14.

Αξιολόγηση - Τεκμηρίωση

Εισαγωγή

Η αξιολόγηση είναι το τελευταίο στάδιο στην παραγωγή ενός προγράμματος. Σ' αυτό το κεφάλαιο θα αναφέρουμε τα κριτήρια βάση των οποίων αξιολογείται ένα πρόγραμμα, θα δώσουμε διάφορα παραδείγματα και θα απαντήσουμε ερωτήσεις όπως "γιατί αυτή η λύση είναι προτιμότερη από την άλλη", στηρίζοντας την απόφασή μας σε αυτά τα κριτήρια. Θα συζητήσουμε την αναγκαιότητα, τη σημασία, το σκοπό και τις κατηγορίες της τεκμηρίωσης, καθώς και τη σημασία της κατά φάση συντήρησης του προγράμματος. Θα δώσουμε οδηγίες για την σύνταξη ενός VTOC (Visual Table Of Contains), μιας διαγραμματικής τεχνικής που χρησιμοποιείται στην τεκμηρίωση ενός προγράμματος. Θα συζητήσουμε για τις φάσεις ανάπτυξης ενός προγράμματος και θα δούμε πως σχετίζονται με τις φάσεις του κύκλου ζωής ενός προγράμματος.

Διδακτικοί στόχοι

Στόχοι του κεφαλαίου αυτού είναι οι μαθητές:

- να διατυπώνουν για το ίδιο πρόβλημα εναλλακτικές προγραμματιστικές λύσεις, να τις συγκρίνουν και να τις αξιολογούν με βάση προκαθορισμένα κριτήρια.
- να προσδιορίζουν τα όρια χρήσης κάθε προγράμματος που δημιουργούν.
- να αναζητούν και να διερευνούν τις δυνατότητες επέκτασης των προγραμμάτων που δημιουργούν (νέες πρόσθετες λειτουργίες, κ.λπ.).
- να αξιολογούν τα προγράμματα που δημιουργούν.
- να διακρίνουν ποιες είναι οι αναγκαίες σημειώσεις για την τεκμηρίωση του προγράμματος.
- να συντάσσουν το φάκελο τεκμηρίωσης της εφαρμογής.
- να διακρίνουν τα στάδια του κύκλου ζωής ενός προγράμματος

Προερωτήσεις

- ✓ Πιστεύεις ότι το τέλος της δημιουργίας ενός προγράμματος, είναι το να το θέσεις σε παραγωγή;
- ✓ Νομίζεις ότι η λύση του προβλήματος που διατύπωσες είναι μοναδική;
- ✓ Γνωρίζεις τι είναι πακέτο λογισμικού και ποια η διαφορά του με το πρόγραμμα;
- ✓ 'Εχεις ακούσει για τον φάκελο τεκμηρίωσης προγράμματος;

14.1 Κριτήρια αξιολόγησης προγράμματος

‘Ενα πρόγραμμα αξιολογείται όχι μόνο από το αν λειτουργεί ή όχι, πράγμα αυτονόητο, αλλά και από το αν πληρεί κάποια κριτήρια.

‘Ενα πρόγραμμα όπως όλα τα προϊόντα αξιολογείται και από τον κατασκευαστή του αλλά και από το χρήστη του. Επομένως τα κριτήρια αξιολόγησης πρέπει να καλύπτουν και τις δύο πλευρές. ‘Ετσι ένα ποιοτικό λογισμικό πρέπει να έχει:

Οι προτεινόμενες λύσεις των προβλημάτων του κεφαλαίου, ακολουθούν σε μεγάλο βαθμό τα κριτήρια αξιολόγησης προγράμματος, προσπαθώντας να κρατήσουν μια ισορροπία μεταξύ τους.

- ✓ Απλότητα – Τυπικότητα (απλές δομές, ίδια αντιμετώπιση ίδιων προβλημάτων)
- ✓ Φιλικότητα (ανοχή στα λάθη, πληροφόρηση και πολλά άλλα)
- ✓ Ευελιξία (δυνατότητα επεκτάσεων και τροποποιήσεων)
- ✓ Αξιοπιστία (εκτέλεση χωρίς λάθη)
- ✓ Ταχύτητα

Τα κριτήρια της απλότητας και τυπικότητας, της ευελιξίας, της αξιοπιστίας και της ταχύτητας, αφορούν τεχνικά θέματα και είναι τα κριτήρια που χρησιμοποιεί ο προγραμματιστής, ο κατασκευαστής του προγράμματος, προκειμένου να αξιολογήσει το έργο του. Το κριτήριο της φιλικότητας είναι το κατ’ εξοχήν κριτήριο που αξιολογείται από το χρήστη της εφαρμογής. Το τελευταίο λόγω της πολυπλοκότητας του έχει αναπτυχθεί διεξοδικά σε ξεχωριστό κεφάλαιο. Εδώ θα αναπτύξουμε και θα δώσουμε παραδείγματα για τα υπόλοιπα.

Αρκετές φορές προσπαθώντας να τηρήσει κανείς τα κριτήρια αυτά, βρίσκεται μπροστά σε διλήμματα. Πράγματι στην προσπάθεια να ικανοποιηθεί κάποιο από αυτά, μπορεί το πρόγραμμά να υστερήσει σε κάποιο άλλο. Σ’ αυτές τις περιπτώσεις, η πείρα σε συνδυασμό με τη σπουδαιότητα του συγκεκριμένου χαρακτηριστικού, θα δώσουν την απάντηση στο δίλημμα.

14.1.1 Απλότητα - τυπικότητα

Στα χρόνια που ο σχεδιασμός ενός προγράμματος ήταν τέχνη, κάθε αλγόριθμος έφερνε την “υπογραφή” του κατασκευαστή του. Συνήθεια της εποχής ήταν η υιοθέτηση περίπλοκων τεχνικών (κάτι σαν πνευματικές ασκήσεις), που άλλοτε είχαν στόχο να λύσουν υπαρκτά προβλήματα, άλλοτε είχαν στόχο το πρόγραμμα να είναι ακατανόητο για οποιονδήποτε τρίτο και άλλοτε να δείξουν την επιδεξιότητα του προγραμματιστή. Η πορεία έδειξε ότι οι λύσεις που παράγονταν με αυτό τον τρόπο, δεν ήταν οι καλύτε-

ρες. Οι ανάγκες συντήρησης των προγραμμάτων, αλλά και η εξέλιξη του υλικού, έφεραν σταδιακά αυτήν την τακτική στο περιθώριο. Στο προσκήνιο βρέθηκε η απλότητα. Σήμερα πλέον στα προγράμματα που κατασκευάζονται:

Το απλό είναι ωραίο !!

- ✓ Εύκολα γίνεται αντιληπτό τι λειτουργίες εκτελούν.
- ✓ Εύκολα μπορούν να συμπληρωθούν από άλλους.

Παρ όλα αυτά και σήμερα προικισμένοι προγραμματιστές, ολισθαίνουν προς το σχεδιασμό ενός περίπλοκου προγράμματος, χρησιμοποιώντας συνήθως "ισχυρότερες" δομές από αυτές που είναι αναγκαίες.

Με το επόμενο παράδειγμα δίνονται για το ίδιο πρόβλημα τρεις διαφορετικές λύσεις και σχολιάζονται κατά πόσον ικανοποιούν το κριτήριο της απλότητας και της τυπικότητας.

Παράδειγμα 1

Δίδεται ταξινομημένος μονοδιάστατος πίνακας ακεραίων αριθμών 1000 θέσεων. Ζητείται να κατασκευαστεί πρόγραμμα, το οποίο να βρίσκει τη συχνότητα εμφάνισης κάθε αριθμού του πίνακα. Τα αποτελέσματα να εμφανίζονται στην οθόνη.

Ένας απλός και τυπικός αλγόριθμος θα πρέπει:

- Να δίνει τις ίδιες τιμές στις μεταβλητές, είτε είναι η αρχή του προγράμματος, είτε πρόκειται για νέο αριθμό.
- Αυτό που γίνεται σπανιότερα (<>), εδώ η αλλαγή του αριθμού, να ξεχωρίζει μέσα σε ένα **ΑΝ**, ενώ αυτό που γίνεται συνήθως (=) να βρίσκεται στην κανονική ροή του προγράμματος.
- Ένα **ΑΝ** μπορεί να πραγματοποιεί τον επιθυμητό έλεγχο.

Έστω ότι τα περιεχόμενα του πίνακα είναι 2, 2, 3, 3, 3, 3, 5, 6, 6, 6, 6, 6, 7, 8, 8, 8, 8, 8, 8, 9, 9, 9, 9, 9, 9. Τότε το πρόγραμμα θα πρέπει να εμφανίσει τα εξής αποτελέσματα:

Αριθμός	Συχνότητα
2	2
3	4
5	1
6	5
7	1
8	6
9	6

Τα επόμενα τρία προγράμματα λύνουν το πρόβλημα.

```
ΠΡΟΓΡΑΜΜΑ Συχνότητα1
ΜΕΤΑΒΛΗΤΕΣ
    ΑΚΕΡΑΙΕΣ: i, S, Προηγ_Α, Α[1000]
ΑΡΧΗ
i <- 1
S <- 0
Προηγ_Α <- Α[i]
ΟΣΟ i < 1001 ΕΠΑΝΑΛΑΒΕ
    ΑΝ Προηγ_Α <> Α[i] ΤΟΤΕ
            ΓΡΑΨΕ Προηγ_Α, S
            Προηγ_Α <- Α[i]
            S <- 0
    ΤΕΛΟΣ_ΑΝ
    S <- S+1
    i <- i+1
ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
ΓΡΑΨΕ Προηγ_Α, S
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ Συχνότητα1

ΠΡΟΓΡΑΜΜΑ Συχνότητα2
ΜΕΤΑΒΛΗΤΕΣ
    ΑΚΕΡΑΙΕΣ: i, S, Προηγ_Α, Α[1000]
ΑΡΧΗ
i <- 1
S <- 0
Προηγ_Α <- Α[i]
ΟΣΟ i < 1001 ΕΠΑΝΑΛΑΒΕ
    ΟΣΟ Προηγ_Α = Α[i] ΚΑΙ i < 1001 ΕΠΑΝΑΛΑΒΕ
        S <- S+1
```

```
        i <- i+1
    ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
    ΓΡΑΨΕ Προηγ_Α, S
    Προηγ.Α <- Α(i)
    S <- 0
ΤΕΛΟΣ_ ΕΠΑΝΑΛΗΨΗΣ
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ Συχνότητα2

ΠΡΟΓΡΑΜΜΑ Συχνότητα3
ΜΕΤΑΒΛΗΤΕΣ
    ΑΚΕΡΑΙΕΣ: i, S, Προηγ_Α, Α[1000]
ΑΡΧΗ
i <- 1
S <- 0
Προηγ_Α <- Α[i]
ΟΣΟ i < 1001 ΕΠΑΝΑΛΑΒΕ
    ΑΝ Προηγ_Α= Α[i] ΤΟΤΕ
        S <- S+1
        i <- i+1
    ΑΛΛΙΩΣ
        ΓΡΑΨΕ Προηγ_ Α, S
        Προηγ_Α <- Α[i]
        S <- 0
    ΤΕΛΟΣ_ΑΝ
ΤΕΛΟΣ_ ΕΠΑΝΑΛΗΨΗΣ
ΓΡΑΨΕ Προηγ_Α, S
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ Συχνότητα3
```

Αρχικά παρατηρούμε ότι όλες οι λύσεις πληρούν όλους τους κανόνες του δομημένου προγραμματισμού και γενικότερα όλων των αρχών και κανόνων που θέσαμε σ' αυτό το βιβλίο.

