιεκτικές προτάσεις που ακολουθούν αντίστοιχη τυποποίηση με αυτή μιας γλώσσας προγραμματισμού. Παρ' όλα αυτά οι προτάσεις αυτές είναι άμεσα αντιληπτές από τον άνθρωπο και δεν υπεισέρχονται σε λεπτομέρειες.

[†] Σχηματική αναπαράσταση του αλγορίθμου επίλυσης ενός προβλήματος με χρήση γεωμετρικών σχημάτων τα οποία συνδέονται με βέλη τα οποία και καθορίζουν την ροή του προγράμματος.

```
PROGRAM EMBADON
      IMPLICIT NONE
      ! Variables
       REAL :: A,B,C,S,E
      ! Body of EMBADON
      PRINT*,"A,B,C="
      READ*, A,B,C
      S=(1/2.0)*(A+B+C)
      E=SQRT(S*(S−A)*(S−B)*(S−C))
      PRINT*,'-----------'
      PRINT*,A,B,C
      PRINT*,'E=',E
 END PROGRAM EMBADON
```

Στη γραμμή 1 και 13 δηλώνεται η αρχή και το τέλος του προγράμματος. Στη γραμμή 4 δηλώνουμε ότι οι μεταβλητές που χρησιμοποιούμε στο πρόγραμμα θα πρέπει να δέχονται πραγματικές τιμές. Στη γραμμή 7 δεχόμαστε τιμές για τις μεταβλητές A,B,C από το χρήστη. Στις γραμμές 8 και 9 υπολογίζουμε το εμβαδόν E, ενώ στις γραμμές 11-12 εκτυπώνουμε τα αποτελέσματα.

- Μεταφράζουμε το πρόγραμμα και εντοπίζουμε τα συντακτικά λάθη, και στη συνέχεια εκτελούμε το πρόγραμμα για να βρούμε τυχόν λογικά λάθη ή λάθη εκτέλεσης (*εκσφαλμάτωση*). Πράγματι, διαπιστώνουμε λάθη όταν οι τιμές των A,B,C δεν αποτελούν πλευρές τριγώνου. Στην περίπτωση αυτή, η υπόριζη ποσότητα είναι αρνητική και μας οδηγεί σε λάθη εκτέλεσης. Θα έπρεπε δηλαδή να ελέγξουμε:

  α) αν τα μήκη των πλευρών είναι θετικά (A>0 & B>0 & C>0) και

  β) αν το άθροισμα των μηκών κάθε δύο πλευρών είναι μεγαλύτερο του μήκους της τρίτης πλευράς (A+B>C, B+C>A, C+A>B).

  Σε παρακάτω ενότητες θα εξετάσουμε την πλήρη επίλυση του προβλήματος.

1.7 Πως μετράμε την απόδοση ενός αλγορίθμου;

Για την μέτρηση της απόδοσης ενός αλγορίθμου υπάρχουν δύο τρόποι:

α) Ο ***εμπειρικός τρόπος*** κατά τον οποίο υπολογίζεται ο χρόνος εκτέλεσης και η χωρητικότητα μνήμης του αλγόριθμου που απαιτούνται για την υλοποίηση του για ένα

σύνολο δεδομένων. Η μέθοδος αυτή έχει το μειονέκτημα ότι δίνει διαφορετικά αποτελέσματα σε διαφορετικούς υπολογιστές ή ακόμα και στο ίδιο μηχάνημα (αν έχει διαφορετική δομή). Επίσης εξαρτάται από την γλώσσα προγραμματισμού που χρησιμοποιήθηκε αλλά και το σύνολο των δεδομένων που μπορούν να διαφέρουν από περίπτωση σε περίπτωση. Θα μπορούσαμε λοιπόν να εκτελέσουμε δύο αλγόριθμους στον ίδιο υπολογιστή με τα ίδια δεδομένα και να συγκρίνουμε τους χρόνους εκτέλεσης.

β) Ο ***θεωρητικός τρόπος*** όπου η μέτρηση της αποδοτικότητας εξαρτάται από το μέγεθος της εισόδου το οποίο προσδιορίζεται από μια μεταβλητή *n*. Για παράδειγμα σε ένα πρόβλημα αντιστροφής ενός τετραγωνικού πίνακα $k \times k$ θα έχουμε $n = k$, σε ένα πρόβλημα ταξινόμησης k αριθμών θα έχουμε $n = k$ ή τέλος σε ένα πρόβλημα πολλαπλασιασμού δύο αριθμών, των οποίων η αναπαράσταση απαιτεί k, m bits αντίστοιχα, θα έχουμε $n = k + m$. Αυτό που μας ενδιαφέρει λοιπόν είναι η επίδραση που θα έχει η αύξηση του μεγέθους των εισόδων στην ταχύτητα εκτέλεσης του αλγορίθμου. Ποιά συνέπεια θα έχει για παράδειγμα ο διπλασιασμός του μεγέθους εισόδου στον χρόνο εκτέλεσης του αλγορίθμου; Θα διπλασιαστεί (γραμμική εξάρτηση); Θα τετραπλασιαστεί (τετραγωνική εξάρτηση); Ας συμβολίσουμε με την συνάρτηση *f*(*n*) τον χρόνο εκτέλεσης (**χρονική πολυπλοκότητα** ή time complexity) (ή την χωρητικότητα μνήμης (**χωρική πολυπλοκότητα** ή space complexity)) σε σχέση με το μέγεθος της εισόδου *n*. Ο χρόνος εκτέλεσης *f*(*n*) εξαρτάται:

1) από τον *αριθμό των στοιχειωδών βημάτων* που πρέπει να εκτελεστούν σε έναν αλγόριθμο. Τα στοιχειώδη αυτά βήματα που λαμβάνουμε υπόψη διαφέρουν από αλγόριθμο σε αλγόριθμο. Σε ένα πρόβλημα ταξινόμησης για παράδειγμα μας ενδιαφέρει ο αριθμός των συγκρίσεων μεταξύ φυσικών αριθμών ή/και οι αντιμεταθέσεις που θα χρειαστούν. Σε ένα πρόβλημα αντιστροφής πίνακα μας ενδιαφέρουν οι πολλαπλασιασμοί και οι διαιρέσεις που θα εκτελεσθούν.

