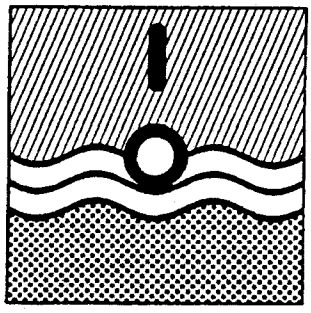
ΥΔΡΟΣΚΟΠΙΟ

ΠΡΟΓΡΑΜΜΑ STRIDE ΕΛΛΑΣ

ΔΗΜΙΟΥΡΓΙΑ ΕΘΝΙΚΗΣ ΤΡΑΠΕΖΑΣ
ΥΔΡΟΛΟΓΙΚΗΣ ΚΑΙ ΜΕΤΕΩΡΟΛΟΓΙΚΗΣ
ΠΛΗΡΟΦΟΡΙΑΣ

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΤΟΜΕΑΣ ΥΔΑΤΙΚΩΝ ΠΟΡΩΝ,
ΥΔΡΑΥΛΙΚΩΝ ΚΑΙ ΘΑΛΑΣΣΙΩΝ ΕΡΓΩΝ

NATIONAL TECHNICAL UNIVERSITY OF ATHENS
DEPARTMENT OF WATER RESOURCES,
HYDRAULIC AND MARITIME ENGINEERING



ΠΡΟΔΙΑΓΡΑΦΕΣ ΓΙΑ ΤΗΝ ΤΥΠΟΠΟΙΗΣΗ ΣΤΗΝ ΑΝΑΠΤΥΞΗ ΛΟΓΙΣΜΙΚΟΥ

GUIDELINES FOR STANDARDISED SOFTWARE DEVELOPMENT

Ν. Παπακώστας
Ομάδα Πληροφορικής Ε.Μ.Π.

Αριθμός τεύχους 1/6
Report number

ΑΘΗΝΑ - ΣΕΠΤΕΜΒΡΙΟΣ 1992
ATHENS - SEPTEMBER 1992

HYDROSCOPE

STRIDE HELLAS PROGRAMME

DEVELOPMENT OF A NATIONAL DATA
BANK FOR HYDROLOGICAL AND
METEOROLOGICAL INFORMATION

ΠΕΡΙΕΧΟΜΕΝΑ

Περίληψη
Abstract

1 ΕΙΣΑΓΩΓΗ 1
1.1 Λόγοι Υπαρξης Προδιαγραφών 1
1.2 Κατηγορίες Προδιαγραφών 1
1.3 Στόχοι Ανάπτυξης 2

2 ΓΛΩΣΣΕΣ - ΕΡΓΑΛΕΙΑ - ΠΕΡΙΒΑΛΛΟΝ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ 4
2.1 Γλώσσα C 4
2.2 Γλώσσα SQL 4
2.3 Γλώσσες και εργαλεία INGRES 4
2.4 Περιγραφή εφαρμογής και χρησιμοποιούμενες γλώσσες 5
2.5 Αλλα εργαλεία 7
2.6 Φιλοσοφία και Περιβάλλον Ανάπτυξης 7

3 ΥΠΟΧΡΕΩΤΙΚΑ ΣΤΟΙΧΕΙΑ ΚΑΙ ΠΡΟΔΙΑΓΡΑΦΕΣ 9
3.1 Γενικά 9
3.2 Υποχρεωτικά Στοιχεία και Προδιαγραφές 9

4 ΠΡΟΑΙΡΕΤΙΚΑ ΣΤΟΙΧΕΙΑ ΚΑΙ ΠΡΟΔΙΑΓΡΑΦΕΣ 15
4.1 Γενικά 15
4.2 Προαιρετικά Στοιχεία και Προδιαγραφές 15

5 ΠΑΡΑΔΕΙΓΜΑΤΑ 18
5.1 Γενικά 18
5.2 Γλώσσες C και ESQL/C 18
5.3 Γλώσσα INGRES Windows/4GL 26
5.4 Γλώσσα SQL 32
5.5 UNIX shell & tools 34

6 ΒΙΒΛΙΟΓΡΑΦΙΑ 37

ΠΕΡΙΛΗΨΗ

Κατα την ανάπτυξη ενος μεγάλου και σύνθετου εργου λογισμικού, όπως το ΥΔΡΟΣΚΟΠΙΟ, είναι επιτακτική η ανάγκη, για λόγους αναγνωσιμότητας, συντηρησιμότητας και μεταφερτότητας των εφαρμογών, συμμόρφωσης όλων των προγραμματιστών με κοινές προδιαγραφές τυποποίησης της ανάπτυξης. Οι εξυπηρετούμενοι στόχοι είναι, εκτός απο την τήρηση των προδιαγραφών λειτουργίας και των χρονοδιαγραμμάτων, η αναγνωσιμότητα, συντηρησιμότητα και μεταφερτότητα, καθώς και η ελαχιστοποίηση του χρόνου εκτέλεσης και των απαιτήσεων σε μνήμη και δίσκο. Η ανάπτυξη του λογισμικού του ΥΔΡΟΣΚΟΠΙΟΥ θα βασισθεί σε ενα πλήρες σύνολο γλωσσών και εργαλείων προγραμματισμού (C, SQL, ESQL/C, INGRES/4GL, INGRES Windows/4GL, UNIX Tools) και στις αρχες του δομημένου προγραμματισμού. Κατάλληλες επικεφαλίδες και κανόνες ονοματολογίας, καθώς και άλλες προγραμματιστικές τεχνικές και πρακτικές, είτε υποχρεωτικές είτε προαιρετικές - συνιστώμενες για τους προγραμματιστές, είναι το πλαίσιο μέσα στο οποίο θα αναπτυχθεί το λογισμικό του Έργου.

ABSTRACT

During the development of a big and complex software project, like HYDROSCOPE, it is essential for all programmers, due to readability, maintainability and portability reasons, to adhere to common guidelines for standardised software development. Regarding the goals of the software development process, these are, apart from conforming to functional requirements and time schedules, readability, maintainability and portability, as well as minimisation of execution time and disk and memory space requirements. HYDROSCOPE software development will be based on a complete set of programming languages and tools (C, SQL, ESQL/C, INGRES/4GL, INGRES Windows/4GL, UNIX Tools), along with structured programming principles. Suitable headers and naming conventions and other programming techniques and practices, either mandatory or optional - suggested to the programmers, compose the HYDROSCOPE software development framework.

1 ΕΙΣΑΓΩΓΗ

1.1 Λόγοι ύπαρξης προδιαγραφών

Στο κείμενο αυτό περιλαμβάνονται οι βασικές αρχές και προδιαγραφές με τις οποίες θα πρέπει να συμμορφώνονται όσοι θα συμμετάσχουν στην ανάπτυξη λογισμικού για τις ανάγκες του ΥΔΡΟΣΚΟΠΙΟΥ. Οι λόγοι ύπαρξης κοινών για όλους τους προγραμματιστές προδιαγραφών είναι οι εξής:

(α) η μεγιστοποίηση της αναγνωσιμότητας (readability) και της συντηρησιμότητας (maintainability) του λογισμικού. Να είναι δηλαδή δυνατό, για ενα οποιδήποτε πρόγραμμα, κάποιος άλλος προγραμματιστής, διαφορετικός απο τον αρχικό δημιουργό, να διαβάσει και να κατανοήσει σωστά τον κώδικα και τη λογική του προγράμματος. Κατά συνέπεια, να είναι σε θέση να το τροποποιήσει αν χρειαστεί χωρίς να απαιτείται παρέμβαση του αρχικού προγραμματιστή. Γενικεύοντας, να δίνεται η δυνατότητα σε μια ομάδα προγραμματιστών να εργάζεται ταυτόχρονα ή διαδοχικά για τη δημιουργία και την αναβάθμιση μιας εφαρμογής χωρίς να σπαταλιέται πολύτιμος χρόνος για εξοικείωση κάθε μέλους της ομάδας με τις αρχές στις οποίες υπακούει κάθε άλλο μέλος, διαδικασία που γίνεται απαγορευτική όταν πρόκειται για σχετικά μεγάλο μέγεθος ομάδας. Το αποτέλεσμα της ελαχιστοποίησης (και, ιδεατά, της εξαφάνισης) προβλημάτων συννενόησης, παραξηγήσεων, έντασης, εκνευρισμού κτλ. είναι αυξημένη παραγωγικότητα και μείωση του απαιτούμενου χρόνου ανάπτυξης. Απλά: ενα έργο εκτελείται βέλτιστα αν - μεταξύ άλλων- τα μέλη της ομάδας που το έχει αναλάβει "μιλάνε την ίδια γλώσσα".

(β) η μεταφερτότητα (portability) του παραγόμενου λογισμικού. Θα πρέπει το λογισμικό να είναι ανεξάρτητο απο τυχόν περιορισμούς που επιβάλλει το σύστημα ανάπτυξης (υλικό (hardware) - λογισμικό συστήματος (system software) πχ. λειτουργικό σύστημα, μεταγλωττιστές (compilers) κτλ.) ώστε να είναι δυνατή η αυτόματη ή με ελάχιστη προσπάθεια μεταφοράς του σε άλλη υπολογιστική πλατφόρμα. Πχ. δεν είναι δυνατό κάποιος προγραμματιστής να αναπτύσει προγράμματα που απαιτούν 32MB μνήμης, αν όλα τα συστήματα στα οποία θα "τρέξει" το πρόγραμμα δεν έχουν 32MB μνήμης.

(γ) η συμμόρφωση με τις προδιαγραφές λειτουργίας (functional requirements) γίνεται ευκολότερη όταν υπάρχει κοινή "βάση" ανάπτυξης, σε σχέση με την οποία έχουν συνταχθεί και οι προδιαγραφές λειτουργίας. Ετσι τελικά είναι πιθανότερη η παραγωγή αξιόπιστου λογισμικού που ακολουθεί αυτές τις προδιαγραφές και εκπληρώνει το σκοπό για τον οποίο κατασκευάστηκε.

1.2 Κατηγορίες προδιαγραφών

Είναι γεγονός, παρόλα αυτά, πως η θέσπιση υπερβολικά αυστηρών και δεσμευτικών κανόνων μπορεί να έχει και αντίθετα αποτελέσματα. Αυτό συμβαίνει όταν ο προγραμματιστής αισθάνεται πως περιορίζεται και "καταπιέζεται", προσπαθώντας να συμμορφωθεί με αυστηρούς κανόνες. Το φαινόμενο είναι τόσο εντονότερο όσο εμπειρότερος είναι ο προγραμματιστής, οπότε έχει αναπτύξει δικές του αρχές και μεθόδους -καλές, κακές ή αδιάφορες (πάντα σε σχέση με το ζητούμενο αποτέλεσμα)- οι οποίες έρχονται κάποτε σε σύγκρουση με το πλαίσιο κανόνων. Αρα είναι σημαντικό να μπορεί κανείς να διατηρήσει την προσωπική του ελευθερία σε σχέση με τις επιλογές που κάνει όταν προγραμματίζει, αυτό που γενικά ονομάζεται "στύλ προγραμματισμού". Η προσωπική αυτή ελευθερία έχει όμως και τα όρια της: υπάρχουν πρακτικές τις οποίες, αν ακολουθήσει κανείς, μπορούν να αποβούν επιζήμιες για το έργο. Πχ. τμήμα του στύλ μπορεί να είναι η συχνή χρήση εντολών τύπου goto ("spaghetti code"). Δεδομένου ότι κάτι τέτοιο περιορίζει και πιθανά

μηδενίζει την ικανότητα τρίτων να κατανοήσουν και να επέμβουν σε ενα πρόγραμμα, η χρήση του πρέπει να περιοριστεί στα απολύτως αναγκαία πλαίσια, ακόμα και σε βάρος της "προσωπικής ελευθερίας". Απο την άλλη, το προσωπικό "indent style" κάθε προγραμματιστή, στο βαθμό που δεν κάνει τον κώδικα δυσνόητο μπορεί να είναι θέμα επιλογής.