Στην πρώτη λύση συναντούμε μία λύση με αδρές γραμμές. Πριν την επανάληψη τοποθετούνται αρχικές τιμές και δίνεται η πρώτη τιμή στο Α(i). Μέσα στο βρόχο ξεχωρίζει το τμήμα που είναι για όλα τα Α(i), και το τμήμα που εκτελείται, όταν διαπιστωθεί διαφορετικό Α(i). Τέλος μετά την επανάληψη έχουμε ένα μέρος των εντολών που εκτελούνται και όταν βρίσκεται διαφορετικό Α(i).

Στη δεύτερη λύση για κάποιες τιμές του i θα γίνει δύο φορές η ερώτηση i<1001. Αυτό θα συμβεί όταν βρεθεί το Προηγ_Α(i). Τότε η εσωτερική επαναληπτική διαδικασία θα τερματιστεί, θα γίνουν η εκτύπωση κ.λπ. και αμέσως μετά θα γίνει η ερώτηση i<1001, χωρίς στο μεταξύ το i να έχει αυξηθεί.

'Ενας λόγος που συνέβη αυτό είναι ότι, χρησιμοποιήθηκε ή δομή **ΟΣΟ** ... **ΕΠΑΝΑΛΑΒΕ** που είναι ισχυρότερη της **ΑΝ** που πραγματικά χρειάζεται.

Στην τρίτη λύση δεν έχουμε τυποποίηση, γιατί η λύση δίνει την ίδια βαρύτητα στο συνηθισμένο γεγονός (=) και στο σπάνιο (<>), τοποθετώντας τα μέσα σε ένα **ΑΝ...ΤΟΤΕ...ΑΛΛΙΩΣ**.

Συμπερασματικά:

- ✓ Η πρώτη λύση είναι απλούστερη και τυπικότερη.
- ✓ Η δεύτερη λύση δεν πληρεί το κριτήριο της απλότητας, μια που χρησιμοποιεί δομή περισσότερο σύνθετη απ' όσο χρειάζεται.
- ✓ Η τρίτη λύση είναι απλή, υστερεί όμως σε τυπικότητα.

Παρατηρήστε ότι και στις τρεις λύσεις, οι ενέργειες που γίνονται όταν αλλάζει το περιεχόμενο του Α(i) χωρίζονται σε δύο επιμέρους τμήματα. Το ένα "ΓΡΑΨΕ Προηγ.Α, S" επαναλαμβάνεται το ίδιο πριν το τέλος του προγράμματος, ενώ το άλλο "Προηγ.Α<-Α(i), s<-0" έχει γραφτεί για πρώτη φορά πριν την αρχή της επαναληπτικής διαδικασίας. Ο χωρισμός γίνεται γιατί, στο κάθε τμήμα μπορούμε να διακρίνουμε συγκεκριμένο σκοπό. Το τμήμα με την εντολή "ΓΡΑΨΕ Προηγ.Α, S" είναι η βασική επεξεργασία, η καρδιά του προγράμματος, εκτελείται κάθε φορά που αλλάζει ο αριθμός, αλλά και μετά το τέλος της επαναληπτικής διαδικασίας. Το άλλο τμήμα, με τις εντολές "s<-0, Προηγ.Α<-Α(i)" έχει σκοπό να δώσει τις αρχικές τιμές στις μεταβλητές και φυσικά εκτελείται πριν ακόμα αρχίσει η επαναληπτική διαδικασία αλλά και μέσα σε αυτήν. Σ' αυτά τα προγράμματα υπάρχει συνήθως, μία ακόμα ενότητα που είναι η προετοιμασία της βασικής επεξεργασίας. Εδώ αυτή αποτελείται από μία μόνο εντολή την s<-s+1.

14.1.2 Ευελιξία

'Ενα από τα κριτήρια αξιολόγησης του προγράμματος είναι και η δυνατότητα επέκτασης του. Παρατηρώντας τον κύκλο ζωής προγράμματος, που αναφέρεται παρακάτω, παρατηρούμε ότι η μισή ζωή του προγράμματος θα καταναλωθεί για τις βελτιώσεις, προσθήκες, γενικά για τις αλλαγές στο αρχικό πρόγραμμα. Αυτό δίνει μεγαλύτερη βαρύτητα στο κριτήριο της ευελιξίας. Ο σχεδιασμός ενός ευέλικτου προγράμματος απαιτεί περισσότερο χρόνο για την επιλογή της μεθόδου, που θα ακολουθήσουμε για τη λύση του.

'Ενα κλασικό παράδειγμα, όπου ο αρχικός σχεδιασμός δεν επιτρέπει την επέκταση του αλγόριθμου παρ' ότι είναι λογικά σωστός, είναι η σύγκριση 3 αριθμών και η κατάταξή τους σε σειρά. Λογική επέκταση του θα ήταν ένα

πρόβλημα που ζητά την κατάταξη 4 αριθμών. Αν η μετάβαση από τον ένα αλγόριθμο στον άλλο είναι γρήγορη, τότε έχουμε σχεδιάσει έναν αλγόριθμο σε σωστή βάση.

Ας δούμε την λογική του εξέλιξη, δεδομένου ότι το πρόβλημα είναι εξαντλημένο από μαθηματικής πλευράς.

Παράδειγμα 2

Δίνεται ζεύγος αριθμών από το πληκτρολόγιο και ζητείται να εμφανίζονται με αύξουσα σειρά στην οθόνη.

```
ΠΡΟΓΡΑΜΜΑ Σειρά_2α
ΜΕΤΑΒΛΗΤΕΣ
    ΠΡΑΓΜΑΤΙΚΕΣ α, β
ΔΙΑΒΑΣΕ α, β
ΑΝ α<β ΤΟΤΕ
    ΓΡΑΨΕ α, β
ΑΛΛΙΩΣ
    ΓΡΑΨΕ β, α
ΤΕΛΟΣ_ΑΝ
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ Σειρά_2α
```

Το παραπάνω πρόγραμμα είναι φυσικά σωστό, απλό, και πληρεί όλους τους όρους που έχουν τεθεί.

Είναι όμως επεκτάσιμο;

Ας δούμε την πιθανότερη επέκταση του.

Παράδειγμα 3

Δίνεται τριάδα αριθμών από το πληκτρολόγιο και ζητείται να εμφανιστούν με αύξουσα σειρά στην οθόνη.

Οι διαφορετικές διατάξεις 3 αριθμών είναι 6 (3!=1*2*3=6). έτσι η παρακάτω λύση για τρεις αριθμούς, είναι η λογική επέκταση της προηγουμένης.

```
ΠΡΟΓΡΑΜΜΑ Σειρά_3α
ΜΕΤΑΒΛΗΤΕΣ
    ΠΡΑΓΜΑΤΙΚΕΣ α, β, γ
ΔΙΑΒΑΣΕ α, β, γ
ΑΝ α<β ΤΟΤΕ
    ΑΝ β<γ ΤΟΤΕ
        ΓΡΑΨΕ α, β, γ
```

```
    ΑΛΛΙΩΣ
        ΑΝ α<γ ΤΟΤΕ
            ΓΡΑΨΕ α, γ, β
        ΑΛΛΙΩΣ
            ΓΡΑΨΕ γ, α, β
        ΤΕΛΟΣ_ΑΝ
    ΤΕΛΟΣ_ ΑΝ
ΑΛΛΙΩΣ
    ΑΝ α<γ ΤΟΤΕ
        ΓΡΑΨΕ β, α, γ
    ΑΛΛΙΩΣ
        ΑΝ β<γ ΤΟΤΕ
            ΓΡΑΨΕ β, γ, α
        ΑΛΛΙΩΣ
            ΓΡΑΨΕ γ, β, α
        ΤΕΛΟΣ_ΑΝ
    ΤΕΛΟΣ_ΑΝ
ΤΕΛΟΣ_ΑΝ
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ Σειρά_3α
```

Αν τώρα μας ζητηθεί η ταξινόμηση τεσσάρων αριθμών, θα πρέπει πρώτα να βρούμε τις 24 (4!=1*2*3*4=24) διαφορετικές διατάξεις των τεσσάρων αριθμών και να γράψουμε φυσικά τα ανάλογα **ΑΝ,** προκειμένου να αναγνωρίσει το πρόγραμμα για ποια διάταξη πρόκειται, ώστε να την εμφανίσει στην οθόνη.

Παρατηρούμε ότι ενώ έχουμε λύσει το πρώτο πρόβλημα με απλό τρόπο, εν τούτοις όταν τα δεδομένα στην είσοδο αυξάνουν, γίνεται όλο και περισσότερο επίπονο να κατασκευαστεί αλγόριθμος που να καλύπτει την νέα ανάγκη, παρ' όλο ότι αυτή δεν φαίνεται αρκετή να προκαλέσει τέτοια αναστάτωση.

Ας δούμε λοιπόν τι πήγε λάθος, ώστε ένα πρόγραμμα που ο βασικός του κορμός δείχνει να έχει σχεδιαστεί σωστά, δεν αντέχει αυτό που από πολλούς θεωρείται φυσική του επέκταση.

Το λάθος προήλθε από το ότι στηριχθήκαμε για τη λύση του, σε μία λύση γνωστή από τα μαθηματικά, που είναι πολύ κοντά στην ανθρώπινη αντιμετώπιση του προβλήματος. Θεωρήσαμε ότι αυτή είναι "καλή λύση" και προχωρήσαμε και στην πρώτη επέκταση της, χωρίς να υπολογίσουμε ότι στον κύκλο ζωής ενός προγράμματος, ο αρχικός σχεδιασμός είναι μόνο το 16%, ενώ η συντήρηση, που περιλαμβάνει και τις επεκτάσεις, είναι 50%.

Ας δούμε μία λύση που στοχεύει στο μεγαλύτερο ποσοστό. Μια λύση που θα είναι εύκολα επεκτάσιμη.

Συμπερασματικά λοιπόν μπορούμε να πούμε ότι λύσεις που είναι καλές για μας μπορεί να μην είναι οι κατάλληλες για τον υπολογιστή. Αυτό θα πρέπει να εξετάζεται σαν μία σοβαρή παράμετρος στην επιλογή του αλγόριθμου.

```
ΠΡΟΓΡΑΜΜΑ Σειρά_2β
ΜΕΤΑΒΛΗΤΕΣ
    ΠΡΑΓΜΑΤΙΚΕΣ α, β
ΔΙΑΒΑΣΕ α, β
ΑΝ α>β ΤΟΤΕ
    temp <- α
    α<-β
    β<-temp
ΤΕΛΟΣ_ΑΝ
ΓΡΑΨΕ α, β
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ Σειρά_2β
```

Στο παραπάνω πρόγραμμα τυποποιήθηκε η έξοδος, ώστε πάντα να τυπώνεται το περιεχόμενο του α και μετά του β. Αυτό προς στιγμήν αυτή η τυποποίηση της εξόδου, δεν δείχνει την σημασία της, αντίθετα θα έλεγε κανείς ότι η λύση είναι "περίεργη" και θα είχε δίκιο.