2) από την *δομή των δεδομένων*. Για παράδειγμα ο χρόνος εκτέλεσης *f*(*n*) που απαιτείται για την ταξινόμηση σε αύξουσα σειρά ενός συνόλου *n* δεδομένων που προκύπτουν από ένα σύνολο δεδομένων ίδιου μεγέθους έστω $D = \{d_1, d_2, ..., d_n\}$ που είναι ήδη ταξινομημένα (*πολυπλοκότητα καλύτερης περίπτωσης* $\min(f(d)), d \in D$), είναι διαφορετικός από τον χρόνο που θα χρειαστεί για ένα σύνολο δεδομένων που είναι ήδη ταξινομημένα σε φθίνουσα σειρά (*πολυπλοκότητα χειρότερης περίπτωσης* $\max(f(d)), d \in D$). Η ανάλυση της χειρότερης περίπτωσης μας δίνει ένα άνω φράγμα για την απόδοση του αλγορίθμου, ενώ της καλύτερης περίπτωσης ένα κάτω φράγμα. Εκτός της πολυπλοκότητας καλύτερης και χειρότερης περίπτωσης, έχουμε και την μέση πολυπλοκότητα που ορίζεται ως

$\frac{\sum_{d\in D} f(d)}{|D|}$ που είναι ο μέσος όρος των πολυπλοκοτήτων για όλα τα δεδομένα του ίδιου μεγέθους και συνήθως δύσκολα υπολογίζεται λόγω του μεγάλου αριθμού των περιπτώσεων.

Αυτό που συνήθως μας ενδιαφέρει είναι η εκτίμηση της συμπεριφοράς του αλγορίθμου καθώς ο αριθμός των δεδομένων αυξάνει ($n \to \infty$) ή διαφορετικά η ασυμπτωτική εκτίμηση του χρόνου εκτέλεσης $f(n)$ (κυρίως για την χειρότερη περίπτωση) και όχι η αναλυτική μορφή του $f(n)$ προκειμένου να συγκρίνουμε διαφορετικούς αλγορίθμους. Για τον σκοπό αυτό δίνουμε τους παρακάτω συμβολισμούς για τον ακριβή προσδιορισμό της τάξης μεγέθους της συνάρτησης $f(n)$.

Ορισμός 1.1 [Knuth, 1976] Ορίζουμε ως $\Theta(g(n))$ το σύνολο των συναρτήσεων

$$\Theta(g(n)) = \{f(n) : \exists c_1, c_2 \in \mathbb{R}^+, n_0 \in \mathbb{N}, \quad 0 \le c_1 g(n) \le f(n) \le c_2 g(n) \ \ \forall n \ge n_0\}$$

Συνεπώς $f(n) \in \Theta(g(n))$ αν και μόνο αν **υπάρχουν** θετικές σταθερές c_1, c_2 τέτοιες ώστε η γραφική παράσταση της $f(n)$ να βρίσκεται μεταξύ των $c_1 g(n), c_2 g(n)$ για αρκετά μεγάλα n. Στην περίπτωση αυτή αντί να γράψουμε $f(n) \in \Theta(g(n))$ συνήθως γράφουμε $f(n) = \Theta(g(n))$. Συνεπώς ο συμβολισμός Θ χρησιμοποιείται όταν θέλουμε να δηλώσουμε ότι έχουμε ένα **ασυμπτωτικά άνω και κάτω φράγμα** για την συνάρτηση $f(n)$. Αν $f(n) = \Theta(g(n))$ τότε θα έχουμε $\lim_{n\to+\infty} \frac{f(n)}{g(n)} = c, c \in \mathbb{R}^+$ (εφόσον υπάρχει το όριο) που δηλώνει ότι η συνάρτηση $f(n)$ αυξάνει με ανάλογο ρυθμό με την συνάρτηση $g(n)$. Αν μάλιστα $\lim_{n\to+\infty} \frac{f(n)}{g(n)} = 1$ τότε η συνάρτηση $f(n)$ αυξάνει με τον ίδιο ρυθμό με την συνάρτηση $g(n)$ και γράφουμε $f(n) \approx g(n)$.

Ορισμός 1.2 [P. Bachman, 1892]. Ορίζουμε ως $O(g(n))$ το σύνολο των συναρτήσεων

$$O(g(n)) = \{f(n) : \exists c \in \mathbb{R}^+, n_0 \in \mathbb{N}, \quad 0 \le f(n) \le c g(n) \ \ \forall n \ge n_0\}$$

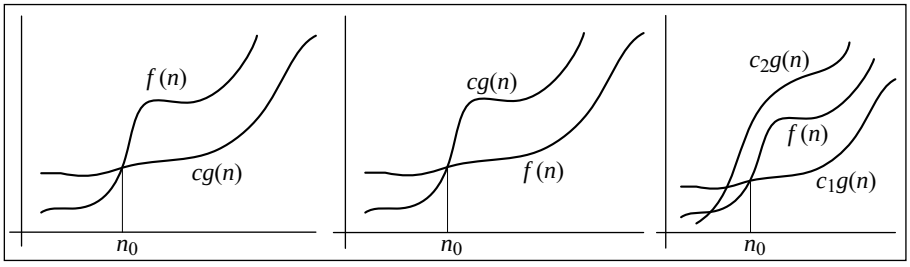
Συνεπώς $f(n) \in O(g(n))$ αν και μόνο αν ***υπάρχει*** θετική σταθερά c τέτοια ώστε η γραφική παράσταση της $f(n)$ να βρίσκεται κάτω από την γραφική παράσταση της