Για αυτούς τους λόγους, θεσπίζονται δύο κατηγορίες κανόνων / αρχών / προδιαγραφών. Η μια κατηγορία είναι οι πολύ σημαντικοί κανόνες, αυτοί τους οποίους κάθε προγραμματιστής θα πρέπει να τηρεί με κάθε θυσία, διότι άπτονται της συνολικής ανάπτυξης και λειτουργίας του έργου, τυχόν δε παραβίαση τους θα έχει σοβαρές επιπτώσεις και στη δουλειά των άλλων. Η δεύτερη κατηγορία δεν είναι καν κανόνες, απλά προτάσεις και παραινέσεις μη "υποχρεωτικού" χαρακτήρα. Το αν και ποιούς θα ακολουθήσει ο προγραμματιστής είναι δική του επιλογή.

1.3 Στόχοι ανάπτυξης

Στο σημείο αυτό πρέπει να γίνει μια αναφορά και στους στόχους που πρέπει να διέπουν την ανάπτυξη μιας εφαρμογής -και σε σχέση με τις ανάγκες του ΥΔΡΟΣΚΟΠΙΟΥ-. Αν εξαιρεθεί ο στόχος (α), για τον οποίο δεν γίνεται κανένας συμβιβασμός, οι άλλοι στόχοι μπορούν να είναι αντικείμενα συμβιβασμού, ώστε να βρεθεί -ει δυνατόν- η "χρυσή τομή". Πχ. αν μια βελτίωση του χρόνου εκτέλεσης κατα 10% απαιτεί προσπάθεια 20% μεγαλύτερη, τότε πιθανά να μην αξίζει τον κόπο, εκτός και αν ο χρόνος εκτέλεσης είναι απαράδεκτα μεγάλος, οπότε δικαιολογείται η επιπλέον προσπάθεια. Δεν είναι δυνατό βέβαια να υπάρχουν στόχοι που δεν εκπληρώνονται καθόλου (πχ. μεγάλη ταχύτητα λειτουργίας σε ενα πρόγραμμα του οποίου η κατανόηση είναι αδύνατη). Σε κάθε περίπτωση, η ομάδα αλλά και προσωπικά κάθε μέλος της επωμίζεται την ευθύνη της αιτιολόγησης των επιλογών που γίνονται. Η ιεράρχηση των στόχων είναι επίσης ευθύνη της ομάδας και του προγραμματιστή. Παρακάτω υπάρχει μια πρόταση ιεράρχησης:

(α) συμμόρφωση με τις προδιαγραφές λειτουργίας. Οποιοδήποτε πρόγραμμα πρέπει να εκτελεί αξιόπιστα και επακριβώς τις λειτουργίες που έχουν προδιαγραφεί γιαυτό. Καμία άλλη δυνατότητα ή χαρακτηριστικό δεν είναι δυνατό να επιτυγχάνεται σε βάρος αυτού του στόχου. Εννοείται φυσικά πως ο στόχος αυτός περιλαμβάνει και την τήρηση των χρονοδιαγραμμάτων που έχουν τεθεί για την εργασία, δεδομένου πως η χρονική διάρκεια του έργου είναι σχετικά περιορισμένη. Αυτό βέβαια δεν σημαίνει πως είναι αποδεκτές λύσεις "quick-and-dirty" ("γρήγορη" ανάπτυξη χωρίς σεβασμό στην εκπλήρωση άλλων στόχων).

(β) αναγνωσιμότητα και συντηρησιμότητα του κώδικα. Το γεγονός πως η ανάπτυξη του λογισμικού δεν γίνεται απο εναν αλλά είναι αποτέλεσμα συνεργασίας υπαγορεύει ως βασικό στόχο της ανάπτυξης την αναγνωσιμότητα και συντηρισιμότητα του κώδικα. Παρόλα αυτά, δεδομένου πως το έργο δεν είναι εκπαιδευτικού χαρακτήρα αλλά θα πρέπει να έχει σαν τελικό αποτέλεσμα ενα λειτουργικό προϊόν, υπάρχουν περιθώρια συμβιβασμού. Πχ. η ανάπτυξη ενός νέου δυσνόητου αλγόριθμου που συνεπάγεται μεγάλα οφέλη για τους άλλους στόχους πρέπει να τεκμηριώνεται κατάλληλα, ενω θα πρέπει να καταβάλλεται κάθε δυνατή προσπάθεια για εγγενή αναγνωσιμότητα, όπως επιλογή κατάλληλων ονομάτων για τις μεταβλητές και τις συναρτήσεις κτλ.

(γ) μεταφερτότητα του κώδικα. Το ΥΔΡΟΣΚΟΠΙΟ βασίζεται σε κοινή για όλους υπολογιστική βάση, με σχετικά μικρές διαφορές. Εν τούτοις, είναι σημαντικό να αποφεύγονται οποιεσδήποτε τεχνικές βασίζονται σε υποθέσεις σε σχέση με το περιβάλλον λειτουργίας. Έτσι, πρέπει να αποφεύγεται η ανάπτυξη εφαρμογών εξαρτώμενων απο στοιχεία όπως πχ. το είδος του υπολογιστή, το

μέγεθος της μνήμης, η ανάλυση της οθόνης (ιδίως λόγω της ανάγκης ταυτόχρονης λειτουργίας σε περιβάλλον UNIX/X-Windows & DOS/MS-Windows), το λειτουργικό σύστημα και γενικά το λογισμικό συστήματος κοκ. Σε περίπτωση που κάτι τέτοιο δεν είναι δυνατό να αποφευχθεί, να γίνονται σαφείς οι παραδοχές και να ορίζονται με κατάλληλο τρόπο (πχ. conditional compilation) τα σημεία στα οποία υπάρχει μη μεταφερτός κώδικας.

(γ) ελαχιστοποίηση του χρόνου εκτέλεσης. Λόγω της φύσης του έργου θα υπάρχουν πολλά σημεία στα οποία θα απαιτείται μεγαλύτερη ταχύτητα επεξεργασίας για την ικανοποίηση των αιτημάτων όλο και πιο ανυπόμονων και απαιτητικότερων χρηστών (σημ. γνωστές και σταθερές ιδιότητες όλων των χρηστών όλων των συστημάτων σε όλο τον κόσμο...). Ο χρόνος εκτέλεσης είναι κλασσικό παράδειγμα στάθμισης του κόστους που απαιτείται και της απαιτούμενης λήψης αποφάσεων. Πχ. δεν ωφελεί τόσο εναν απομακρυσμένο (remote) χρήστη μια αύξηση της ταχύτητας ανάγνωσης δεδομένων απο το δίσκο (η οποία είναι της τάξης των MB/sec) κατα 10% όταν η ταχύτητα του δικτύου είναι 3 τάξεις μεγέθους (KB/sec) μικρότερη, ίσως μάλιστα κάτι τέτοιο να επιτείνει το πρόβλημα (μεγαλύτερη κίνηση στο δίκτυο). Αντίθετα, αν με ίδια προσπάθεια ήταν δυνατό να βελτιωθούν πχ. οι αλγόριθμοι, ώστε να μην απαιτείται συχνή προσπέλαση σε απομακρυσμένες πληροφορίες, το τελικό κέρδος σε ταχύτητα εκτέλεσης θα ήταν πολλαπλάσιο.

(δ) ελαχιστοποίηση των απαιτήσεων σε μνήμη και χώρο στο δίσκο. Οχι μόνο υπάρχει σχετική επάρκεια στην υπολογιστική υποδομή ως προς αυτά τα στοιχεία, αλλά είναι και αυτά των οποίων το κόστος "πέφτει" συνέχεια, και άρα δικαιολογεί μια -λογική- "σπατάλη". Πχ. μεγάλη προγραμματιστική προσπάθεια για μείωση του μεγέθους ενος προγράμματος μπορεί να είναι λογικό να αποφευχθεί με την αγορά επιπλέον μνήμης (οι άνθρωποι είναι "ακριβότεροι" απο το υλικό...).

2 ΓΛΩΣΣΕΣ - ΕΡΓΑΛΕΙΑ - ΠΕΡΙΒΑΛΛΟΝ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

2.1 Γλώσσα C

Μια απο τις γλώσσες προγραμματισμού που θα χρησιμοποιηθούν για την ανάπτυξη των εφαρμογών του ΥΔΡΟΣΚΟΠΙΟΥ είναι η C. Η γλώσσα αυτή έχει πολλά πλεονεκτήματα όπως:

(α) ευρύ φάσμα δυνατών εφαρμογών, απο επιστημονικές και εμπορικές μέχρι τον προγραμματισμό συστημάτων. Ιδιαίτερα στον τελευταίο τομέα, στις περιπτώσεις όπου απαιτείται προγραμματισμός σε στενή συσχέτιση με το υλικό (πχ. λειτουργικά συστήματα, συστήματα διαχείρισης βάσεων δεδομένων (ΣΔΒΔ), μεταγλωττιστές, συστήματα γραφικών κτλ.) είναι η πιθανότερη επιλογή

(β) εξαιρετικά διαδεδομένη, σε βαθμό που στο λειτουργικό σύστημα UNIX να αποτελεί αδιαμφισβήτητο standard.

(γ) πολύ ισχυρή, με ταχύτητα εκτέλεσης κοντά σε αυτήν της γλώσσας assembly.

(δ) εξαιρετικά μεταφερτή σε μεγάλο πλήθος και πολλά είδη υπολογιστών.

(ε) ενσωματώνει εξελιγμένες δομές ελέγχου, ισχυρούς και ποικίλους τελεστές, πολλά είδη μεταβλητών, λιτότητα έκφρασης και "ανοικτή" και επεκτάσιμη αρχιτεκτονική.

Στο ΥΔΡΟΣΚΟΠΙΟ θα χρησιμοποιηθεί για τον προγραμματισμό συστήματος και "χαμηλού επιπέδου" (system & low level programming), όπου απαιτείται υψηλή ταχύτητα ή/και στενή αλληλεπίδραση με τον πυρήνα και διάφορα άλλα στοιχεία του ΣΔΒΔ (πχ. για άμεσο προγραμματισμό του λειτουργικού συστήματος ή του πυρήνα του ΣΔΒΔ (user defined data types)).

2.2 Γλώσσα SQL

Η SQL είναι η standard γλώσσα χειρισμού δεδομένων (data manipulation language) για σχεσιακές βάσεις δεδομένων. Τα πλεονεκτήματα της είναι τα εξής:

(α) εξαιρετικά διαδεδομένη, αποτελεί επίσημο standard σε όλο τον κόσμο και ιδίως αναφορικά με σχεσιακές βάσεις δεδομένων, ενω η δημοτικότητα της επεκτείνει την εφαρμογή της και σε μη σχεσιακές ΒΔ.

(β) ιδιαίτερα καλά προσαρμοσμένη σε σχεσιακές ΒΔ (θεώρηση δεδομένων ως στήλες (columns) σε πίνακες (tables)).

(γ) όχι διαδικαστική (non-procedural) γλώσσα.

(δ) αρκετά ισχυρή και ευέλικτη

(ε) εύκολη στην εκμάθηση

Στο ΥΔΡΟΣΚΟΠΙΟ θα χρησιμοποιηθεί για κάθε είδους ερώτηση (query) προς τη ΒΔ, τόσο σε τοπικούς όσο και σε απομακρυσμένους κόμβους, αποτελώντας το standard τρόπο επικοινωνίας της εφαρμογής με τη ΒΔ. Η γλώσσα ESQL/C (Embedded SQL/C) ενσωματώνει την SQL σε προγράμματα C, ώστε να είναι δυνατό απο ενα πρόγραμμα C να γίνει μια ερώτηση σε SQL προς τη ΒΔ και τα αποτελέσματα να αποθηκευτούν σε μεταβλητές του προγράμματος C, συνδυάζοντας τα πλεονεκτήματα και των δύο. Η ESQL/C θα χρησιμοποιείται πρακτικά όπου και η C, εναλλακτικά με τη γλώσσα 4ης γενεάς (4GL).