Τώρα ας επεκτείνουμε αυτήν τη λογική για να λύσουμε το πρόβλημα για τρεις αριθμούς.

```
ΠΡΟΓΡΑΜΜΑ Σειρά_3β
ΜΕΤΑΒΛΗΤΕΣ
    ΑΚΕΡΑΙΕΣ: σ
    ΠΡΑΓΜΑΤΙΚΕΣ α, β, γ
σ <- 0
ΔΙΑΒΑΣΕ α, β, γ
ΟΣΟ σ=0 ΕΠΑΝΑΛΑΒΕ
    σ <- 1
    ΑΝ α>β ΤΟΤΕ
        temp <- α
        α <- β
        β <- temp
        σ <- 0
    ΤΕΛΟΣ_ΑΝ
    ΑΝ β>γ ΤΟΤΕ
        temp <- β
        β <- γ
        γ <- temp
        σ <- 0
    ΤΕΛΟΣ_ΑΝ
ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
ΓΡΑΨΕ α, β, γ
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ Σειρά_3β
```

Η μεταβλητή σ δείχνει, αν έχει γίνει αντιμετάθεση στα ζεύγη που συγκρίνονται. Αν δεν έχει γίνει καμία τότε σ=1 και οι αριθμοί είναι στη σωστή σειρά και άρα μπορούν να εμφανιστούν. Εξετάζοντας το προηγούμενο πρόγραμμα παρατηρούμε ότι, υπάρχει μια σειριακή ακολουθία από ανεξάρτητα **ΑΝ** (μπορούν να αλλάξουν σειρά), στα οποία συγκρίνονται οι αριθμοί με τη σειρά εισαγωγής τους. Έτσι γίνονται οι συγκρίσεις α>β και β>γ και γίνεται η αντιμετάθεσή τους, αν χρειάζεται. Δηλαδή από τη σύγκριση των δύο (α>β) περάσαμε στην σύγκριση των τριών (α>β, και β>γ).

Τώρα η σύγκριση δύο αριθμών με τη χρήση αντιμετάθεσης δεν φαίνεται πια "περίεργη".

Στα παραπάνω είδαμε τα βήματα **βελτιστοποίησης** της δομής ενός προγράμματος με την ευκαιρία μιας επέκτασης που δεν μπορούσε να γίνει.

14.1.3 Αξιοπιστία

Ο όρος αναφέρεται στη συνεχή και ομοιόμορφη λειτουργία ενός προγράμματος. Πράγματι χρησιμοποιώντας ένα πρόγραμμα περιμένουμε από αυτό, να συμπεριφέρεται βάση των προδιαγραφών, ακόμα και κάτω από τις δυσμενέστερες συνθήκες.

Συνήθως τα προβλήματα αξιοπιστίας παρουσιάζονται σε πραγματικές συνθήκες λειτουργίας των προγραμμάτων, όταν ο χρήστης του προγράμματος δεν είναι πλέον ο κ. Τέλειος αλλά αντίθετα ο κ. Άσχετος.

Συνήθη λάθη χειρισμού, ή ακόμη ηθελημένα λάθη που επιζητούν τα όρια του προγράμματος, μπορούν να καταρρακώσουν την αξιοπιστία του.

- Η αξιοπιστία ενός προγράμματος αυξάνεται με επιτυχείς ελέγχους επί των δεδομένων.
- Η αξιοπιστία ενός προγράμματος μειώνεται, όταν αποδέχεται δεδομένα εκτός προδιαγραφών και προχωρά στην επεξεργασία τους.
- Σε ένα αξιόπιστο πρόγραμμα δεν πρέπει να υπάρχουν εντολές ή ενότητες που δεν εκτελούνται.
- Ένα πρόγραμμα για να παραμένει αξιόπιστο, θα πρέπει μετά από κάθε αλλαγή, να επανελέγχεται με όλα τα δεδομένα ελέγχου που είχαν χρησιμοποιηθεί στους προηγούμενους ελέγχους.
- Σε ένα αξιόπιστο πρόγραμμα πρέπει να αντιμετωπίζονται όλες οι ακραίες περιπτώσεις, δεδομένων και ενεργειών.

Ας δώσουμε ένα παράδειγμα. Ας υποθέσουμε ότι έχουμε ένα πρόγραμμα που διαβάζει δύο αριθμούς και εμφανίζει το άθροισμα τους. Προφανώς από το πρόγραμμα περιμένουμε, όσο του δίνουμε αριθμούς, να μας δίνει το άθροισμα. Τι θα κάνει όμως το πρόγραμμα, αν του δώσουμε ετερόσημους αριθμούς; θα πρέπει να μας δώσει το αλγεβρικό τους άθροισμα. Το ίδιο θα

πρέπει να συμβεί και αν του δώσουμε δύο αρνητικούς αριθμούς. Αν του δώσουμε έναν αριθμό και ένα γράμμα όμως; Εδώ έχουμε αλλαγή στον τύπο δεδομένων και ένα πρόγραμμα που δουλεύει αξιόπιστα, πρέπει να απορρίψει το γράμμα σαν δεδομένο ακατάλληλο για επεξεργασία και να ζητήσει άλλο αποδεκτό τύπο δεδομένου.

Σε άλλη περίπτωση δίνεται ημερομηνία από το πληκτρολόγιο (ημέρα, μήνας, έτος), και ζητείται να βρεθεί σε ποια ημέρα της εβδομάδας αντιστοιχεί. 'Αραγε πως θα συμπεριφερθεί το πρόγραμμα αν του δώσουμε ενδεικτικά: Αρνητικό αριθμό ή μηδέν στην θέση της ημέρας, του μήνα ή του έτους; Αριθμό μήνα μεγαλύτερο του 12; Αριθμό ημέρας 32; Εδώ έχουμε τιμές εκτός ορίων υπολογισμού, προφανώς ένα αξιόπιστο πρόγραμμα πρέπει να απορρίπτει κάθε τέτοια ημερομηνία και να ζητά άλλη αποδεκτή τιμή ημερομηνίας.

Παραδείγματα τιμών άλλου τύπου από τον αναμενόμενο είναι και τα:

- ✓ Τι πρέπει να κάνει ένα πρόγραμμα που στα γράμματα ενός ονοματεπώνυμου δίνονται και αριθμοί;
- ✓ Τι πρέπει να κάνει ένα πρόγραμμα όταν τα ψηφία του ταχυδρομικού κώδικα είναι λιγότερα από πέντε;
- ✓ Τι πρέπει να κάνει ένα πρόγραμμα που του δίδονται γράμματα στη θέση αριθμών;

Παραδείγματα τιμών εκτός ορίου υπολογισμών είναι και τα:

- ⇨ Δίνεται δυαδικός αριθμός. Ζητείται να μετατραπεί στον αντίστοιχό του δεκαδικό. Προφανώς στην είσοδο πρέπει να γίνονται δεκτά μόνο ψηφία 0 και 1.
- ⇨ Δίνεται ρωμαϊκός αριθμός. Ζητείται να μετατραπεί στον αντίστοιχό του δεκαδικό. Προφανώς στην είσοδο πρέπει κατ' αρχήν να γίνονται δεκτοί μόνο χαρακτήρες I, V, X, L, C, D, M, αλλά και μία ακόμα σειρά ελέγχων προκειμένου να γίνει αποδεκτός ο συνδυασμός των ορθών χαρακτήρων.

Για τις περιπτώσεις που η αξιοπιστία του προγράμματος κινδυνεύει από διαφορετικό τύπο δεδομένου, ο τρόπος ελέγχου προσφέρεται από τη γλώσσα προγραμματισμού. Πράγματι κάθε γλώσσα έχει τη δυνατότητα να ορίσει τύπο μεταβλητής και έτσι να αποκλείσει κάθε άλλο τύπο δεδομένων μέσα στη μεταβλητή αυτή. Αυτά τα προβλήματα είναι απλούστερο να αντιμετωπιστούν, δεδομένου ότι ο έλεγχος του τύπου πραγματοποιείται αυτόματα από τη γλώσσα προγραμματισμού.

Για την περίπτωση των τιμών εκτός ορίου, ο τρόπος βελτίωσης της αξιοπιστίας θα στηριχθεί στους ελέγχους που θα συμπεριληφθούν στο πρόγραμμα. Προσπαθώντας για τη λύση τέτοιου είδους προβλημάτων, διαπιστώνεται ότι πρώτα πρέπει να σχεδιαστούν όλοι οι έλεγχοι και μετά η λύση του προβλήματος. Υπάρχει δηλαδή ένα *"πρόβλημα μέσα στο πρόβλημα"* .

Χωρίς πολύ καλή γνώση του χώρου του βασικού προβλήματος είναι αδύνατον να σχεδιαστούν σωστά οι απαραίτητοι έλεγχοι, ώστε το πρόγραμμα να είναι αξιόπιστο.

Διαπιστώνεται ακόμη, ότι ένα πρόγραμμα λίγων γραμμών, λόγω των ελέγχων, κύρια των ελέγχων αποδεκτών τιμών διογκώνεται υπέρμετρα.

Δεν θα αναφερθούμε σε παραδείγματα της πρώτης περίπτωσης, επειδή η αντιμετώπιση τους είναι στενά συνδεδεμένη με τη χρησιμοποιούμενη γλώσσα προγραμματισμού.

Ας δούμε λοιπόν ένα παράδειγμα για τη δεύτερη περίπτωση.

Παράδειγμα 4

Να κατασκευαστεί πρόγραμμα που θα ελέγχει μια ημερομηνία που δίνεται από το πληκτρολόγιο.

Κατ΄ αρχήν μερικές γνωστές και "άγνωστες" πληροφορίες για τον χώρο των ημερομηνιών, προκειμένου να υπάρχει ένα ελάχιστο ποσό γνώσεων για να περιγραφούν οι σχετικοί έλεγχοι.