$cg(n)$ για αρκετά μεγάλα n. Στην περίπτωση αυτή αντί να γράψουμε $f(n) \in O(g(n))$ συνήθως γράφουμε $f(n) = O(g(n))$. Ο συμβολισμός O χρησιμοποιείται όταν θέλουμε να δηλώσουμε ότι έχουμε ένα **ασυμπτωτικά άνω φράγμα** για την συνάρτηση $f(n)$. Μάλιστα αν $f(n) = O(g(n))$ τότε θα έχουμε $\lim_{n\to+\infty} \frac{f(n)}{g(n)} = c, c \in \mathbb{R}^+ \cup \{0\}$ (εφόσον υπάρχει το όριο) που δηλώνει ότι η συνάρτηση $f(n)$ αυξάνει με πιο αργό ή το πολύ ίσο ρυθμό σε σχέση με την συνάρτηση $g(n)$.

Ορισμός 1.3 [Knuth, 1976] Ορίζουμε ως $\Omega(g(n))$ το σύνολο των συναρτήσεων

$$\Omega(g(n)) = \{f(n) : \exists c \in \mathbb{R}^+, n_0 \in \mathbb{N}, \quad 0 \le cg(n) \le f(n) \quad \forall n \ge n_0\}$$

Συνεπώς $f(n) \in \Omega(g(n))$ αν και μόνο αν ***υπάρχει*** θετική σταθερά c τέτοια ώστε η γραφική παράσταση της $f(n)$ να βρίσκεται πάνω από την γραφική παράσταση της $cg(n)$ για αρκετά μεγάλα n. Στην περίπτωση αυτή αντί να γράψουμε $f(n) \in \Omega(g(n))$ συνήθως γράφουμε $f(n) = \Omega(g(n))$. Ο συμβολισμός Ω χρησιμοποιείται όταν θέλουμε να δηλώσουμε ότι έχουμε ένα **ασυμπτωτικά κάτω φράγμα** για την συνάρτηση $f(n)$. Αν $f(n) = \Omega(g(n))$ τότε θα έχουμε $\lim_{n\to+\infty} \frac{f(n)}{g(n)} = c, c \in \mathbb{R}^+ \cup \{\infty\}$ (εφόσον υπάρχει το όριο) που δηλώνει ότι η συνάρτηση $f(n)$ αυξάνει με το λιγότερο ίσο ή πιο γρήγορο ρυθμό σε σχέση με την συνάρτηση $g(n)$.



$f(n) = \Omega(g(n))$ $\quad$ $f(n) = O(g(n))$ $\quad$ $f(n) = \Theta(g(n))$

Αν για παράδειγμα ο χρόνος εκτέλεσης είναι $f(n) = \Theta(n^3)$ (κυβική πολυπλοκότητα) αυτό σημαίνει ότι αν διπλασιάσω το μέγεθος του προβλήματος τότε απαιτείται $((2n)^3 = 8n^3)$ οκταπλάσιος χρόνος για την εκτέλεση του αλγορίθμου, αν τριπλασιάσω

το μέγεθος του προβλήματος τότε απαιτείται ($(3n)^3 = 27n^3$) εικοσιεπταπλάσιος κ.ο.κ.. Αλγόριθμοι των οποίων ο χρόνος εκτέλεσης τους είναι της μορφής $\Theta(n^k), k \in \mathbb{R}$ ονομάζονται **πολυωνυμικοί** και συνήθως δεν απαιτούν τόσο μεγάλη υπολογιστική προσπάθεια σε σχέση με αλγορίθμους που έχουν πολυπλοκότητα $\Theta(n^n), \Theta(n!), \Theta(2^n)$ και οι οποίοι ονομάζονται **μη πολυωνυμικοί**. Στον παρακάτω πίνακα δίνεται ο χρόνος εκτέλεσης (σε ns) ενός αλγορίθμου σε σχέση με την πολυπλοκότητα του αλγορίθμου αν υποθέσουμε ότι κάθε στοιχειώδης πράξη απαιτεί 1ns της CPU του υπολογιστή μας.

Πολυπλοκότητα αλγορίθμου	**Μέγεθος προβλήματος (n)**		
	10	30	50
$\Theta(n)$	0.00001	0.00003	0.00005
$\Theta(n\log_2 n)$	3.0103×10^{-6}	6.11385×10^{-6}	8.85919×10^{-6}
$\Theta(n^2)$	0.0001	0.0009	0.0025
$\Theta(n^3)$	0.001	0.027	0.125
$\Theta(2^n)$	0.001024	1073.74	1.1259×10^{9}
$\Theta(n!)$	3.6288	2.65253×10^{26}	3.04141×10^{58}

Μπορούμε να παρατηρήσουμε ότι $3n^2 = O(n^2)$ αλλά και $3n = O(n^2)$. Το φράγμα στην πρώτη περίπτωση είναι ασυμπτωτικά πολύ στενό ενώ στην δεύτερη περίπτωση δεν είναι. Για να διαχωρίσουμε τις δύο αυτές περιπτώσεις δίνουμε τον συμβολισμό $o(g(n))$.

Ορισμός 1.4 [E. Landau, 1909] Ορίζουμε ως $o(g(n))$ το σύνολο των συναρτήσεων

$$o(g(n)) = \{f(n) : \forall c \in \mathbb{R}^+, \exists n_0 \in \mathbb{N}, \quad 0 \le f(n) \le cg(n) \quad \forall n \ge n_0\}$$

Συνεπώς $f(n) \in o(g(n))$ αν και μόνο αν ***για κάθε*** θετική σταθερά c η γραφική παράσταση της $f(n)$ βρίσκεται κάτω από την γραφική παράσταση της $cg(n)$ για αρκετά μεγάλα n. Στην περίπτωση αυτή γράφουμε $f(n) = o(g(n))$.