2.3 Γλώσσες και Εργαλεία INGRES

Το ΣΔΒΔ INGRES επιλέχθηκε για να καλύγει τις ανάγκες του ΥΔΡΟΣΚΟΠΙΟΥ σε πυρήνα σχεσιακής βάσης δεδομένων και τα αντίστοιχα εργαλεία ανάπτυξης. Τα εργαλεία ανάπτυξης της INGRES περιλαμβάνουν, εκτός απο τα "κλασσικά" τμήματα (πχ. SQL, Report-Writer κτλ.), πολύ καλή γλώσσα προγραμματισμού 4ης γενεάς (4GL) για περιβάλλον τερματικών χαρακτήρων (character based), γεννήτρια κώδικα /

εφαρμογών (code / application generator) για αυτό το περιβάλλον και επιπλέον μια εξαιρετικά ισχυρή γλώσσα 4ης γενεάς για προγραμματισμό σε περιβάλλον γραφικών (Bitmapped Windows Icons Menus Mouse Pointers Graphical User Interface). Τα πλεονεκτήματα των εργαλείων αυτών είναι:

(α) εξαιρετική συνεργασία με τον πυρήνα της ΒΔ.

(β) ισχυρές και ευέλικτες γλώσσες 4ης γενεάς, η χρήση των οποίων έχει σαν αποτέλεσμα τη μείωση (ακόμα και σε ποσοστό μεγαλύτερο του 50 - 60%) του απαιτούμενου χρόνου ανάπτυξης.

(γ) ευελιξία σε σχέση με το περιβάλλον ανάπτυξης (χαρακτήρων ή γραφικών) και δυνατότητα χρήσης γραφικών (με ευνοϊκά αποτελέσματα για τη φιλικότητα προς τον τελικό χρήστη και τις παρεχόμενες δυνατότητες) σε αρκετά μεγάλο ποσοστό της τελικής εφαρμογής.

(δ) δυνατότητα ανάπτυξης μεγάλου μέρους των απαιτούμενων εφαρμογών με γλώσσες 4GL χωρίς ανάγκη χρήσης της (πιο δύσχρηστης) C.

(ε) ολοκληρωμένο περιβάλλον ανάπτυξης και άμεση επικοινωνία με SQL. Επίσης, αυτές οι γλώσσες είναι ειδικά σχεδιασμένες για την λειτουργία σε περιβάλλον βάσεων δεδομένων, σε αντίθεση με τις "κλασσικές" γλώσσες προγραμματισμού, και προσφέρουν μεγάλες δυνατότητες επικοινωνίας τόσο με τη ΒΔ όσο και με την οθόνη ή τον εκτυπωτή, για την παρουσίαση (presentation) των αποτελεσμάτων.

Οι γλώσσες και τα εργαλεία INGRES θα χρησιμοποιηθούν για τη μεγάλη πλειοψηφία των απαιτούμενων εφαρμογών ως βασικές γλώσσες προγραμματισμού εφαρμογών (application programming) του ΥΔΡΟΣΚΟΠΙΟΥ. Μόνο στις (λίγες) περιπτώσεις που θα απαιτηθεί "χαμηλότερου" επιπέδου προγραμματισμός (ή η μέγιστη δυνατή ταχύτητα) θα απαιτηθεί χρήση C - ESQL/C.

2.4 Περιγραφή εφαρμογής και χρησιμοποιούμενες γλώσσες

Θα πρέπει επίσης να γίνουν σαφή ορισμένα σημεία σχετικά με το είδος των εφαρμογών που θα χρειαστεί να αναπτυχθούν για το ΥΔΡΟΣΚΟΠΙΟ και με τις γλώσσες προγραμματισμού που θα χρησιμοποιηθούν για αυτές. Συγκεκριμένα, στις εφαρμογές του ΥΔΡΟΣΚΟΠΙΟΥ περιλαμβάνονται:

(α) ο προγραμματισμός της Βάσης Δεδομένων. Δηλαδή, η υλοποίηση του σχήματος της ΒΔ, η δημιουργία και τυχόν τροποποίηση των πινάκων της, η δημιουργία των user defined data types, των rules, των database procedures κτλ., η διαχείριση της πρόσβασης των χρηστών κοκ. Ο προγραμματισμός αυτός θα γίνει με τα παρεχόμενα απο το ΣΔΒΔ εργαλεία (SQL, σπανιότερα 4GL κτλ.) καθώς και σε C ή ESQL/C. Επίσης είναι δυνατή και η χρήση standard εργαλείων του UNIX (πχ. shell, awk, sed κτλ.).

(β) ο προγραμματισμός των εφαρμογών που θα βασισθούν στα αποθηκευμένα στη ΒΔ στοιχεία. Εφαρμογές εισαγωγής δεδομένων, ερωτήσεων προς τη βάση, παρουσίασης αποτελεσμάτων (πιθανά και σε γραφική μορφή), εξαγωγής αναφορών (reports) κτλ. Ολες αυτές οι εφαρμογές θα αναπτυχθούν με τις ειδικά σχεδιασμένες για αυτή τη λειτουργία γλώσσες 4ης γενεάς (INGRES/4GL, INGRES Windows/4GL) και όπου απαιτείται μεγαλύτερη ταχύτητα θα χρησιμοποιηθεί ESQL/C.

(γ) διάφορες βοηθητικές εφαρμογές, όπως πχ. επικοινωνίας με άλλες εφαρμογές (πχ. μοντέλα, τα οποία δεν περιλαμβάνονται στο ΥΔΡΟΣΚΟΠΙΟ) ή με δη υπάρχουσες τράπεζες πληροφοριών, αυτοματοποιημένων λειτουργιών (πχ. backup) κτλ. Για το σκοπό αυτό θα χρησιμοποιηθούν τόσο εργαλεία του UNIX όσο και ESQL/C, για τη δημιουργία κατάλληλων φίλτρων για "εξαγωγή" δεδομένων σε αρχεία προς άλλα προγράμματα.

Εννοείται φυσικά πως σε κάθε περίπτωση ο προγραμματιστής μπορεί να χρησιμοποιήσει οποιαδήποτε απο τις παραπάνω γλώσσες -C, ESQL/C, INGRES SQL, 4GL, Windows/4GL & other tools, UNIX tools- κρίνει σκόπιμο σε σχέση με τις ανάγκες της συγκεκριμένης εφαρμογής.

Αναφορικά με τυχόν χρήση άλλων γλωσσών προγραμματισμού, πέρα απο τις παραπάνω, αυτή δεν είναι επιθυμητή, για τους εξής λόγους:

(α) Οι γλώσσες προγραμματισμού που έχουν επιλεχθεί είναι οι βέλτιστες ανα περίπτωση για τη συγκεκριμένη εργασία, καθώς έχουν ειδικά σχεδιαστεί και δοκιμαστεί διεξοδικά, πχ. η C σε προγραμματισμό συστημάτων, η SQL σε ερωτήσεις σε σχεσιακές βάσεις δεδομένων, οι 4GL σε εφαρμογές ανάκτησης και παρουσίασης στοιχείων απο βάσεις δεδομένων κοκ.

(β) Η διεθνής εμπειρία έχει καταδείξει τα σημαντικά οφέλη σε παραγωγικότητα και χρόνο που υπάρχουν απο τη χρήση τέτοιων εξειδικευμένων εργαλείων.

(γ) Είναι αδύνατο να έχει τη δυνατότητα κάθε προγραμματιστής να αναπτύξει στη γλώσσα που προτιμά, διότι τότε θα προκύψει μια χαώδης κατάσταση έλλειψης συνενόησης, συντονισμού, δυνατότητας έστω και ελάχιστης συντήρησης και εξέλιξης των προγραμμάτων με τελικό αποτέλεσμα αναμφίβολα την αποτυχία του έργου.

(δ) Υπάρχει ασφαλώς μια δυσκολία, ακόμα και απροθυμία, για την εκμάθηση μιας νέας γλώσσας, και διάθεση για χρήση των ήδη γνωστών, και αν ακόμα δεν είναι οι πλέον κατάλληλες για το συγκεκριμένο έργο. Η εκπαίδευση στις συγκεκριμένες γλώσσες που έχουν επιλεχθεί (C, ESQL/C, SQL, INGRES 4GL, INGRES Windows/4GL, UNIX tools) θα δώσει ασφαλώς μια αρκετά περιεκτική εικόνα τους, ώστε με ελάχιστη (πχ. ενός -δυο μηνών) εξάσκηση να μπορεί κάποιος να γίνει παραγωγικός. Ο χρόνος αυτός θα αντισταθμιστεί ασφαλώς απο τα χρονικά και άλλα οφέλη που θα συνεπάγεται η μελλοντική χρήση αυτών των γλωσσών (και όχι άλλων).

(ε) Παρέχεται σαφώς η δυνατότητα χρήσης ήδη υπάρχοντων προγραμμάτων (πχ. μοντέλων) και επικοινωνίας τους με τη ΒΔ με την (αρκετά απλή) κατασκευή των κατάλληλων "φίλτρων" που θα ανακτούν τα στοιχεία απο τη ΒΔ και θα τα μετατρέπουν στην απαιτούμενη απο το πρόγραμμα μορφή (format) σε αρχεία. Σε κάθε περίπτωση άσχετα με το ΥΔΡΟΣΚΟΠΙΟ, είναι δυνατή η τροποποίηση των προγραμμάτων αυτών με "μεταγραφή" τους στην κατάλληλη Embedded SQL γλώσσα (πχ. ESQL/FORTRAN) για άμεση (online) ανάκτηση στοιχείων απο τη ΒΔ, αν είναι επιθυμητό.

Παρόλα αυτά, είναι κατανοητό πως υπάρχει ενας αρκετά μεγάλος όγκος έτοιμου κώδικα, σε μορφή βιβλιοθηκών άλλης γλώσσας προγραμματισμού (κυρίως FORTRAN), η χρήση του οποίου είναι δυνατό να έχει θετικά αποτελέσματα στην επιθυμητή μείωση του χρόνου ανάπτυξης. Αρα, είναι δυνατή η χρήση άλλης γλώσσας προγραμματισμού, και συγκεκριμένα της FORTRAN, υπο τις παρακάτω προϋποθέσεις:

(α) είναι δυνατή η χρήση ήδη έτοιμων (ή προσαρμοσμένων με μικρές τροποποιήσεις) τμημάτων κώδικα FORTRAN σε μορφή βιβλιοθηκών, για τις συγκεκριμένες εκείνες εργασίες στις οποίες η FORTRAN εξειδικεύεται (μαθηματικά, στατιστική). Αυτό σημαίνει πως η χρήση της FORTRAN πρέπει να περιοριστεί σε συγκεκριμένες ρουτίνες, ενω δεν είναι εφικτή η υλοποίηση του "κυρίως" προγράμματος και της λογικής της εφαρμογής σε FORTRAN. Για το σκοπό αυτό πρέπει να χρησιμοποιηθεί κάποια απο τις άλλες γλώσσες.

(β) "κορμός" του αναπτυσσόμενου για τις ανάγκες του ΥΔΡΟΣΚΟΠΙΟΥ κώδικα θεωρούνται οι γλώσσες C, ESQL/C, INGRES/4GL, INGRES Windows/4GL,

UNIX tools, όπως αναφέρθηκε παραπάνω. Η χρήση άλλης γλώσσας (FORTRAN) θα πρέπει να γίνεται πάντα με πρόβλεψη σύνδεσης (linking) με προγράμματα των παραπάνω γλωσσών. Ο προγραμματιστής που χρησιμοποιεί FORTRAN είναι αποκλειστικά υπεύθυνος για εξεύρεση τρόπου σύνδεσης (linking - interfacing) των προγραμμάτων του με τα υπόλοιπα και επίλυσης τυχόν προβλημάτων (πχ. calling convention, μη δυνατότητα περάσματος πινάκων και records, ανυπαρξία interface με common blocks κτλ.). Οι ρουτίνες σε FORTRAN μπορούν να θεωρηθούν ως παρελκόμενα, τα οποία με κατάλληλη προσαρμογή -απο τον προγραμματιστή- συνδέονται με την κυρίως εφαρμογή, χωρίς οποιαδήποτε τροποποίηση της, και χωρίς αυτή να "γνωρίζει" το γεγονός της χρήσης FORTRAN (δηλ. οι ρουτίνες FORTRAN θα είναι "μαύρα κουτιά" για την υπόλοιπη εφαρμογή).