- Κάθε έτος έχει 365 ημέρες, εκτός αν είναι δίσεκτο, οπότε έχει 366.
- Οι μήνες Ιανουάριος, Μάρτιος, Μάιος, Ιούλιος, Αύγουστος, Οκτώβριος και Δεκέμβριος έχουν 31 μέρες.
- Οι μήνες Απρίλιος, Ιούνιος, Σεπτέμβριος και Νοέμβριος, έχουν 30 μέρες.
- Ο Φεβρουάριος έχει 28 μέρες, εκτός αν το έτος είναι δίσεκτο, οπότε έχει 29.
- 'Ενα έτος λέγεται δίσεκτο αν ο αριθμός του είναι πολλαπλάσιο του 4, αλλά όχι του 100, εκτός αν είναι πολλαπλάσιο του 400. Για παράδειγμα: το 1984 είναι δίσεκτο (πολλαπλάσιο του 4), το 1900 δεν είναι (πολλαπλάσιο του 4 αλλά και του 100), το 2000 είναι δίσεκτο (πολλαπλάσιο του 4, του 100, αλλά και του 400), τέλος το 1993 δεν είναι.
- Το ημερολόγιο που ισχύει σήμερα είναι το Γρηγοριανό. Το εφάρμοσε ο πάπας Γρηγόριος ο ΙΓ' την $4^{η}$ Οκτωβρίου του 1582 προσθέτοντας δέκα ημέρες. 'Ετσι η επόμενη της $4^{ης}$ Οκτωβρίου ήταν η $15^{η}$ Οκτωβρίου. Στην Ελλάδα εφαρμόστηκε στις 10 Μαρτίου 1923. 'Ετσι η επόμενη της $10^{ης}$ Μαρτίου ήταν η $23^{η}$ Μαρτίου 1923.

Προσέξτε λοιπόν γιατί δεν υπάρχουν στα ελληνικά ημερολόγια οι ημερομηνίες 11/3/1923 έως 22/3/1923. Ενώ στα Ευρωπαϊκά δεν υπάρχουν οι ημερομηνίες 5/10/1582 έως 14/10/1582. Αυτό σε κάποιες περιπτώσεις μπορεί να είναι σημαντικό, ενώ για άλλες αδιάφορο.

Μετά από αυτά σκεπτόμενοι τη λύση του προβλήματος, διαπιστώνουμε ότι θα πρέπει να κάνουμε τους παρακάτω ελέγχους.

- ✓ Πρώτα να ελέγξουμε, αν δόθηκαν αρνητικοί αριθμοί στην ημερομηνία.
- ✓ Μετά να ελέγξουμε, αν η ημέρα είναι υπαρκτή. Βλέπε πληροφορία για το Γρηγοριανό ημερολόγιο.
- ✓ Στη συνέχεια να ελέγξουμε, αν το έτος που δόθηκε είναι δίσεκτο. Αυτή την πληροφορία θα τη χρησιμοποιήσουμε στον έλεγχο της ημέρας, όταν ο μήνας είναι ο Φεβρουάριος.
- ✓ Και τέλος να ελέγξουμε, αν συνδυάζονται σωστά ο αριθμός της ημέρας με τον αριθμό του μήνα.

```
ΠΡΟΓΡΑΜΜΑ Ημερομηνία
ΔΙΑΒΑΣΕ η, μ, ε
ΚΑΛΕΣΕ Ελεγχος_1(η, μ, ε, c)
ΑΝ c=0 ΤΟΤΕ ΓΡΑΨΕ 'Αρνητικές τιμές'
ΚΑΛΕΣΕ Ελεγχος_2(η, μ, ε, c)
ΑΝ c=0 ΤΟΤΕ ΓΡΑΨΕ 'Ανύπαρκτη ημερομηνία'
ΚΑΛΕΣΕ Ελεγχος_3(η, μ, ε, c)
ΑΝ c=0 ΤΟΤΕ ΓΡΑΨΕ 'Λανθασμένη ημερομηνία'
ΤΕΛΟΣ

ΔΙΑΔΙΚΑΣΙΑ Ελεγχος_1(η, μ, ε, c)
! Αρνητικές ημερομηνίες
c <- 0
ΑΝ ε>0 ΚΑΙ μ>0 ΚΑΙ η>0 ΤΟΤΕ c <- 1
ΤΕΛΟΣ_ΔΙΑΔΙΚΑΣΙΑ Ελεγχος_1

ΔΙΑΔΙΚΑΣΙΑ Ελεγχος_2(η, μ, ε, c)
! Ιουλιανό ημερολόγιο
c <- 1
ΑΝ (10<η ΚΑΙ η<23) ΚΑΙ μ=3 ΚΑΙ ε=1923 ΤΟΤΕ c <- 0
ΤΕΛΟΣ_ΔΙΑΔΙΚΑΣΙΑ Ελεγχος_2

ΔΙΑΔΙΚΑΣΙΑ Ελεγχος_3(η, μ, ε, c)
! 'Ελεγχος δίσεκτου έτους (αν δ=1 ΔΙΣΕΚΤΟ)
ε4 <- ε mod 4
ε100 <- ε mod 100
```

```
ε400 <- ε mod 400
δ <- 0
ΑΝ  ε4=0  ΤΟΤΕ
    ΑΝ  ε100=0 ΤΟΤΕ
        ΑΝ  ε400=0  ΤΟΤΕ
            δ <- 1
        ΤΕΛΟΣ_ΑΝ
    ΑΛΛΙΩΣ
        δ <- 1
ΤΕΛΟΣ_ΑΝ
ΤΕΛΟΣ_ΑΝ
! Ελεγχος αποδεκτής ημερομηνίας (c=1 ΑΠΟΔΕΚΤΗ)
c <- 0
ΕΠΙΛΕΞΕ μ
    ΠΕΡΙΠΤΩΣΗ 1, 3, 5, 7, 8, 10, 12
        ΑΝ η<=31 ΤΟΤΕ c <- 1
    ΠΕΡΙΠΤΩΣΗ 4, 6, 9, 11
        ΑΝ η<=30 ΤΟΤΕ c <- 1
    ΠΕΡΙΠΤΩΣΗ 2
        ΑΝ δ=0 ΚΑΙ η<=28 ΤΟΤΕ c <- 1
        ΑΝ δ=1 ΚΑΙ η<=29 ΤΟΤΕ c <- 1
ΤΕΛΟΣ_ΕΠΙΛΟΓΩΝ
ΤΕΛΟΣ_ΔΙΑΔΙΚΑΣΙΑ Ελεγχος_3
```

Ας δοθεί προσοχή στη σύνθετη συνθήκη στην ενότητα Ελεγχος_2. Ο έλεγχος σωστά γίνεται με σύνθετη συνθήκη, παρά τα όσα αντίθετα έχουν γραφτεί. Εδώ γίνεται μια ακίνδυνη καταστρατήγηση του κανόνα για λόγους συντομίας. Πραγματικά στο συγκεκριμένο σημείο δεν πρόκειται να υπάρξει καμία αλλαγή, σε οποιαδήποτε επέκταση του προγράμματος.

Το παραπάνω πρόγραμμα χρησιμοποιείται απαραιτήτως ως υποπρόγραμμα από οποιοδήποτε πρόγραμμα επεξεργάζεται ημερομηνίες που εισάγονται από το πληκτρολόγιο. Ας σημειωθεί ότι για τον έλεγχο της αποδεκτής ημερομηνίας χρειάστηκαν αρκετές γραμμές. Αυτή θα είναι η επιπλέον επιβάρυνση κάθε τέτοιου προγράμματος.

14.1.4 Ταχύτητα

'Ενα πρόγραμμα είναι ταχύτερο κάποιου άλλου, αν λύνει το ίδιο πρόβλημα σε λιγότερο χρόνο. Αυτό μπορεί να επιτυγχάνεται είτε:

- με αξιοποίηση πληροφοριών για την ποιότητα των δεδομένων. Για παράδειγμα ο αλγόριθμος ταξινόμησης, που είναι γενικά ταχύτερος (quick

sort), για ορισμένες ακραίες περιπτώσεις (περίπου ταξινομημένα στοιχεία) γίνεται βραδύτερος από άλλους,

- με διάφορες τεχνικές, όπως συμβαίνει με την περίπτωση των ψηφίων ελέγχου όπου δεν αλλάζει ο αλγόριθμος, αλλά ένα επιπλέον τμήμα που προστίθεται στην αρχή της λύσης, έχει ως αποτέλεσμα τη συνολική αύξηση της ταχύτητας του προγράμματος,
- με διαφορετικούς αλγόριθμους, όπως συμβαίνει με τους αλγόριθμους ταξινόμησης ή αναζήτησης, όπου για τα ίδια δεδομένα κάποιοι αλγόριθμοι αποδεικνύονται ταχύτεροι άλλων.
- με χρήση περισσότερης μνήμης

Το τελευταίο ξεφεύγει από τα όρια του βιβλίου, ενώ η τρίτη περίπτωση έχει αναπτυχθεί διεξοδικά στα κεφάλαια αλγοριθμικής και προγραμματισμού, επομένως θα δώσουμε παραδείγματα για τις δύο άλλες περιπτώσεις.

Παράδειγμα 5

Έστω ότι έχουμε να συγκρίνουμε μια σειρά αριθμών που εισάγονται από το πληκτρολόγιο και να απαντήσουμε πόσοι από αυτούς είναι αρνητικοί, θετικοί και μηδέν.

Προφανώς το πρόβλημα λύνεται με δύο εμφωλευμένα **ΑΝ** (τρεις έξοδοι).

Οι παρακάτω λύσεις είναι όλες αποδεκτές.

```
ΑΝ number>0 ΤΟΤΕ
    θ <- θ+1
ΑΛΛΙΩΣ_ΑΝ number<0 ΤΟΤΕ
    α <- α+1
ΑΛΛΙΩΣ
    μ <- μ+1
ΤΕΛΟΣ_ΑΝ

ΑΝ number<0 ΤΟΤΕ
    α <- α+1
ΑΛΛΙΩΣ_ΑΝ number>0 ΤΟΤΕ
    θ <- θ+1
ΑΛΛΙΩΣ
    μ <- μ+1
ΤΕΛΟΣ_ΑΝ
```

```
ΑΝ number=0 ΤΟΤΕ
    μ <- μ+1
ΑΛΛΙΩΣ_ΑΝ number<0 ΤΟΤΕ
    α <- α+1
ΑΛΛΙΩΣ
    θ <- θ+1
ΤΕΛΟΣ_ΑΝ
```

Ποια θα πρέπει να θεωρήσουμε ταχύτερη λύση;

Τέτοια ερώτηση δεν έχει απάντηση σε μία γενική θεώρηση, αν όμως μπορούμε να έχουμε την πληροφορία, του πως περίπου θα είναι τα δεδομένα μας, τότε πραγματικά μπορούμε να διαλέξουμε μια από τις τρεις που θα είναι η ταχύτερη για τα συγκεκριμένα δεδομένα. Για παράδειγμα αν γνωρίζουμε ότι στα δεδομένα μας οι περισσότεροι αριθμοί είναι μικρότεροι του μηδενός τότε η $2^{η}$ λύση θα δώσει ταχύτερα το τελικό αποτέλεσμα.

Παράδειγμα 6

Δίνεται πίνακας 1000 κωδικών αριθμών. Ζητείται να σχεδιαστεί πρόγραμμα που θα δέχεται έναν κωδικό από το πληκτρολόγιο, θα τον αναζητά στον πίνακα και θα δίνει τη θέση του πίνακα που βρέθηκε ο κωδικός.