Η διαφορά δηλαδή των συμβολισμών $O(g(n))$ και $o(g(n))$ είναι ότι στην πρώτη περίπτωση ***υπάρχει*** κάποια θετική σταθερά c για την οποία ισχύει η ανισότητα $0 \le f(n) \le cg(n)$ ενώ στην δεύτερη περίπτωση η ανισότητα ισχύει για κάθε θετική σταθερά c. Μάλιστα αν $f(n) = o(g(n))$ τότε θα έχουμε $\lim_{n\to+\infty} \frac{f(n)}{g(n)} = 0$ πρδ. $3n = o(n^2)$ μιας και έχουμε $\lim_{n\to+\infty} \frac{3n}{n^2} = 0$, αλλά $3n^2 \notin o(n^2)$ εφόσον

$\lim_{n\to+\infty}\frac{3n^2}{n^2}=3\neq 0$. Αυτό σημαίνει ότι η συνάρτηση $g(n)$ μεγαλώνει με πιο γοργούς ρυθμούς σε σχέση με την $f(n)$ καθώς το n πλησιάζει στο άπειρο. Παρόμοια με τον συμβολισμό $o(g(n))$ έχουμε τον συμβολισμό $\omega(g(n))$.

Ορισμός 1.5 Ορίζουμε ως $\omega(g(n))$ το σύνολο των συναρτήσεων

$$\omega(g(n))=\{f(n):\forall c\in\mathbb{R}^+,\exists n_0\in\mathbb{N},\quad 0\le cg(n)\le f(n)\quad \forall n\ge n_0\}$$

Συνεπώς $f(n)\in\omega(g(n))$ αν και μόνο αν ***για κάθε*** θετική σταθερά c η γραφική παράσταση της $f(n)$ βρίσκεται πάνω από την γραφική παράσταση της $cg(n)$ για αρκετά μεγάλα n. Στην περίπτωση αυτή γράφουμε $f(n)=\omega(g(n))$.

Ισχύει ότι $f(n)=\omega(g(n))$ αν και μόνο αν $g(n)=o(f(n))$. Μάλιστα αν $f(n)=\omega(g(n))$ τότε θα έχουμε $\lim_{n\to+\infty}\frac{f(n)}{g(n)}=\infty$ αν υπάρχει το όριο. Αυτό σημαίνει ότι η συνάρτηση $f(n)$ μεγαλώνει με πιο γοργούς ρυθμούς σε σχέση με την $g(n)$ καθώς το n πλησιάζει στο άπειρο.

Στόχος του βιβλίου αυτού *δεν είναι* να ασχοληθεί με την πολυπλοκότητα των αλγορίθμων που θα παρουσιάσουμε, αλλά να δώσει ένα έναυσμα στον αναγνώστη, μέσω απλών παραδειγμάτων, για περαιτέρω μελέτη της περιοχής της Πληροφορικής που ασχολείται με την θεωρία των αλγορίθμων και της πολυπλοκότητας.

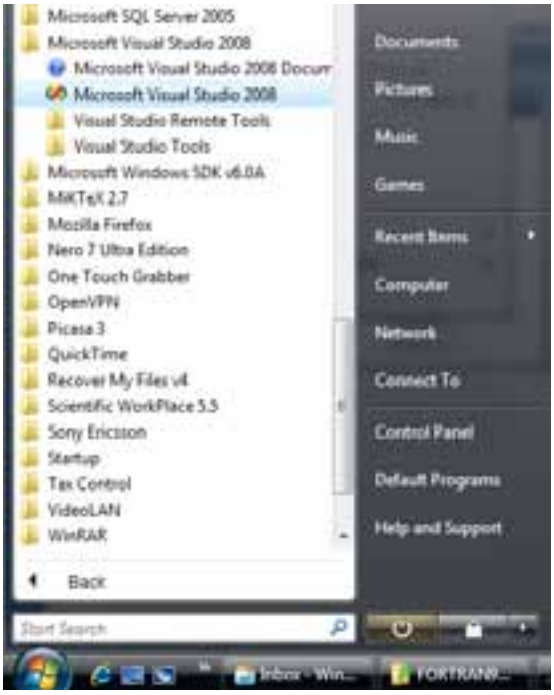
1.8 Πώς θα μεταφράσουμε και θα εκτελέσουμε ένα πρόγραμμα στην Intel Visual Fortran;

Παρακάτω δίνουμε όλα τα βήματα που πρέπει να κάνετε για να εισάγετε ένα πρόγραμμα που έχετε γράψει στο περιβάλλον της Intel Visual Fortran*, να το μεταφράσετε και τέλος να το εκτελέσετε.