(γ) κλήση ρουτινών FORTRAN είναι δυνατή μόνο μέσα απο C και ESQL/C και όχι απο INGRES 4GL και INGRES Windows/4GL. Για κλήση μέσα απο αυτά τα INGRES tools πρέπει να γραφτεί κατάλληλη interface ρουτίνα σε C, η οποία θα καλείται απο το πρόγραμμα πχ. σε INGRES Windows/4GL με κατάλληλο πέρασμα παραμέτρων, και αυτή με τη σειρά της θα καλεί -έμμεσα πια- τη ρουτίνα FORTRAN.

(δ) πρέπει να καταβάλλεται προσπάθεια συμμόρφωσης των ρουτινών FORTRAN με τους κανόνες προγραμματισμού του ΥΔΡΟΣΚΟΠΙΟΥ που παραουσιάζονται παρακάτω.

2.5 Αλλα εργαλεία

Η χρήση άλλων εργαλείων (όχι γλωσσών) προγραμματισμού, πέρα απο αυτά που αναφέρθηκαν (πχ. UNIX tools, INGRES tools) για ανάλυση και πιθανά παραγωγή κώδικα (ή ψευδοκώδικα) κάποιας απο τις προτεινόμενες γλώσσες προγραμματισμού, όπως CASE tools, application generators κτλ. είναι δυνατή. Μελετάται μάλιστα και η πιθανότητα προμήθειας τέτοιων εργαλείων και συνολικής χρήσης τους απο το ΥΔΡΟΣΚΟΠΙΟ. Επίσης είναι δυνατή (και συνίσταται) η χρήση standard UNIX programming tools όπως: make, lex, yacc, sccs, rcs κτλ. ([PIKE], [ROCH]).

2.6 Φιλοσοφία και περιβάλλον ανάπτυξης

Η φιλοσοφία ανάπτυξης λογισμικού θα είναι η client-server. Δηλαδή, ο server είναι ο πυρήνας του σχεσιακού ΣΔΒΔ που "τρέχει" στο σύστημα UNIX και θα διαχειρίζεται αποκλειστικά τα αποθηκευμένα στοιχεία. Κάθε εφαρμογή (client) που ερωτά για κάποια αποθηκευμένα στη ΒΔ στοιχεία, επικοινωνεί με αυστηρά καθορισμένο τρόπο (συνήθως εντολές SQL) με το server, ο οποίος είναι υπεύθυνος για τη διεκπεραίωση της ερώτησης και την αποστολή των αποτελεσμάτων. Η εφαρμογή είναι υπεύθυνη για τη λογική που ακολουθείται, για την παρουσίαση των στοιχείων και την επικοινωνία με τον χρήστη (user interface). Ο server θα λειτουργεί πάντα στο σύστημα UNIX, ενώ οποιοσδήποτε client θα μπορεί να λειτουργήσει (θεωρητικά και ιδανικά εντελώς όμοια) τόσο σε περιβάλλον UNIX όσο και σε περιβάλλον PC και DOS. Σημειωτέον ότι όλα τα εργαλεία ανάπτυξης είναι κατασκευασμένα με βάση αυτή τη φιλοσοφία και άρα η δημιουργία εφαρμογών client - server είναι ο "φυσικός" και "σωστός" τρόπος δημιουργίας εφαρμογών.

Η ανάπτυξη του λογισμικού θα γίνει ολοκληρωτικά σε περιβάλλον UNIX για το τμήμα που αφορά στον server, και είτε σε UNIX είτε σε PC και DOS για τους clients.

(α) Για τις εφαρμογές γραφικών θα χρησιμοποιηθεί το γραφικό περιβάλλον INGRES Windows/4GL σε δυο εκδόσεις:

(i) X-Windows, είτε απευθείας στη κύρια μονάδα του σταθμού εργασίας, είτε απο τερματικό γραφικών (X-Terminal) μέσα απο το δίκτυο

(Ethernet TCP/IP LAN). Το τερματικό γραφικών θα είναι ενα ισχυρό PC (τουλάχιστον 386/33 με 4MB RAM) με κατάλληλο λογισμικό (X-Server/DOS) ή ενα εξειδικευμένο γραφικό τερματικό.

(ii) MS-Windows μέσα απο το δίκτυο. Το PC το οποίο θα λειτουργεί με αυτό τον τρόπο θα πρέπει να είναι ισχυρό (τουλάχιστον 386/40 με 16MB RAM).

Πρέπει εδώ να γίνει ιδιαίτερη μνεία στην ανάπτυξη γραφικών εφαρμογών στο περιβάλλον INGRES Windows/4GL τόσο σε UNIX/X-Windows όσο και σε DOS/MS-Windows. Είναι εξαιρετικά σημαντικό να γίνει η ανάπτυξη με συνεχή προσοχή στη μεταφερτότητα των εφαρμογών ανάμεσα στα δύο περιβάλλοντα. Για αυτό το λόγο, πρέπει να τηρούνται σχολαστικά οι οδηγίες μεταφερτότητας που δίνει η INGRES στο αντίστοιχο εγχειρίδιο ([INGRES]) και να λαμβάνονται υπόψη οι διαφορές των δυο δυνατών περιβαλλόντων (πχ. διαφορετική ανάλυση οθόνης, άλλα fonts, άλλα χρώματα κτλ.). Είναι λογικό πως η ανάπτυξη θα πρέπει να γίνεται σε σχέση με το πιο περιοριστικό περιβάλλον, δηλ. το DOS/MS-Windows. Θα πρέπει λοιπόν να καταβάλλεται προσπάθεια η ανάπτυξη να γίνεται κυρίως με βάση το περιβάλλον DOS/MS-Windows, ώστε να είναι πιθανότερη η δυνατότητα μεταφοράς προς το πιο εξελιγμένο περιβάλλον UNIX/X-Windows. Αλλά και εφαρμογές που αναπτύσσονται στο περιβάλλον UNIX/X-Windows θα πρέπει να δοκιμάζονται αμέσως μετά τη συμπλήρωση κάθε μέρους τους για το αν "τρέχουν" στο άλλο περιβάλλον.

(β) Για τις εφαρμογές χαρακτήρων μπορεί να χρησιμοποιηθεί όποια μέθοδος κρίνεται πρόσφορη (πχ. απευθείας στην "κονσόλα" του σταθμού εργασίας, κάτω απο X-Windows και xterm, μέσω του δικτύου απο PC και κατάλληλο TCP/IP telnet λογισμικό σε περιβάλλον χαρακτήρων DOS ή μέσα απο τα MS-Windows κτλ.).

(γ) Για να επιτευχθεί τη μεταφερτότητα των εφαρμογών είναι απαραίτητη η αποφυγή τεχνικών που εξαρτώνται απο το υλικό του υπολογιστή (hardware dependent), ακόμα και σε βάρος της ταχύτητας εκτέλεσης. Μόνο σε εξαιρετικές περιπτώσεις και με κατάλληλη αιτιολόγηση και τεκμηρίωση είναι δυνατή η χρήση τέτοιων τεχνικών. Επίσης, ως προγραμματιστικό περιβάλλον UNIX εννοείται αυτό το οποίο ορίζουν τα διεθνή πρότυπα POSIX ([POSIX]), SVID ([SVID]) και X/Open. Κατα συνέπεια, χρήση διαφόρων στοιχείων του προγραμματιστικού περιβάλλοντος (HP/UX) που δεν περιλαμβάνονται στα παραπάνω πρότυπα πρέπει να αποφεύγεται.

3 ΥΠΟΧΡΕΩΤΙΚΑ ΣΤΟΙΧΕΙΑ ΚΑΙ ΠΡΟΔΙΑΓΡΑΦΕΣ

3.1 Γενικά

Τα παρακάτω στοιχεία είναι υποχρεωτικά, ανεξάρτητα απο τη χρησιμοποιούμενη γλώσσα προγραμματισμού. Με τον όρο "υποχρεωτικά" εννοείται η ανάγκη συμμόρφωσης όλων των προγραμματιστών του ΥΔΡΟΣΚΟΠΙΟΥ με αυτά. Τυχόν εξαιρέσεις θα πρέπει να αιτιολογούνται σαφώς και να τυγχάνουν της έγκρισης των οργάνων του έργου, διαφορετικά θα θεωρούνται οι αντίστοιχες εφαρμογές "μη ικανοποιητικές".

3.2 Υποχρεωτικά στοιχεία και προδιαγραφές

(α) Οι γλώσσες προγραμματισμού που χρησιμοποιούνται για την ανάπτυξη εφαρμογών του ΥΔΡΟΣΚΟΠΙΟΥ είναι οι εξής: C, (ANSI) SQL, ESQL/C, INGRES/4GL, INGRES Windows/4GL, UNIX Bourne shell (/bin/sh) και διάφορα εργαλεία UNIX (πχ. awk, sed κτλ.). Μπορεί επίσης να χρησιμοποιηθεί και C++ αν κριθεί απαραίτητος ο αντικειμενοστραφής (object oriented) προγραμματισμός. Σε κάθε περίπτωση ο προγραμματιστής μπορεί να διαλέξει με ποιά απο τις παραπάνω γλώσσες θα εργαστεί, αρκεί φυσικά η επιλογή του να είναι λογική και να μπορεί να αιτιολογηθεί. Η χρήση άλλων γλωσσών (FORTRAN) διέπεται απο τους κανόνες και περιορισμούς που αναπτύχθηκαν στην παράγραφο 2.4., καθώς και απο όλες τις παρακάτω προδιαγραφές.

(β) Ολα τα στοιχεία (μεταβλητές, συναρτήσεις κτλ.) του προγράμματος γράφονται στα Αγγλικά και είναι Αγγλικές λέξεις, για λόγους ενιαίου συμβολισμού και τελικά ευκολότερης κατανόησης -αδυναμία χρήσης Ελληνικών με Ελληνικούς χαρακτήρες σε όλες τις γλώσσες προγραμματισμού, έλλειψη κοινά αποδεκτού τρόπου αναπαράστασης Ελληνικών με λατινικούς χαρακτήρες (πχ. xarakthres, charactires κτλ.), έλλειψη ικανοποιητικής μετάφρασης πολλών όρων (πχ. bit, byte κτλ.) ή και ασαφής μετάφραση (πχ. index = δείκτης, pointer = δείκτης κτλ.), ανάγκη κατανόησης του κώδικα μόνο απο προγραμματιστές οι οποίοι, κατα τεκμήριο, γνωρίζουν Αγγλικά.

(γ) Ολα τα σχόλια στον κώδικα του προγράμματος γράφονται επίσης στα Αγγλικά, για εξοικονόμιση χρόνου κατα τον προγραμματισμό και αποφυγή προβλημάτων με διαφορετικά Ελληνικά character sets (πχ. ΕΛΟΤ-928, IBM-437).

(δ) Ολα τα μηνύματα του προγράμματος προς τον χρήστη θα πρέπει να είναι σε Ελληνικά ΕΛΟΤ-928. Ωστόσο, θα πρέπει να ληφθεί πρόνοια ώστε να είναι δυνατή η απρόσκοπτη μετάφραση σε άλλη γλώσσα. Ετσι, δεν συνιστάται η ενσωμάτωση στον κώδικα των μηνυμάτων. Η βέλτιστη λύση είναι να υπάρχει ενα αρχείο ή πίνακας μηνυμάτων, το οποίο να διαβάζει το πρόγραμμα κατα την εκκίνηση του και να λειτουργεί ανάλογα στην αντίστοιχη γλώσσα.