Η αναζήτηση σε ένα πίνακα είναι θέμα το οποίο έχει ήδη αναπτυχθεί στο κεφάλαιο 3. Έτσι εδώ δεν θα λύσουμε το πρόβλημα, αλλά θα αναπτύξουμε μία ιδέα για την βελτίωση του χρόνου επεξεργασίας, που μπορεί να χρησιμοποιηθεί σε κάθε περίπτωση. Είτε δηλαδή ο πίνακας είναι ταξινομημένος πίνακας και κάνουμε δυαδική αναζήτηση, είτε είναι μη ταξινομημένος και κάνουμε σειριακή αναζήτηση.

Η ιδέα έχει τη βάση της στις εξής παρατηρήσεις:

- Η αναζήτηση σε πίνακα είναι ούτως ή άλλως χρονοβόρα.
- Αν αναζητηθεί κωδικός που δεν υπάρχει στον πίνακα, το πρόγραμμα θα εκτελέσει όλα τα βήματα πριν το διαπιστώσει.
- Ο κωδικός που δεν υπάρχει, συνήθως είναι αποτέλεσμα λανθασμένης πληκτρολόγησης.

Επομένως αν αναπτυχθεί μία μέθοδος που θα ελέγχει τους κωδικούς που πληκτρολογούνται και θα επιτρέπει την αναζήτηση μόνο για τους κωδικούς που είναι σωστοί, αποδεκτοί, θα έχουμε μια σημαντική εξοικονόμηση του συνολικού χρόνου επεξεργασίας.

Οι κωδικοί με τους οποίους μπορούμε να πετύχουμε αυτόν τον έλεγχο περιέχουν ένα ακόμα ψηφίο το **ψηφίο ελέγχου** και παράγονται με ειδικό τρόπο. Πώς παράγεται ένα ψηφίο ελέγχου κωδικού;

'Εστω ότι έχουμε πενταψήφιους κωδικούς και ένας από αυτούς είναι ο 12345. 'Ενας τρόπος για την παραγωγή του 6^{ου} ψηφίου είναι ο εξής:

$$\Psi_6 = (\Psi_1*1+\Psi_2*3+\Psi_3*5+\Psi_4*7+\Psi_5*13) \bmod 11$$
$$= (1*1+2*3+3*5+4*7+5*13) \bmod 11 = (1+6+15+28+65) \bmod 11 = 5.$$

'Αρα ο κωδικός που θα πληκτρολογείται θα είναι ο 123455. Με τον ίδιο κανόνα το πρόγραμμα θα ελέγχει, μετά την πληκτρολόγηση, αν ο εισαγόμενος 6ψήφιος, πλέον, κωδικός είναι αποδεκτός και μετά θα προχωρά στην αναζήτησή του στον πίνακα των 1000 θέσεων.

Ο κανόνας δημιουργίας του 6^{ου} ψηφίου είναι ο εξής: Πολλαπλασιάζονται όλα τα ψηφία του κωδικού με έναν πρώτο αριθμό, και προστίθονται τα γινόμενα. Το άθροισμα διαιρείται με 11. Το υπόλοιπο της διαίρεσης είναι το ψηφίο ελέγχου. Το υπόλοιπο της διαίρεσης με το 11 είναι από 0 μέχρι 10. Αν βρεθεί υπόλοιπο 10, το 6^{ο} ψηφίο είναι 0.

Υπάρχουν πολλοί τρόποι για τη δημιουργία του ψηφίου ελέγχου με πλεονεκτήματα και μειονεκτήματα.

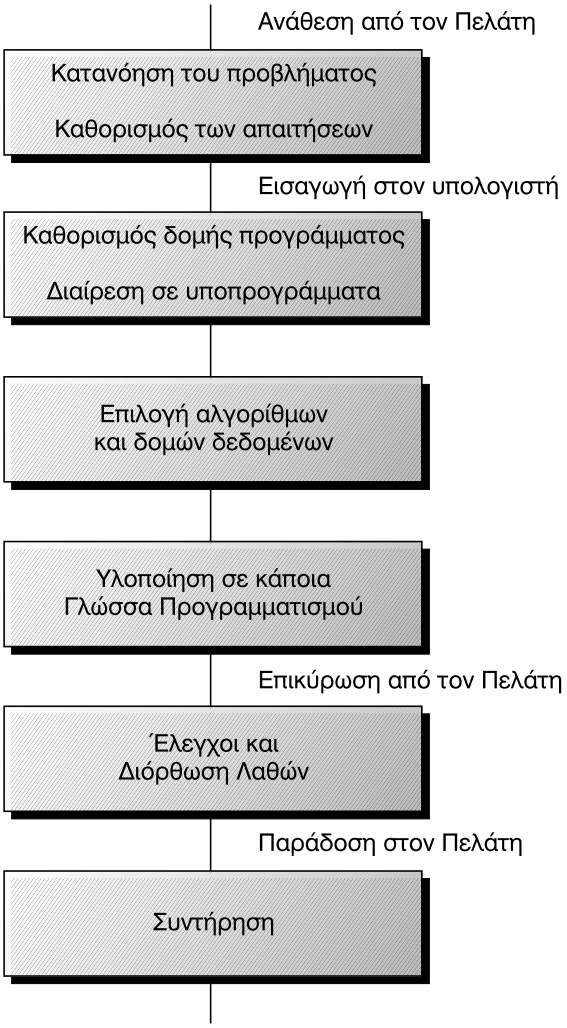
Το ψηφίο ελέγχου λοιπόν χρησιμοποιείται για λόγους ταχύτητας, όταν ο εισαγόμενος κωδικός πληκτρολογείται. Πράγματι ο χρόνος που χρειάζεται για να γίνουν οι απαραίτητες πράξεις για τον έλεγχο του κωδικού, ώστε αυτός να απορριφθεί, είναι κατά πολύ μικρότερος από το το χρόνο που χρειάζεται το πρόγραμμα για να κάνει 1000 συγκρίσεις, πριν καταλήξει στο συμπέρασμα ότι, ο κωδικός είναι λανθασμένος και φυσικά δεν υπάρχει.

Η μέθοδος εφαρμόζεται στους κωδικούς τραπεζικών λογαριασμών, στον κωδικό ΑΦΜ, στον κωδικό ISBN και γενικά όπου οι κωδικοί πληκτρολογούνται και η αναζήτηση τους πρόκειται να γίνει σε πολύ μεγάλους πίνακες ή αρχεία.

14.2 Τεκμηρίωση του Προγράμματος

Με τον όρο τεκμηρίωση εννοείται το σύνολο του γραπτού υλικού που περιγράφει τα συστατικά μέρη και τις λειτουργίες ενός νέου προγράμματος ή τις τροποποιήσεις που έγιναν σε ένα υπάρχον πρόγραμμα.

Στο σχήμα 14.1 φαίνεται η διαδοχή των φάσεων ανάπτυξης ενός προγράμματος.



Σχ. 14.1. Φάσεις ανάπτυξης προγράμματος

Η τεκμηρίωση δεν αποτελεί μια ξεχωριστή φάση στην ανάπτυξη ενός προγράμματος, αλλά μία παράλληλη διαδικασία που συμπληρώνει όλα τα στάδια ανάπτυξης ενός προγράμματος.

Οι παλιοί προγραμματιστές λένε: "Η τεκμηρίωση είναι χάσιμο χρόνου." Παραλείπουν όμως να πουν πόσο χρόνο αφιέρωσαν για να κάνουν μία αλλαγή στο παλιό τους πρόγραμμα που δεν είχε τεκμηρίωση.

Ας δούμε λοιπόν επιγραμματικά, ποιοι είναι οι λόγοι για τους οποίους χρειάζεται να γίνεται τεκμηρίωση, ποιες κατηγορίες τεκμηρίωσης υπάρχουν και τέλος ποιες ανάγκες εξυπηρετεί η ύπαρξη της τεκμηρίωσης.

14.2.1 Λόγοι τεκμηρίωσης

- **Παρόμοια εργασία.** Αν το πρόγραμμα που ζητείται, περιέχει παρόμοιες λειτουργίες με άλλα που έχουν ήδη δημιουργηθεί, τότε χρησιμοποιώντας την τεκμηρίωση, μπορούν να γίνουν οι απαραίτητες προσαρμογές σε αυτά.
- **Κάθε τροποποίηση γίνεται ευκολότερα.** Η ύπαρξη τεκμηρίωσης υποβοηθά στην επιτυχή και γρήγορη επέμβαση στο τμήμα του προγράμματος που πρέπει να μεταβληθεί ή να αντικατασταθεί ή τέλος να προστεθεί μια νέα λειτουργία στη σωστή θέση.
- **Δεν μπορεί να γίνει έλεγχος της λειτουργίας του αλγόριθμου.** Η απουσία τεκμηρίωσης ουσιαστικά απαγορεύει τον έλεγχο του αλγόριθμου. Αντίθετα η ύπαρξη του αρχείου δοκιμής (test file), μαζί με τα αποτελέσματα και τα σχόλια που το συνοδεύουν, θα περιορίσει έναν νέο έλεγχο μόνο στα σημεία που χρειάζεται.

14.2.2 Κατηγορίες τεκμηρίωσης

- **Τεκμηρίωση ανάπτυξης.** Αναφέρεται στις προδιαγραφές του προβλήματος, τι πρόκειται να κάνει το πρόγραμμα και τη σύνδεσή του με άλλα προγράμματα, αν υπάρχει. Περιέχει επίσης την περιγραφή των χρησιμοποιούμενων αλγορίθμων.
- **Τεκμηρίωση ελέγχου.** Αναφέρεται στα δεδομένα ελέγχου (test file) που χρησιμοποιήθηκαν προκειμένου να δοκιμαστεί το πρόγραμμα, αν έχουν διερευνηθεί οι ακραίες περιπτώσεις και ποιες είναι αυτές και τέλος αν έχουν καθιερωθεί κάποιες και ποιες συμβάσεις για τη λύση του προβλήματος. Κάθε πρόγραμμα έχει κάποια όρια, τα οποία πρέπει να αναφέρονται στην τεκμηρίωση.
- **Τεκμηρίωση χρήστη**. Περιέχει όλες τις οδηγίες που πρέπει να δοθούν στο χρήστη του προγράμματος προκειμένου να το χρησιμοποιήσει αποδοτικά. Οι οδηγίες αυτές συνήθως αναφέρονται ως εγχειρίδιο χρήστη (users manual).
- **Τεκμηρίωση προγράμματος.** Αναφέρεται στον τρόπο με τον οποίο έχουν λυθεί τα επιμέρους προβλήματα, τις υποθέσεις που έχουν γίνει και τους περιορισμούς που έχουν τεθεί. Ακόμα πληροφορίες για "ειδικές" τεχνικές που έχουν χρησιμοποιηθεί. Πρέπει σε κάθε βήμα να αναφερθεί με λεπτομέρεια, πολλές φορές κουραστική, αλλά δυστυχώς αναγκαία, η λύση που χρησιμοποιήθηκε και τα δεδομένα με τα οποία ελέγχθηκε. 'Ενα υπόμνημα των μεταβλητών που χρησιμοποιούνται είναι χρήσιμο για την τεκμηρίωση αλλά και για την αρχική ανάπτυξη του αλ-

γόριθμου. Η απλούστερη και πλέον στοιχειώδης τεκμηρίωση αυτής της μορφής γίνεται με την εισαγωγή γραμμών σχολίων μέσα στον κώδικα του προγράμματος.