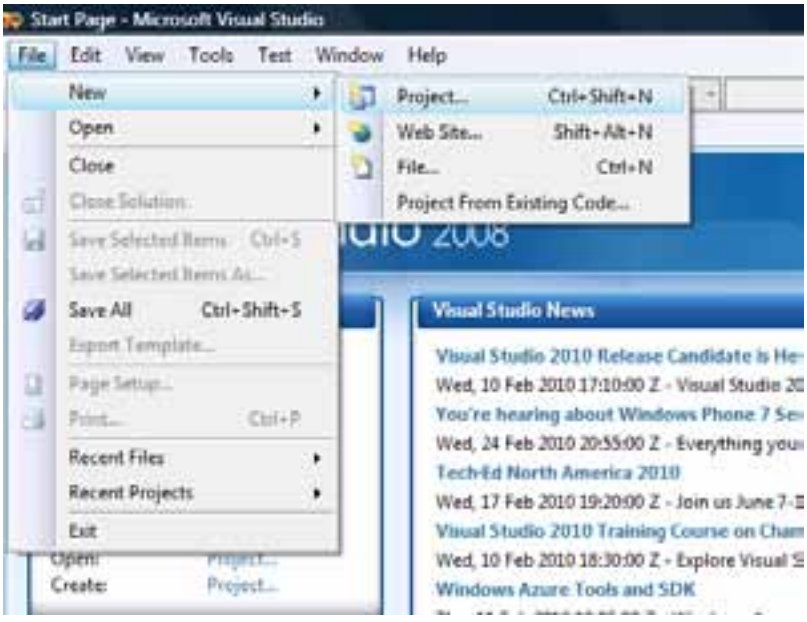
Βήμα 1. Εκκίνηση προγράμματος

Από την επιλογή *Έναρξη* (*Start*) διαλέγουμε *Προγράμματα* (*All Programs*) στη συνέχεια *Microsoft Visual Studio 2008* και τέλος *Microsoft Visual Studio 2008.*

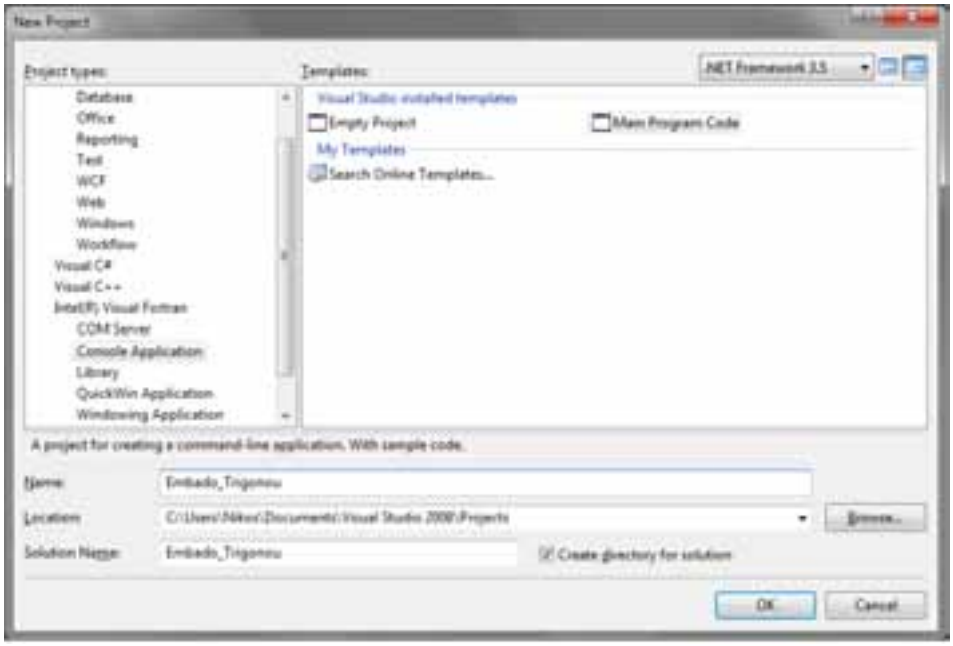
* Στο Παράρτημα Γ, θα βρείτε οδηγίες για τον τρόπο δημιουργίας ενός προγράμματος στο περιβάλλον της Compaq Visual Fortran.



Βήμα 2. Από την επιλογή *File* διαλέγουμε *New* και στη συνέχεια *Project*.



Στην επιλογή *Project Types* επιλέγουμε ***Intel(R) Visual Fortran***. Στη συνέχεια επιλέγουμε *Console Application* για την δημιουργία ενός τυπικού προγράμματος στην Fortran. Θα πρέπει να επιλέξουμε μεταξύ *Empty Project* (κενό project όπου θα πρέπει να ενσωματώσουμε τα αρχεία που θέλουμε) και *Main Program Code* (δημιουργείται αρχείο το οποίο έχει ήδη την δομή ενός προγράμματος στην Fortran και στο οποίο θα ενσωματώσουμε το πρόγραμμα μας).

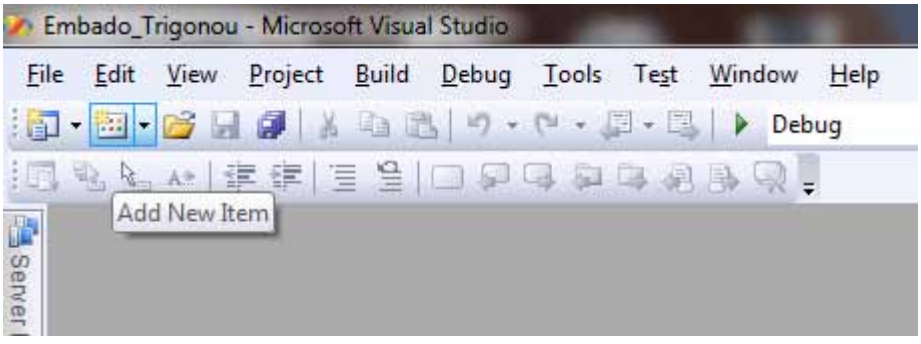


Στο πλαίσιο *Name* συμπληρώνουμε το όνομα του *Project Workspace* (π.χ. Embado_Trigonou) ενώ στο *Location* συμπληρώνουμε τον κατάλογο και την δευτερεύουσα μονάδα μνήμης που θέλουμε να δημιουργηθεί το *Project Workspace.* Τέλος κάνουμε κλικ στο OK. Το *Project Workspace* δημιουργείται για να τοποθετήσουμε το σύνολο των αρχείων που θα χρειαστούμε στο πρόγραμμα μας. Αν θέλουμε οι εφαρμογές μας να περιέχουν και γραφικά ή να εκτελούνται σε παράθυρο των Windows θα πρέπει να επιλέξουμε μια διαφορετική κατηγορία (QuickWin Applications, Windowing Applications). Στις περιπτώσεις αυτές το Project Workspace θα αποτελείται από περισσότερα από ένα αρχεία, όπως το κυρίως πρόγραμμα και τα αρχεία που περιέχουν τις εξωτερικές συναρτήσεις/διαδικασίες/modules.