(ε) Οι εφαρμογές χωρίζονται σε αρχεία. Κάθε αρχείο περιέχει κώδικα μιας μόνο γλώσσας προγραμματισμού, σε μορφή συναρτήσεων (functions) και άλλων στοιχείων της γλώσσας (πχ. ορισμοί τύπων, μεταβλητών κτλ.). Τα κριτήρια χωρισμού σε αρχεία είναι "ποιοτικά" με την έννοια πως ενα αρχείο περιλαμβάνει συγγενή στοιχεία (πχ. η κύρια συνάρτηση, η συνάρτηση αρχικοποίησης, οι ορισμοί παγκόσμιων μεταβλητών κτλ. σε ενα αρχείο, όλες οι μαθηματικές συναρτήσεις σε άλλο, οι βοηθητικές συναρτήσεις σε τρίτο κοκ.). Ενα αρχείο, αυτό το οποίο περιέχει την "κύρια" συνάρτηση (και, κατα προτίμηση, όλους ή τους περισσότερους ορισμούς παγόσμιων μεταβλητών) θεωρείται το "κύριο" αρχείο της εφαρμογής και καλό είναι να έχει και κατάλληλο όνομα (πχ. "main"). Ολα τα αρχεία της εφαρμογής περιλαμβάνονται σε ενα directory tree και αντίστροφα, ενα directory tree αντιστοιχεί σε μια

μόνο εφαρμογή. Σε κάθε τέτοιο directory tree (πχ. αυτό που αρχίζει απο το directory /usr/hydroscope/myprogram) υπάρχουν τα subdirectories bin (πχ. /usr/hydroscope/myprogram/bin) που περιέχει τον τελικό εκτελέσιμο κώδικα, src (πχ. /usr/hydroscope/myprogram/src) που περιέχει τον κώδικα της εφαρμογής, lib (πχ. /usr/hydroscope/myprogram/lib) που περιέχει τυχόν βιβλιοθήκες και άλλα βοηθητικά αρχεία όπως αρχεία μηνυμάτων, man (πχ. /usr/hydroscope/myprogram/man) που περιέχει τεκμηρίωση, examples (πχ. /usr/hydroscope/myprogram/examples) που περιέχει παραδείγματα, καθώς και κάθε άλλο αρχείο/directory κριθεί σκόπιμο. Συνιστάται επίσης και η παρουσία ενός αρχείου README (πχ. /usr/hydroscope/myprogram/README) το οποίο ο προγραμματιστής μπορεί να χρησιμοποιήσει για οποιαδήποτε πληροφορία για την εφαρμογή. Οι μικρές ή οι βοηθητικές εφαρμογές, τα προγράμματα σε SQL καθώς και σε UNIX shell, μπορούν παρόλα αυτά να ομαδοποιούνται σε ενα directory (πχ. bin). Τα ονόματα των αρχείων πρέπει να δίνουν μια εικόνα για το περιεχόμενο τους. Ο αριθμός των αρχείων, σε σχέση με το μέγεθος της εφαρμογής, δεν πρέπει να είναι ούτε μικρός (με λίγα πολύ μεγάλα αρχεία) ούτε μεγάλος (με πολλά πολύ μικρά αρχεία).

(στ) Σε περίπτωση που στοιχεία της εφαρμογής αποθηκεύονται στη ΒΔ (πχ. INGRES Windows/4GL frames) και όχι σε αρχεία, εννοείται πως ο παραπάνω κανόνας (ε) δεν ισχύει, αν και ισχύουν όσα αναφέρονται παρακάτω για τις επικεφαλίδες κτλ. Στην περίπτωση του κύριου "αρχείου", αυτό είναι το κύριο frame.

(ζ) Ολες οι επικεφαλίδες και τα άλλα στοιχεία της τεκμηρίωσης γράφονται σε μορφή σχολίων της οικείας γλώσσας προγραμματισμού. Οι χαρακτήρες σχολίου είναι οι `/*` (αρχή σχολίου) και `*/` (τέλος σχολίου) για όλες τις χρησιμοποιούμενες γλώσσες εκτός απο το UNIX shell, στο οποίο οι χαρακτήρες σχολίου είναι οι # (αρχή σχολίου μέχρι το τέλος της γραμμής).

(η) Κάθε αρχείο ή άλλο στοιχείο (πχ. frame) της εφαρμογής αρχίζει με την επικεφαλίδα του ΥΔΡΟΣΚΟΠΙΟΥ (σχ. 3.1).

(θ) Το κύριο αρχείο/frame της εφαρμογής αρχίζει με την επικεφαλίδα του κύριου αρχείου (σχ. 3.2), η οποία δίνει η οποία δίνει τις απαιτούμενες πληροφορίες για την εφαρμογή.

(ι) Κάθε αρχείο/frame της εφαρμογής να αρχίζει με την επικεφαλίδα αρχείου (σχ. 3.3), η οποία δίνει τις απαιτούμενες πληροφορίες για το αρχείο.

(ια) Πρίν απο κάθε συνάρτηση της εφαρμογής υπάρχει η επικεφαλίδα συνάρτησης (σχ. 3.4). η οποία δίνει τις απαιτούμενες πληροφορίες για τη συνάρτηση.

(ιβ) Αμέσως πριν ή μετά απο τον ορισμό (όχι την απλή δήλωση (πχ. σαν "extern" στη C)) μιας παγκόσμιας ή τοπικής μεταβλητής πρέπει να υπάρχει σχόλιο που να επεξηγεί τη χρήση και το περιεχόμενο της. Εξαιρούνται οι "τετριμμένες" μεταβλητές (πχ. δείκτες πινάκων, προσωρινές μεταβλητές κτλ.).

(ιγ) Οι κανόνες ονοματολογίας είναι οι εξής:

(i) όλες οι μεταβλητές και συναρτήσεις (όπου στις συναρτήσεις περιλαμβάνονται και οι "macros") της εφαρμογής πρέπει να έχουν περιγραφικά (στο μέτρο του δυνατού και λογικού) ονόματα που δίνουν σαφή εικόνα της χρήσης και του περιεχόμενου της, χωρίς βέβαια υπερβολές (πχ. πολύ μεγάλα ονόματα). Εξαιρούνται και πάλι οι τετριμμένες ή οι προσωρινές μεταβλητές με κατα προτίμηση σύντομα - αλλά και περιγραφικά, όπου είναι δυνατό- ονόματα (πχ. `i`, `j`, `k`, `flag1`, `tmp1`, `tmp2` κτλ.).

(ii) οι μεταβλητές και συναρτήσεις των οποίων το όνομα αποτελείται απο μια λέξη αρχίζουν απο πεζό γράμμα και μπορούν να περιέχουν και αριθμούς (πχ. `"main"`, `"func1"`, `"int2float"` κτλ.).

(iii) οι μεταβλητές και συναρτήσεις των οποίων το όνομα αποτελείται απο δύο ή περισσότερες λέξεις αρχίζουν με κεφαλαίο γράμμα και κάθε λέξη αρχίζει επίσης με κεφαλαίο (πχ. `"FunctionName"`, `"BigVariableName"`, `"MyVar2"` κτλ.).

(iv) οι σταθερές και οι δηλώσεις νέων τύπων γράφονται με κεφαλαία γράμματα. Ο χαρακτήρας `'_'` (underscore) μπορεί να χρησιμοποιηθεί για αύξηση της αναγνωσιμότητας (πχ. `"CONSTANT1"`, `"MYTYPE"`, `"MY_NEW_TYPE"` κτλ.).

(ιδ) Κάθε συνάρτηση πρέπει να εκτελεί μια και μόνο μια λειτουργία. Εξαίρεση αποτελεί η main function, η οποία έχει διπλό ρόλο: επικοινωνία με τον έξω κόσμο (πχ. command line parameters) και υλοποίηση της βασικής δομής του προγράμματος με κλήση των κατάλληλων "αρχικών" συναρτήσεων.

(ιε) Κάθε εφαρμογή πρέπει να περιλαμβάνει μια συνάρτηση init (initialisation) η οποία αρχικοποιεί τα στοιχεία της εφαρμογής (αρχικές τιμές μεταβλητών, άνοιγμα αρχείων, αρχική δέσμευση μνήμης κτλ.). Ολες οι μεταβλητές πρέπει να αρχικοποιούνται, ιδίως οι δείκτες (pointers - indexes).

(ιστ) Η αναγνωσιμότητα του κώδικα επιτυγχάνεται με χρήση κατάλληλων ονομάτων (βλ. (η)), με αποφυγή πολύπλοκων και ασαφών κατασκευών και, φυσικά, με χρήση σχολίων. Εκτός απο τις επικεφαλίδες, σχόλια πρέπει υποχρεωτικά να χρησιμοποιούνται σε σημεία που -δικαιολογημένα, για σημαντική πχ. αύξηση ταχύτητας- ο κώδικας είναι πολύπλοκος. Κατα τ΄ άλλα, η χρήση των σχολίων πρέπει να είναι μετρημένη και λογική, διότι τυχόν υπερβολή οδηγεί και πάλι σε έλλειψη αναγνωσιμότητας ([KNUTH]).

(ιζ) Ελεγχος τιμών επιστροφής απο συναρτήσεις, ιδίως όταν τυχόν λάθος σε αυτές μπορεί να σημαίνει αποτυχία του προγράμματος (πχ. αποτυχία "ανοιγματος" σημαντικού αρχείου).

(ιη) Ελεγχος "νόμιμων" - κανονικών δεδομένων εισόδου (πχ. δεδομένα χρήστη) για συμμόρφωση με τα αναμενόμενα

(ιθ) Η έξοδος καθώς και σε κάθε περίπτωση η επικοινωνία με το χρήστη (έξοδος - είσοδος) συνοδεύεται απο κατάλληλα επεξηγηματικά μηνύματα.

(κ) Αποφυγή όλων εκείνων των δομών ελέγχου που διακόπτουν την ομαλή ροή του προγράμματος: `goto`, `break` κτλ. Η χρήση τους δεν απαγορεύεται αν είναι μετρημένη και δικαιολογημένη, πχ. για έξοδο απο πολλούς διαδοχικούς βρόχους κτλ. Απο την άλλη, απαγορεύεται για την υλοποίηση της λογικής του προγράμματος (πχ. `"if ... goto ... else goto ..."`) και γενικά σαν ενεργό τμήμα της προγραμματιστικής τεχνικής ([WIRTH]).

(κα) Λειτουργία με βάση τις αρχές του δομημένου προγραμματισμού ([KNUTH]), ορισμένες απο τις οποίες περιγράφονται παρακάτω:

(i) αποφυγή goto

(ii) λειτουργική ομαδοποίηση σε συναρτήσεις (μια λειτουργία ανα συνάρτηση).

(iii) ευρεία χρήση δομών ελέγχου (πχ. `if - else if - else - endif`, `for`, `while`, `do - while`, `switch` κτλ.)

(iv) χρησιμοποίηση των τιμών επιστροφής (return values) των συναρτήσεων

(v) ομαδοποίηση συσχετισμένων μεταβλητών σε `structs (records)` και πίνακες `(arrays)`.

(vi) χρήση συμβολικών ονομάτων αντι για αυθαίρετους αριθμούς (πχ. `if var == CONST` αντί για `if var == 34`).

(vii) έλεγχος των συνθηκών τερματισμού βρόχων.

(viii) αποφυγή δημιουργίας πολύ μεγάλων και πολύπλοκων συνθηκών.

(ix) χρησιμοποίηση παρενθέσεων σε εκφράσεις για αποφυγή προβλημάτων προτεραιότητας τελεστών.

(χ) μία μόνο εντολή (statement) ανα γραμμή κώδικα. (πχ.
```
if(condition)
      statement
```
και όχι
```
if(condition) statement
```
κτλ.)