Η τεκμηρίωση εξυπηρετεί λειτουργίες όπως:

- **Επικοινωνία μεταξύ χρήστη - αναλυτή - προγραμματιστή**. Στη φάση της κατανόησης του προβλήματος πολλές ερωτήσεις γίνονται, ενώ δίδονται οι αντίστοιχες απαντήσεις. Οι απαντήσεις αυτές αποτελούν τις οδηγίες βάση των οποίων θα γίνει το πρόγραμμα, επομένως είναι δεσμευτικές και για τα δύο μέρη, άρα είναι χρήσιμο να καταγράφονται. Η τεκμηρίωση πραγματοποιείται σε όλα τα στάδια της ανάπτυξης ενός προγράμματος. Κάθε διευκρίνιση που έχει δοθεί σε ερωτήματά μας, σε όλα τα στάδια ανάπτυξης πρέπει να βρίσκεται στο φάκελο τεκμηρίωσης.
- **Ιστορική αναφορά για τροποποιήσεις διορθώσεις**. Μια ιστορική αναφορά θα προσφέρει τη δυνατότητα συνολικής εκτίμησης του έργου που έχει γίνει.
- **Ποιοτικό έλεγχο.** Κάνει δυνατό έναν έλεγχο και δεν σπαταλάται χρόνος για τη γενική αναζήτηση του λάθους. Αν ακολουθήθηκε μία ευφυής μέθοδος, ποια είναι αυτή, καθώς και ποια είναι τα δεδομένα ελέγχου με τα οποία δοκιμάστηκε.
- **Αναφορά για διδακτικούς λόγους**. 'Εχοντας καταγράψει τη διαδρομή της σκέψης μας για το συγκεκριμένο αλγόριθμο, στη διάρκεια ζωής του προγράμματος μπορεί να επινοηθεί ένας άλλος καλύτερος τρόπος λύσης του ίδιου προβλήματος, λόγω της εμπειρίας που στο μεταξύ θα έχει αποκτηθεί. Να καταγράφεται η μέθοδος λύσης και ο λόγος που επιλέχθηκε. Να περιγράφεται το στιγμιότυπο στο οποίο βασίστηκε η λύση.

Παράδειγμα 7

Το μηχανογραφικό κέντρο του Υπουργείου Παιδείας πρόκειται να εκτυπώσει καταστάσεις με τις βαθμολογίες των πανελληνίων εξετάσεων που τελικά θα αποσταλούν στα σχολεία. Σε κάθε σελίδα αυτής της κατάστασης θα πρέπει να τυπώνεται η ονομασία του σχολείου. Όλες οι ονομασίες των σχολείων βρίσκονται σε ένα πίνακα από τον οποίο θα αναζητούνται. Σε κάθε γραμμή της σελίδας θα τυπώνεται το ονοματεπώνυμο, πατρώνυμο, και ο τελικός βαθμός του μαθητή.

'Ενα τέτοιο πρόβλημα είναι φανερό ότι θα πρέπει να χωριστεί σε ενότητες προκειμένου να λυθεί. Για τη συντήρηση του προγράμματος είναι απαραίτητο να έχει αποτυπωθεί η σύνδεση αυτών των ενοτήτων. Είναι προ-

Το πρόβλημα του 2000 από τη σκοπιά της τεκμηρίωσης

Η ιδέα του να κρατά κανείς για το έτος δύο ψηφία, αντί τέσσερα που ήταν το ορθό, βρήκε πρόσφορο έδαφος και απλώθηκε παντού, σαν μία "έξυπνη" ιδέα που εξοικονομούσε χώρο μνήμης για το πρόγραμμα, αλλά και για την αποθήκευση των δεδομένων. 'Οπως φάνηκε όμως, δεν μελετήθηκαν όλες οι πιθανές επιπτώσεις, με αποτέλεσμα το κόστος της επαναφοράς των τεσσάρων ψηφίων για το έτος, να ανέλθει σε ιλιγγιώδη ποσά και χωρίς να είναι σίγουρο ότι βρέθηκαν όλες οι γραμμές κώδικα που έπρεπε να διορθωθούν. Αυτό το κόστος ήταν τόσο μεγάλο, γιατί είτε υπήρχαν παλιά προγράμματα που είχε χαθεί η τεκμηρίωση τους και άρα η επέμβαση στον κώδικα αφορούσε όλο το πρόγραμμα, είτε γιατί επειδή η "μέθοδος" ήταν πολύ διαδεδομένη, θεωρήθηκε ότι τέτοιες παρατηρήσεις δεν ήταν απαραίτητο να αναφέρονται στην τεκμηρίωση.

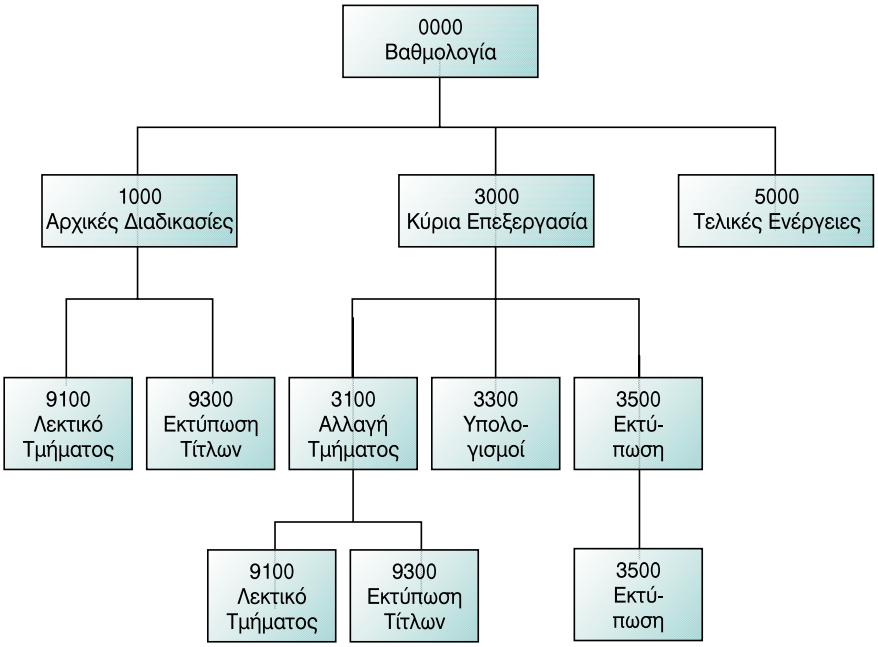
φανές ότι ο ψευδοκώδικάς του, θα εκτείνεται σε αρκετές σελίδες, όπου δύσκολα θα μπορεί κανείς να εντοπίσει τις διάφορες ενότητες, αλλά και τη σύνδεση που έχουν αυτές μεταξύ τους.

Για να καλυφθεί αυτή η ανάγκη είναι προτιμότερο να χρησιμοποιηθεί κάποια διαγραμματική τεχνική. Μία από τις πολλές που υπάρχουν είναι η HIPO (Hierarchy plus Input – Process –Output). Η τεχνική αυτή και χρησιμοποιεί διαγράμματα με τα οποία απεικονίζει την είσοδο, την έξοδο και τις λειτουργίες ενός προγράμματος. Απαρτίζεται από τρία είδη.

- Τον Οπτικό Πίνακα Περιεχομένων (Visual Table Of Contains, **VTOC**). Είναι ένα ιεραρχικό διάγραμμα αναπαράστασης των λειτουργιών του προγράμματος και της σχέσης μεταξύ τους.
- Το διάγραμμα Επισκόπησης Ιεραρχίας και Εισόδου – Επεξεργασίας - Εξόδου (Overview Hierarchy plus Input – Process – Output, **Overview HIPO**). Είναι διάγραμμα απεικόνισης εισόδου, διεργασιών και εξόδου, των λειτουργιών που εμφανίζονται στην κορυφή ενός οπτικού πίνακα περιεχομένων.
- Το Λεπτομερές Ιεραρχικό διάγραμμα Εισόδου, Επεξεργασίας, Εξόδου (**Detail HIPO**). Είναι διάγραμμα απεικόνισης εισόδου, λειτουργιών που εμφανίζονται στο χαμηλότερο τμήμα ενός οπτικού πίνακα περιεχομένων.

Από τα τρία διαγράμματα αυτό που είναι χρήσιμο για την τεκμηρίωση του προγράμματος, είναι το VTOC, τα δύο άλλα βρίσκουν εφαρμογή στο πεδίο της τεκμηρίωσης Πληροφοριακών Συστημάτων. 'Ενας οπτικός πίνακας περιεχομένων, είναι ένας μακροσκοπικός τρόπος αποτύπωσης των λειτουργιών του προγράμματος, που παρουσιάζει όχι τη λογική του, αλλά τα συστατικά του. Σ' αυτόν εμφανίζονται οι ενότητες από τις οποίες αποτελείται το πρόγραμμα, καθώς και ο ιεραρχικός τρόπος που σχετίζονται μεταξύ τους. 'Ετσι σε μία μόνο σελίδα αποτυπώνεται η εσωτερική διάρθρωση του προγράμματος, όση πολυπλοκότητα και αν παρουσιάζει.

Το VTOC είναι ένα σημαντικό εργαλείο για την καλή τεκμηρίωση του προγράμματός σας.



***Σχ. 14.2.** Διάγραμμα VTOC*

Στο σχήμα 14.2 φαίνεται ότι το πρόγραμμα αποτελείται από οκτώ διαφορετικές ενότητες, που αναπτύσσονται σε τέσσερα επίπεδα. Η τοποθέτηση μιας ενότητας κάτω από μία άλλη σημαίνει ότι, η εκτέλεσή της εξαρτάται από αυτήν. Το βήμα που ακολουθεί αυτό το ιεραρχικό διάγραμμα, είναι η αναλυτική σχεδίαση κάθε ενότητας. Ας σημειωθεί ότι, ένας οπτικός πίνακας περιεχομένων διαβάζεται από πάνω προς τα κάτω και από αριστερά προς τα δεξιά.

Εκτός από το όνομα που δόθηκε σε κάθε ενότητα και που πρέπει να είναι ενδεικτικό της λειτουργίας της, δίνεται και ένας, συνήθως τετραψήφιος αριθμός. 'Ετσι στο μηδενικό επίπεδο που υπάρχει πάντοτε (ενότητα του προβλήματος), δίδεται ο αριθμός 0000. Στην πρώτη ενότητα του πρώτου επιπέδου δίδεται ο αριθμό 1000, στην επόμενη αυτής ο αριθμό 3000 κ.ο.κ.Ο αριθμός 9000 κρατείται για τις ενότητες που καλούνται από διάφορα επίπεδα και από διαφορετικούς κλάδους (κλάδος 1000, κλάδος 3000 κλπ.). Αυτές χαρακτηρίζονται ως ενότητες κοινής χρήσης, και είναι συνήθως η ενότητα της ανάγνωσης και των τίτλων.