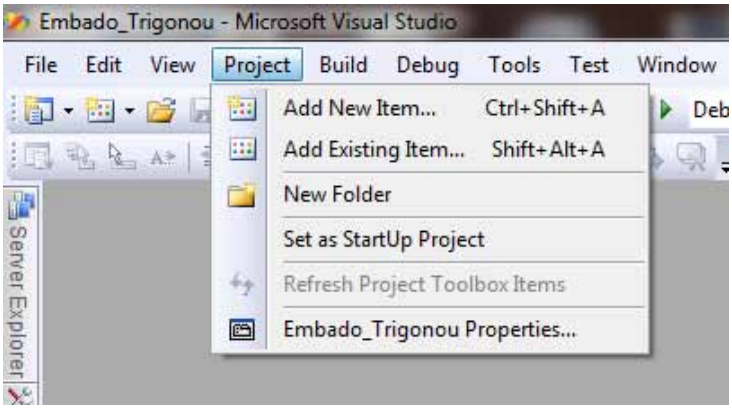
Αν επιλέξουμε Main Program Code εμφανίζεται αρχείο το οποίο έχει ήδη την δομή ενός προγράμματος στην Fortran όπως φαίνεται παρακάτω:

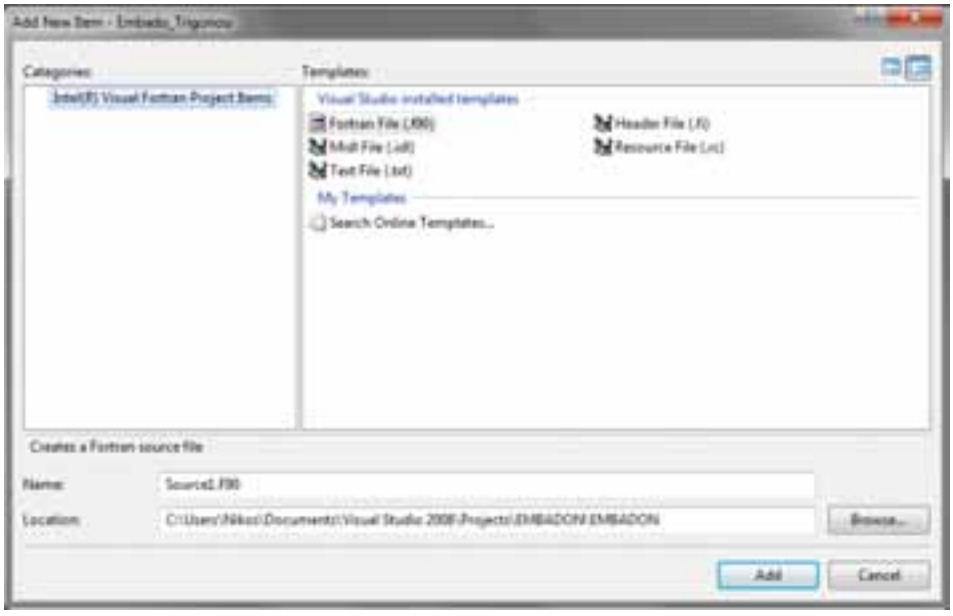
```
Embado_Trigonou.f90
!  Embado_Trigonou.f90
!
!  FUNCTIONS:
!  Embado_Trigonou - Entry point of console application.
!

!*****************************************************************************
!
!  PROGRAM: Embado_Trigonou
!
!  PURPOSE:  Entry point for the console application.
!
!*****************************************************************************

    program Embado_Trigonou

    implicit none

    ! Variables

    ! Body of Embado_Trigonou
    print *, 'Hello World'

    end program Embado_Trigonou
```

Αν επιλέξουμε *Empty Project* δημιουργείτε ένα κενό project όπως στην παρακάτω εικόνα:



Στη συνέχεια πρέπει να προσθέσουμε ένα αρχείο για να γράψουμε τον κώδικα του προγράμματος. Από το menu επιλέγουμε *Project* και στη συνέχεια *Add New Item*. Στον παρακάτω διάλογο που θα εμφανιστεί:

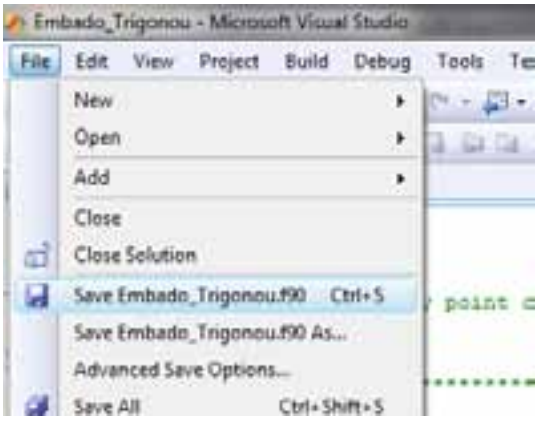




Επιλέγουμε Fortran File (.f90) για να προστεθεί ένα νέο αρχείο όπου θα γραφεί ο κώδικας μας και εφόσον συμπληρώσουμε το όνομα του αρχείου στην επιλογή *Name* κάνουμε κλίκ στο *Add*. Αν τώρα θέλουμε να επιλέξουμε ένα αρχείο που ήδη υπάρχει στο project μας θα πρέπει να ακολουθήσουμε την πορεία *Project->Add Existing Item* και στη συνέχεια προσθέτουμε το αρχείο ή τα αρχεία που θέλουμε να προσθέσουμε.