(κβ) Σε σχέση με τα command line arguments, αυτά πρέπει να αρχίζουν απο '-' αν είναι προαιρετικά. Αν πρέπει να ακολουθούνται απο δεύτερη λέξη (πχ. όνομα αρχείου) μεσολαβεί ενα κενό. Υποχρεωτικά command line arguments δεν αρχίζουν απο '-'. Μπορούν να είναι ολόκληρες λέξεις ή συντομογραφίες λέξεων ή μόνο γράμματα (πχ. "`program -argument1 -arg2 file1 -a -ab -argument5 file2 mandatory`" είναι μια μορφή command line).

```
/*
*****************************************************************
   HYDROSCOPE:   CREATION OF A NATIONAL DATABANK FOR
                 HYDROLOGICAL AND METEOROLOGICAL INFORMATION
*****************************************************************
*/
```

ΣΧ.3.1: ΕΠΙΚΕΦΑΛΙΔΑ ΥΔΡΟΣΚΟΠΙΟΥ

```
/*
** Application:
      Το όνομα της εφαρμογής
** Authors:
      Τα ονόματα των προγραμματιστών, ο φορέας στον οποίο ανήκουν
      και τα αρχικά τους
** Work:
      Εργασία στο πλαίσιο της οποίας αναπτύχθηκε η εφαρμογή
** Date:
      Ημερομηνία τελευταίας τροποποίησης
** Version:
      Τρέχουσα έκδοση
** Files:
      Αρχεία (modules) ή άλλα στοιχεία (πχ. frames, 4GL procedures
      κτλ.) απο τα οποία αποτελείται η εφαρμογή
** Language:
      Χρησιμοποιούμενες γλώσσες προγραμματισμού
** Purpose:
      Σκοπός της εφαρμογής
** History:
      Ιστορικό της ανάπτυξης των εκδόσεων της εφαρμογής, με
      ημερομηνίες, αρχικά προγραμματιστή και επιγραμματική
      περιγραφή προστιθέμενων χαρακτηριστικών
** Misc:
      Διάφορα άλλα σχόλια
*/
```

ΣΧ. 3.2: ΕΠΙΚΕΦΑΛΙΔΑ ΕΦΑΡΜΟΓΗΣ

```
/*
** File/Frame:
      Το όνομα του αρχείου ή του στοιχείου της εφαρμογής
** Application:
      Το όνομα της εφαρμογής στην οποία ανήκει το αρχείο
** Main File/Frame:
      Κύριο αρχείο της εφαρμογής
** Other Files/Dependencies:
      Αρχεία απο τα οποία εξαρτάται το παρόν αρχείο, πχ. include
      files
** Language:
      Η γλώσσα προγραμματισμού που χρησιμοποιείται στο
      αρχείο/frame
** Items:
      Συναρτήσεις και άλλα στοιχεία του προγράμματος που
      δηλώνονται στο αρχείο
*/
```

ΣΧ.3.3: ΕΠΙΚΕΦΑΛΙΔΑ ΑΡΧΕΙΟΥ

```
/*
** Function/Procedure:
      Το όνομα και ο τύπος της συνάρτησης
** Authors:
      Οι συγγραφείς της συνάρτησης (τα αρχικά τους)
** Date:
      Ημερομηνία τελευταίς τροποποίησης
** Purpose:
      Περιγραφή του σκοπού της συνάρτησης
** Arguments:
      Λίστα των παραμέτρων της συνάρτησης, των τύπων τους και του
      σκοπού τους
** Return/Exit values:
      Τι επιστρέφει η συνάρτηση, τόσο σαν return values όσο και σαν
      τροποποίηση παραμέτρων (call by reference)
** Algorithms:
      Σε περίπτωση που στη συνάρτηση χρησιμοποιούνται διάφοροι
      γνωστοί αλγόριθμοι (πχ. Runge-Kutta, quicksort κτλ.)
** Misc:
      Διάφορα άλλα σχόλια
*/
```

ΣΧ.3.4: ΕΠΙΚΕΦΑΛΙΔΑ ΣΥΝΑΡΤΗΣΗΣ

4 ΠΡΟΑΙΡΕΤΙΚΑ ΣΤΟΙΧΕΙΑ ΚΑΙ ΠΡΟΔΙΑΓΡΑΦΕΣ

4.1 Γενικά

Τα παρακάτω στοιχεία είναι προαιρετικά. Η χρήση τους δεν είναι δεσμευτική, απλά αποτελούν ενα σύνολο κανόνων με τους οποίους προτείνεται η συμμόρφωση, προκειμένου να ακολουθηθεί ενα κοινό "προγραμματιστικό στύλ".

4.2 Προαιρετικά στοιχεία και προδιαγραφές

(α) Λογική χρήση παγκόσμιων μεταβλητών, περιορισμένη σε αυτές ακριβώς τις μεταβλητές τις οποίες πρέπει να προσπελαύνουν όλες οι συναρτήσεις. Σε άλλη περίπτωση, χρήση τοπικών (στο παρόν αρχείο ή συνάρτηση) μεταβλητών.

(β) Συναρτήσεις με υπερβολικά πολλές παραμέτρους (περισσότερες απο 5 - 6) καλό είναι να αποφεύγονται.

(γ) Σύγκριση αριθμών κινητής υποδιαστολής (floating point), καθώς και δεικτών (pointers) καλό είναι να αποφεύγεται. Επίσης, καλό είναι να αποφεύγεται η σύγκριση τιμών διαφορετικού τύπου (πχ. integer με pointer), ιδίως χωρίς μετατροπή τύπου (casting). Σε κάθε περίπτωση σύγκρισης ή ανάθεσης τιμών διαφορετικού τύπου θα πρέπει να γίνεται μετατροπή.

(δ) Οταν είναι δυνατή, προτιμάται η χρήση του `switch / case` σε σχέση με το `if`, λόγω σαφέστερου κώδικα και αυξημένης ταχύτητας.

(ε) Οταν είναι εκ των προτέρων γνωστός ο αριθμός των επαναλήψεων προτιμάται το `for` απο το `while`.

(στ) Χρήση κατάλληλης στοίχισης (indentation) ώστε να είναι σαφή τα όρια της δομής ελέγχου. Η προτεινόμενη στοίχιση (όπου `'{'/'}'` είναι οι χαρακτήρες αρχής/τέλους της δομής ελέγχου στη C και τις INGRES/4GL και Windows/4GL, για τις οποίες ο συμβολισμός `'{'/'}'` προτιμάται απο τον `'BEGIN'/'END'`) έχει ως εξής (με ‹SPACE› εννοείται ενα κενό, με ‹TAB› εννοείται ενα ή περισσότερα TAB (διάστημα, 4 ή 8 κενοί χαρακτήρες), όσα απαιτούνται για να "ξεχωρίσει" σαφώς (περισσότερο απο ενα ‹SPACE›) η μια λέξη απο την προηγούμενη της):

(i) if/for/while/else/else if(condition){
‹TAB›statements;
}

πχ. (C)

```
if(var1 == CONST1) {
        x = y + z;
        w++;
}
else if(var1 == CONST2) {
        x = y - z;
        w--;
}
else
        x = z;
```

Εναλλακτικά, και σε σχέση με τη θέση του "{" / "BEGIN" μετά την εντολή της γλώσσας (πχ. if, for, while), εξίσου διαδεδομένη είναι και η μορφή:

command
{
‹TAB›statements
}

πχ. (C)

```
if(condition)
```

```
{
x = y;
z = x + w / 2;
}
```

(ii) switch(condition){
case‹SPACE›value:
‹TAB›statements;
‹TAB›break;
...
default:
‹TAB›statements;
‹TAB›break;
}

πχ. (C)

```
switch (var1) {
case CONST1:
        x = y + z;
        w++;
        break;
case CONST2:
        x = y - z;
        w--;
        break;
default:
        x = z;
        break;
}
```

(iii) (για δήλωση μεταβλητής)
type‹TAB›name1,‹SPACE›name2...

πχ. (C)

```
int   var1, var2;
char  var3[SIZE + 1], *p[SIZE];
struct      s {
        int   i;
        float f;
};
struct s    *var3, func1();
```

(iv) (για συνάρτηση)
type‹TAB›name(argument1,‹SPACE›argument2, ...,‹SPACE›argument-n)
type‹TAB›argument1;
...
{
‹TAB›local variables
‹BLANK LINE›
‹TAB›statements;
}

πχ. (C)

```
int   function(arg1, arg2)
char  arg1;
struct s    *arg2;
{
        int   LocalArray[SIZE], *p;
        char  var;

        var = arg1;
```

```
        ...
    }
```

(ν) (για εντολές SQL)
πχ.

```
SELECT column1, column2
FROM table1, table2
WHERE table1.column2 = table2.column3
      AND
      table2.column4 = table1.column5
```

(ζ) Οι τελεστές και εντολές της SQL γράφονται με κεφαλαία, ενω η χρήση της SQL σε συνδυασμό με άλλες γλώσσες (πχ. ESQL/C, 4GL) ακολουθεί τους κανόνες της SQL.

(η) Καταβάλλεται προσπάθεια κάθε συνάρτηση να έχει ενα -και μόνο ενα- σημείο εξόδου. Συναρτήσεις που δεν επιστρέφουν τίποτε πρέπει να έχουν μια εντολή "return" και να μην αφήνονται να φτάσουν στο τελευταίο '}' (ή 'END').

(θ) Προτιμάται η χρήση των τελεστών ανισότητας (πχ. '<') απο ισότητας (πχ. '==').

(ι) Αποφεύγεται η αναδρομή (recursion).

(ια) Αποφεύγονται οι πολύπλοκες εκφράσεις, ακόμα και αν η σωστή σειρά αποτίμησης τους (προτεραιότητα τελεστών) έχει εξασφαλισθεί μέσω παρενθέσεων.

(ιβ) Αναφορικά με τη συμμόρφωση (για τη γλώσσα C) με το πρότυπο ANSI-C ([ANSI]) (πχ. function prototypes, argument syntax κτλ.): Είναι επιλογή του προγραμματιστή αν θα χρησιμοποιήσει το "παραδοσιακό" στύλ ([K&R], 1st edition) ή το νέο, ωστόσο, λόγω της προς το παρόν πολλές φορές πλημμελούς υλοποίησης απο τα προγραμματιστικά περιβάλλοντα του προτύπου ANSI-C, προτιμάται το παραδοσιακό στύλ προγραμματισμού σε C.

5 ΠΑΡΑΔΕΙΓΜΑΤΑ

5.1 Γενικά

Ακολουθούν παραδείγματα για όλες τις γλώσσες προγραμματισμού, τα οποία επιδεικνύουν τόσο τα υποχρεωτικά όσο και τα προαιρετικά στοιχεία της ανάπτυξης λογισμικού. Ιδιαίτερη προσοχή πρέπει να δοθεί στην ονοματολογία, τις επικεφαλίδες και τη χρήση των σχολίων.