Μετά την οριστικοποίηση των ενοτήτων πρώτου επιπέδου, κάθε ενότητα των άλλων επιπέδων θα συνοδεύεται από ένα αριθμό που θα είναι της ενότητας που την καλεί. 'Ετσι θα έχουμε τις ενότητες με αριθμούς 1100 και 1300 που θα καλούνται από την ενότητα 1000 κ.λπ.

Παρατηρήστε ότι οι αριθμοί που προηγούνται και συμπληρώνουν τα ονόματα των ενοτήτων έχουν ένα πολύ σημαντικό ρόλο. Δείχνουν μέσα στον κώδικα, αμέσως, την θέση της ενότητας στο πρόγραμμα, δηλαδή, από ποιες καλείται, ποιες καλεί και από ποιες είναι ανεξάρτητη. 'Ετσι σε μία δεδομένη επέμβαση γνωρίζετε που πρέπει να επέμβετε, αλλά και ποιες ενότητες πρέπει να ελέγξετε, μήπως επηρεάστηκαν από την επέμβασή σας.

Παρατηρείστε ακόμη ότι, σε κάθε κλάδο και σε κάθε επίπεδο η αρίθμηση δεν είναι συνεχόμενη. Αυτό σας δίνει την δυνατότητα σε μία μελλοντική συντήρηση, αν διαπιστώσετε ότι χρειάζεστε μια νέα ενότητα, να την τοποθετήσετε στη σωστή θέση με το σωστό αριθμό, χωρίς να χρειαστεί να επαναριθμήσετε όλες τις άλλες.

Κανόνας 3Ε

- *Είσοδος*
- *Επεξεργασία*
- *Έξοδος*

Σημειώνεται ότι

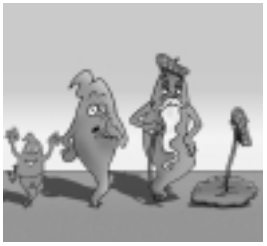
- Ο βαθμός ανεξαρτησίας καθορίζει το αν και πόσο οι αλλαγές σε μία ενότητα, επιβάλλουν αλλαγές σε άλλες ενότητες.
- Κάθε ενότητα έχει μία είσοδο, μία έξοδο και εκτελεί μία καθορισμένη επεξεργασία (κανόνας 3 Ε, **ε**ίσοδος, **ε**πεξεργασία, **έ**ξοδος).
- Κάθε ενότητα καλείται από μία άλλη ενότητα και επιστρέφει σ' αυτήν όταν τελειώσει.
- Κάθε ενότητα αποτελείται από τις βασικές δομές, ακολουθία, επιλογή και επανάληψη.
- Σε κάθε ενότητα δεν πρέπει να υπάρχει απότομη διακοπή, άπειρη επανάληψη.
- Πολλές ενότητες κάνουν το πρόγραμμα πολύπλοκο.

14.2.3 Φάκελλος Προγράμματος

Η ολοκλήρωση του προγράμματος σημαίνει την ώρα που σταματά η καθημερινή απασχόληση με αυτό. Η εργασία που έχει γίνει μέχρι τώρα, αλγόριθμοι, κωδικοποίηση, δεδομένα ελέγχου και αποτελέσματα, σχόλια από την πρώτη στιγμή που το πρόβλημα διατυπώθηκε μέχρι τώρα, διευκρινήσεις που ζητήθηκαν, απαντήσεις που δόθηκαν κάθε στιγμή της ανάπτυξης του προγράμματος, πρέπει να αρχειοθετηθούν. Συγκροτείται έτσι ο φάκελος τεκμηρίωσης, ο φάκελος του προγράμματος.

Ο φάκελος τεκμηρίωσης πρέπει να ολοκληρωθεί αμέσως μετά την ολοκλήρωση του προγράμματος. Η επόμενη φορά που θα γίνει προσπέλαση στο φάκελο αυτό, θα είναι όταν πρέπει να διαβαστεί για να αντληθούν οι απαραίτητες πληροφορίες για τη συντήρηση του προγράμματος.

14.3 Κύκλος Ζωής Λογισμικού



'Ενα πρόγραμμα αρχίζει την ζωή του από την στιγμή που θα καθοριστούν οι απαιτήσεις του, οι προδιαγραφές του και παύει να ζει όταν εξαντληθούν όλα τα περιθώρια συντήρησής του (προσθήκες, αλλαγές, βελτιώσεις).

Θα έχετε προσέξει ότι στα διάφορα πακέτα λογισμικού, είτε είναι γλώσσες προγραμματισμού, είτε πακέτα εφαρμογών, είτε λειτουργικά συστήματα, δίπλα στο εμπορικό όνομα του λογισμικού, υπάρχει ο αριθμός της έκδοσής του (version).

Ο αριθμός έκδοσης που συνοδεύει την ονομασία κάθε πακέτου λογισμικού, δείχνει ακριβώς τις αλλαγές που έχουν πραγματοποιηθεί από την αρχική του εμφάνιση. Ο κανόνας λέει ότι, όταν οι αλλαγές είναι σημαντικές, δηλαδή έχουν προστεθεί νέες λειτουργίες, εντολές, προγράμματα, ο αριθμός αυξάνει κατά ακέραιο αριθμό (DOS ver.5 DOS ver.6), όταν οι αλλαγές είναι μικρότερες, τότε αυξάνεται κατά δέκατα (Windows v.3.1, Windows v.3.11). Τα τελευταία χρόνια αυτή η εξέλιξη των δυνατοτήτων ενός λογισμικού έχει συνδεθεί με την πολιτική πωλήσεων των εταιρειών παραγωγής. 'Ετσι οι αριθμοί που ακολουθούν τις ονομασίες του λογισμικού, έχουν έντονη την χροιά του τμήματος πωλήσεων και όχι του τμήματος ανάπτυξης της εταιρείας (Windows 95, Windows98, Office 97 κ.λπ).

Θα έχετε παρατηρήσει ακόμα ή θα έχετε ακούσει, ότι το τάδε λογισμικό αντικαταστάθηκε από το δείνα, για παράδειγμα το MS-DOS αντικαταστάθηκε από τα Windows.

Οι αριθμοί έκδοσης και οι αντικαταστάσεις του λογισμικού δείχνουν με τον καλύτερο τρόπο ότι, κάθε λογισμικό έχει ένα κύκλο ζωής, όπου στον κύκλο αυτό γεννιέται και πεθαίνει όπως ένας ζωντανός οργανισμός.

Τι σημαίνει όμως "κύκλος ζωής"; Ποιες είναι οι ενδιάμεσες φάσεις του;

Μια σύγκριση θα μας δώσει καλύτερα το περιεχόμενο του όρου "Κύκλος Ζωής". Ας δούμε πρώτα τον "κύκλο ζωής" ενός αυτοκινήτου.

Η δημιουργία ενός αυτοκινήτου ξεκινά συνήθως από μία έρευνα που θα καταγράψει τις επιθυμίες του κοινού, αυτοκίνητο πόλης, εκτός δρόμου, φθηνό, γρήγορο και την τάση της αγοράς. Τα αποτελέσματα της έρευνας, σε συνδυασμό με άλλα δεδομένα, όπως άλλα μοντέλα του εργοστασίου, μοντέλα του ανταγωνισμού κ.λπ, θα καθορίσουν τις προδιαγραφές του νέου αυτοκινήτου, μικρό ή μεγάλο, γρήγορο, νεανικό, με πόσες πόρτες, με ποια πρόσθετα, σε ποια τιμή κ.λπ. (φάση ανάλυσης).

Οι μηχανικοί της εταιρείας έχοντας υπ' όψη τις προδιαγραφές θα σχεδιάσουν το νέο αυτοκίνητο επιλέγοντας τα κατάλληλα υλικά, δίνοντάς του την εξωτερική μορφή και τα άλλα χαρακτηριστικά που έχουν προδιαγραφεί, ιπποδύναμη, κατανάλωση, κ.λπ. (φάση σχεδιασμού).

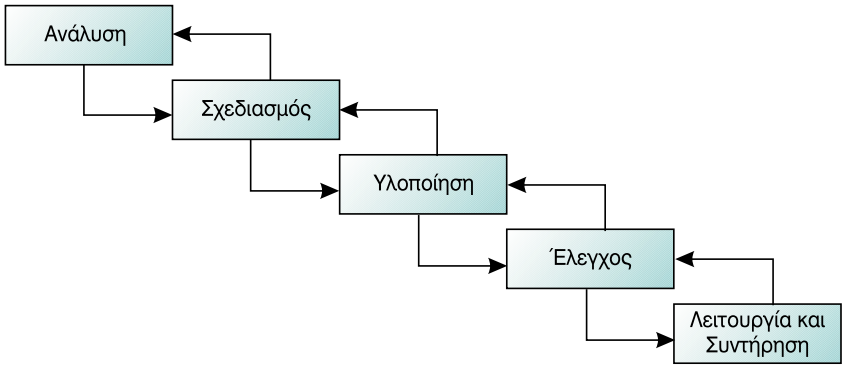
Το τμήμα παραγωγής του εργοστασίου θα πάρει τα σχέδια και θα προσαρμόσει τη γραμμή παραγωγής από πλευράς διαδικασιών και ελέγχων, ώστε να μπορεί να παράγεται το νέο αυτοκίνητο (φάση υλοποίησης).

Το έτοιμο αυτοκίνητο θα πρέπει να περάσει ένα τελικό έλεγχο λειτουργίας στο δρόμο και κάτω από ακραίες συνθήκες. Αυτός ο έλεγχος θα διενεργηθεί μέσα στις εγκαταστάσεις της εταιρείας, ώστε το νέο αυτοκίνητο να δοκιμαστεί σαν σύνολο (φάση ελέγχου).

Το δοκιμασμένο αυτοκίνητο είναι έτοιμο για πώληση. Πωλούμενο μπαίνει πλέον σε κανονική λειτουργία, η οποία εξασφαλίζεται με την κατάλληλη συντήρηση από τους εξουσιοδοτημένους μηχανικούς (φάση λειτουργίας και συντήρησης).

Αυτός λοιπόν είναι ο κύκλος ζωής ενός αυτοκινήτου, παρόμοιος σε αρκετά σημεία με τον κύκλο ζωής ενός προγράμματος, που παρουσιάζεται διαγραμματικά στο σχήμα 14.3.

Τα βέλη δείχνουν την αλληλεπίδραση των διαδοχικών φάσεων του κύκλου ζωής ενός προγράμματος. Δηλαδή ότι το τέλος μιας φάσης οδηγεί στην επόμενη, αλλά μπορεί να οδηγήσει και στην προηγούμενη. Όταν το τέλος μιας φάσης οδηγεί στην προηγούμενη, σημαίνει ότι ορισμένα στοιχεία, χρειάζεται να επανακαθοριστούν.