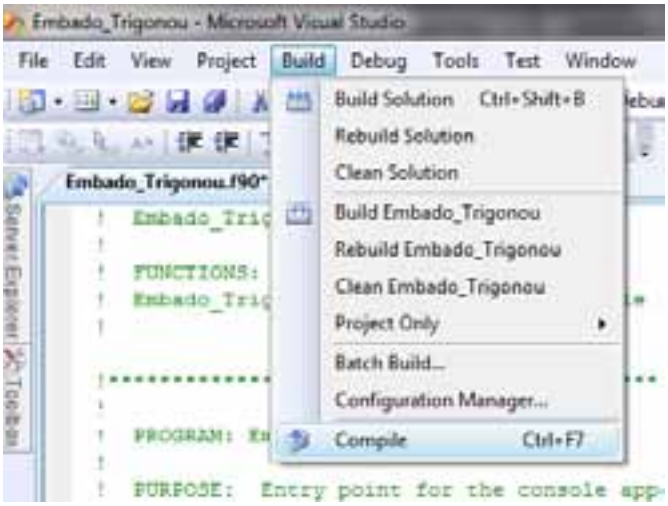
Βήμα 3. Θα γράψουμε το πρόγραμμα μας, τοποθετώντας κάτω από την γραμμή «!Variables» τις μεταβλητές μας και κάτω από την γραμμή «!Body of Embado_TrigonouS=(1/2.0)*» το κυρίως πρόγραμμα

Βήμα 4. Αποθηκεύουμε το πρόγραμμά μας. Από την επιλογή File διαλέγουμε Save



Βήμα 5. Μετάφραση του προγράμματος

Από την επιλογή *Build* επιλέγουμε *Compile* (ή Ctrl+F7 ή το αντίστοιχο εικονίδιο απο την μπάρα με τα εικονίδια)



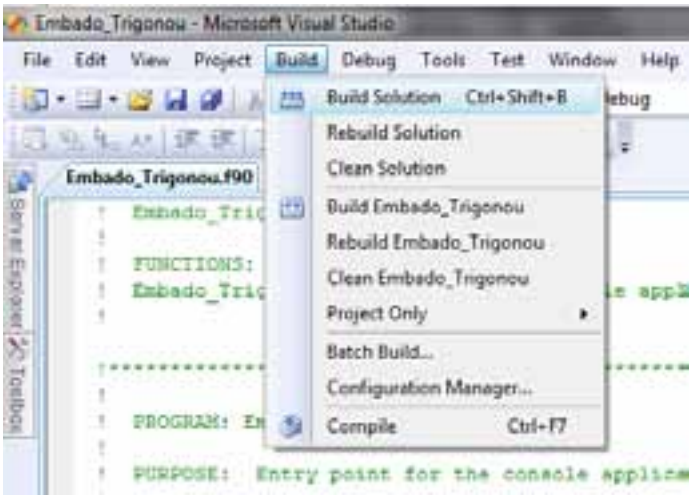
Αν όλα πάνε καλά, δηλαδή δεν βρεθεί κάποιο συντακτικό λάθος τότε θα εμφανισθεί στο τέλος της οθόνης μας το παρακάτω παράθυρο:

το οποίο δηλώνει ότι δε βρέθηκε κανένα συντακτικό λάθος.

Σε περίπτωση που υπήρχαν λάθη, θα έπρεπε να τα εντοπίσουμε από τα μηνύματα του παραπάνω παράθυρου, να τα διορθώσουμε και να προσπαθήσουμε να ξαναμεταφράσουμε το πρόγραμμά μας. Αν παραλείπαμε το * στην 24η γραμμή του προγράμματος (S=(1/2.0)(A+B+C)) και προσπαθούσαμε να κάνουμε compile θα είχαμε το παρακάτω μήνυμα

Βήμα 6. Δημιουργία εκτελέσιμου προγράμματος

Εφόσον έχει μεταφρασθεί χωρίς λάθη το πρόγραμμά μας, διαλέγουμε απο το Build την επιλογή Build Solution (ή Ctrl+Shift+B ή το αντίστοιχο εικονίδιο του BUILD απο την μπάρα με τα εικονίδια)

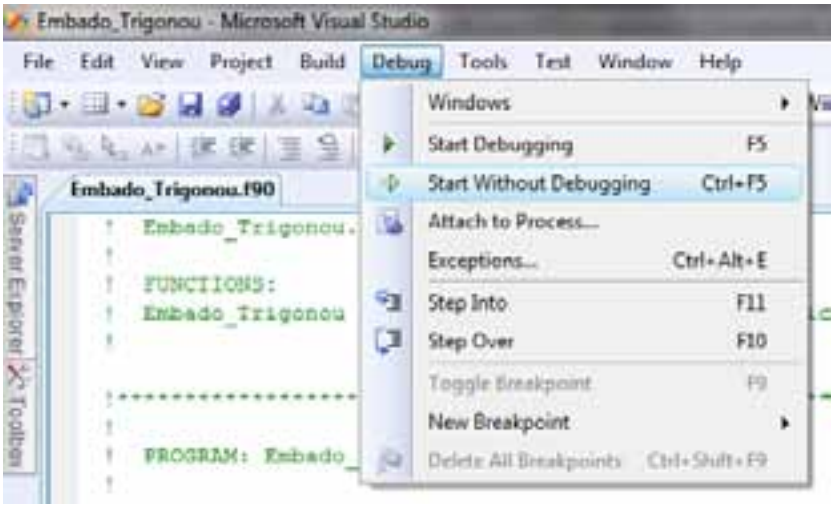


Αν όλα πάνε καλά, χωρίς λάθη έχουμε το παρακάτω μήνυμα:

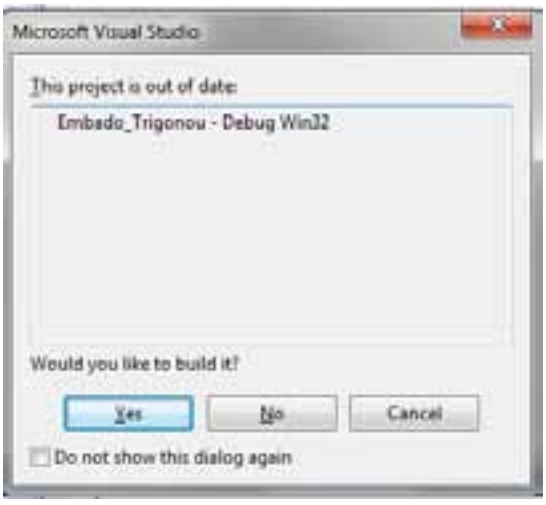
Αν έχουμε λάθη, τα διορθώνουμε και ξαναεκτελούμε τα τελευταία 2 βήματα. Για την διόρθωση αρκεί να κάνουμε διπλό κλίκ πάνω στο κάθε μήνυμα λάθους για να μας οδηγήσει στην αντίστοιχη γραμμή που έγινε το λάθος. Καλό θα είναι να ξεκινούμε πάντα απο το πρώτο λάθος και έπειτα απο την διόρθωση του να ξανακάνουμε compile μιας και το λάθος αυτό μπορεί να επηρέασε και τις υπόλοιπες εντολές του προγράμματος. Σε περίπτωση που είμαστε σίγουροι ότι δεν έχουμε λάθη, μπορούμε να παραλείψουμε το βήμα 5 και να εκτελέσουμε κατευθείαν το βήμα 6, το οποίο θα εκτελέσει και τα δύο βήματα.

Βήμα 7. Εκτέλεση του προγράμματος

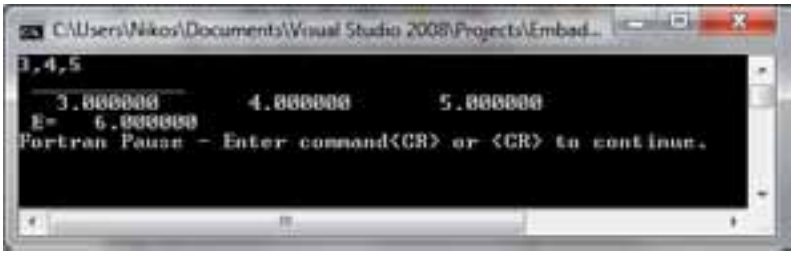
Από την επιλογή Debug διαλέγουμε Start Without Debugging (ή Ctrl+F5 ή επιλέγουμε το αντίστοιχο εικονίδιο από την μπάρα εικονιδίων)



Εμφανίζεται το παράθυρο



στο οποίο επιλέγουμε Yes. Στο επόμενο παράθυρο του MS-DOS εισάγουμε τα δεδομένα και εκτελείται το πρόγραμμά μας:



Η παραπάνω διαδικασία αφορά τα απλά προγράμματα με τα οποία θα ασχοληθούμε στη συνέχεια. Για περισσότερες λεπτομέρειες ο αναγνώστης μπορεί να ανατρέξει στον οδηγό βοήθειας του αντίστοιχου προγράμματος.

Δραστηριότητα 1.2 Προσπάθησε να γράψεις, να μεταφράσεις και να εκτελέσεις το παρακάτω πρόγραμμα στο περιβάλλον της Intel Visual Fortran:

```
PROGRAM TOKOS
    IMPLICIT NONE
    ! Variables
    INTEGER  ::  N
    REAL        :: K,E,TK
    ! Body of Tokos
    PRINT*,'GIVE ME THE INITIAL VALUE'
    READ*, K
    PRINT*,'GIVE ME THE YEARS'
    READ*,N
    PRINT*,'GIVE ME THE PERCENTAGE'
    READ*,E
    TK=K*(1+E)**N
   PRINT*,'THE FINAL VALUE AFTER',N,'YEARS WILL BE',TK
 END PROGRAM TOKOS
```

1.9 Σύνοψη

Ο υπολογιστής αποτελείται από το υλικό και το λογισμικό. Η δομή του υλικού περιγράφηκε στην ενότητα 1.2, ενώ οι βασικές κατηγορίες λογισμικού περιγράφηκαν στην ενότητα 1.3. Στην ενότητα 1.4 ορίσαμε τι είναι αλγόριθμος ενώ στην ενότητα 1.5 προσπαθήσαμε να περιγράψουμε τι είναι γλώσσα προγραμματισμού, καθώς και τις 2 βασικές κατηγορίες στις οποίες χωρίζονται οι γλώσσες προγραμματισμού. Η έννοια του προγραμματισμού αναλύθηκε επιγραμματικά στην ενότητα 1.6 και υλοποιήθηκαν τα στάδια του μέσω ενός απλού παραδείγματος. Η μέτρηση της απόδοσης ενός αλγορίθμου βάσει της ασυμπτωτικής συμπεριφοράς της χρονικής πολυπλοκότητας μελετήθηκε στην ενότητα 1.7. Η διαδικασία μετάφρασης και εκτέλεσης ενός προγράμματος στο περιβάλλον της Intel Visual Fortran αναλύθηκε στην ενότητα 1.8.

1.10 Απαντήσεις στις δραστηριότητες

Ενδεικτική απάντηση Δραστηριότητας 1.1

Η διαδικασία είναι αυτή που περιγράφεται στο σχήμα 1.7. Στην περίπτωση που το πρόγραμμα ήταν γραμμένο σε γλώσσα μηχανής, δεν θα χρειαζόταν η διαδικασία μετάφρασης, η οποία ουσιαστικά μεταφράζει το πρόγραμμα σε γλώσσα μηχανής.

Ενδεικτική απάντηση Δραστηριότητας 1.2

Ακολουθούμε τη διαδικασία που περιγράφηκε στην ενότητα 1.8.