5.2 Γλώσσες C και ESQL/C

```
/*
**************************************************************
   HYDROSCOPE:   CREATION OF A NATIONAL DATABANK FOR
                 HYDROLOGICAL AND METEOROLOGICAL INFORMATION
**************************************************************
*/
/*
** Application:
      myprogram
** Authors:
      C. Programmer - NTUA (CP)
      P. Programmer - NMS (PP)
** Work:
      GENERAL ANALYSIS 11: Coding Standards and Specifications
** Date:
      31/8/92
** Version:
      2.1
** Files:
      main.c, myesqlc.sc, myinclude.h
** Language:
      C, ESQL/C
** Purpose:
      This is not a real application. It is just segments of
      code to demonstrate programming style etc.
** History:
      31/7/92, ver. 1.0, CP: Initial version
      5/8/92, ver. 1.1, CP:  Bugs correction
      15/8/92, ver. 1.2, PP: Added capability C1
      30/8/92, ver. 2.0, PP: Added capabilities C2 & C3
      31/8/92, ver. 2.1, PP: Bugs correction and added
            capabilities C4 & C5
** Misc:
      Usage (command line arguments, if they exist) here:
      myprog [-a arg1] [-arg2] [-arg3 file1] [-b] [-arg4] file2
      -a arg1:    argument -a (one letter). If it exists it must be
                  suceeded by argument (eg. a file name) arg1.
      -arg2:      argument -arg2 (a word).
      -arg3 file1:      argument -arg3 succeded by file name file1
      -b:         argument -b.
      -arg4:      argument -arg4
      file2:      mandatory argument file2 (optional argument in
                  [ ... ]).
*/
/*
** File:
      main.c
```

```
** Application:
      myprogram
** Main file:
      main.c
** Other files/Dependencies:
      myinclude.h
** Language:
      C
** Items:
      Types:
            struct      s1
      Globals:
            int   array1[ARRAY1_SIZE]
            NEWTYPE     array2[ARRAY2_SIZE]
            float       var1
            struct s1   var2, MyOwnVar, *StructS1Pointer
      Functions:
            void  main()
            int   FunctionName()
** Misc:
*/

/* 1. Standard include files */
#include   <stdio.h>
#include   <sys/time.h>

/* 2. Application include files */
#include   "myinclude.h"

/* 3. (Local) types definition */
struct      ls{
      float       FloatStructMember10, f20;
      int   IntStructMember;
};

/* 4. Global variables definition */
int   array1[ARRAY1_SIZE],          /* Dummy global variable */
      FunctionName();               /* Dummy function */
NEWTYPE     array2[ARRAY2_SIZE];  /* Another dummy global
                                     variable */
float       var1;                 /* Yet another dummy global
                                     variable */
struct s1   var2,                 /* ... */
            MyOwnVar,             /* ... */
            *StructS1Pointer;     /* pointer... */

/* 5. Functions */
/*
** Function:
      void  main()
** Authors:
      CP, PP
** Date:
      31/8/92
** Purpose:
      Main function. Interfaces with outer world via command
      line parameters and implements main program logic via
      the proper controlling loops and function calls.
```

```
** Arguments:
      int   argc:       number of command line arguments (In)
      char  *argv[]:    vector of command line arguments (In)
** Return/Exit values:
      No return values
      Exit values:
            0:    normal completion
            1:    error #1
            2:    error #2
** Algorithms:
** Misc:
*/
void  main(argc, argv)
int   argc;
char  *argv[];
{
      int   i, j, s,
            MyIntegerVariable;     /* Dummy variable */
            ParseArguments();      /* Function to parse command
                                      line arguments */
      char  s[CONST3],             /* Another dummy variable */
            *sp,                   /* Yet another dummy variable */
            *StringArray[CONST4]; /* ... */

      /* First parse command line arguments and act
         accordingly (set flags etc.) */
      while((s = ParseArguments(argc, argv)) > 0){
            if((s > CONST1) && (s != CONST3))
                  if(DoSomething() == FALSE){
                        fprintf(stderr, "Error.\n");
                        exit(2);
                  }
            switch(s){
            case CONST3:
                  DoSomething1();
                  break;
            case CONST:
                  DoSomethingElse();
                  break;
            default:
                  exit(1);
                  break;
            }
      }
      /* Initialise (give global variables initial values,
         open files etc.) */
      init();
      /* Main program logic follows */
      for(i = 0; i < CONST; i++){
            array1[i] = MyIntegerVariable = DoSomething();
            var2.FloatStructMemeber1 = CONST2 *
                  var1 - array2[i + j] + (float) k;
      }
      while(TRUE){            /* For ever */
            func1(var2);
            if(FunctionName(&MyIntegerVariable, &var1, k)
                  != TRUE){
                        printf("Terminated.\n");
```

```
                        exit(0);
                }
        }
}

/*
** Function:
        int   FunctionName()
** Authors:
        CP
** Date:
        31/8/92
** Purpose:
        Do nothing, just illustrate various aspects of programming
        style, specifications and rules (especialy for C).
** Arguments:
        int   *arg1:                    variable to be changed (In/Out)
        float       *FloatVarP:         a float variable (Out)
        int   k:                        some index (In)
** Return/Exit values:
        TRUE:       if success
        FALSE:      if not
** Algorithms:
        Using some strange new algorithm
** Misc:
*/
int   FunctionName(arg1, FloatVarP, k)
int   *arg1, k;
float       *FloatVarP;
{
        struct s1   LocalVar,           /* Some local variable */
                    *S1Pointer;         /* and a pointer to it */

        /* Various struct/struct pointers assignement and
          expressions */
        LocalVar.f2 = CONST2;
        S1Pointer = &LocalVar;
        *FloatVarP = S1Pointer->f20 - CONST2 * (CONST % 3);
        if((*arg1 > 0) && (k < CONST)){
                /* Shift left *arg1 CONST3 | CONST1 bits, that is
                   (CONST3 = 3 or 4, CONST1 = 1) 3 or 5 bits,
                   depending on value of CONDITION1 (myinclude.h) */
                *arg1 <<= CONST3 | CONST1;
                S1Pointer->UnionStructMember.LongStructMember1 =
                        (long) *arg1;
        }
        else if(k > CONST)
                return(FALSE);
        else{
                *arg1 = k--;
                return(*arg1 + (int) LocalVar.LongStructMember1);
        }
        return(TRUE);
}
```

```
/*
*****************************************************************
   HYDROSCOPE:   CREATION OF A NATIONAL DATABANK FOR
                 HYDROLOGICAL AND METEOROLOGICAL INFORMATION
*****************************************************************
*/
/*
** File:
      myesqlc.sc
** Application:
      myprogram
** Main file:
      main.c
** Other files/Dependencies:
      myinclude.h
** Language:
      ESQL/C
** Items:
** Misc:
*/

/* 1. Standard include files */
#include   <stdio.h>
#include   <sys/types.h>

/* 2. Application include files */
#include   "myinclude.h"

/* 3. Mandatory ESQL/C include file */
exec sql    include    sqlca;

/* 4. ESQL/C global definitions */
exec sql    begin      declare section;
char  data[ARRAY1_SIZE + 1];       /* Array where some character
                                      data will be stored */
struct s1   MyESQLCVar;            /* Another dummy variable */
exec sql    end   declare section;

/* Cursor definition */
exec sql    declare    curs  for
      SELECT                       /* SQL statement */
            empname, salequantity, empno, dept
      FROM
            department, sales, employee
      WHERE
            sales.empno = department.empno
            AND
            department.name = employee.name;

/* 5. ESQL/C Error Handling */
exec sql    whenever sqlerror      call Abort;
exec sql    whenever not found     call NotFound;

/* 6. Other definitions */
static char *     StringTable[] = {      /* Initialisation */
      "String One", "String Two", "String Three",
      "String Four", "String Five",
      (char *) 0
```

```
};

/* 7. Functions */
/*
** Function:
      void  func1()
** Authors:
      PP, CP
** Date:
      31/8/92
** Purpose:
      Do nothing, just illustrate various aspects of programming
      style, specifications and rules (especialy for ESQL/C).
** Arguments:
      struct s1   var:        some input variable (In)
** Return/Exit values:
      None
** Algorithms:
** Misc:
*/
void  func1(var)
struct s1   var;
{
      int   i, k, t = FALSE,
            rc,               /* Return Code */
            *p,               /* Pointer to some integer */
            SomeOtherVar;     /* Dummy */
      /* ESQL/C local definitions */
      exec sql    begin declare section;
      struct      sss{
            char  name[CONST + 1];  /* Employee name */
            long  sales;            /* Employee sales */
            int   employee,         /* Employee number */
                  dept;             /* Employee department */
      };
      struct sss  NameRecord;       /* Fetch from cursor */
      exec sql    end declare section;

      InitDB();                     /* Open database etc. */
      for(i = 0; i < CONST && t != TRUE; i++, k--){
            exec sql    fetch curs into :NameRecord;
            if(employee < 0)
                  return;
            exec sql    UPDATE
                              emptable
                        SET
                              empsal = empsal * 1.2    /* 20% raise */
                        WHERE
                              /* Condition missing */
            ...
      }
      return;
}
```

```
/*
*****************************************************************
   HYDROSCOPE:   CREATION OF A NATIONAL DATABANK FOR
                 HYDROLOGICAL AND METEOROLOGICAL INFORMATION
*****************************************************************
*/
/*
** File:
      myinclude.h
** Application:
      myprogram
** Main file:
      main.c
** Other files/Dependencies:
** Language:
      C
** Items:
      Defines:
            CONST1, CONST2, PI, YOUR_STRING, MYSTRING,
            CONST3, VERY_BIG_CONSTANT_NAME, CONST, ARRAY1_SIZE,
            ARRAY2_SIZE
      Macros:
            MacroName1(), macro2()
      Types:
            NEWTYPE
** Misc:
*/

/* 1. Constant and macro defines */
#define     CONST1      1
#define     CONST2      2.0
#define     PI          3.14

#define     YOUR_STRING      "This is your string\n"
#define     MYSTRING    "This is my string\n"

/* Conditional compilation */
#ifdef CONDITION1
#define     CONST3      3
#define     VERY_BIG_CONSTANT_NAME      \
      "Very big string define for a very big constant"
#else
#define     CONST3      4
#endif

#define     CONST       40
#define     ARRAY1_SIZE      100
#define     ARRAY2_SIZE      50

#define     MacroName1(a)          (a * a)

#if CONDITION2 && !CONDITION3
#define     macro2(a, b)      (a > b ? a + b : a - b)
#elif CONDITION3
#define     macro2(a, b)      (a > b ? a - b : a + b)
#else
#define     macro2(a, b)      (a > b ? a - b : a * b)
#endif
```

```
/* 2. New (global) types definition */
typedef     struct s{
      int   i;
      float f;
}     NEWTYPE;

struct      s1{
      float       FloatStructMember1, f2;
      union       u1{
            long  LongUnionMember1;
            double      DoubleUnionMember1;
      }     UnionStructMember;
};

/* 3. Global variables and functions declarations (extern) */
extern int  array1[];
extern NEWTYPE    array2[];
extern void       func1();
```