Σχ. 14.3. *Κύκλος ζωής προγράμματος*

Στη φάση **Ανάλυσης και Σχεδίασης** που ακολουθεί τον ορισμό του προβλήματος από τον πελάτη, μπορούμε να διακρίνουμε τις εξής ενέργειες:

- ✓ Καταγράφονται αναλυτικά τα δεδομένα και τα ζητούμενα του προβλήματος.
- ✓ Ζητούνται οι απαραίτητες διευκρινήσεις από τον πελάτη, σε όσα σημεία οι προδιαγραφές παρουσιάζουν ασάφεια.
- ✓ Καθορίζεται η δομή του προγράμματος.
- ✓ Καθορίζονται οι ενότητες (ρουτίνες, υποπρογράμματα) από τις οποίες θα αποτελείται το πρόγραμμα.
- ✓ Αναζητούνται έτοιμες ενότητες (modules) από παλιότερα προγράμματα που μπορούν να χρησιμοποιηθούν και σ' αυτό το πρόγραμμα.
- ✓ Επιλέγονται οι αλγόριθμοι και οι δομές δεδομένων που θα χρησιμοποιηθούν σε κάθε ενότητα.

Στην επόμενη φάση της **Υλοποίησης** του προγράμματος σε κάποια γλώσσα προγραμματισμού, ακολουθούμε τα εξής βήματα με τη σειρά:

- ✓ Επιλέγεται η γλώσσα προγραμματισμού για το συγκεκριμένο πρόγραμμα. Σημειώστε ότι όλες γλώσσες δεν είναι κατάλληλες για όλα τα προβλήματα.
- ✓ Εισάγεται το κωδικοποιημένο πρόγραμμα στον υπολογιστή. Το πρόγραμμα αυτό είναι το αρχικό πρόγραμμα (source program).
- ✓ Ζητείται η μετάφραση του προγράμματος από ένα μεταγλωττιστή, ώστε αυτό να γίνει κατανοητό από τον υπολογιστή. Το πρόγραμμα αυτό είναι το τελικό πρόγραμμα (object program).

Προσοχή: Ο έλεγχος ενός προγράμματος, διαπιστώνει την ύπαρξη ενός λάθους, ποτέ όμως δεν πιστοποιεί την ανυπαρξία λάθους.

- ✓ Η μετάφραση θα αποκαλύψει λάθη "ορθογραφίας" και συντακτικού" της γλώσσας προγραμματισμού.
- ✓ Διορθώνονται τα λάθη και ακολουθεί ξανά μετάφραση του προγράμματος, έως την οριστική εξάλειψή τους.

Η φάση των **Ελέγχων** αναφέρεται στον έλεγχο των λογικών λαθών που πιθανόν να υπάρχουν σε ένα πρόγραμμα. Η σειρά των ελέγχων είναι η ακόλουθη.

- ✓ Το πρόγραμμα ελέγχεται, με τα δεδομένα ελέγχου που είχε ελεγχθεί και ο αλγόριθμος, για να διαπιστωθεί αν παράγει τα επιθυμητά αποτελέσματα.
- ✓ Διαπιστώνονται λάθη που οφείλονται είτε σε λάθη κατά την κωδικοποίηση του αλγόριθμου είτε από λανθασμένη επικοινωνία ενοτήτων.
- ✓ Τα τυχόν λάθη διορθώνονται και οι έλεγχοι συνεχίζονται μέχρι το πρόγραμμα να απαλλαγεί από αυτά.
- ✓ Παρουσιάζονται τα αποτελέσματα στον πελάτη προκειμένου να έχουμε την οριστική επικύρωση εκ μέρους του.
- ✓ 'Αν παρουσιάζεται απόκλιση από τις προδιαγραφές, θα πρέπει να επιστρέψουμε στην φάση της ανάλυσης και σχεδίασης προκειμένου να γίνουν οι κατάλληλες διορθώσεις.

Στο τέλος της φάσης των ελέγχων έχουμε ένα έτοιμο πρόγραμμα που μπορούμε να παραδώσουμε στον πελάτη για χρήση. Δεν πρέπει να ξεχνάτε, ότι ο έλεγχος του προγράμματος διαπιστώνει την ύπαρξη ενός λάθους που πρέπει να διορθωθεί, δυστυχώς όμως δεν πιστοποιεί την απουσία λάθους. 'Ετσι ένα πρόγραμμα κατ' ουσία βρίσκεται κάτω από ένα διαρκή έλεγχο της ορθής λειτουργίας του.

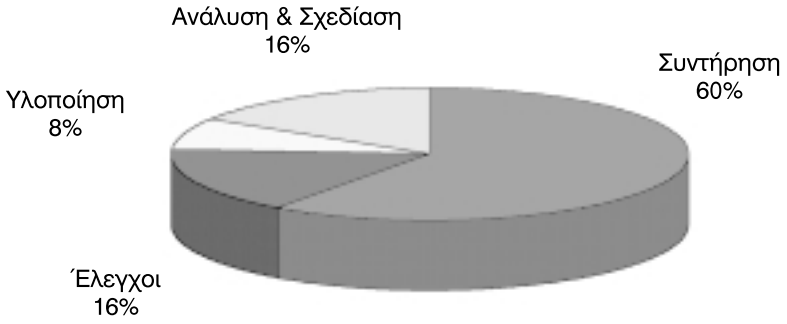
Στην επόμενη φάση της **Συντήρησης** θα γίνουν όλες οι προσαρμογές και βελτιώσεις που χρειάζονται προκειμένου το πρόγραμμα σας να συνεχίσει να χρησιμοποιείται. Η φάση αυτή διαρκεί όσο θα χρησιμοποιείται το πρόγραμμα σας. Εδώ θα παρατηρήσουμε ότι:

- ✓ Οι προσαρμογές είναι αναπόφευκτες όταν διαφοροποιούνται τα δεδομένα του προβλήματος ή όταν ο χρήστης ζητήσει νέες λειτουργίες.
- ✓ Κάποιες προσαρμογές μπορεί να απαιτήσουν την εκτέλεση της φάσης της ανάλυσης και σχεδίασης και άρα όλων των υπολοίπων φάσεων.
- ✓ Οι βελτιώσεις προκύπτουν από την εμπειρία που αποκτάται με τον καιρό και μας κάνει να "βλέπουμε" τα ίδια πράγματα με "άλλο μάτι".

- ✓ Κάθε προσαρμογή ή βελτίωση θα πρέπει να καταλήγει σε συνολικό έλεγχο του προγράμματος και φυσικά στην καταγραφή των σχετικών σχολίων για την τεκμηρίωση.
- ✓ Το τελευταίο στάδιο αυτής της φάσης έρχεται με την τελειοποίηση του προγράμματος. Τώρα το πρόγραμμά σας δουλεύει υποδειγματικά ... μέχρι την επόμενη αλλαγή.

Στο σχήμα 14.4 παρουσιάζεται η προσπάθεια που απαιτείται κατά τις διάφορες φάσεις του κύκλου ζωής ως ποσοστό του συνολικού έργου.

Κύκλος Ζωής Προγράμματος



Σχ. 14.4*. Κατανομή της προσπάθειας που απαιτείται στις διάφορες φάσεις του κύκλου ζωής προγράμματος*

Ανακεφαλαίωση

- Μάθαμε να απορρίπτουμε λύσεις που δεν πληρούν τα βασικά χαρακτηριστικά αξιολόγησης προγράμματος.
- Είδαμε ότι το μεγαλύτερο μέρος της ζωής ενός προγράμματος διανύεται στην φάση συντήρησης του.
- Μάθαμε να εκτιμούμε την φιλικότητα ενός προγράμματος σαν χρήστες, αλλά και να διαπιστώνουμε τις δυσκολίες που έχουν τα φιλικά προγράμματα από πλευράς προγραμματισμού.
- Μάθαμε ότι πριν προχωρήσουμε στη λύση ενός προβλήματος πρέπει να γνωρίσουμε τον χώρο που ανήκει το πρόβλημα.
- Μάθαμε ποια είναι τα βήματα ανάπτυξης λογισμικού και σε ποια θέματα κατανέμεται ο χρόνος στον κύκλο ζωής ενός προγράμματος.

- Είδαμε πως μπορούμε να αυξήσουμε την ταχύτητα ενός αλγόριθμου, αξιοποιώντας πληροφορίες και τεχνικές. Για παράδειγμα με τη χρήση ψηφίων ελέγχου κωδικού δημιουργούμε ταχύτερα προγράμματα.
- Μάθαμε τι είναι το διάγραμμα VTOC και πως αυτό συμπληρώνει την τεκμηρίωση της λύσης ενός προβλήματος.

Λέξεις Κλειδιά

Κύκλος ζωής προγράμματος, Συντήρηση προγράμματος, HIPO, VTOC, Οπτικός Πίνακας Περιεχομένων, Δεδομένα ελέγχου.

Ερωτήσεις

1. Ποιες είναι οι φάσεις του κύκλου ζωής προγράμματος;
2. Σε ποια φάση της ανάπτυξης ενός προγράμματος αρχίζει η τεκμηρίωση και πότε τελειώνει;
3. Ποιος είναι ο κανόνας 3 Ε;
4. Ποια η σημασία του αριθμού που συνήθως ακολουθεί το όνομα ενός πακέτου λογισμικού;
5. Ποια η σημασία των αριθμών που συνοδεύουν το όνομα μιας ενότητας σε έναν οπτικό πίνακα περιεχομένων (VTOC).
6. Ποιο κριτήριο αξιολόγησης προγράμματος θεωρείτε ότι είναι σπουδαιότερο;
7. Ποιοι λόγοι επιβάλλουν την τεκμηρίωση του προγράμματος.
8. Τι εξυπηρετεί η τεκμηρίωση ενός προγράμματος;
9. Ποια στοιχεία περιέχει ο φάκελος προγράμματος;
10. Πιστεύετε ότι η συντήρηση του προγράμματος είναι σημαντική εργασία;
11. Είναι δυνατόν ένα πρόγραμμα να περιέχει λογικό λάθος και να λειτουργεί;

Βιβλιογραφία

1. *Εγκυκλοπαίδεια Πληροφορικής και Τεχνολογίας Υπολογιστών*, Εκδόσεις Νέων Τεχνολογιών, Αθήνα 1986.
2. Β. Λαοπόδης, *Ανάλυση και Σχεδίαση Συστημάτων*, Εκδόσεις Νέων Τεχνολογιών, Αθήνα 1992.

Διευθύνσεις Διαδικτύου

- http://diamond.idbsu.edu/˜philmac/cs125/980331.htm

Ιστοσελίδα με ενδιαφέρον υλικό σχετικά με τον κύκλο ζωής λογισμικού.

- http://129.219.88.111/softeval.html

Στην ιστοσελίδα αυτή μπορείτε να βρείτε πληροφορίες σχετικές με τα κριτήρια αξιολόγησης λογισμικού.

- http://www.acm.org/pubs/toc/Abstracts/cacm/62963.html

Τα κριτήρια αξιολόγησης μέσα από την ιστοσελίδα της παλαιότερου και μεγαλύτερου, σε παγκόσμιο επίπεδο, επιστημονικού και εκπαιδευτικού οργανισμού υπολογιστών, του ACM (Association of Computing Machinery).