5.3 Γλώσσα INGRES Windows/4GL και INGRES/4GL

```
/*
*****************************************************************
   HYDROSCOPE:   CREATION OF A NATIONAL DATABANK FOR
                 HYDROLOGICAL AND METEOROLOGICAL INFORMATION
*****************************************************************
*/
/*
** Frame:
      main
** Application:
      parts
** Authors:
      PP
** Date:
      31/8/92
** Purpose:
      This frame displays a list of parts from the parts table in
      the database, and allows the interactive user to select any
      number of parts from the table for display and optional
      update.It records in the 'DisplayIndicator' column of the
      table field whether or not the picture is currently being
      displayed. A global variable, called 'UpdatesMade' is used
      to record whether any data has been updated, and if so to
      commit the work to the database.
** Arguments:
** Return/Exit values:
** Algorithms:
** Misc:
*/
initialize(i = integer) =
{
      /* Initialize the path for the picture files for updating */
      SELECT      :PathForPictures = BitmapPathName
      FROM  PathToSystem
      WHERE opsystem = :CurSession.OperatingSystem;
      COMMIT;
      /* Initialize the other global data */
      UpdatesMade = FALSE;
      /* Now read in the parts data, and initialize the
        display_indicator column in the table field */
      i = 1;
      SELECT DISTINCT  :PartTable[i].partno = partno,
            :PartTable[i].ShortDescription = ShortDesc
      FROM  Part
      {
            i = i + 1;
      };
};

/* When the display button is pressed, or when the middle mouse
   button is clicked while positioned over the table field;
   1. open the new window (DisplayFrame) and display picture of
      the part that is currently selected along with its name.
   2. turn the display indicator on.
*/
on click    DisplayButton =
```

```
{
      /* The [] refers to the current row in part_table */
      if PartTable[].DisplayIndicator = TRUE then
            /* Frame is already displayed */
            CurFrame.InfoPopup(MessageText =
                  'Part Detail Frame already open.');
      else
            openframe DisplayFrame(partno = PartTable[].partno,
                  RowNumber = field(PartTable).CurRow,
                  ParentFrame = CurFrame);
            PartTable[].DisplayIndicator = TRUE;
      endif;
};

/* The UserEvent 'IndicatorOff' is sent by the DisplayFrame when
   is closed to direct the MainFrame to turn off the display
   indicator for the row.
*/
on userevent      'IndicatorOff' =
{
      /* The row number will be returned in the messageinteger
         attribute */
      PartTable[CurFrame.MessageInteger].DisplayIndicator =
            FALSE;
};
```

```
/*
*****************************************************************
   HYDROSCOPE:   CREATION OF A NATIONAL DATABANK FOR
                 HYDROLOGICAL AND METEOROLOGICAL INFORMATION
*****************************************************************
*/
/*
** Frame:
      RegionDetail
** Application:
      sales
** Authors:
      CP
** Date:
      31/8/92
** Purpose:
      This is called up by the SummaryMap frame.  It takes the
      RegionName given and gets the set of salespeople from the
      database for that region in the year 1988 and displays them
      in the table field.  For salespeople who made over 100% of
      quota, part of the row is shaded and a 'good job' message is
      displayed in the row.  This uses the special CellAttributes
      for table field cells. When this closes, it sends a message
      back to the mapbw frame so that the highlighted map can be
      unhighlighted.
** Arguments:
      varchar(30)      RegionName:      name of the region
      ImageTrim  RegionMap:             map field object of region
      FrameExec  ParentFrame:           parent caller
** Return/Exit values:
** Algorithms:
** Misc:
*/
initialize(RegionName = varchar(30), ParentFrame = FrameExec,
      RegionMap = ImageTrim,
      i = integer4,                 /* Temp variable */
      j = integer4) =
{
      /* Load up table field */
      i = 1;
      REPEATED SELECT  :SalesmanTable[i].SalesmanName = sh.name,
            :SalesmanTable[i].SalesmanPicture.DBHandle =
                  sh.portrait,
            :SalesmanTable[i].QuotaSubform.QuotaBar =
                  100 * (sl.sales / sl.quota),
            :SalesmanTable[i].QuotaSubform.QuotaValue =
                  100 * (sl.sales / sl.quota)
      FROM  SalesmanHeader sh, SalesHeader sl
      WHERE sh.region = :RegionName
            AND   sh.number = sl.SalesmanNumber
            AND   sl.year = '1988'
      {
            if SalesmanTable[i].QuotaSubform.QuotaBar > 100 then
                  SalesmanTable[i].QuotaSubform.QuotaCongrats =
                        'GOOD JOB!!';
                  field(SalesmanTable[i].QuotaSubform).Bg =
                        FP_SHADE;
            else
```

```
                    SalesmanTable[i].QuotaSubform.QuotaCongrats =
                        '';
                    field(SalesmanTable[i].QuotaSubform).Bg =
                        FP_CLEAR;
                endif;
                i = i + 1;
        };

        /* move map to image field */
        RegionImage = RegionMap.Image.Duplicate();
};

on click    CloseButton =
{
        ParentFrame.SendUserEvent(eventname = 'CLOSE_REGION',
            MessageObject = RegionMap);
        return;
};
```

```
/*
*****************************************************************
   HYDROSCOPE:   CREATION OF A NATIONAL DATABANK FOR
                 HYDROLOGICAL AND METEOROLOGICAL INFORMATION
*****************************************************************
*/
/*
** Procedure:
      GenerateData
** Application:
      linegraph
** Authors:
      PP
** Date:
      31/8/92
** Purpose:
      Given a range of integer values, generate a more or less
      random integer between the two of them. This uses some
      database queries and the random_integers global array
      (which is initialized the first time through) to generate
      the value.
** Arguments:
      integer    LowValue:        low end of range.
      integer    HiValue:         high end of range.
      integer    StartAgain:      set to TRUE if first time.
** Return/Exit values:
      integer    result:     value between lowvalue and highvalue.
** Algorithms:
** Misc:
*/
procedure  GenerateData(LowValue = integer not null,
      HiValue = integer not null,
      StartAgain = integer not null,
      ValueRange = integer not null,      /* Temp value for range */
      result = integer not null) =
{
      /* If first time through, set up the RandomIntegers array */
      if StartAgain = TRUE then
            RandomIntegers[1].Value = 7;
            RandomIntegers[2].Value = 2;
            RandomIntegers[3].Value = 5;
            RandomIntegers[4.Value = 4;
            RandomIntegers[5].Value = 9;
            RandomIntegers[6].Value = 1;
            RandomIntegers[7].Value = 3;
            RandomIntegers[8].Value = 4;
            RandomIntegers[9].Value = 8;
            RandomIntegers[10].Value = 6;
            NextRandomInteger = 1;
      endif;
      /* Now get a data value. For random effect, this example uses
        a number of measures to randomize */
      REPEATED SELECT  :result = _cpu_ms() + _bio_cnt() +
            _pfault_cnt();
      result = result * RandomIntegers[NextRandomInteger].Value;
      NextRandomInteger = NextRandomInteger + 1;
      if NextRandomInteger > 10 then
            NextRandomInteger = 1;
```

```
        endif;
        /* Now normalize the value to the range given */
        ValueRange = HiValue - LowValue;
        result = mod(Result, ValueRange);
        result = HiValue - Result;
        return result;
};
```

5.4 Γλώσσα SQL

```
/*
*****************************************************************
   HYDROSCOPE:   CREATION OF A NATIONAL DATABANK FOR
                 HYDROLOGICAL AND METEOROLOGICAL INFORMATION
*****************************************************************
*/
/*
** Query:
      query1
** Application:
      mysqlapp
** Authors:
      CP
** Date:
      31/8/92
** Purpose:
      Select the employee number (empno), employee name
      (empname), department name (deptname) and the
      quantity of sales (salequant) this employee (salesman)
      achieved if this quantity is > 1000 or < 200, ordered by
      the number of the employee
** Misc:
*/
SELECT      empno, empname, deptname, salequant
FROM  employee, department, sales
WHERE (sales.salequant > 1000 OR sales.salequant < 200)
      AND   sales.emp = employee.empno
      AND   employee.dept = department.deptno
ORDER BY    salequant;

/*
** Query:
      query2
** Application:
      mysqlapp
** Authors:
      PP
** Date:
      31/8/92
** Purpose:
      Select the employee name (empname) for all employees with
      a salary bigger than the average salary who work on a 3rd
      floor department and their name begins with 'Alex'
** Misc:
*/
SELECT      empname
FROM  employee emp1, department
WHERE salary >
      (SELECT     avg(salary)
      FROM  employee emp2
      WHERE emp1.dept = emp2.dept)
      AND   empname like 'Alex%'
      AND   deptno in
            (SELECT     deptno
            FROM  department
            WHERE department.floor = 3);
```

```
/*
** Query:
      query3
** Application:
      mysqlapp
** Authors:
      PP
** Date:
      31/8/92
** Purpose:
      Give a 20% raise to all employees who work for manager
      'Smith'
** Misc:
*/
UPDATE      employee
SET   salary = 1.2 * salary
WHERE dept in
      (SELECT     deptno
      FROM  department
      WHERE department.manager in
            (SELECT     empno
            FROM  employee
            WHERE empname = 'Smith'));
```

5.5 UNIX shell & tools

```
:
#
#*************************************************************
#   HYDROSCOPE:  CREATION OF A NATIONAL DATABANK FOR
#                HYDROLOGICAL AND METEOROLOGICAL INFORMATION
#*************************************************************
#
#
# Application:
#     lastlogin
# Authors:
#     C. Programmer - NTUA (CP)
# Work:
#     GENERAL ANALYSIS 11: Coding Standards and Specifications
# Date:
#     31/8/92
# Version:
#     1.0
# Files:
#     lastlogin.sh
# Language:
#     Bourne shell
# Purpose:
#     Determine when user last logged in
# History:
#     31/8/92, ver. 1.0, CP: Initial Version
# Misc:
#     For this program to work, the existence of a .lastlogin
#     file in the user's home directory is required. The file's
#     modication date is updated by login(1) each time the user
#     logs in
while [ $# -gt 0 ]; do
      user=$1
      # Get the user's home directory from /etc/passwd
      home=`egrep "^$user" /etc/passwd | awk -F: '{ print $6} '`
      if [ -z "$home" ]; then
            echo "${user} : no such user"
            shift
            continue
      fi
      file=$home/.lastlogin
      if [ ! -r $file ]; then
            echo "${user} : never logged in"
      else
            # Print date field from ls -l
            when=`ls -l $file | awk '{ print $6 " " $7 " " $8} '`
            echo "${user} : ${when}"
      fi
      shift
done
```

```
:
#
#*************************************************************
#   HYDROSCOPE:  CREATION OF A NATIONAL DATABANK FOR
#                HYDROLOGICAL AND METEOROLOGICAL INFORMATION
#*************************************************************
#
#
# Application:
#     filesets
# Authors:
#     C. Programmer - NTUA (CP)
# Work:
#     GENERAL ANALYSIS 11: Coding Standards and Specifications
# Date:
#     31/8/92
# Version:
#     1.0
# Files:
#     filesets.sh
# Language:
#     Bourne shell
# Purpose:
#     Determine which filesets are loaded on the system's disk
# History:
#     31/8/92, ver. 1.0, CP: Initial Version
# Misc:
#     This program works only on HP/UX systems, which load
#     software in the form of filesets, described in
#     subdirectories of /system eg. /system/UX-CORE etc.
#
if [ $# -lt 1 ]; then
      echo "usage: $0 fileset [fileset]"
      exit 1
fi
prog=$0
cd /system
while [ $# -ge 1 ]; do
      fileset=$1
      if [ ! -d $1 ]; then
            echo "$prog: invalid fileset $fileset"
            shift
            continue
      fi
      cd $fileset
      echo "-- ${fileset} :"
      # If no index file, just continue
      if [ ! -r index ]; then
            shift
            cd ..
            continue
      fi
      # Use awk to parse index file info
      cat index | awk '
            BEGIN             { depindex=1 }
            /^fd:/            { split($0, fd, ":\t") }
            /^pn:/            { split($0, pn, ":\t") }
            /^pd:/            { split($0, pd, ":\t") }
```

```
            /^fs:/              { split($0, fs, ":\t") }
            /^dep:/             { split($0, dep, ":\t");
                                  alldep[depindex++] = dep[2] }
            END                 { printf("\tSIZE=%d. FILESET
DESCR=%s.\n\tPARTITION=%s. PARTITION DESCR=%s.\n",
                                  fs[2] / 1024, fd[2], pn[2], pd[2]);
                                  if(depindex != 1)
                                      for(i in alldep)
                                          printf("\tDEPEND=%s.\n",
                                              alldep[i]) } '
      cd ..
      shift
done
```

6 ΒΙΒΛΙΟΓΡΑΦΙΑ

[K&R] Kernighan, B. - Ritchie, D., *The C Programming Language*,
1st edition, Prentice Hall, 1978
2nd edition, Prentice Hall, 1987

[ANSIC] American National Standards Institute, *The C Programming Language*,
ANS X3.159-1989, 1989

[POSIX] Institute of Electric and Electronics Engineers, *Portability Guide*,
IEEE 1003.1-1988 POSIX.1-1988, 1988
IEEE 1003.1-1990 POSIX.1-1990, 1990

[SVID] American Telegraph and Telephone, *System V Interface Definition*,
Volumes 2 and 3, AT&T, 1985 - 1986

[KNUTH] Knuth, D., *The Art of Computer Programming*,
Volumes 1 - 3, Prentice Hall, 1972 - 1975

[WIRTH] Wirth, N., *Algorithms + Data Structures = Programs*,
Addison - Wesley, 1975

[INGRES] ASK/Ingres Corporation, *INGRES RDBMS and Tools Manuals*,
ASK/Ingres, 1991

[PIKE] Pike, R. - Kernighan, B., *The UNIX Programming Environment*,
Prentice Hall, 1984

[ROCH] Rochkind, M., *Advanced UNIX Programming*,
Prentice Hall, 1984